Software Agents for the Warfighter

Information Technology Assessment Consortium (ITAC)
Lt. Commander Dylan Schmorrow, DARPA ITO Technical Coordinator
Maylene Duenas, NASA Ames Technical Officer

Software Agents for the Warfighter

October 2002

General Editor
Jeffrey M. Bradshaw (UWF/IHMC)

Principal Authors
Guy Boy (EURISCO)
Ed Durfee (University of Michigan)
Michael Gruninger (NIST)
Henry Hexmoor (University of Arkansas)
Niranjan Suri (UWF/IHMC)
Milind Tambe (USC/ISI)
Mike Uschold (The Boeing Company)
Jan Vitek (Purdue University)

Contributors
Geoff Arnold (Sun Microsystems)
Patrick Beautement (Qinetiq)
Keith Decker (University of Delaware)
Paul Feltovich (UWF/IHMC)
Richard Fikes (Stanford University)
Robert Gray (Dartmouth College)
Dominic Greenwood (Fujitsu Labs)
Pat Hayes (UWF/IHMC)
Robert Hoffman (UWF/IHMC)
Wayne Jansen (NIST)
Renia Jeffers (UWF/IHMC)
Nick Jennings (University of Southampton)
Lewis Johnson (USC/ISI)
Frank McCabe (Fujitsu Labs)
Jeremy Pitt (Imperial College)
Austin Tate (University of Edinburgh/AIAI)
Andrzej Uszok (UWF/IHMC)

Domain Expert Group
Major General Dean Cash, Commander, Joint Forces Command J9
Vice Admiral Art Cebrowski, (USN, Ret.) Director, Transformation Office, OSD
Admiral Archie Clemins, (USN, Ret.), former CINCPACFLT
Dr. Doug Cooke, NASA Johnson Space Flight Center
Rear Admiral Jack Dantone, (USN, Ret.) Founding Dir. of National Imaging and Mapping Agency
Dr. George Donahue, George Mason University, former Associate Administrator of the FAA
General Howell Estes, (USAF, Ret.) Former CINCSPACE
Admiral Jim Hogg, (USN, Ret.), Director of Navy CNO Strategic Studies Group, Newport, R.I.
Vice Admiral Tim Wright (USN, Ret.,UWF/IHMC), former COMSEVENTHFLT

Software Agents for the Warfighter
Part 1: Overview and Introduction

Background

The Information Technology Assessment Consortium (ITAC)

The objective of the ITAC is to provide a rapid, in depth evaluation of the potential impacts of emerging new IT technologies to meet DoD future needs. It will also be a key resource for senior decision makers to evaluate the impact of global IT activity upon their missions. The ITAC's independent assessment activities will be led by the Institute for Human and Machine Cognition (IHMC) and performed by a cadre of exceptionally well-qualified teams drawn from universities, government, and industry.

The founding members of the ITAC are NASA's Ames Research Center and DARPA's Information Technology Office, while the University of West Florida's Institute for Human and Machine Cognition functions as the executive management agent for the Consortium. It is anticipated that other government agencies will be added to the Consortium over time. Each participating government agency would contribute financially to the Consortium and jointly manage it programmatically.

The success of the ITAC rests on its ability to rapidly assemble virtual teams of experts to perform the technology assessments. These teams are drawn from all available sources and represent the best available expertise. The ITAC functions as a kind of virtual assessment organization with a small office and business development staff and little or no fulltime scientific staff.

Origins of the Study

In the spring of 2001, a comprehensive study was proposed to assess the potential of software agents for the warfighter of the future. The study was intended to:

- describe specific scenarios and use cases under which benefits of agent technology are most likely to accrue—as well as those for which agents are unnecessary or inappropriate;
- evaluate selected aspects of current agent research and development to determine which of these offer the most promise in the future;
- summarize the state-of-the-art in relevant commercial technologies and pinpoint gaps between available technologies and military requirements over the next several years;

- recommend strategies to increase the likelihood that key research breakthroughs and relevant commercial software solutions will be available in time to meet warfighter needs;
- identify risks associated with various aspects of agent technology, and the likelihood of future aggravation or mitigation of these risks.

Upon approval of the proposal, a core assessment team spanning the various areas of expertise required was rapidly assembled and began the substantial task of study and authoring to develop the body of the report. As part of this process, a seasoned corps of domain experts were contacted to help focus the study and provide military expertise in key areas on the basis of their experience. Following the initial authoring and discussion phases, a broad team of agent researchers performed in-depth reviews of the material relevant to their area of expertise. Their efforts were intended to ensure technical accuracy and clarity. In most cases these individuals made substantial contributions in the form of additions and recommended revisions, and thus they are listed as contributors. Following an additional period of critical review and revision, and with the kind encouragement of our sponsoring organizations at DARPA and NASA, we hope to publish this report for wider distribution in book form.

Overview of Software Agents for the Warfighter

Software Agent Technology

(To be added)

Synopsis of the Major Topics

(To be added)

Structure of the Report

The remainder of the report is structured into two major parts. Part 1 gives examples of various relevant military application areas and describes how different aspects of agent technology can be woven into each of them. Part two is organized around various technical components of agent technology.

The military application areas discussed in the next section of Part 1 are:

- Advanced Sensor Grids

- Unmanned Autonomous Systems
- Advanced Command Posts
- Mobile Operations
- Joint/Coalition Operations
- Logistics
- Information Assurance

Part 2 is organized around discussions of the following technical components:

- Agent Architecture and Capabilities
- Agent-Agent Interaction
- Human-Agent Interaction
- Semantic Integration
- Mobile Agents
- Agent Infrastructure

In each chapter of Part 2, we give a brief overview of the technical component, followed by a discussion of five topics:

1. Relevance to the warfighter. Use cases under which benefits this agent technology component are most likely to accrue, as well as those for which this it is irrelevant or inappropriate. A key theme that cuts across most of these application areas is that of network-centric warfare.

2. Technical description. Summary of the state-of-the-art in relevant research and commercial technologies. An evaluation of this aspect of current agent research and development to determine which of the many alternatives being pursued offer the most promise now and in the future, and which have most relevance to the selected military scenarios.

3. Risks. Identification of risks associated with the use of this aspect of agent technology, and the likelihood of future aggravation or mitigation of these risks.

4. Forecast. Summary forecasts for each element of this agent technical component in an extended table. Where appropriate, assumptions behind the forecast are stated, as well as indications of confidence in each the projections.

5. Summary and recommendations. Gaps between available technologies and military requirements over the next several years are pinpointed and strategies to increase the likelihood that key research breakthroughs and relevant commercial software solutions will be available in time to meet warfighter needs are recommended.

Military Scenarios

Advanced Sensor Grids

Overview

Advanced Sensor Grids are networks of large numbers of autonomous sensor devices with onboard processing, networking, and, potentially, storage capabilities. An example of such an approach is the Navy's Expeditionary Sensor Grid program.

A sensor grid may consist of several different types of sensors ranging from satellites to unmanned vehicles of all types to ground or underwater sensors with acoustical, seismic, and/or visual data collectors to undersea sensors. The collection of sensors may range from static to highly dynamic (with various mobile sensors such as UAVs joining and leaving the sensor network).

The goal of advanced sensor grids is to provide information regarding entities of interest in a comprehensive and timely manner. This information must be accessible at individually appropriate levels of detail to all echelons of warfighters from individual soldiers to high-level commanders.

Sensor grids pose several challenging problems. Software agent technologies are well suited to address many of these:

1. One of the key problems of sensor grids is the significant increase in the quantity of raw data that is gathered and made available. This raw data needs to be appropriately processed and made available to users based on their individual requirements. Confidence values need to be computed for observed phenomena based on the performance characteristics of the class of sensors in question as well as the historical accuracy of individual sensors. Moreover, data from several different sources (sensors of different types or at different locations) needs to be integrated in order to confirm or deny hypotheses regarding observed phenomena or to speed up classification of phenomena.

2. Another challenging problem facing sensor grids is management of the large numbers and varieties of sensors. Management includes turning sensors on and off in order to conserve battery power, detecting failed sensors, configuring sensors to provide the necessary information at the right resolution and update rates, and enforcing any policies governing access to the sensors by the various users. Management also includes compensating for failed sensors and balancing load across the set of available sensors - though much of this behavior can be achieved through self-organization among the entities in this kind of distributed system.

3. Bandwidth optimization is yet another problem. The available network bandwidth in a battlefield environment is often limited and could be easily overloaded by many sensors simultaneously transmitting large quantities of data. Transmitting data also

consumes power and hence should be controlled when sensors are operating on limited power sources. Again, sensor-grid management can be sensitive to resource availability and can adapt dynamically as necessary.

4. Agents in advanced sensor grids will interact to exchange and fuse information to cooperatively synthesize more precise and complete understandings of the environment. Active sensing might require the coordinated efforts of agents managing several sensors (e.g., triangulation). Agents will also negotiate over competing demands in highly-dynamic situations and coordinate over the combined application of sensors to improve precision and reduce uncertainty. For the warfighter, this means that the sensor grid is applying its assets in a coordinated fashion to ensure the best feasible picture of the ongoing operational environment.

Agent Scenario

A conflict has occurred and we are trying to track down an elusive adversary in mountainous country that is hard to patrol. We deploy a Battlefield Mesh (Libicki 1995)[1] built from millions of sensors, emitters, and sub-nodes dedicated to the task of collecting every interesting signature and assessing its value and location for targeting purposes. They are delivered by air and distribute themselves in nooks and crannies all over the area of interest. Many of these sensors have already appeared, albeit in rudimentary form. In the future, they will be cheaper and more sensitive, capable, collectively, of receiving signals from the various parts of the electromagnetic spectrum. Some would be optical sensors -- perhaps small charge-coupled devices tied to neural net processors; they could cover not only the visible range, but also near-ultraviolet, and all shades of infrared. Others would act like small radar detectors, either singly, or in computational harmony with their like-minded neighbors. Chemical sensors could detect the passage of machines or their men. Some would sense changes in magnetism, air pressure, sounds, vibration, or even gravity, and so on.

Why have we deployed so many sensor types? The easy answer is that warfighting conditions differ and we are unsure of our opponents' counter-measures. Some environments (e.g., open desert) and targets (e.g., surface vehicles) are easy to surveil; other environments and targets are tougher. To detect the latter may require exploiting the inherent differences between machinery and background that register on other senses. The hard answer is that single-sensor surveillance gives the target a single-dimension problem to solve. Tanks strive to be hard to see and thus employ camouflage and night movement. Submarines strive to stay quieter, using size, baffling, and ultra-smooth running machinery. Aircraft are stealthy by controlling their radar reflections and by engineering special shapes and coatings. Multi-sensor surveillance, however, complicates the single-dimensional problem by obviating techniques which dampen emissions of one type at the expense of another; moreover, the multi-dimensional problem they create becomes that much more difficult to solve.

[1] The following material is adapted liberally from Libicki 1995.

No one sensor need necessarily detect every emanation from a target. The more capabilities a sensor combines, the more expensive it gets, thus the fewer would be used and the easier each target would be to find and kill. Alternatively, specialized, perhaps even single-purpose sensors, can each collect signatures, exchange them with subnodes and collectively form a picture of a target in its environment. Eventually, the Mesh self-organizes and settles down to a relatively 'steady state' -- now it is ready to do useful work.

The Battlefield Mesh would contain cheap disposable emitters to illuminate targets with reflected radio waves, generate confusing signatures, and broadcast local positioning signals for precise targeting. Although accurate positioning systems are critical for the operation of a Mesh, full GPS capability need not be ubiquitous (GPS can also be jammed). Emitters that know where they sit and can broadcast relative distances to the other elements of the Mesh may suffice and positions can be inferred locally.

Some sensors may be equipped to move; they may have little cilia-like feet on land, fins in the water, and an airfoil (see below) in the air. Mobility would help right errantly laid sensors, take high ground (trees, houses, hills) in appropriate terrain, and cluster to where other cuing systems suggest the presence of target-rich environments. Movable sensors fitted with precise chemicals or explosives (e.g., for taking out a critical piece of electronics) could be the killing mechanism in some cases.

Perhaps the prototypical sensor would be a sandwich the size of a penny. On top would sit a photovoltaic energy source or optical sensors; next would be a sliver of microprocessor, perhaps a chemical or acoustic sensor, and then a penny-sized battery, a transmitter for an antenna jutting out to the side, and finally some anchoring pod on the bottom. Another design would make the sensor look like a weed plant of a meter or two length. The shaft would be the antenna; the head a spectral sensor device capable of seeing as far as a human can, and the roots would be acoustic and vibration sensors, as well as anchors. To use yet another analogy, sensors might be the size of bottle caps; emitters, the size of soda straws; and mini-projectiles the size of coke bottles.

Unmanned aerial vehicles act as relay stations and help collate, fuse, and transmit the collective vision of the Mesh -- returning a coherent image, much as the retina does to the brain. Or opponent's every move it detected by one of the sensors - every change in temperature, noise, or other signature creates a disturbance in the Mesh. Patterns can be detected and hideouts located. Offensive action can now be taken and the enemy defeated.

Unmanned Autonomous Systems

Overview

Unmanned Autonomous Systems (UAS) will be an increasingly important part of future military programs. Benefits of such systems include reduced manning, fewer casualties, the possibility of independent (split) operations, and the ability to deal more adequately with certain types of complex threats. Currently, there are over 45 UAV's and UCAV's, over 30 UUV's, more than 10 UGV's, and more than 10 USV's in various phases of testing and development,[2] besides a variety of unmanned space platforms. While a greater degree of standardization and modularity is foreseen in the future, hard requirements for different physical configurations and capabilities as well as the continued evolution of technology mean that multiple hardware platforms in each family of vehicles will continue to be a necessity. Agent technology can provide help in dealing with this heterogeneity by implementing capabilities-based modeling, management, lookup, and engagement of autonomous systems. Using sophisticated agent-based matchmaking and teamwork-based coordination approaches, these diverse platforms can be managed as a system of collaborating systems with common high-level interfaces and communicating in human-intelligible dialogues rather than as independent entities with diverse low-level API's and incomprehensible system messages.

Agent technology can be of assistance in many areas of automation, including providing automated launch and recovery, automated sustainment, and especially a means of common control across diverse platforms and capabilities. While some efforts toward commonality within platform types are underway (e.g., Navy Tactical Control System (TCS) for UAV's, Army Joint Architecture for Unmanned Ground Systems (JAUGS)), there is a need for an unmanned system-common control architecture across all vehicle types. In such an architecture, high-level requests from a single warfighter user could simultaneously task and control multiple heterogeneous vehicles, and each vehicle could provide services and data products in an integrated fashion. Such an architecture would go beyond current prototypes that attempt to control autonomous vehicles as a relatively simple "swarm" to allow for rapid sophisticated configuration and delivery of effects based on operator intention, across the equivalent of a battle group package. Going beyond mere coordination of flight and simple task execution, agents operating on-board or off-board can assist controllers in directing the high-level behavior of a diverse set of vehicles that are capable of effecting many error recovery procedures autonomously.

An additional role of agents is in increasing the span of human attention by providing assistance in monitoring and detection. Like air traffic controllers

[2] The acronyms for these unmanned vehicles (U*V) stand for: aerial, combat aerial, underwater, ground, and surface types.

and other radar monitors, these warfighters must identify multiple radar targets simultaneously and keep track of (i.e., maintain in short-term memory) multiple contacts that must be classified (light or heavy, friend or foe) and level of threat determined (if any). This difficulty is compounded by the stress of time urgency and threat to human life. Under high workload and stressful conditions, a decision must be made as to whether or not to intercept or redirect the contact. The likelihood of an error increases with the number of targets present. Human operators are prone to performance changes due to fatigue or altered workload, and to procedural errors, such as habit capture errors. By enabling detection of these changes, an intelligent system could transparently hand off tasks to autonomous agents to reduce workload or prevent mishaps.

Unmanned systems will operate in teams that must collectively achieve mission goals as specified by human operators. The systems will interact to share intelligence and to reactively readjust team activities to achieve objectives in the dynamic environment. The warfighter will be able to task these systems to pursue objectives that people should not pursue because the missions are too dangerous, remote, or prolonged.

In other areas of communication and control of unmanned autonomous systems, agent domains and policy-based management can be used to express complex requirements and to establish explicit behavioral constraints that put absolute bounds on the kinds of actions to be performed and the way in which they are performed. Mobile agent technology can help reduce network bandwidth usage in communication of information, reduce network latency for real-time control, reduce polling by installing custom monitors, and support disconnected operation.

Mobile agent technology can be used to implement dynamic configuration at the software level, by carrying new capabilities to unmanned systems at runtime: functions can be transferred from one platform (vehicle or sensor) to another, and new algorithms, missions, and functions can be dynamically downloaded.

Agent Scenario

UAVs are already acting as airborne relay stations, fusing and forwarding information being gathered from the Mesh. However, the UAS are also an extension of human warfighters into the battlespace and can be thought of as a kind of cognitive prosthesis. Having detected several active hideouts, it is now time to move against the enemy, and the UAS allow us to reach out and touch the enemy remotely.

However, friendly forces are also acting on the ground, and as the operation moves on to the point of attack, the UAS operators become tense. Agents located with the warfighters sense physiological indicators of increased cognitive and physiological stress and inform the UAS of this fact. The UAS alter their behavior, increasing their level of autonomy slightly and compensating for uncharacteristic inputs from the human operators.

In the few minutes leading up to the start of the assault, the UAS operators hand over targeting authority to friendly forces on the ground who are in the best position to provide exact and decisive guidance to the UAS.

The attack is launched. Meanwhile, other disturbances are noted by the Mesh that might indicate enemy forces deploying from a nearby cave exit, some apparently trying to escape, others trying to mount a flanking attack. Specific simple warning is given and the related information made available to those who are interested-- on demand. The friendly forces are not deluged with information at this critical time. Air support is called in and the UAS transfer information: sensor-to-shooter.

Advanced Command Posts

Overview

Currently, the configuration of C4I systems and command centers is largely determined in advance of conflict. This means that processes (which may turn out to be faulty) are unwittingly embedded into the software and enforce rote procedures. This is fine until the military imperative demands flexibility or until strategies, tactics, or processes change. However, this difficulty is compounded by the fact that most of battle management is event-driven and not process-driven -- a fact not acknowledged in most requirements-capture exercises. What is really required is a set of tools and mechanisms to enable Command Post reconfiguration on-the-fly, so that warfighters can respond to dynamic military imperatives in the battlespace.

Also, increasingly, terms such as 'distributed cognition' (Hollan, Hutchins and Kirsch 2000) or augmented cognition'[3] are being used for the way humans off-load cognitive tasks into the information and physical domain. Clearly, C4I systems currently *increase* cognitive load and make things worse. Instead, they need to be designed in a different way so that they are malleable and are receptive to the tasks we give them. So, in future Command Posts, humans will not be deluged with information, because they will be able to shape the behavior of the command center and C4I systems to their will. They will be able to drive the enterprise – i.e., be active decision-makers, not dumb process followers -- such that the C4I tools and infrastructures fit their behavior to the human decision-making requirements and imperatives, not the other way round.

Multi-agent systems, therefore, aim to address three facets of the command and control problem. First, there is the problem of delegation and execution of directives down from higher levels of command. Agent technology aims to enable a higher degree of autonomy and flexibility than has been possible with rigid command systems of the past. The claim is that on-board automated planning and inference capabilities in such agents would allow them to cope more adequately with situations involving significant uncertainties and unforeseen contingencies. Second, there is the corresponding problem of information flow, as data and analytic results mov upwards from levels of sensing and observing to inform the actions of commanders at different levels. It has been proposed that agent technology assist in providing secure intelligent filtering, data mining, abstraction, and routing of information to those who need to know. Finally, there are the problems of collaboration among diverse command and control operatives. Multi-agent systems typically incorporate some kind of theory of coordination and teamwork that aims to provide a greater degree of robustness to the complexities of interorganizational interaction.

Agent Scenario

Agent technology has enabled the size of the deployed command post to be reduced and has streamlined command post operations by allowing the

[3] See http://www.darpa.mil/ito/research/ac/index.html.

human commanders to offload routine and more mechanical activities to automated agents. These agents, in turn, can interact with each other to monitor, assess, deconflict, and redirect activities occurring across the command.

Each warfighter has its own interface / mediator agents associated with it that are aware of the current shared work context. Some are involved in controlling the UAS, and others are monitoring the execution of the plan and trying to detect deviations or new events which will require corrective action. They are assisted in this by their agents who are monitoring resource allocation, network setup, and overall control functions across the distributed entities. The supporting agent architecture accounts for this by modeling other agents at different levels. At the simplest level agents have abilities to predict, constrain, and set policies for behaviors of other agents. At more complex levels agents model and understand other agents and account for their motivational and informational states. Humans can neither do this, nor would they want to, and can rely on the monitoring and housekeeping agents to do this task for them, confident that malicious or atypical behavior will be reported.

While the offensive operation is underway in the mountains, the command post is 'in the back seat.' However, they are still monitoring the wider Mesh though their 'decision-desktops,' which are configured to meet their individual warfighting requirements. Their interface agents monitor what the humans are doing, their stress-levels, where their attention is on the desktop and the tasks they are carrying out. The agents will update information on demand and will cue other agents to start gathering updates on areas of interest, often before the human actually asks for it.

Mobile Operations

Overview

Mobile operations are characterized by low and intermittent connections as the warfighters may not be able to communicate or may be forced to communicate in short bursts to avoid detection. In this setting, mobile-agent technology eases the task of developing software systems. A mobile agent, capable of performing some task in its entirety, can be sent to or from the warfighters to avoid the need for continuous data transmission. This agent can continue its task even if the network link becomes completely unavailable, either because of mission requirements for emissions control or because of physical interference, physical separation, or hardware problems.

Moreover, the agent can change its behavior or relocate itself as mission and network conditions change. For example, if emissions control becomes essential, an agent running on a remote sensor platform might relocate itself to a less powerful sensor on one of the warfighters themselves, avoiding all RF emissions at the expense of lower sensor resolution.

Finally, the entire communications infrastructure, not just application-level tasks, can be implemented with mobile agents. As conditions change, new communication agents can be distributed to all the warfighters in a particular unit. For example, if emissions control has become important, the unit leader's device might deploy communication agents that compress and buffer all data messages so that the messages can be sent in short bursts at mission-appropriate times.

Mobile agents allow bandwidth conservation and latency reduction in many information retrieval and management applications. For example, in a low-bandwidth environment, a mobile agent that performs a multi-step query against multiple databases can be dispatched close to the location of the databases, avoiding the transmission of intermediate results across the network. Similarly, in an unreliable network environment, the same mobile agent can continue its query task even if the network goes down temporarily.

In the reverse direction, code that provides high-level access to a particular database or service can be dynamically dispatched to a warfighter's machine, further reducing the warfighter's reliance on the network. For example, if a warfighter is directly or indirectly making heavy use of a particular database, code to cache query results can be dynamically dispatched to the warfighter's machine. Queries that can be answered from the cached results will never be transmitted across the network, even though the warfighter had no pre-installed query caching capabilities.

Agent Scenario

In a Military Operation in Urban Terrain (MOUT) scenario, adversaries are hard to locate and engage because they are embedded within the infrastructure—not urban infrastructure but buildings around cave entrances and structures within the caves themselves.

From activity detected by the Mesh, snipers are suspected along the planned route of advance of one of the ground units. Fortunately, each soldier is equipped with a set of personal software assistants (each an autonomous software agent) capable of communicating with the soldier out in the open environment. These personal assistants, partly because of commands from the soldiers and partly through their own initiative, react to the nearby sniper threats by negotiating with nearby UASs which relay the information from the Mesh. Some UASs agree to help the soldiers by flying above the area where the soldiers are located; they provide critical information and thus improve the soldiers' situation awareness. In one case a party of people is sensed which represents no threat. In another case a sniper is located and identified, so the soldier is warned and he or she may be able to take the sniper out in advance.

Unfortunately, in one case, the UASs fail to identify the sniper's location relative to a group of enemy soldiers. One friendly soldier finds that he is in a position to see what is going on and instructs his personal assistant to inform other soldiers' personal assistants about dangers in this area of the terrain. The personal assistants find out about each other through registering with the UAS relay / Mesh network. Together, either with the command from their users or on own initiative, they may now plot an alternative route for advancing toward the objective. Of course agents must always take the safety of humans into account in their deliberations.

This replanning of the mission requires software agents to communicate what is happening with command and control units outside the area. The fluid and constantly evolving nature of the situation as new information flows in poses special challenges. A warfighter will have difficulty knowing the relevant status of the ongoing mission as he or she undergoes sporadic disconnections from the network and changes to positions and relationships with other warfighters. Through the interactions between the personal assistants and the warfighters they serve, a warfighter can better understand the changing circumstances and make other levels of command aware of the needs and progress of the distributed mission.

Joint/Coalition Operations

Overview

Many of the Joint coalition issues are the same as those discussed for the Command Post discussion. However, they pose specific problems. The nature of coalition operations implies the need to rapidly configure incompatible or foreign systems into a cohesive whole. Several key principles apply:

- the issues relate to those involved in the creation and maintenance of a coherent coalition organization (with real and virtual parts) from the diverse and disparate 'come-as-you-are' elements provided by the coalition partners (people, processes, and systems);
- all coalitions are a dynamic (ever-changing) mix of heterogeneous and disparate elements and maintaining the cohesiveness of the coalition requires a continuous, pro-active readjustment process;
- multiple coalitions may be active at any one time ('competing' for resources, etc.) and a decision in one may affect another concurrent operation;
- partners may be part of a coalition, but their contributions may be anonymous (to protect sources, etc.);
- coalition elements should be supported by appropriate IT in achieving 'unity of action';
- "interoperability of the mind" is as important as interoperability of systems, if not more so;
- the difficulties are compounded in the virtual organization of the coalition since there will be a mix of doctrines equipment, operational procedures, languages, etc.;
- most coalitions will have commercial / civilian elements, and appropriate interoperability will have to be provided with their infrastructures,
- the Command Process is 'command led' and is characterized by a mix of deterministic and naturalistic decision-making styles,
- coalitions consist of loosely connected elements working semi-autonomously, and within their delegated authority, toward a common goal (as defined in the Commander's Intent); elements need to rendezvous (and synchronize) only occasionally and must be free to optimize locally / snatch fleeting opportunities etc.,
- supporting the achievement of command agility (working in a flexible, unpredictable manner -- where the decision-maker is the only thing on the critical path -- leading to decision-dominance over the opponent) is vital; this is especially so in Execution and Battle Management;

- enabling commanders to access relevant coalition-wide information as and when they demand it to support their decision-making is crucially important to a successful outcome. Information should not be pushed according to some rigid, pre-determined process.
- there is a pressing need to set up coalition organizations / systems rapidly (in order to respond decisively to emerging crises);
- systems provided to support the humans must be robust, secure, dynamic, and adaptable and must not constrain human actions;
- there must be no single point of failure in the coalition, and performance must 'degrade gracefully' and / or systems must self-heal.

Coalition operations, therefore, are complex and heterogeneous and change dynamically, so it is difficult to achieve and maintain coherent operations with shared information and battlespace visualizations. In involving software agents, it is important to focus on these coalition-specific issues, particularly the fact that we embrace heterogeneity, not exclude it.

Agent Scenario

To guide the command and control operations of the coalition, intelligence gathering by partner nations must be coordinated, and information must be accessible to appropriate commanders. The networked communications infrastructure can support this in principle, but each country has various firewalls in place to avoid unauthorized access. Agent technologies can secure access to intelligence, permitting the flexible modification of policies that dictate the degree to which partners can differentially share information and the protocols that they should obey. Furthermore, these technologies can support the translation and transformation of information to allow consistent semantic interpretation among coalition partners.

During the planning phases for coalition activities, agent-encapsulated capabilities for logistics planning, battle planning, etc. can be teamed to support the construction of effective coalition tasks that can then be handed over to associated functional units. Because the partners might follow different doctrines, these functional units might formulate specific plans that unintentionally conflict with each other or with humanitarian aid groups operating in the theater, possibly in catastrophic ways (e.g., friendly fire). Agent technologies can support the planning and coordination phases to detect and help resolve unintended interactions, working in mixed-initiative mode with human operators to modify and synchronize tasking as needed and to monitor and repair ongoing mission plans. Agents can also help speed the right data to the right destinations.

It has been revealed that a new partner is to join the coalition, as this will provide intelligence about the area, over the border, on the other side of the mountains that are being attacked.

The information available from this new coalition partner must be integrated as quickly as possible into the shared coalition grid. Several agent-based mechanisms are used to do this. Translator and mediator agents are employed to create an area of local interoperability between the new partner

and the coalition. This is the quickest way to get the most important information into the grid. At the same time, Sysads create wrappers and 'grid helpers' to agent-enable some of the new partner's technology, leading to a greater level of interoperability.

Logistics

Overview

Logistic operations are carried out by highly decentralized systems. The scheduling and allocation of resources must be carefully coordinated so that personnel, supplies, and support converge in the right amounts at the right times in the right places, despite originating from various locations and moving using different means. Using effective interactions between automated agents that support logistics means that the warfighter will only be placed in combat situations where the other critical ingredients to success are available.

Indeed, one could foresee a situation where simple agents embedded in the barcode label of every package could interrogate the surrounding packages, the pallete and the transport platform to see if it had been routed 'correctly'. A high-priority package could, therefore, alert a decision-maker if it had been incorrectly placed on a slow transport platform such as a ship. This could lead to packages being delegated with a certain degree of self-determination, leading to a certain amount of self-organization in logistics delivery.

Agent Scenario

It has been decided to deploy a combined force of Army, Navy, Marine, and Air Force units within the region to provide support for a wider military campaign in the area. The first challenge is deployment of operational units and the establishment of materiel distribution networks. Given the operational requirements, logistics plans are generated for different command levels, but the dynamic nature of the situation requires constant updates of the plans. Front-line commanders interact with the software agents associated with their units to provide updates to tactical situations and resource requirements. At command levels, agents identify the deviations from existing plans and notify the appropriate personnel. These agents share modified plans and new operational data with agents associated with the operational units. Once the distribution networks have been established, transportation plans for the delivery of materiel must routinely be generated and maintained. Once again, the dynamic nature of operations requires that the agents associated with each unit be able to communicate relevant changes in the plans and the environment.

The distribution networks also require real-time integration between military units and suppliers (DoD and commercial vendors) to ensure just-in-time delivery of materiel in support of military operations. A big problem is the high-volume of data that needs to be integrated and dealt with. Through a combination of shared ontologies, rich semantic markup of the data, and agent capabilities, this data is processed more efficiently and with fewer errors, and with less human intervention required.

Information Assurance and Survivability

Overview

Future networked systems will range from interconnected grids to wireless swarms of semi-autonomous sensors and wearables. As the underlying networked software systems scale in the hundreds of thousands of nodes and each of these nodes running small parts of a number of interdependent distributed applications spread over multiple administrative domains, new behaviors are likely to emerge, giving rise to hitherto unforeseen vulnerabilities of the overall infrastructure. The fundamental goal of information assurance is to make large-scale information systems robust and able to tolerate security breaches and hostile attacks without failing. Information assurance is a key requirement for all warfighter scenarios in which enemy forces employ network warfare technology. Three key challenges have to be addressed:

How can trustworthiness be achieved for large-scale, complex systems spanning multiple organizations? Security policies will have to dynamically adapt to enforce global security properties at trust- and administrative-domain boundaries. Furthermore, the security infrastructure must react automatically to reports of security flaws and breaches by installing patches, modifying firewalls settings, and training intrusion detection systems to recognize new attack signatures.

How can systems be made robust in the face of environmental disruption, attacks, and instability due to rapid growth? The trustworthiness of a system must be enforced and/or validated as that system evolves and adapts. Composition (and decomposition) of systems out of (and into) components must maintain specified levels of trustworthiness. The difficulty is that most security properties are global and specifications are often local; thus there is a need for tools to evaluate dynamic, and sometimes short-lived, composition of heterogeneous software components.

How can secure ubiquitous computing be provided with small mobile devices, ad hoc communications, and mobile code? The warfighters will be equipped with small mobile devices that will require frequent software upgrades to fix defects and overcome hardware space constraints. The code and platforms implementing advanced information systems must be secured from external threats, and technologies for detecting when a device has been compromised are needed.

Agents pose both new risks and new solutions to security problems. Granting agents sufficient latitude so that autonomous actions can be taken without constant human supervision also creates the possibility that rogue agents that manage to penetrate security defenses will also be allowed a longer leash for mischief. Agent mobility opens a door of attack to malicious hosts designed to prey on visiting agents. Uncontrolled and unauthorized agent cloning or deliberately programmed resource-consuming agents can be used to mount denial-of-service attacks on unprotected hosts. These are but a few of the many scenarios that must be analyzed and countered.

On the other hand, research in agent-related technologies has begun to demonstrate promising new approaches to preventive defense of critical systems and responsiveness to attacks once they occur. For example, research in control of agent "domains" by explicit policy decisions and sophisticated policy enforcement mechanisms claims the advantages of formal offline policy analysis and verification, fine-grained real-time control, and perhaps most important, high-level administrative tools that can be effectively used by non-security experts. As another example, multi-agent approaches to intrusion detection promise increased modularity and coordination of information and strategy sharing among defensive components.

Well-defined interactions and protocols between computational agents that comprise the information network permit the kinds of authorization, understanding, and awareness that are necessary for detecting and thwarting security risks, ensuring that the warfighter can depend on the information he or she is provided.

Agent Scenario

The attack on the enemy installation in the mountain caves has been partially successful, but it is suspected that some enemy forces are escaping across the border into the neighboring country.

Information reports are being received from the new coalition partner covering exactly this area. UASs are maneuvered into position based on the information being received, but the enemy forces are not detected. For political reasons, it is not possible to deploy more Mesh entities and so there is a risk of the enemy escaping owing to lack of information about their position.

At this point, friendly-force Sysads become aware that a denial-of-service attack it being mounted and become suspicious about its source. Information agents are queried about the reliability and quality of information that has been being received from the new coalition partner, and they reveal serious inconsistencies.

The source of the denial-of-service attack is tracked down to the new partner, and agent policies are changed to stop the malicious agents from using coalition resources. Meanwhile, the interoperability gateways are disabled. The Command Post and the friendly soldiers in the mountains are informed of this, and information from the suspect source is now discounted by all agents. The search proceeds.

References

C. Libicki, August 1995 "The Mesh and The Net" Martin Center for Advanced Concepts and Technology, Institute for National Strategic Studies. NATIONAL DEFENSE UNIVERSITY.

Hollan, J., Hutchins, E., & Kirsh, D. (2000) Distributed cognition: Toward a new foundation for human-computer interaction research. University of California, San Diego.

Software Agents for the Warfighter
Part 2: Technology Components

Agent Architectures and Capabilities

Brief Overview

Description

Agent architectures provide the blueprints for the design and development of individual agents. The role of an architecture is to define an agent's capabilities and to delineate a separation of concerns; thus it defines the features of individual components, the information and control flows between them, and the modes of interaction between the amalgamated components and an environment. Agent capabilities instantiate components of the agent architecture, either for operation within the architecture itself, or as a means for providing a capability to the users of the agent. There is no universally applicable agent architecture and so many different types of architectures have been developed. Each architecture has its strengths and weaknesses that make it suitable for particular roles or particular types of problems. Given this fact, this chapter considers a progression of agent architectures, from those that define simple reactive agents all the way up to self-modifying, time-constrained, proactive agents. At each stage in the progression, we consider some of the capabilities that agents must have to succeed at that level and summarize the state of the art of such capabilities. The chapter concludes by outlining the main challenges still remaining before agents with various capabilities can satisfy the needs of the scenario applications.

Relevance to the Warfighter

In a sense, the topics in this chapter are at the crux of developing agent-based technologies that are relevant to the warfighter. While issues of interaction and interoperability are clearly important, none of these matter if the agents with whom humans and other agents interoperate are unable to perform the tasks for which they were designed. Thus, in this chapter, we talk about capabilities such as information fusion, management of uncertainty, planning, time-critical responsiveness, and adaptation, without which the use of agent technology for applications like advanced sensor grids, command and control, and autonomous and mobile operations, would be impossible. Successful agent technologies will allow the warfighter to delegate routine or hazardous tasks to agents, which in turn will work "behind the scenes" to provide timely information or suggestions about courses of action to the warfighter as dictated by the evolving situation and the expertise embodied in the agent's capabilities.

Risks

The main risks of current trends in agent architectures and capabilities are the following. First, there is a tendency to want to create and employ the most sophisticated agent for an application domain; a better understanding of the different gradations of agent complexity and how they could be matched to need could lead to more robust, efficient, and cost-effective agent systems. Second, except for the simplest kinds of agents, verification and

validation techniques fall short of permitting us to characterize in a principled way exactly what we can expect of an agent. This situation is similar to that of all software engineering paradigms when they involve the development of complex systems. Nevertheless, quantifiable tradeoffs between whether "unpredictable but powerful" outweighs "predictable but limited" must be made. Third, as individual agent capabilities mature, it will become increasingly important, but difficult, to insert these modules into alternative agent architectures while remaining true to the spirit and assumptions of the different agent architectural frameworks. Fourth, it is still not a common practice to re-use architectures or architectural components. This leads to significant duplicated effort and hinders the general rate of progress in architecture development.

Forecast

Each year, the capabilities of agents expand to address problems that were once considered far too difficult for mere computational agents. While agents have indeed come a long way the vision outlined in this document extends far beyond the current state of the art. As existing architectural designs and agent capabilities mature, they will be augmented by the cutting-edge results being developed (and yet to be developed) by researchers to move agents the next steps, so that the ambitious visions laid out in this document can be reached.

Summary and Recommendations

Agent architectures at all levels of sophistication, from those supporting reactive agents up to self-modifying and self-aware agents, need further development. Reactive architectures need to support verification and validation of performance within guaranteed time-critical boundaries. Agents with world models need more efficient and powerful mechanisms for combining evidence and drawing inferences about the world. Goal-based agents need to be able to form plans in highly complex situations despite resource limitations. Agents operating under less certainty need to formulate policies to do the best they can in response to the uncertain unfolding of their environment. Resource-limited and time-constrained agents need to balance responsiveness with farsightedness - all in haste, and self-modifying agents need to make changes that are justifiable and still ensure sufficient predictability in behaviors upon which operators depend. Much progress has been made along many of these fronts, but much remains yet to be done. In particular, further work should be strongly encouraged in developing agents that are capable within the constraints of dynamic and uncertain application domains. In addition, encouraging research in fundamental agent theory is critical to the future development of sound agent and multiagent architectures, and a future "science of agent architectures".

Relevance to the Warfighter

Advanced Sensor Grids

Advanced sensor grids can bring together far-flung information to potentially improve a warfighter's awareness dramatically, but they risk overwhelming the warfighter with too much data. Advanced sensor grids can be implemented using agent architectures and capabilities that incorporate complex perceptual methods for correlating disparate data and information to present a more consolidated picture. The agents should use world models designed to explicitly represent uncertainty and to index information based on temporal and spatial characteristics. By modeling the activities as well as unique characteristics of warfighters, agents in advanced sensor grids should be capable of actively monitoring for just those crucial aspects of the world model about that the warfighter needs to be aware in the current context.

Unmanned Autonomous Systems

Unmanned autonomous systems are typically dispatched in hazardous applications, where timely reaction to mission-critical emergent phenomena is vital, and where assurances about correct performance is crucial as the systems operate autonomously. Emerging agent architectures and capabilities promise to provide time-critical responsiveness, along with the ability of a system to autonomously reformulate its plans of action and possibly even to modify its behavior based on its experiences. These capabilities will free the warfighter from overseeing the activities of the system, and allow the warfighter to depend on the autonomous operation of a system

Advanced Command Posts

Command post operations can involve the attention of numerous warfighters to monitor and control the battlefield situation. The use of agents with advanced capabilities for information fusion, situation assessment, planning, resource allocation, and so on, can offload tasks (especially more routine tasks) from humans, freeing up warfighters for other duties. Such agents will need to be verifiably capable of performing command and control tasks at a level at or above human performance, and where they fall short will need to know their limitations and when to bring human intelligence into the loop.

Mobile Operations

Mobile operations put particular stress on architectural capabilities for drawing inferences about a dynamic environment that is only partially accessible for sensing. Associating agents with a mobile warfighter can allow those agents to continuously process incoming information and fuse that information with past experiences and knowledge of the environment to provide the warfighter with awareness that takes into consideration the changing context (including physical location, role, and objectives). Given the complex and often incompletely-achievable objectives of a mission, agents associated with a mobile warfighter can formulate and continuously revise projections about the future, along with measures of likelihood, to help the warfighter make the most rational decision possible, despite rapid changes to the situation and to relationships with other warfighters due to mobility.

Agent Architectures and Capabilities, continued

Joint/Coalition Operations

Joint/Coalition Operations will assume many of the same capabilities as advanced command posts, augmented by agent capabilities for coordinating action and sharing information across multiple warfighting units. Those further considerations are predominantly the subject of other chapters (agent-to-agent and agent-to-human interaction); it is important, however, that the agent architecture support these augmented capabilities for multi-agent reasoning.

Logistics

Logistics operations require advanced capabilities for planning and resource scheduling; such capabilities can be embedded in agent architectures to support automated planning and scheduling in complex, uncertain, and rapidly evolving situations. Furthermore, logistics operations require monitoring the execution of logistics plans, and rapid recovery when some operations fail. Such operations challenge not only the management of world models, but also time-constrained reasoning for quick recovery. Used appropriately, these agent technologies can offload from human warfighters the logistics operations that are either routine or require a level of continual attention surpassing human abilities. In addition, the warfighter in the field will benefit from the use of these agent technologies because the appropriate materiel will be where the warfighter needs it, when it is needed, more reliably.

Information Assurance

The integrity and pedigree of data are important for information assurance. Agents need to be able to associate justifications for all their inferences, and remember dependencies among data for backtracking. These justifications allow an agent to track down erroneous conclusions, and retract data that lead to bad conclusions. A warfighter should be able to act with agents to probe their justifications for assertions about the world or recommendations about actions, to allow the warfighter to understand not only the agent's results, but also the limitations behind those results. The ability to provide justification for decisions should be part of an agent architecture for agents that are providing decision support. Providing such justification can be particularly challenging for agents that are self-modifying, where they need to be able to show evidence for how previous experiences have led to changes in their internal capabilities.

Technical Description

The fundamental behavior of an agent is to act to bring about some desired change to its environment. Typically, this requires the agent to perceive whatever it can (through its own sensing or through receipt of information in some other way, such as through messages from other agents) in order to form a view about the current state of its overall environment. The agent will then deliberate based on this view to decide on an action to take that it believes will change the environment for the better (at least based on the agent's preferences). Finally, the agent will act on its decision. An agent will cycle through this perceive-decide-act cycle repeatedly.

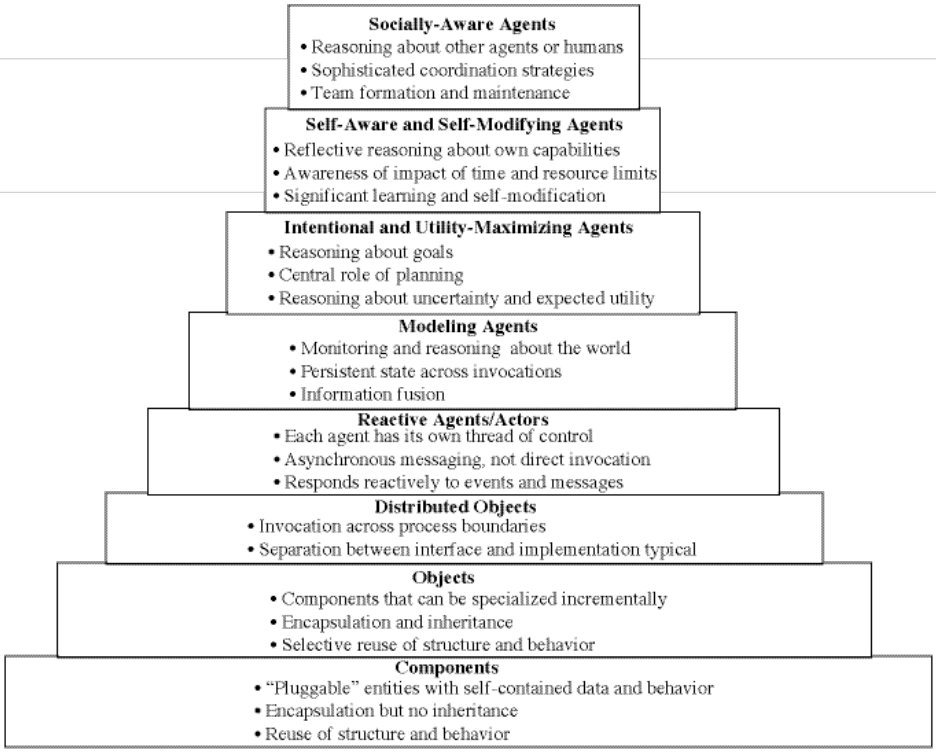
Other chapters in this report describe how an agent interacts with other agents and with humans in the course of its activities. In this chapter, we look more directly at what goes on inside an agent: the kinds of capabilities that agents might need to have to operate in the challenging applications we envision for them, and the way those capabilities fit together within architectures to structure agent programs that mesh with an environment and task.

Before we begin, however, we should point out that the principled development of agents and of agent architectures is a subject that is currently undergoing investigation on many fronts and by many people. Therfore, our treatment here will only discuss a few of the representative ideas. Readers interested in more can consult various collections (Huhns and Singh 1998), conference proceedings (Agent. 2001 and 200), and monographs (e.g. Wooldridge et al. 1996, 1997, 1998). Furthermore, the subject of developing capabilities that help make computational agent programs more "intelligent" has been the ongoing study of the field of AI, and here we cannot hope to do full justice to that field. Again, our hope is to highlight relevant and representative examples from the literature.

As has been done by others (e.g., Russell and Norvig 1995), we will approach the subject of agent architectures and capabilities by starting out with very simple agents that are capable of being successful only in very limited ways in simple environments. We will then successively ratchet up both the expectations on the agents and the challenges posed by their environments. As we go along, we will be identifying architectural components and agent capabilities that are germane, both in terms of the state of the art and in terms of opportunities for research that will support deployment of increasingly sophisticated agents in warfighter applications.

Realistically, large multifaceted systems for the types of applications of interest in this report would typically be composed of standard components and objects, supplemented by a variety of agent types (figure **). These would range from agents that are relatively simple "automatons" with very specific and limited roles in the system, up to agents that are more "human-

like" in the diversity of capabilities that they embody and the degree to which heavy demands can be placed on them. Between these extremes are other varieties of agents. Just as in traditional human organizations where a smaller number of top-level decision makers have the most latitude for action and are in turn supported by successively large numbers of more highly-constrained agents at layers below, we would expect that the number of agents of each type in the system would decrease as one moves up the pyramid. The figure is also meant to illustrate that there is a more or less unbroken continuum between sophisticated distributed object architectures and simple agent architectures. As distributed object systems become more capable and as agent-based systems of different varieties become more common, the line between what is truly an object and what is truly an agent (at least in these middle-levels) will become increasingly difficult to draw. This issue will not be of any great concern for most people.



In this same spirit, we will start our review by considering the simplest agents, which interact directly with the world, generally manifesting at least the minimal marker of autonomy common to most agents: an independent thread that responds to asynchronous messaging *(reactive agents)*. The next levels of sophistication for agents come from maintaining internal state *(modeling agents)*, then from explicitly representing and reasoning about goals *(intentional agents)*, and finally adding to that the ability to reason

about uncertainties to assess expected utilities *(utility-maximizing agents)*. Above these levels, agents are capable of reasoning not only about their external environments, but also about their own capabilities, including how their own resource limitations impact how they should decide to pursue their current (often time-critical) goals *(self-aware agents)*. Beyond that, they can reason about how they might modify their own internal machinery (their "programs," so to speak) based on experience in order to perform even better in the future *(self-modifying agents)*. Finally, at the top of the pyramid is the agent that not only reasons about itself and its environment but also about the other agents (including humans) around it, including possibly their goals, uncertainties, limitations, and adaptations *(socially-aware agents)*. Issues relating to agent sociality are more properly and completely covered elsewhere in this document (specifically, the chapters on human-agent and agent-to-agent interactions), and so our treatment here will work upward from reactive agents and end with self-modifying agents.

It should be noted that our hierarchical ordering of the spectrum of agent capabilities is meant to be a heuristic organizing device for expository purposes, not a strict model of dependency relationships among the layers. One could arguably contend for different orderings for different purposes. Moreover, basic aspects of social capabilities (such as agent messaging) and of rudimentary self-modifying behavior (such as reinforcement learning) are often present in simple agents. Furthermore, some important system features not explicitly called out here, such as mobility, are a significant and pervasive feature that can be exploited by entities populating any of the levels. For example, code mobility in simple systems can be exploited by distributed object systems for purposes such as load-balancing or software distribution; mobility can equally be well-used by sophisticated agents who have their own private reasons for wanting to move around the network.

Reactive Agents and Actors

Reactive agents employ architectures that offer a direct and rapid connection between sensing and acting. By analogy, this is similar to reflex-like (also known as "stimulus-response") behavior of biological systems. Typical control and automation systems would fall into this category; for example, a thermostat could arguably be classified as this rudimentary type of reactive agent.

Strongly influenced by the behaviorist approach, Rod Brooks' Subsumption Architecture is the best-known example of an agent architecture for reactive agents (Brooks 1986). In the Subsumption Architecture, the outputs of sensors are directly connected to the inputs for actuators. When multiple sensors are connected to the same actuator, the mechanism for how these influences are combined is hardwired into the architecture. No centralized control of behaviors is provided, and there is no internal representation of the state of the world, much less about how the actions of the agent might affect the world.

The hardwired arbitration between multiple stimuli to the same actuator must be carefully designed. For example priority might be given to some behaviors over others. Consider a robot with three behaviors: *avoid obstacles*, *follow another robot*, and *wander*. *Avoid obstacle* may suppress the other two behaviors, *follow another robot* may suppress *wander*, and *wander* does not suppress anything. As this illustrates, the design of the policies by which reactions are chosen and prioritized itself can require careful deliberation (and can be automated, as we shall see), but once designed can be implemented directly into the simpler reactive architecture.

The advantage of this type of reactive architecture is the speed between sensing and acting because there is no deliberation. Not surprisingly, it is difficult to scale this scheme to a system with nontrivial sets of behaviors or where sequential behaviors are needed. Nonetheless, for many kinds of applications where the responses and their priorities are well-understood, a reactive architecture can be used to implement effective agents.

Actor architectures (Agha 1986; Agha and Jamali 1999) seek to build on lessons learned from well-engineered concurrent distributed object systems. They are particularly useful as a formal foundation for agents that perform well in applications requiring flexible and efficient naming, migration, and coordination capabilities

Modeling Agents

The simplest addition to a reactive agent architecture is to include an internal state. The internal state can serve several purposes. It can store a (partial) history of the perceptions and actions taken by the agent. This helps; for example, when the agent should take different actions in the same immediate state depending on how the state was reached or on what actions it has already tried. Remembering history can prevent an agent from repeatedly "banging its head into a wall," sometimes quite literally.

A second advantage to maintaining state information is that an agent can base its decisions not only on specific sensor inputs, but also on processed interpretations of those inputs, informed by other information or knowledge that the agent has at its disposal. By fusing information into a more comprehensive and confident view of its true situation, an agent can exploit this more complete awareness to make subtle distinctions between its choices of action and can thus act more appropriately given its whole context.

Finally, maintaining internal state gives an agent the ability to extrapolate from the history of states and the current state into the future, projecting forward to see what states could or must arise. In our classification of agents, this capability would typically be part of an intentional agent (because projecting forward implies that an agent can reason about which future states it prefers, so as to decide on its next actions to take). We therefore postpone discussion of this usage of state until the section on

Intentional Agents.

World model

The state information that an agent maintains needs to be stored in a repository. Such a repository is often called a "world model" because it represents the agent's internal model of the present, and possibly past and future, status of the world. A world model captures the agent's observations and memories about the context in which it is operating, augmented with inferences it has drawn about other aspects of its context based on the combination of information available in its world model. A world model is thus potentially useful for reasoning that requires more than one action and where reasoning about prior experiences is required (Albus 1981; Zimmer 1996; Balakirsky and A. Lacaze 2000).

Because a world model contains data, a variety of alternative data structures have been proposed for world modeling. At one extreme, data structures that impose little structure on the world model permit flexible application to a wide variety of applications. For example, a world model might simply be an expandable list of assertions (often in a language like predicate logic) about what is known or believed about the world. Updates to the world model involve adding and deleting assertions, as well as often tracking the implications of those changes in terms of making further assertions and retractions. Using this kind of world model typically requires that an agent scan through the list of assertions to find assertions that match against conditions that would lead it to take particular actions which, while flexible, can be slow for sizable world models.

At the other extreme, world models can be highly structured, where particular kinds of information are placed in predefined places. For example, in a blackboard-based agent architecture (Lesser and Corkill 1981), the blackboard data structure is typically partitioned into regions intended to house information that has undergone different levels of processing. The application of a blackboard architecture to speech understanding (Reddy 1976), for example, placed lower-level interpretations of speech such as phonemes into one region of the blackboard, and higher-level interpretations such as words and phrases into other regions. Through this structuring, the agent could more efficiently match data in the blackboard with knowledge about what to do with (and about) this data.

At this time, there is no agreement about how world models should be implemented for agents. A general-purpose data structure and representation language will, by definition, be widely applicable, but so far such approaches have tended to be slow for any particular application. On the other hand, special-purpose approaches can be streamlined, but they seldom carry over to other agent applications. It seems likely that, for the near future, there will be no convergence on a standard world-modeling technique. Instead, developers will formulate world models that strike the right balance between generality and efficiency, and face the thorny issues of how agents that represent what they know in different world models can still effectively exchange information (see the chapters on Semantic Interoperability and on Agent-to-Agent Interaction).

Information Fusion

One capability often manifested in a state-based agent architecture is the ability to perform information fusion. Information fusion is the process by which an agent combines different pieces of information, acquired perhaps at different times, or through different sensors, or even perhaps from different agents, into a unified, "fused" view of the situation. (See http://www.inforfusion.org/ and http://www.inforfusion.org/proposals/index.htm).

Information fusion is important in many applications where rapid synthesis of data from different modalities is required. Information fusion can be used to bring together disparate data from multiple sensors, where the data should be correlated as evidence for the same phenomenon being observed. A common technique for this type of information fusion is Kalman filtering used in image detection, distributed signal detection, or detecting trajectories of mobile targets (Draper et al. 1993). Information fusion is also relevant when interacting agents share overlapping views of the same phenomena and should combine their information to gain a better understanding of a situation (Granlund et al. 2001). Generally speaking, the information fusion of multiple types of data, with varying degrees of uncertainty, from multiple sources is a long-term research challenge. In multi-agent situations, information fusion might also need to be augmented with the ability to explain conclusions drawn from this process, to permit argumentation and supports agreement among the agents (see chapter on Agent-to-Agent Interaction).

Intentional Agents

Reactive agents, whether state-based or not, are limited in their capabilities because their mode of behavior is to monitor their sensors (and world model if applicable) and then respond when an information pattern matches the conditions for firing an action. The agent's actions are thus reflexive, rather than deliberately chosen to further the agent's agenda – at least as far as the agent is concerned. (The designer of the agent might have made such deliberate decisions and then embodied them into the agent.)

The next step up that we consider in agent capabilities, as embodied in agent architectures, is proactivity. In this context, by proactivity, we mean that an agent explicitly represents its goals and at any given time can evaluate alternative courses of action to select one (if it chooses to) to further its pursuit of these goals. Thus, rather than waiting for some stimulus in order to act, an agent can initiate action based on an assessment of what it could potentially do and whether any of those options will progress it toward meeting its objectives.

A goal-based agent uses its world model to represent projections about what new states of the world might arise depending on alternative actions it could take. In other words, the agent can project forward through possible sequences of actions and formulate a plan that it anticipates will result in achieving one or more of its goals.

More specifically, it is generally assumed that the agent's world model contains its *beliefs*. These beliefs match patterns sought for triggering projections about states of the world that are believed possible in the future. Furthermore, it is assumed that an agent has explicitly represented its goals or *desires*, such that it can properly evaluate its alternative plans of action in order to select one. This choice on the part of the agent results in forming an *intention* to actually pursue its chosen plan of action. For the most part, a goal-directed agent will at least remember its (recent) past intentions and could well have a more complex representation of the intentions it has, the reasons it has them, the conditions under which it should reconsider them, and so on.

Agent architectures that have these components are often referred to as *Belief-Desire-Intention* (BDI) architectures and are viewed as being able to exhibit intentional behavior (Dennett 1987). A BDI agent adopts intentions, often at an abstract level, that it then gradually refines into primitive actions that can be executed. BDI agents continually go through a cycle that typically involves the following steps (Rao and Georgeff 1998). First, beliefs are revised to accord with previous beliefs and new sensory input (world modeling and information fusion). Next, a set of options is generated (plans are formulated or retrieved and adapted) that follow from the agent's beliefs and consistently extend prior desires and current intentions. Options are pruned to a minimal set for adoption, where these options are milestones in accomplishing current intentions. At the third step, intentions are revised for consistency with the agent's most up-to-date beliefs and desires. The revised intentions (or intentions with renewed commitment) augment the agent's intention structure. There are now many systems and toolkits that implement a BDI view of agency. These include:

- JACK (see http://www.agent-software.com.au),
- JAM (see http://members.home.net/marcush/IRS/index.html),
- Zeus (see http://www.btexact.com/projects/agents/zeus/).

Although desires might be inconsistent, intentions must be consistent. BDI agents spend a lot of effort making sure that they have the right amount of commitment to their intentions. Therefore, BDI agents are reluctant to give up on their old intentions unless they are clearly unattainable or suboptimal given alternatives that might now be selected. Research on intention revision and commitments is a growing field. The research has suggested constructing agents with specific policies toward commitments (Jennings 1993). However much of this work is empirical (Kinny and Georgeff 1991; Jennings 1995) and further effort is required to develop a theoretical underpinning such that these policies can be shown to be optimal in a wide range of realistic environments.

As can be seen, one of the fundamental capabilities for a goal-based agent is the ability to form plans. Indeed, so central to the operation of a goal-based

agent is its means of formulating, executing, monitoring, and revising its plans that agent architectures for goal-based agents typically revolve around the manner in which planning and plan execution are done. Thus, in the following, we characterize several relevant planning approaches and, by extension, some of the alternative architectural designs for goal-based agents. For a survey article on the more recent work in the planning field, see (Weld 1999).

Logic-based Planners

Logic-based planning has its roots deep in the AI field, dating all the way back to some of the earliest and most influential planning work such as STRIPS (Fikes and Nilsson 1972) and the situation calculus (McCarthy 1963). Basically, a logic-based planner represents the states of the world and the actions that can be taken in terms of a set of propositions/predicates that hold and can be changed. Planning then amounts to proving that a sequence of actions must transform the world from an initial state to a state where desired goals are satisfied. The declarative specification for states and actions provides advantages such as ease of access and manipulation, while the underlying logical formalism supports reasoning about the correctness and validity of agent behavior. For example, simple mappings between perceptions and actions have been implemented with a theorem-prover (Amir and Maynard-Reid 1999) such that correctness can be validated, which is important for agents operating in safety-critical domains.

However, the drawbacks of logic-based approaches have included the challenges in dealing with dynamic knowledge and dynamic situations, and the difficulties in keeping the computation tractable as problems scale up. Among the attempts to extend logic-based approaches to deal with dynamic worlds have been the high level logic-based language GoLog (Reiter 1998, 2001), and Minerva which also offers a logic programming approach to addressing the scaling issues (Aciego et al. 1999; Leite et al. 2001). In general, these languages start from predicate calculus and extend the logic to include temporal elements in order to express temporal ordering of actions, enabling the expression of procedures that involve concurrent, iterative actions. The difficulty in developing these systems is semantic consistency.

Other recent work in logic-based planning involves the design of graph-based algorithms (such as in GraphPlan (Blum an Furst 1997)) that change the planning problem into a problem of proving that a consistent path can be found in a graph. Because of the recent rapid evolution in algorithms for solving satisfiability problems, into which these graph-based techniques can be mapped, a powerful set of planners has recently been developed, including Blackbox (Kautz 1998), which can formulate a logistics plan involving 105 actions in six minutes despite a search space of 10^{16} possible states.

A variety of implementations of these types of logic-based planning systems have been made available, including:

GraphPlan (www.cs.cmu.edu/afs/cs.cmu.edu/user/avrim/www/graphplan.html)

IPP (www.informatik.uni-freiburg.de/~koehler)

STAN (www.dur.ac.uk/~dcs0www/research/stanstuff/stan)

SGP (www.cs.washington.edu/research/projects/ai/www/sgp.html)

Blackbox (www.research.att.com/~kautz/blackbox/index.html)

Medic (ftp://ftp.cs.washington.edu/pub/ai/medic.tar.gz)

Heuristic Planners

Heuristic planners incorporate knowledge about the particular application domain to yield richer plans in less time. A famous precursor of heuristic planning methods was the General Problem Solver (GPS) developed by Newell and Simon (Newell and Simon 1963) in which the heuristic knowledge was captured in a table that prioritized "differences" between current and target states and suggested appropriate operations to reduce those differences first. These heuristics led to a "means-ends" search strategy.

Recent heuristic planning techniques are exemplified in SIPE-2 (Lee an Wilkins 1996), a technology developed by SRI that allows tractable planning for complex, resource-bounded problems. Quoting from the SIPE-2 homepage (http://www.ai.sri.com/~sipe/): SIPE-2 is a performance-oriented, general-purpose software system for generating and monitoring the execution of plans. It plans hierarchically (see below), using different levels of abstraction, and provides formalism for describing actions as operators. Given an arbitrary initial situation and a set of goals, SIPE-2, either automatically or under interactive control, combines operators to generate plans to achieve the prescribed goals in the given world.

Hierarchical Planners

Hierarchical planners take advantage of structure in the space of plan operators, allowing a plan to be constructed by successively refining an abstract plan into a detailed plan. Tracing their lineage at least as far back as ABSTRIPS (Sacerdoti 1974) and NOAH (Sacerdoti 1977), hierarchical planners are now predominantly equated with the so-called HTN (Hierarchical Task Network) planning systems (Erol el at. 1994). As mentioned above, SIPE-2 includes hierarchical planning as well; thus it can be seen that there is not a clean division between these planner categories.

HTN planning is especially appropriate for applications where there is a rich body of plan abstractions and hierarchical structure already in place. For example, in military settings, it can be the case that the role of an officer is to take a goal from a superior, formulate a plan to a particular level of detail, and pass on sub-goals to relevant subordinates, who in turn break these down into more concrete actions, and so on. HTN planning works this same way, in terms of successive refinement, and thus can be a natural match to a military setting or a setting with similar kinds of command chains. However, if in the given domain there is not a rich hierarchy of predefined doctrine, then HTN methods need to be augmented with other planning technologies (such as logic-based or heuristic-based planners) that can search through sequences of primitive operations to find previously undeveloped plans for

unique situations. Cypress (http://www.ai.sri.com/~cypress/) is an example of a planning system that combines both HTN planning and heuristic planning.

Partial-Order Planners

Partial-Order Planners develop plans by incrementally inserting and linking together tentative plan steps in order to span the difference between a given initial state and a goal state. This category of planners can overlap each of the previous categories. Like logic-based planning, partial order planners focus on proving that the goal situation is caused by applying the (eventually discovered) steps to the initial state of the world. Like heuristic and hierarchical approaches, a partial-order planner is not committed to building a plan either from the initial state forward or from the goal state backward, but instead can work from the middle outward.

Partial-order planning technologies are available for experimentation and prototyping (http://www.cs.washington.edu/research/projects/ai/www/ucpop.html).

Mixed Initiative Planning and GUIs

Mixed Initiative planning has been advanced as a way to combine the talents and capabilities of people with those of software planning systems. Planning technologies for complex application domains are increasingly turning to this strategy. It is called mixed-initiative because, given a target problem to plan for, the software system sometimes takes the reins and suggests aspects of a plan, and at other times the human can take control and extend and revise the developing plan.

Mixed-initiative, and Graphical User Interfaces that allow human participation in planning and specification of plans, are features of some systems and are a focus of ongoing research. Among the systems that provide GUIs for specifying and editing plans are SIPE-2 (described above) and the Interaction Plan Editor for TAIPE (Durfee et al. 1997). Further discussion of interacting agent and human systems for accomplishing complex tasks such as planning can be found in the chapter on Human-Agent Interaction.

Reactive Planners

"Reactive planner" is something of an oxymoron, since "planner" implies that there is some foresight going on, but "reactive" implies more of a stimulus-response kind of behavior. These planners do a limited abstract form of lookahead, interleaving planning and execution and postponing planning decisions as long as possible. Therefore, these systems elaborate abstract plans into immediate actions and wait to elaborate later plan steps until earlier ones are taken. This of course presumes that the planner has a sufficiently rich repertoire of plan elaboration to be able to handle a variety of possible contexts for future plans. Thus, these planning technologies are typically most useful when there are many ways of getting something done, and the hard part is deciding which way will make the most sense in the evolving context of the world.

Examples of such planning systems include the Procedural Reasoning System and its descendents (Georgeff and Ingrand 1990; Lee et al. 1994). Other examples include RAPS (Firby 1987) and Soar (Laird et al. 1987).

The last of these is somewhat different, in that it also includes possibilities for self-adaptation, where the system can learn what worked well last time and more quickly retrieve and do it in similar future situations. This capability is discussed in a later section.

Utility-Maximizing Agents

As agents are placed in increasingly challenging application environments, the ability to formulate and execute plans that are provably able to reach goals can become impossible for a number of reasons. In this section, we look at some of these possible reasons as well as agent architectural features and capabilities that can be employed to address these challenges. Subsequent sections elaborate on the challenges that environments can pose to agents.

Active Perception

Sometimes it may be difficult for agents to be able to access needed information to populate its world model. Up until now, this chapter has assumed that an agent has information from its sensors or other means to be able to model the current state (and possibly past states) of the world, so that it can make decisions based on complete and accurate knowledge. What if some information is *inaccessible*?

Obviously, one possible recourse is for the agent to plan additional actions whose purpose is not to achieve its primary goal(s) but rather to achieve subsidiary goals of having an appropriate world model. For example, an autonomous vehicle can move to an observation point in order to gather crucial reconnaissance data, or an agent supporting logistics operations can send out a request-for-bids to potential commercial transport companies. Rather than passively awaiting whatever sensor information might arrive, this kind of agent is engaging in "active perception" to go out and get the information it needs to make decisions (Bajczy and Liberman1988).

A particular type of active perception, specifically intended to resolve uncertainties in whether an agent's plan is working, is often referred to as "monitoring" or sometimes "tracking." In this, an agent builds expectations about the trajectory of its environment based on its world model and the actions it has planned to take. As it pursues those actions, therefore, an agent knows exactly what it expects the world to be like and can explicitly watch for the features that it considers important for the successful completion of its plan.

Interleaved Planning and Execution

At the core of some agent architectures is the idea of active perception for the monitoring and revision of agent activities *during* the course of executing a plan (instead of prior to planning). That is, as an agent pursues its plan, it stops between actions and perceives some aspects of the environment to decide either which alternative branch of its plan to take, or whether its plan should be revised in some way. In some agent architectures, this "stopping and perception" is explicitly built into its plan, while in others the architecture implicitly requires the agent to monitor progress after every

action it takes.

At one extreme is the case where, despite its uncertainty, an agent has a sufficiently complete model of the world to be able to identify, ahead of time, all of the conditional branches that might be required. An agent of this type uses what is called *conditional planning*, where a conditional plan is formulated before the agent begins to act, and then the agent decides which of the branching alternative subplans to pursue as it gets to a choice point and can reconnoiter.

At the other extreme is the approach called *plan monitoring and replanning*, where either an agent does not assume that it can predict possible plan deviations, or the space of deviations is too large to consider all of them. In such cases, an agent builds a single plan, based on its best guess as to the course of future events, and executes this plan. However, it monitors the environment against the projections of its world model and detects deviations from expectations. Because it had not planned ahead for these, if they are detected the agent halts its execution at this point, and formulates another plan that fits the new situation. This new plan might simply repair the previous plan by inserting actions to get it back "on track," or it might formulate an entirely new plan from scratch, depending on the cost of planning and the significance of the deviation.

Uncertainty Management

In many cases, despite all of its efforts to form a sufficiently complete and accurate world model in time for when it must make decisions, an agent will not be able to eliminate all uncertainty. There might be features of the environment that, despite engaging all get resources, cannot be perceived. That is, there might be features that are inherently inaccessible; for example, an advanced command post would love to have a complete and accurate model of the enemy's plans, but unless some incredibly fortuitous event occurs, it will not have such a model (at least not before those plans are executed).

Moreover, even what the agent knows about its plans and the plans of others might involve uncertainty. Some actions, because of inherent randomness or to complexities that cannot be fully fleshed out, are *non-deterministic*. An agent might be able to enumerate possible outcomes of its actions, but it cannot provably claim that any particular outcome will be reached.

When operating in applications with inaccessibility and non-determinism, an agent needs a world model that is amenable to being revised as information changes, and one that is able to represent the inherent uncertainties. For example, a traditional logic-based world model stores sentences that are considered true, and thus is not amenable to handling cases where sentences might, upon further discoveries, toggle from true to false. Efforts to extend logical world models to handle these kinds of changes have included work in non-monotonic logics, truth maintenance systems, default logics, and circumscription reasoning (Cadoli and Schaerf 1993; Forbus and deKleer 1993; McCarthy 1980).

Alternatively, rather than maintaining a world model where assertions about

the world are considered categorically true (until proven otherwise), the assertions themselves can be annotated with a measure of how certain the agent currently is of the truth of the assertion. Most often, this measure is in the form of a probability. Substantial work in AI (and elsewhere) has been directed at developing practical mechanisms for representing and propagating probabilistic information in agent world models (Pearl 1988).

One such way that has found increasing use is to represent a world model (or at least that portion of the world model that involves uncertainty) in a probabilistic belief network. The possible assertions about the state of the world are linked together based on correlations between their probabilities. Therefore, as evidence appears that changes the probability associated with one assertion, the effects of that change can be propagated through the network graph as needed to update the probabilities associated with other assertions. Considerable work has gone into the development of efficient algorithms for performing these updates (algorithms for polytree networks (Dodier 1999) or existing tools like JavaBayes (http://www-2.cs.cmu.edu/~javabayes/index.html)).

Commonly, uncertainty might is handled by Bayesian decision theory and non-monotonic logics [Gabbay, 1994]. Bayesian networks are based on independence of probabilities among causes of circumstances. Dependencies are built by directed acyclic graphs that links causes and events. If two branches culminate at a node, the immediate nodes from the two terminating edges are independent causes. There are many nonmonotonic systems and by and large they allow the reasoning system to make conclusions that are in some way non-incremental [Brewka, 2001]. Defeasible logic is one such example. The idea here is that conclusions are accepted until contradictory information erodes our confidence in defeasible drawn conclusions. Both Bayesian methods as well as nonmonotonic logics have been used extensively and continue to be important in military domains where uncertainty is commonplace. Architecturally, methods for handling uncertainty must be incorporate din the world model.

Probabilistic Planning

As planning technologies have increasingly been applied in these kinds of complex domains where unpredictable changes and uncertain events can occur, planning systems that capture probabilistic models of uncertainty and develop plans that maximize the probability of success over the space of possible world evolutions have been devised. These are the so-called probabilistic planners, such as Buridan (http://www.cs.washington.edu/research/projects/ai/www/bur.html). For the most part, probabilistic planners focus on a subset of issues that are addressed using Markov Decision Process models, described below, and thus we do not go into further detail here.

Expected Utility Maximization

On top of the challenges mentioned about inaccessibility and non-determinism, two more crucial challenges must be faced in most non-trivial applications. One of these is simply that goals can sometimes be unattainable. In most realistic problems, developing a plan that accomplishes absolutely everything desired is impossible. Instead, an agent must be able to

consider what can be accomplished, and select from among the alternatives the best one. Thus, as we progress up towards more sophisticated agents, we introduce the notion that an agent's desires are represented more richly than simply a goal that is either achieved or is not. Instead outcomes are characterized as preferences, which can be achieved to different degrees. Generally, for an agent of this type, we assume that it has some function that can assign a numerical *utility* to a situation, representing the degree to which that situation satisfies the agent's preferences.

The second crucial challenge is that executing plans usually incurs cost. Thus, given two plans that reach the same state of the world (and thus achieve the same utility), an agent will prefer executing the plan that involves the lowest cost.

Putting these together with the notions of inaccessibility and non-determinism, the agent faces a daunting task: it must decide among alternative plans of action to select the plan that it expects (probabilistically) will lead to the best outcome, based both on the utility of the states that will be reached and the costs of the actions that are needed to reach those states.

Adopting a "plan-then-execute" approach, this amounts to choosing a single sequence of actions. For each candidate sequence of actions, the agent can project forward using its models of uncertainty to identify all of the possible states that can be reached, their utilities, and the probabilities of reaching each of them. The agent can multiply the utility of each outcome by the probability of that outcome arising, and sum these together. From this sum, it can subtract the anticipated cost of the plan, to compute the overall *expected utility* (von Neumann and Morgenstern 1947; DesJardins 1995) of the plan.

Taken another step, however, the notions of conditional planning can also be introduced, where instead of planning a sequence of actions to be taken blindly, the agent can instead build a conditional plan that specifies different actions depending on which states are actually reached during execution. This is the fundamental idea behind the formation of a *policy* using a Markov Decision Process (MDP). The MDP (Boutilier et al. 1999) associates with each state that the environment might reach the optimal action to take from that state (assuming that its planned optimal actions are taken in all subsequent states). Techniques such as value-iteration and policy-iteration have been developed for MDPs to make the computation that it takes to do this practical.

Even MDPs do not address the whole story, however. Agents might not be able to observe enough to know exactly what state they are in at each step, so techniques for reasoning about uncertainty and active perception need to be considered for Partially Observable MDPs (POMDPs). (http://www.cs.duke.edu/~mlittman/topics/pomdp-page.html). (Kaebling et al 98, Berstein et al 2000, Pynadath and Tambe 2002). Furthermore, MDPs, as their name implies, make the Markovian assumption that the action to take in a state is independent of how that state was reached. When the history of

states or actions leading up to a state does impact the decision of an agent, then the problem becomes much harder, requiring the expansion of the space of states to capture (in any of a number of ways) the relevant additional information, typically leading to a combinatorial explosion in the size of the policy computation.

Finally, it should be noted that, once a policy is formulated, it could be embedded in an agent that simply matches its current state with the prescribed action: a reactive agent (Schoppers 1987). This potential decoupling between an agent (or an agent component) that is simply reactive, and another that configures that reactive agent (component), is a theme that recurs as we shall see.

Self-Aware Agents

We now move onto the case where agent architectures need to operate in situations in which resource limitations and time criticality issues come to the fore. Up until now we have not been placing any constraints on how long an agent has in order to decide what it will do next. Naturally, as we have moved up our progression of agents to those that look farther into the future and consider alternative (probabilistic) futures and try to reason about all of them, the time that an agent takes to decide what to do increases dramatically. This is fine if the environment it is acting in will wait for the agent to make up its mind. However, in the kinds of applications considered in this report, that will seldom be the case.

An agent will view its environment as *dynamic* when the environment can change at times other than when the agent is taking an action. In particular, while an agent is deciding what it should do, another agent (whether friend or adversary) could be acting and changing the environment. Either our agent should anticipate these changes and form a plan that begins with the anticipated state, or it should try to plan faster so that the state in what it executes its plan is as close as possible to the state it had planned for.

A related challenge that an agent might face is that not only might it have limited time, but it also might have limited resources for formulating and/or executing plans. Employing a planning technique that it has insufficient resources to complete, or formulating a plan that is beyond its ability to execute, will not gain an agent anything.

In simple terms, once an agent faces time and resource limits, it becomes important for the agent to incorporate into its architecture the ability to model its own capabilities and limitations, along with modeling the external environment. The paradox is that this increased awareness on the part of the agent further increases the number of things the agent needs to reason about! That is, not only must an agent reason about how to achieve its goals, but it must also reason about how the world might be changing and how it should allocate its resources in order to do this reasoning.

Anytime deliberation

One answer to this challenge is to develop reasoning mechanisms that an agent can use such that the reasoning can take a variable amount of time. Thus the agent will trade off the quality of its decisions (such as its plans) for the timeliness in formulating those decisions (plans). Classes of algorithms called *anytime algorithms* have been developed and characterized that permit this kind of successive improvement. By employing anytime algorithms, an agent can respond to real-time deadlines for reasoning while still returning an answer that is the best that time has allowed it to find (Dean and Boddy 1988; Srinivas and Horvitz 1995).

Furthermore, a careful characterization of such algorithms can allow the definition of performance profiles, where the "payoff" of using an algorithm can be mapped against the time put into the algorithm (and possibly the time/quality of algorithms used that feed results into this algorithm). Then, given some objective, an agent can perform *deliberation scheduling* to explicitly schedule its decisions about which algorithms will execute at which times and for how long so as to maximize the overall quality of the results within time bounds (Mouaddib and Zilberstein 1998; Musliner and Boddy 1997; Garvey and Lesser 1993).

Meta-Level Agent Architectures

Meta-level agent architectures view the decision problems faced by an agent in terms of what amounts to a ladder of decisions. An agent must decide on an action to take. It might identify several possible candidate actions, and so it must decide on some means of selecting one of these. However, it might have several means for selecting one, so it needs to choose a procedure for selecting a means. If several such procedures exist, then the decision problem is pushed up yet again. Each of these levels represents a "meta-level" to the problem below it, because it does not solve the lower problem, but does solve the problem about how to decide how to solve the lower problem.

When agents act in dynamic and uncertain environments, a meta-level architecture can be useful because of the flexibility such an agent has in what machinery is brought to bear when the agent faces a problem. Rather than going through a set number of steps to reach a decision, an agent might go through very few (if there is little or no meta-level indecision) or many, as need dictates. Thus, for decisions that are clear-cut, the architecture can respond quickly; for decisions that are not, the architecture will flexibly add meta-levels to resolve the decisions.

One example of a meta-level architecture is PRS, the Procedural Reasoning System (Georgeff et al. 1987). In PRS, an agent matches its goals against the plans that it knows about for achieving those goals. From this matching, it identifies a set of possible plans to pursue. If this set contains only one element, then it can pursue that plan directly. However, if the set has multiple elements, then the agent can pose the meta-level goal of selecting one of these and will retrieve plans that are capable of solving this meta-level goal. Again, if there is one choice, the agent takes it and proceeds; otherwise, it can push the problem farther up the meta-level chain.

A second architecture that has this character is Soar (Laird et al. 1987). Soar behaves similarly: when it faces an impasse because it lacks knowledge to choose one out of several options, it forms a problem space to solve the problem of resolving the impasse. This process can occur recursively as needed. An additional feature of Soar, however, is that once the impasse is solved, Soar adds new rules to its knowledge base so that, in similar situations in the future, the choice can be made directly rather than by appealing to the meta-levels. Thus, a Soar agent can become faster at doing tasks as it gains more experience.

Finally, it has been recognized that sometimes deeper deliberations should be short-circuited based on broader overriding concerns. As a human example, panic serves to cut through rationalization and lead quickly to a decision. Emotion-based research has been gaining a foothold in the agent community (see http://www.ai.mit.edu/people/jvelas/ebaa.html) for the possibility it provides for bypassing chains of reasoning to protect the agent in dangerous situations or to enable it to work with agents that have not been beneficial in the past (Picard 1997). Emotions such as pain, pleasure, fear, and anger are used in making quick decisions. CogAff is an example of an agent architecture that makes use of emotions (Sloman and Logan 1999). In our DoD scenarios, it is possible that mechanisms based on computational equivalents of emotions will be useful for self-protection and defense or for pro-active behavior and offense. However research in this area is still quite immature.

Layered Agent Architectures

The concepts of having layers of decision making, as in the meta-level architectures just described, can be institutionalized in an agent architecture by defining a fixed number of levels where each successive level is responsible for looking at a "bigger" picture than the level below. That is, lower levels act very simply and reactively, while higher levels can employ increasing amounts of reasoning about goals, plans, and uncertainties.

The clear delineation of scope and responsibility for each layer in these architectures can simplify the development of the agent, support modularization, and clarify the flow of control in the system. However, from the perspective of time-criticality, the most important advantage of this type of architecture is that, by developing the layers appropriately, each of the layers can be provided with its own computational resources. Thus, a reactive layer can immediately output actions in reflex to the current situation without waiting for higher levels to chime in. At the next higher level, more goal-directed reasoning is simultaneously in action; while this higher level cannot keep up with the reactive behavior in terms of specifying an external action, its output can instead be directed towards modifying the parameters of the reactive layer, such that the reactions can be tuned toward achieving current goals more effectively. Even higher layers can simultaneously be operating, influencing the goal-directed reasoning layer to, for example, change the priorities associated with different goals based on a broader multi-agent situation.

Examples of these kinds of architectures include AuRA (Arkin 1990),

TouringMachines (Ferguson 1992), and Cypress (Wilkins and Meyers 1995).

An advantage of having these explicit layers with their own resources is that a careful characterization of some of a lower (more reactive) layer can permit guarantees about real-time behavior. That is, some minimal level of real-time performance can be assured, and the higher layers of the architecture serve to reconfigure the reactive component as fits the evolving needs of the application domain. Examples of architectures that explicitly make real-time guarantees include Musliner's CIRCA (Musliner et al. 1995; Krebsbach and Musliner 2001; Goldman et al. 2001) and MARINER (http://www.teltec.dcu.ie/mariner/). However, further work is needed on how to combine these various layers in a coherent manner such that the whole agent can act in an appropriate and predictable manner.

Bounded Optimality

At a more conceptual level, Russell and Subramanian (1995) have looked at the problem of agent design as fundamentally a problem of generating executable computer programs. They break from the "gold standard" of evaluating an agent against some measure of rationality, such as whether it always provably achieves its goals or always maximizes its expected utility. Instead, the claim is that the best an agent designer can do is implement an agent that is optimal given the constraints of the language for describing agents and the limitations of the machine on which an agent executes.

Termed bounded optimality, this approach emphasizes being able to prove that, of all of the agents that could have been built, the agent that has been built is the best there is. Notice that this does not claim that the agent is perfect, or even that it is always rational. The agent can make mistakes. As a human comparison, it would be like pointing to a system of government, and arguing that while that system does not always make the best decisions (looking in hindsight), there are no changes to the system that would lead it to do better consistently. Thus, given the bounds in which it has to work, the system is optimal.

At this point, bounded optimality is a criterion for agent evaluation, rather than a prescription for how to build agents. In general, all that we can say is that, if we enumerate all agent programs that could run on the platform that we have at hand, and evaluate them in turn, we should arrive at a boundedly optimal agent. But of course, such a strategy for agent building is intractable. In the future, a tractable means for achieving bounded optimality might be developed, but until then we can still use bounded optimality as an idealization to consider working towards.

Self-Modifying Agents

Finally, we briefly consider agents that have to operate in applications in which it is not only the current state of the environment that can be uncertain, but in addition the agent's knowledge about how its decisions might affect its payoffs is itself very limited. Or, because of other evolving processes in

the environment, decisions that make sense at one time are inappropriate at other times, and the agent needs to continuously modify its decision-making functions.

Self-modification of agents can be done using techniques from machine learning (http://www.aic.nrl.navy.mil/~aha/research/machine-learning.html). Machine learning is a large field, and many of its techniques have been applied to agent-based systems. The types of modification that are most often considered relate to which plans are most successful in achieving a given objective, which goals should be executed locally and which should involve the assistance of other agents, and what is the best means for the agent to coordinate its actions with those of others within the system. The main machine learning technique that is used in these cases is reinforcement learning. Reinforcement learning is particularly useful because it can be applied online [Sun and Peterson, 2000]. In particular, it focuses on an agent interacting with its environment. The agent learns how to optimize its choice of actions so as to maximize its rewards from the environment. Typically, the environment is modeled as a Markov process and the learning is an approximation algorithm for solving Markov Decision Processes.

Fundamentally, though, there are two clearly opposing beliefs when it comes to self-modification. First, most users of a system in which agents can modify themselves are uneasy about this possibility; humans are used to machines that act "mechanically" – computer programs are generally predictable and repetitive. People are afraid that self-modifying agents open the door to unpredictability that could prove fatal in some situations, and thus should be avoided. At the same time, however, many users recognize that, for complex applications where little is known about the operational environment until that environment is entered, adaptation is absolutely required.

This paradoxical situation needs to be resolved if agents are to be maximally useful in complex domains. As the pyramid at the beginning of this chapter indicates, it is possible that somewhere in a large system there will be agents that are self-modifying. However, it is probable that, at least for the foreseeable future, the ratio of such agents to all agents will be quite small, so that humans can keep an eye on them. Only when such agents demonstrate that they can be trusted to modify themselves in ways that people approve of will they be more widely used.

Challenges

Relatively speaking, the design and development of agent architectures and capabilities is still in its adolescence. There are a variety of ideas and strategies that have been developed, and they have reached a level of maturity where it is possible to construct and deploy agents. However, for the most part, agent development has proceeded on a limited or even case-by-case basis, because the kinds of agent capabilities, environmental factors, and user expectations that apply can vary so dramatically from one application

domain to another.

This situation highlights the importance of the emerging field of agent-oriented software engineering (Jennings and Wooldridge 2000). The aim in this endeavor is to develop principled models, techniques and tools to aid the process of developing high quality agent-based systems. Such work is essential if agent technology is to become part of the mainstream of software development. One of the most active lines of research in this area is in the development of methodologies for the analysis and design of agent systems. Here there are broadly two camps. Firstly, there are those that believe that it is best to start with standard (object-oriented) methodologies (such as UML) and add agent specific extensions (e.g., Agent UML (http://www.auml.org/)). Secondly, there are those that believe that these methodologies provide an inappropriate set of abstractions and that it is best to develop agent specific methodologies (e.g, Gaia (Wooldridge et al. 2000) and TROPOS (http://www.cs.toronto.edu/km/tropos/)).

In addition to the general challenge of agent-oriented software engineering, we now turn to the specific challenges that agent researchers face, in order to reach the kinds of agent capabilities that are particularly needed for the scenarios earlier defined.

Reactive Agents and Actors

As the most primitive, and therefore best understood, of agent types, the challenges in reactive agents are at more of an engineering level than at a fundamental science level. Certainly, software and hardware mechanisms for improving the speed at which stimuli can trigger responses, as the number of such mappings increases, continue to be important areas of study. Each time another Boolean feature of the environment is added to be monitored, the number of mappings doubles, so that a reactive agent architecture must scale to mapping spaces that grow exponentially with the sensing capabilities of the agent. Furthermore, being able to make assurances about the timing and correctness of the mapping poses challenges be able to verify and validate that a reactive agent will indeed meet the demands placed on it by the variability of its environment and the expectations of its designer. Finally, work is needed on the methodological level to ensure such agents can be designed and built to provide reliable behavior — presently such engineering is completely ad hoc.

Modeling Agents

Maintaining and managing a world model introduces challenges that have, for the most part, been actively studied but not yet fully solved. Aside from the challenges in information fusion (discussed shortly), there are significant challenges in developing appropriate languages for modeling the world and developing suitable data structures for storing the sentences in those languages used to capture a particular world model. The development of languages goes hand-in-hand with the development of ontologies, which to date has been a process more resembling an art than an engineering science. Numerous ontologies have been proposed (http://www.daml.org/ontologies/), along with languages for general-purpose world modeling (Hayes and Menzel 2001). It remains a challenge to formulate a world modeling language and strategy that is both general-

purpose and efficient, such that it could be widely adopted to both reduce the development time of agent systems and improve the likelihood of semantic interoperability (see chapter on that topic).

Intentional Agents

Goal-based agents perform planning, and planning is in general daunting problem. Most research in planning has focused on developing strategies to overcome the difficulties in tractability, but typically this just shifts the problem elsewhere. For example, hierarchical planning incorporates more knowledge to simplify planning by relating detailed plans with the more abstract plans that they can make concrete; even so, however, hierarchical planners still have to make choices about which abstract plans to refine and which refinement to choose, which collectively comprises yet another exponential search space. For this reason, hierarchical planners have been augmented with heuristic strategies for making these choices (Tsuneto et al. 1998, Clement and Durfee 1999). Other planning techniques have faced similar problems, sometimes devising solutions that finesse the problem with heuristics, and at other times (such as with planners like GraphPlan (http://www-2.cs.cmu.edu/~avrim/graphplan.html)) taking advantage of available computing machinery (such as rapid constraint solvers) to power through problems.

Besides the challenges remaining in improving planning technologies, however, the agent community faces broader challenges in improving planning agents. Planning agents do not plan for the sake of planning, but rather, plan in order to decide what to do next. In this context, sometimes there is no good choice, and a challenge facing the designers of proactive agents is to devise means by which those agents can decide whether the plan is promising enough to continue pursuing.

If a proactive agent can determine when it has reached the limits of its capability, it can choose to enlist the aid of humans in a mixed-initiative manner, but in general it is hard to build an agent architecture that supports this kind of self-assessment. Moreover, even when not reflecting on its own limitations, if a planning agent is willing to question previous decisions, how often should it do so, and how much of its efforts should it put into such questioning versus just pushing ahead (Kinny and Georgeff 1991; Pollack 1995)? Clearly, if goal-based agent technologies are to be depended on for applications such as command posts and logistics planning, these challenges will need to be met in a principled way.

Finally, a proactive agent, capable of modeling plans, could be working in tandem with a warfighter and needs to model the warfighter's plans to monitor the pursuit of those plans. This kind of monitoring is crucial to an agent's role of working behind the scenes to collect, process, and present to the warfighter the right information at the right time in order to address the questions the warfighter needs answered. Another challenge, then, is in devising tasking and planning languages that permit the seamless sharing of plan information between human and computational agents and that support the generation and sharing of justifications for decisions and recommendations from the agent to the human.

Utility-Maximizing Agents

As proactive agents are pushed into domains where uncertainty is high and goals may only be achievable to some matter of degree, the challenges become even more daunting. Agents need to extend their world modeling techniques to represent and manipulate data about the world (and inferences based on such data) in principled yet practical ways. Tractability generally requires some compromise of accuracy and acceptance of approximations that call into question an agent's ability to make optimal decisions. Data collected over time needs to be represented and used without creating indefinitely large world models (such as temporally-extended belief networks). Probabilistic networks must extend into spatial domains, particularly for mobile operations where partial (spatially-restricted) observability of the environment will come to the fore.

The impact of knowledge, or lack thereof, can be explicitly reasoned about, and therefore can provide input to decisions about active sensing. But in general there are too many possible observations to compute the value of each possibility at every step of the way. A challenge is in focusing such decisions, perhaps by connecting up the process of improving the world model with the planning activities to direct sensors toward data that most impact decisions.

In the context of supporting the activities of a warfighter, an expected-utility-based agent must be able to faithfully represent the preferences that the warfighter intends it to follow, and elicitation of preference information can be a complicated matter (D'Ambrosio 1996). Similarly, accurate computations of probabilities associated with possible future states of the world require faithful representations of the probabilities of the kinds of events that might occur beyond the agent's control in the environment. Eliciting concise and correct models of these events is complicated by the interdependencies between events.

In summary, expected-utility-based agents face challenges both from without and within. From without, these agents need information that, for all but the most contrived or circumscribed applications, might be hard to acquire and validate definitively. From within, these agents then face often intractable computation to use whatever information they do get. Along with other challenges, these two will likely keep agent researchers busy for the foreseeable future.

Self-Aware Agents

Resource-limited agents, especially those operating in time-critical applications, face an unenviable challenge. The combination of resource limitations and time criticality makes it crucial that such an agent be extremely smart about how it spends its resources and its time. On the other hand, being smart means, in part, spending resources and time to decide on how to spend resources and time. To avoid spending too much time on this decision, an agent might also spend yet more of its resources and time deciding how much resources and time it can afford to spend on deciding how best to apply its resources and time to solving the initial problem!

It seems likely that there is no magic bullet for solving such problems; instead, such agents might have to make approximate decisions for the sake

of keeping time and resource usages in check. In fact, it is most likely to be the case that, for such agents, "optimal" will need to be replaced by "satisfactory" as a measure of performance. The use of layering architectures, or meta-level architectures that can "short-circuit" long chains of reasoning to rapidly implement stop-gap reactions is one way in which these issues are being addressed.

Making decisions, whether optimal or not, about tradeoffs among alternative places where time and resources can be spent requires the agent to have models of how well it can expect to do on some computational or external task as a function of the resources/time it invests. Such profiles have been developed for some classes of tasks, but in general addressing tradeoffs and efficiency is hard to do short of running a procedure in the laboratory many times and collecting statistics – which still might not match the conditions the agent faces in the real world. Thus, the agent might also need to reason about (and therefore devote resources and time to reasoning about!) the degree to which its current situation matches the conditions for which it was devised, and call for assistance when it is out of its depth (see discussion of adjustable autonomy, for example, in the chapter on Agent/Human Interaction).

Self-Modifying Agents

As was mentioned earlier, self-modification tends to be an agent attribute that most people acknowledge as likely to be necessary, yet people often are wary about machines that can change themselves. The first challenge in developing self-modifying agents, therefore, is in coming up with clear criteria under which such agents should be deployed, that they are only used when there are no simpler options. Alternatively, an agent can be highly constrained in what parts of itself can be self-modified; for example, a layered agent architecture might only be able to modify its own reactive layer (Musliner 1993).

When the use of self-modifying agents is unavoidable, the agents should be crafted carefully, and at least initially their architecture should support interaction with the warfighter to explain what self-modifications have been made and why. This will be challenging in itself; many machine learning algorithms, often resulting in agents self-modification, involve subtle and intricate calculations and underlying biases that could make such explanations to the warfighter (who is not expected to be a computer scientist) non-trivial.

Yet another challenge will be on an agent evaluating its own self-modifications. When an agent changes itself, it should monitor its behavior to see whether that change was indeed for the better, and revert to a previous incarnation of itself if it in fact has not improved. But it is unclear how often an agent should consider a self-modification, how long it should test a change before deciding whether or not to keep it, and, indeed, even what it should compare its performance with in order to evaluate the impact of a change.

Conclusion

There are many different kinds of agents, from simple reactive ones to sophisticated ones that need to deal explicitly with highly demanding environments. Each type of architecture has certain characteristics and with these characteristics come a number of advantages and disadvantages *in particular types of environment*. The mapping between these characteristics and the nature of the target application environment is what should define the choice of architecture for a particular application. Generally speaking, however, as we work up the ladder of sophistication in agents there is both more promise in what we can expect and more pitfalls in what could go wrong (Wooldridge and Jennings 1998). For the types of agent applications of interest in this document, it is likely that the populations of interacting agents may need to span a variety of types.

At all levels of agent complexity, challenges remain. These range from issues of engineering and verification, all the way up to fundamental questions about agent capabilities that require breakthroughs before substantial headway can be made. While a reasonably complete treatment has been attempted in the context of the scenarios in this chapter, it is important to note that this discussion has only touched superficially on the state of the art and the remaining challenges. Longer treatments (indeed textbooks like those of Russell and Norvig (1995), and of Wooldridge (1999)) can provide the interested reader with a much deeper appreciation of the potential of agents and the obstacles in the way to reaching that potential.

References

Abdulla, M., 2000, *Benevolent Agents*. PhD Thesis, University of South Carolina.

Aciego, M., de Guzmán, I., Brewka, G.; and Pereira, L. (eds.), 2000. *Logics in Artificial Intelligence*. Proceedings of JELIA'00 European Workshop, Málaga, LNAI 1919, Springer-Verlag.

Agent. 2001. *Proceedings of the Fifth International Conference on Autonomous Agents (Agents 2001)*. Montreal, Canada, ACM Press.

Agent. 2000. *Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000)*. Barcelona, Catalonia, Spain, ACM Press.

Agha, G. 1986. *Actors: A Model of Concurrent Computation in Distributed Ssytems*. MIT Press, Cambridge, Mass.

Agha, G.; and Jamali, N. 1999. *Concurrent programming for distributed artificial intelligence*. In Weiss, G., *Multi-agent Systems: A Modern Approach to DAI*. MIT Press, Cambridge, Mass., 505-534.

Albus, J. 1981. *Brains, Behavior, and Robotics*. BYTE Books: McGraw-Hill, Peterborough, N.J.

Alford, W., Kawamura, K. and Wilkes, D. *Human-directed local autonomy for motion guidance and coordination in an intelligent manufacturing system*. Proceedings of SPIE Architectures, Networks, and Intelligent Systems for Manufacturing Integration, Vol. 3203, 81-86.

Alonso, E.; and Kudenko, D. 1999. *Machine Learning Techniques for Adaptive Logic-Based Multi-Agent Systems*. GSFC NASA conference.

Amir, E.; and Maynard-Reid, P. 1999. *Logic-Based Subsumption Architecture*, Proceedings of Sixteenth International Joint Conference on Artificial Intelligence.

Arkin, R. 1990. *Integrating behavioral, perceptual and world knowledge in reactive navigation*. Robotics and Autonomous Systems, 6:105-122.

Aylett, R., Horrobin, A., O'Hare, J., Osman, A.; and Polyak, M. 1999. *Virtual teletubbies: reapplying a robot architecture to virtual agents*. Proceedings of 3rd International Conference on Autonomous Agents

Ajzen, I.; and Fishbern,M. 1980. *Understanding Attitudes and Predicting Social Behavior*, Prentice Hall.

Bajczy, R.; and Lieberman, L. 1988. *Active Perception*. Proceedings of the IEEE, 76(8): 996-1005.

Balakirsky, S.; and Lacaze, A. 2000. *World Modeling and Behavior Generation for Autonomous Ground Vehicles*, Proceedings of ICRA 2000.

Bernstein, D. S., Zilberstein, S. and Immerman, N. "The Complexity of Decentralized Control of Markov Decision Processes",Proceedings of the 16th International Conference on Uncertainty in Artificial Intelligence, Stanford, California, 2000

Blum, A.; and Furst, M. 1997. *Fast planning through planning graph analysis*. Artificial Intelligence, 90(1-2): 281-300.

Boella, G. 2000. *Cooperation among economically rational agents*. PhD thesis, Università di Torino.

Boutilier, C., Dean, T.; and Hanks, S. 1999. *Decision Theoretic Planning: Structural Assumptions and Computational Leverage*. Journal of AI Research.

Brewka, G. 2001. *Nonmonotonic Reasoning: From Theoretical Foundation to Efficient Computation,* Cambridge University Press.

Broersen, J., Dastani, M., Huang, Z., Hulstijn, J.; and Van der Torre, L. 2001. *The BOID architecture: Conflicts between beliefs, obligations, intentions and desires*. Proceedings of the Fifth International Conference on

Autonomous Agents (AA2001), 9-16, ACM Press. (see http://www.cs.vu.nl/~boid/)

Brooks, R. 1986. *A Robust Layered Control Systems for a Mobile Robot*. IEEE Journal of Robotics and Automation, RA 2-1:14-23.

Cadoli, M.; and Schaerf, M. 1993. *A Survey on Complexity Results for Non-monotonic Logics*. Journal of Logic Programming.

Clement, B.; and Durfee, E. 1999. *Top-Down Search for Coordinating the Hierarchical Plans of Multiple Agents*. Proceedings of the Third International Conference on Autonomous Agents, 252-259.

Cockburn, D.; and Jennings, N. 1996. *ARCHON: A Distributed Artificial Intelligence System for Industrial Applications*. In O'Hare, G.; and Jennings, N. (eds.). *Foundations of Distributed Artificial Intelligence*. Wiley, 319-344.

D'Ambrosio, J. 1996. *ISMAUT Tools: A Software Tool Kit for Rational Tradeoffs Among Conflicting Objectives*. http://www.eecs.umich.edu/techreports/cse/1996/CSE-TR-289-96.pdf.

Decker, K.; and Sycara, K. 1997. *Intelligent Adaptive Information Agents*, Journal of Intelligent Information Systems, 9 239--260.

Dennett, D. 1987. *The Intentional Stance*, MIT Press.

Dean, T.; and Boddy, M. 1988. *An analysis of time-dependent planning*. Proceedings of AAAI-88, 49—54.

Dodier, R. 1999. *An Algorithm for Inferences in a Polytree with Heterogeneous Conditional Distributions*. http://civil.colorado.edu/~dodier/publications.html.

Durfee, E., Huber, M., Kurnow, M.; and Lee, J. 1997. *TAIPE: Tactical Assistants for Interaction Planning and Execution*. Proceedings of the First International Conference on Autonomous Agents, 443-450.

Erol, K., Hendler, J.; and Nau, D. 1994. *Semantics for Hierarchical Task Network Planning*. Technical Report CS-TR-3239, University of Maryland.

Fasli, M. 2000. *Towards Circumspect BDI Agents: Preliminary Report*. Proceedings of the International Conference on Artificial Intelligence, IC-AI'00, Las Vegas, Nevada, USA, 573-579, CSREA Press.

Ferguson, I. 1992. *Touring Machines: An Architecture for Dynamic, Rational, Mobile Agents*. Ph.D. Thesis, Computer Laboratory, University of Cambridge, UK.

Firby, R. 1987. *An investigation into reactive planning in complex domains*. Proceedings of AAAI-87, 202-206.

Franklin, S., Kelemen, A.; and McCaule, L. 1998. *IDA: a cognitive agent*

architecture. Proceedings of the IEEE Conference on Systems, Man and Cybernetics, 2646-2651.

Fikes, R.; and Nilsson, N. 1972. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2(3-4): 189-208.

Forbus, K.; and deKleer, J. 1993. *Building Problem Solvers*. MIT Press.

Gabbay, D., Hogger, C.; and Robinson, J. 1994. *Handbook of Logic in Artificial Intelligence and Logic Programming*, *Volume 3: Nonmonotonic Reasoning and Uncertain Reasonin*. Oxford University Press.

Garvey, A;. and Lesser, V., 1993. *Design-to-time Real-Time Scheduling*. IEEE Transactions on Systems, Man and Cybernetics - Special Issue on Planning, Scheduling and Control, 23(6).

Georgeff, M. 1994. *Distributed multi-agent reasoning systems* (dmars). Technical report, Australian AI Institute, Level 6, 171 La Trobe Street, Melbourne, Australia.

Georgeff, M.; and Ingrand, F. 1990. *Managing Deliberation and Reasoning in real-time Systems*. Proceedings of the DARPA Workshop on Innovative Approaches to Planning.

Georgeff, M., Lansky, A.; and Schoppers, M. 1987. *Reasoning and planning in dynamic domains: an experiment with a mobile robot*. TN380, AI Center, SRI International, Menlo Park, California.

Goldman, C.; and Rosenschein, J. 1995. *Mutually Supervised Learning in Multi-agent Systems, Adaptation and Learning in Multi-Agent Systems*. Proceedings of IJCAI95 Workshop, Montreal, Canada, Lecture Notes in Artificial Intelligence Vol. 1042, Springer-Verlag.

Goldman, C., Musliner, D.; and Krebsbach, K. 2001. *Managing Online Self-Adaptation in Real-Time Environments*. Proceedings of Second International Workshop on Self-Adaptive Software.

Granlund, R., Johansson, B., Persson, M., Artman, H.; and Mattson, P. 2001. *Exploration of methodological issues in micro-world research - experiences from research in team decision making*. To be presented at a workshop on the use of micro-worlds in research. Granda.

Goschnick, S. 2000, *ShadowBoard: Revisiting the individual in MAS*. University of Melboune, Technical Report.

Gritton, E., Davis, P., Steeb, R.; and Matsumura, J. 2000. *Ground Forces for a Rapidly Employable Joint Task Force*, RAND.

Haugeland, J. 1985. *Artificial Intelligence: The Very Idea*. Cambridge, MA: The MIT Press.

Hayes, P.; and Menzel, C. 2001. *A semantics for the Knowledge Interchange Format*, Workshop on the IEEE Standard Upper Ontology, IJCAI 2001, Seattle.

Hexmoor, H.; and Falcone, R. 2002, *Agent Autonomy*, Kluwer, Forthcoming.

Huhns, M.; and Singh, M. (eds.). 1998. *Readings in Agents*. Morgan Kaufmann.

DesJardins, M. 1995. *Goal-Directed Learning: A Decision-Theoretic Model for Deciding What to Learn Next*. In Leake, N.; and Ram, A. (eds.), *Goal-Driven Learning*. MIT Press.

Jennings, N.; and Wooldridge, M. 2000. *Agent-Oriented Software Engineering*. Proceedings of the 9th European Workshop on Modeling Autonomous Agents in a Multi-Agent World: Multi-Agent System Engineering (MAAMAW-99).

Jennings, N. 1995. *Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems using Joint Intentions*. Artificial Intelligence, 75 (2) 195-240.

Jennings, N. 1993. *Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems*. The Knowledge Engineering Review, (3), 1993, 223-250.

Kaelbling, L. P., Littman, M. L., Cassandra, A. R., "Planning and Acting in Partially Observable Stochastic Domains", Artificial Intelligence, Vol. 101, 1998

Jonker, C.; and Treur, J. 2002. *Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. International Journal of Cooperative Information Systems*. March. In press.

Kautz, H.; and Selman, B. 1998. *Blackbox: A new approach to the application of theorem proving to problem solving*. Proceedings of AIPS Workshop on Planning as Combinatorial Search, 58-60.

Kinny, D.; and Georgeff, M. 1991. *Commitment and effectiveness of situated agents*. Technical Report 17, Australian Artificial Intelligence Institute, Melbourne, Australia.

Krebsbach, K.; and Musliner, D. 2001. *You Sense, I'll Act: Coordinated Preemption in Multi-Agent CIRCA*. Working Notes of the AAAI Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems.

Laird, J., Newell, A.; and Rosenbloom, P. 1987. *SOAR: An architecture for general intelligence*. Artificial Intelligence 33(1):1-64.

Lansky, A. 1990. *Localized search for controlling automated reasoning*. Proceedings of the DARPA Workshop on Innovative Approaches to

Planning, Scheduling, and Control, 115-125.

Lee, T.; and Wilkins, D. 1996. *Using SIPE-2 to integrate planning for military air campaigns*, IEEE Expert 11(6):11-12.

Lee, J., Huber, J., Durfee, E.; and Kenny, P. 1994. *UM-PRS: An Implementation of the Procedural Reasoning System for Multi-robot Applications*. In Proceedings of the AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space, 842-849.

Mouaddib, A.; and Zilberstein, S. 1998. *Optimal Scheduling of Dynamic Progressive Processing*. Proceedings of the 13th Biennial European Conference on Artificial Intelligence, Brighton, UK.

Musliner, D.; and Boddy, M. 1997. *Contract-Based Distributed Scheduling for Distributed Processing*. Working Notes of the AAAI Workshop on Constraints and Agents.

Musliner, D., Hendler, J., Agrawala, A., Durfee, E., Strosnider, J.; and Paul, C. 1995. *The Challenges of Real-Time AI*. IEEE Computer 28(1):58-66.

Musliner, D., Durfee, E.; and Shin, K. 1993. *CIRCA: A Cooperative Intelligent Real-Time Control Architecture*. IEEE Transactions on Systems, Man, and Cybernetics.

Laird, J.; and Rosenbloom, P. 1996. *The evolution of the Soar cognitive architecture*. In Steier, D.; and Mitchell, T. (eds.), *Mind Matters: A tribute to Allen Newell*, 1--50. Erlbaum, Mahwah, NJ.

Leite, J., Alferes, J.; and Pereira, L. 2001. *MINERVA - A Dynamic Logic Programming Agent Architecture*, In Meyer, J; and Tambe, M. (eds.), Pre-Procseedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL'01), 133-145, Seattle, WA, USA.

Lesser, V.; and Corkill, D. 1981. *Functionally accurate, cooperative distributed systems*. IEEE Transaction on Systems, Man, and Cybernetics SMC-11(1):81-96.

von Neumann, J.; and Morgenstern, O. 1947. *Theory of Games and Economic Behavior*, Princeton University Press.

McCarthy, J. 1980. *Circumscription - a form of non-monotonic reasoning*. Artificial Intelligence, 13:27-39.

McCarthy, J. 1963. *Situations, actions, and causal laws*. Memo 2, Stanford AI Project.

Pearl, J. 1988. *Probabilistic Resasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Picard, R. 1997. *Affective Computing*, MIT Press.

Pollock, J. 1995. *Cognitive Carpentry*, Bradford/MIT Press.

Pouchard, L. 2000. *Metrics for Intelligence: the Perspective from Software Agents*, Proceedings of Workshop on Performance Metrics for Intelligent Systems National Institute of Standards and Technology.

Price, R.; and Boutilier, C. 2001. *Imitation and Reinforcement Learning in Agents with Heterogeneous Actions*. Proceedings of 14th Canadian Conference on AI.

Pynadath, D. and Tambe, M. Multiagent teamwork: Analyzing key teamwork theories and models First Autonomous Agents and Multiagent Systems Conference (AAMAS), 2002

Rao, A.; and Georgeff, M. 1998. *Decision Procedures for BDI Logics*. Journal of Logic and Computation, 8(3):293-342.

Reddy, R., Erman, L., Fennell, R.; and Neely, R. 1976. *The HEARSAY speech understanding system: An example of the recognition process*. IEEE Transactions on Computers C-25:427-431.

Reiter, R. 1998. *Sequential, temporal GOLOG. Principles of Knowledge Representation and Reasoning*. Proceedings of the Sixth International Conference KR'98, 547-556.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Russell, S.; and Subramanian, D. 1995. *Provably Bounded-Optimal Agents*. JAIR 2: 575-609.

Russell, S.; and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Schoppers, M. 1987. *Universal Plans for Reactive Robots in Unpredictable Environments*, Proceedings of the 10th International Joint Conference on Artificial Intelligence.

Sloman, A.; and Logan, B. 1999. *Building cognitively rich agents using the Sim_Agent toolkit*. Communications of the Association of Computing Machinery, 42(3):71-77.

Soler, J. Julián, V., Carrascosa, C.; and Botti, V. 2000. *Applying the ARTIS Agent Architecture to Mobile Robot Control*. Proceedings of IBERAMIA-SBIA 2000, 359-368.

Srinivas, S.; and Horvitz, E. 1995. *Exploiting System Hierarchy to Compute Repair Plans in Probabilistic Model-Based Diagnosis*. Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence.

Sun, R.; and Peterson, T. 2000. *Automatic partitioning for multi-agent reinforcement learning*, Proceedings of the International Conference of Simulation of Adaptive Behavior (SAB'2000), Paris, France. MIT Press, Cambridge, MA.

Sycara, K., Paolucci, M., van Velsen, M.; and Giampapa, J. 2001. *The RETSINA MAS Infrastructure*. CMU Technical Report.

Tsuneto, R., Hendler, J.; and Nau, D. 1999. *Analyzing External Conditions to Improve the Efficiency of HTN Planning*. Proceedings of AAAI-98.

Weld, D. 1999. *Recent Advances in AI Planning*. AI Magazine.

Wilkins, D.; and desJardins, M. 2001. *A Call for Knowledge-based Planning*. AI Magazine, 22(1).

Wilkins, D.; and Myers, K. 1995. *A common knowledge representation for plan generation and reactive execution.* Journal of Logic and Computation, 5(6):731-761.

Wilkins, D., Myers, K.; and Wesley, L. 1994. *Cypress: Planning and Reacting under Uncertainity*. ARPA/Rome Laboratory Planning and Scheduling Initiative Workshop Proceedings, Morgan Kaufmann Publishers Inc., San Mateo, CA, 111-120.

Wooldridge, M., Jennings, N.; and Kinny, D. 2000. *The Gaia Methodology for Agent-Oriented Analysis and Design*. Journal of Autonomous Agents and Multi-Agent Systems. 3(3):285-312.

Wooldridge, M.; and Veloso, M. (eds.). 1999. *Artificial Intelligence Today*. Springer-Verlag Lecture Notes in AI Volume 1600.

Wooldridge, M, Singh, M.; and Rao, A. (eds.). 1998. *Intelligent Agents IV*. Springer-Verlag Lecture Notes in AI, Volume 1365.

Wooldridge, M, Mueller, J.; and Jennings, R. (eds.). 1997. *Intelligent Agents III*. Springer-Verlag Lecture Notes in AI, Volume 1193.

Wooldridge, M, Mueller, J.; and Tambe, M. (eds.). 1997. *Intelligent Agents II*. Springer-Verlag Lecture Notes in AI, Volume 1037.

Zimmer, U. 1996. *Robust World-Modelling and Navigation in a Real World*. NeuroComputing, 13(2-4):247-260.

Risks

Depending on agents, just like depending on any software or hardware machinery, involves risk. At the agent capability level, the risk is that an agent that is purported to have particular capabilities (planning, information fusion, etc.) is in fact not competent in its operational environment of doing what it was supposed to be able to do. Or, probably more likely, the expectations that a user has of what the capability entails differ from the underlying semantics of the agent's capability. This is especially the case given the anthropomorphic ways in which agents and agent capabilities are often describes (Wooldridge and Jennings 1998). In either of these cases, however, there is a mismatch of expectations. For mission-critical agent systems, such mismatches can be catastrophic, as important tasks are not done right or fully.

This kind of risk can be mitigated through careful definition and design of the capability, including its iterative testing of the by users of the capability. However, thorough definition, design, and testing can be time-consuming. Thus until more formal verification techniques are developed for agent capabilities, some residual uncertainty in performance is likely to persist.

At the agent architecture level, it could be that all of the component capabilities of the agent work correctly in isolation, but when joined together in the agent architecture they fail to work seamlessly. An architecture makes certain commitments to the relationships and flow of control between capabilities—for example, how perception and world modeling relate to plan formation and execution, or how the amount of planning for goals is balanced against the costs of using time or other resources. Thus, to fit within an architecture, capabilities must meet the expectations of the architecture. An architecture assuming real-time replanning based on monitoring plan execution might depend, for example, on an "anytime" planning component that can return an approximate plan quickly and can improve upon that plan as time allows. To this end, work is needed on models and methods for dealing with capability composition.

As we work our way up the agent pyramid, from reactive, to proactive, to resource-limited, to self-modifying agents, the complexity of the architecture increases. This, in turn, means that the difficulties of developing clear definitions of agent capabilities and their interfaces grow. A risk in using agents for the applications described in this document is that those applications, intentionally, push on the state of the art. Thus we are forced to use more sophisticated architectures, which are less well understood. The challenge to the research community, therefore, is to increase the understanding at least as fast as development and deployment demands.

Finally, the agent architectures and capabilities will be realized in systems that work in human organizations, supporting warfighters. The risks involved in the adoption of these technologies are great, and are discussed and are discussed in the chapter on Human-Agent Interaction.

Forecast

Agent Architectures and Capabilities, continued

Technology element	Near term 2001-2003	Midterm 2004-2006	Long term 2007-2010
Core Agent Technologies			
Reactive Agents	Improvements in scale/speed of techniques for mapping percepts to actions	Verification and Validation techniques	Methodologies and tools for systematically designing and building reliable behaviour
State-Based Agents	Ontology-building tools continue to improve	New standards for world modeling languages proposed	Efficient techniques for fusing multiple types of information of varying degrees of uncertainty from multiple sources
Goal-Based Agents	Faster planning techniques for larger scale problems in more dynamic domains.	Principles emerge for optimal commitment reconsideration	Tools for easily modeling and monitoring human plans Models for knowing when the agent should pass the initiative to the human and when it should claim the initiative from the human
Expected Utility-Based Agents	Refinement of ideas in temporally-extended belief networks Active sensing in MDPs Belief revision about environmental characteristics Initial use of Partially observable markov-decision processes (POMDPs) as a basis for agent architectures	Clear principles for determining when to engage in active perception and when to replan Innovations in spatial belief networks Improved tractability for partially-observable MDPs Automated preference elicitation offline Efficient algorithms for POMDPs	Approximate algorithms for scaling belief networks to very large sizes Automated preference elicitation during operations

Agent Architectures and Capabilities, continued

Technology element	Near term 2001-2003	Midterm 2004-2006	Long term 2007-2010
Core Agent Technologies			
Resource-Limited Agents	Development of anytime algorithms for an agent's action deliberation capability Extensions of performance profiles to more algorithms	Deliberation scheduling foundation within architecture Computational models of emotion to help regulate resource use Develop models for combining the different layers of the layered architectures into a coherent and predictable whole	Approximate algorithms with principled definition and gauarantees of satisfactory performance Development of boundedly optimal architectures
Self-Modifying Agents	Self-modification in layered architectures Safety and invariance guarantees in self modification	Explanation capabilities about self modifications	Well-defined criteria for adopting self-modifying agents in a systematic fashion
All agent types	Well-developed fundamental theory and principles of agent architectures in separate formalisms, BDI logics, markov decision processes (MDP), and partially observable markov decision processes (POMDPs) Analysis tools for agent and multiagent performance	Clear understanding of the relationships among logics and POMDP formalisms; and development of new integrated theories Development of POMDP based formalisms for multiagent architectures Fundamental breakthroughs in theories of distributed POMDPs	Rigorous well-developed mathematical understanding of agent architectures, multiagent architectures Well-established mathematics and science of agents and agent architectures

Figure 2. *Agent Architecture Forecast Table (Part x of n)*

Summary and Recommendations

Computational agents provide an appropriate and powerful abstraction for working with automation that is well-suited to many types of ones, including the military applications that recur in this document. A human operator can delegate tasks and responsibilities to an agent, without needing a computer professional's grasp of how that agent works and how to "program" it. An agent can then pursue its tasks "behind the scenes" as needed, alerting the human at appropriate times to relevant information or recommendations of important decisions. This model of computing is exactly what is needed to support the warfighter and thus we believe that agent technology is fundamental to any modern electronic warfare capability.

In this chapter, we described a progression of agent architectures and capabilities, from simple and limited agents to more complex, versatile, and adaptive agents. The capabilities discussed include world modeling, information fusion, planning, active perception, probabilistic reasoning, resource scheduling, and machine learning. In this rich landscape, an agent architecture provides a framework for the information and control flows between the components that provide these various capabilities. Different architectures take different stances on the relationship between the agent, its tasks, and its environment, ranging from agents that are assumed to know everything about their environments and are able to deterministically move the environment to a state that satisfies goals, to agents that have to work with limited information in rapidly changing environments where doing well enough (and fast enough) is all that can be hoped for.

Agents at all levels of complexity are active foci of research --- some with reasonably short-term prospects of success, others that still require significant scientific breakthroughs. The application domains of advanced sensor grids, unmanned autonomous systems, advanced command posts, and so on, are likely to require agents at all levels, and will push on the state of the art because of the size, complexity, and time criticality of these applications. In this context, the challenges that agent researchers face include:

- Scaling up rapid reactive systems to larger percept input streams.
- Developing world-modeling languages and technologies that can work efficiently and effectively across domains.
- Improving planning technology to increase speed and scale to more complex tasks.
- Integrating planning into agent architectures to recognize when the agent is out of its depth and when the agent should reconsider whether to continue with its current plan.
- Managing world models for temporally and spatially extended probabilistic information.

Agent Architectures and Capabilities, continued

- Automating the acquisition of knowledge about environmental processes and about user preferences to form valid probabilistic projections about future courses of events and the desirability of each of them to the human whose interests are being represented.
- Rapid reasoning about resource allocation to competing demands on a resource-limited agent operating in a time-critical environment.
- Reasoning about tradeoffs and satisfactory performance for problems where resources cannot assure optimal behavior.
- Principled adaptation to improve performance based on experience and explicit justification for self-modifications to satisfy users.

In addition, it is critical to continue the development of fundamental theory of agents, agent architectures and multiagent architectures. BDI logics and partially observable markov decision processes (POMDPs) offer two promising avenues for such theoretical development. This research may provide rigorous foundations for agent architectures, novel analytical tools to understand relationships among architectures, and indeed may lead to fundmental mathematical breakthroughs of importance not only for agents, Artificial Intelligence and Computer Science, but beyond in operations research and other fields.

Software Agents for the Warfighter
Part 2: Technology Components

Agent-Agent Interaction

Brief Overview

Description

In a networked information system that includes human and computational decision-making agents, the competence of the system depends on the coordinated interactions among all of the different types of agents and humans. This chapter will focus on computational techniques for supporting effective interactions between artificial agents. The next chapter will then focus on supporting interactions between agents and humans.

There is a wide range of agent-agent interaction technologies. The style of interaction best suited to a particular application is largely dependent on the ways in which agents' goals and capabilities are related and the overall functionality desired of the system.

In one broad class of interactions, each agent can be viewed as providing some kind of capability or service, and the purpose of agent-agent interaction is to form groups or teams of agents, dynamically, that can collectively provide a functionality needed by system users. Examples of where this need arises in the motivating scenarios include cooperative gathering and interpretation of intelligence data to support warfighter awareness, exploiting formations of unmanned vehicles, and collaboration between agent-based capabilities for command and control planning (mission planning, trajectory planning, intelligence gathering, planning, etc.) to craft battle and logistics plans.

The technologies germane to this type of agent-agent interaction include ubiquitous communications infrastructures, communication languages and protocols, matchmaking and brokering, and models for contracting, cooperative planning, and teamwork.

A second broad class of interactions is where each agent is either acting as an independent entity or is representing an independent entity that has its own objectives and capabilities that might interact with those of other agents. Examples of where this arises in the motivating scenarios include coordinated command and control in loosely-coupled (coalition) operations, and dealing with contention over vital assets such as intelligence-gathering capabilities or warfighter assignments.

The technologies germane to this type of agent-nt-agent interaction span a spectrum corresponding to the attitudes between agents, ranging from cooperative to competitive. At the cooperative extreme, where agent acquaintances are known at compile time, are techniques for distributed constraint satisfaction, dynamic teaming, and coalition formation. In 'middle' cases, where the interaction is between friendly forces that are nevertheless competing over scarce resources, techniques can involve organizational structuring, market-based allocation mechanisms, and

negotiation protocols. At the competitive extreme, where agent acquaintances are discovered at run time and the agents are playing a zero sum game (where what is good for one is bad for another), the interaction can be conditioned by social mechanisms (voting, norms, etc.) for enforcement of regulations.

Relevance to the Warfighter

Agent-agent interaction automates the formation, management, coordination, and operation of agent teams that support information gathering, logistical support, and command/control for enhancing the situation awareness and effective performance of the warfighter. This minimizes the need for warfighter intervention to control unmanned systems operating in missions too dangerous, remote, or prolonged for human warfighters. Moreover, it allows the warfighter to concentrate on tasks requiring human intellect through the use of taskable agents that collaboratively handle routine activities.

Risks

Because it does not include a human in the loop, agent-agent interaction runs the risk of getting into states such as deadlock and livelock that could impact the performance of the automated systems and indeed of the entire network. Other risks include unforseen or "emergent" group interactions that may adversely impact either performance or goal achievement. Agents and their interaction mechanisms will need one or more of the following: careful analysis, incremental introduction, and (when feasible and until trust is established) human oversight.

Forecast

There is significant near-term promise for agent-agent interaction infrastructures that will provide a baseline for the construction of agent-based software systems that advance the state of the art. Looking out to the medium and longer terms, agent interaction technologies that will flexibly support warfighter needs in more open, evolving operations are on the horizon, but will require significant research and development effort.

Summary and Recommendations

Agent-agent interaction within agent-based systems permits agents to pool their capabilities together behind the scenes to provide the user with more comprehensive warfighting capabilities and to help detect and coordinate interactions between independently-operating warfighter operations. Technologies are well on the way for realizing these advantages in circumscribed applications. To be understood well enough to be counted upon in critical military applications, and to be versatile enough to conform to changing circumstances, investments are needed in research that will culminate in adaptive agent-interaction technologies, in principled methods for characterizing and predicting the competence of multiagent systems, and in comprehensive frameworks for dealing with distributed uncertainty in systems where agents must decide when to act unilaterally and when to function in teams or other organizational structures.

Relevance to the Warfighter

Advanced Sensor Grids

Agents in advanced sensor grids will interact to exchange information to cooperatively synthesize, in a robust manner, more precise and complete

understandings of the environment. They will also negotiate over competing demands in highly-dynamic situations and coordinate over the combined application of sensors to improve precision and reduce uncertainty. For the warfighter, this means that the sensor grid is applying its assets in a coordinated fashion to ensure the best feasible picture of the ongoing operational environment.

Unmanned Autonomous Systems

Unmanned systems will operate in teams that must collectively achieve mission goals as specified by human operators. The systems will interact to share intelligence and to reactively readjust team activities to achieve objectives in the dynamic environment. Such systems may also be able to adjust activities without direct communication (using indirect observation and expectations) when the situation warrants it. An unmanned system should also make appropriate decisions when encountering other unmanned systems whose responsibilities or objectives differ from its own. The warfighter will be able to task these systems to pursue objectives that people should not pursue because the missions are too dangerous, remote, prolonged, etc.

Advanced Command Posts

Agent technology can reduce the size of command posts and streamline command post operations by allowing human commanders to offload routine and more mechanical activities to automated agents. These agents, in turn, can interact with each other to monitor, assess, deconflict, and redirect activities occurring across the command.

Mobile Operations

Mobile operations pose significant challenges to the warfighter because of their fluid and constantly evolving nature. A warfighter will have difficulty knowing the relevant status of the ongoing mission as he or she undergoes sporadic disconnections from the network and changes to positions and relationships with other warfighters. Through the interactions between the computational agents associated with warfighters, a warfighter can have assistance in understanding the changing circumstances and making other levels of command aware of the needs and progress of the distributed mission.

Joint/Coalition Operations

Coalition operations increase the possibility of coordination errors, as different functional units in the coalition follow disparate doctrine and may take actions that have unintended consequences on others. Associating agents with functional or organizational units and allowing these agents to operate "behind the scenes" to predict and avoid unintended conflicts can improve coalition performance and help warfighters avoid incompatible actions, including "friendly fire."

Logistics

Logistic operations are carried out by highly decentralized systems. The scheduling and allocation of resources must be carefully coordinated so that personnel, supplies, and support converge in the right amounts at the right times in the right places, despite originating from various locations and moving using different means. Effective interactions between automated agents that support logistics means that the warfighter will only be placed in combat situations where the other critical ingredients to success are available.

Information Assurance

Well-defined interactions and protocols between computational agents that comprise the information network permit the kinds of authorization, understanding, and awareness that are necessary for detecting and thwarting security risks, ensuring that the warfighter can depend on the information he or she is provided. Computational agents can represent and mirror organizational boundaries in information systems, and thus their associated

security policies.

Technical Description

The need for agent-agent interaction illustrated by the scenarios brings to the forefront two related issues. One issue is concerned with the problem of bringing together agent-based components so that they can provide some desired overall capability as a team. In this case, agent-agent interaction concentrates on how agents interact to advertise their capabilities, describe their needs, negotiate the terms for working together, exchange relevant information, build on each others results, plan and coordinate actions, and collectively deliver a service to the user in a timely and effective manner. The other issue looks outside of the agent team, to concentrate instead on what happens between agents (or between teams of agents) as they continuously and concurrently operate in a constantly changing environment, potentially competing for resources[1] and pursuing incompatible goals. In such a context, agent-agent interaction must allow agents to resolve differences of opinion and prioritize the accomplishment of important tasks, despite uncertainty over the availability of resources, conflicts over objectives and the use of resources to achieve them, lack of global knowledge, and absence of any centralized control.

Accordingly, the technical description of agent-agent interaction can be separated into the two main subsections in this technical description:

1. agent-agent interaction for cooperative problem solving, including techniques for:
   - distributed task allocation (the assignment of tasks, actions, or resources to agents);
   - information sharing (ensuring that the right agents are aware of relevant information in a timely way);
   - distributed planning (generalization of the planning problem to allow tasks to be planned and executed by multiple agents);
   - team formation (creation and dissolution of agents into specialized teams to meet transient system requirements).
2. agent-agent interaction for self-interested agents in shared, dynamic environments, including aspects of:
   - discovery of potential conflicts and synergies (knowing enough about other agents to anticipate interactions);
   - agreement on coordination alternatives (contracts, voting mechanisms, organizational structures, etc.);

---

[1] At the level of agent-agent interaction, a resource is considered to be an entity in the system or environment that an agent can use to further its pursuit of its goals. For example, this could be some physical object (e.g., a sensor) or another agent that can provide a service for it. This is in contrast to the sense in which "resource" is used in the Agent Infrastructure section, where resources are at a lower-level such as CPU cycles, memory, and network bandwidth. The agent's attention/processing time may in fact be the resource that is contended for (i.e. the agent must manage its reasoning processes, etc.).

- social mechanisms (trust, reputation, power, permission and obligation);
  emergent properties and legacy systems (global properties from local computations, and seamless integration of newer, better, and 'smarter' agents).

Naturally, there are overlaps between these issues, but for convenience the remainder of this section is structured along the lines above, highlighting the state-of-the-art, current research and documented results in the topics reviewed.

Cooperative Problem Solving

The central concern in cooperative problem solving is collective solution of "a problem" that could not be solved by individuals acting on their own. That is, a successful agent-based system, from the perspective of cooperative problem solving is more than the sum of its parts: the competence of the group is greater than the union of the competences of its individual members.. Therefore, either implicitly or explicitly, the performance/payoff/utility of each agent in the system is based on the collective performance of the group. Note that there is often an underlying assumption that the group of agents have been designed and implemented to operate in a closed system (i.e. the agents are predisposed or programmed to cooperate because each succeeds when the group is successful). This does not imply that cooperative problem solving techniques are only for cooperative or 'benevolent' agents. Cooperative behavior can evolve even in highly antagonistic situations, and many of the techniques discussed here are equally applicable even if all the agents are competitive or 'self-interested' agents. Cooperative problem solving imposes certain requirements on an agent's internal architecture, including that each agent is able to model other agents. Using these models, an agent should be able to reason about how its local activities could impact what other agents can or should be doing, and to reason about how non-local activities of other agents could impact it. Discussion of the appropriate internal agent architectural associated with modeling an agent's social environment is given in the section of this document on Agent Architectures. The local reasoning about group activities, in turn, generally gives rise to communication among agents, as they share information about the problems they are solving, make requests of other agents for help, coordinate the timing of problem-solving activities, etc. It is this type of agent-agent interaction that this section focuses on.

Communication Infrastructure

A precursor to agents being able to communicate to solve problems is that agents are capable of correctly understanding their communications. When agents are developed by a variety of designers and thrown together in an open networked environment, understanding between agents is far from trivial. In this section on agent-agent interaction, we assume that agents communicate using a small number of well-defined message types, such as those defined by KQML (Finin 1997) and FIPA (Chiariglione 1987) (please see the section on Agent Infrastructures for a discussion of these and other communication infrastructures). Within those message types, the message content that describes aspects of the problem being solved or of the agents performing the problem solving will be captured using particular ontologies. The challenges and potential solutions to handling multi-agent systems with multiple ontologies are described in the section of this

document on Ontologies and Semantic Integration.

For the purposes of this section on agent-agent interaction, we assume that problems in devising message types and content languages are outside of the scope of this section. That is, here the focus is on determining how an agent knows what to say, whom to say it to, and when to say it; we assume that an agent can correctly encode what it wants to say to another, and once a message is sent, the recipient can understand its contents.

Problem Decomposition

A fundamental assumption behind cooperative problem solving is that a "problem" being solved is beyond the capabilities of any single agent. Instead, the overall solution requires contributions from two or more agents, meaning that each of the cooperating agents is working on a "piece" of the overall problem. Thus, a necessary step in cooperative problem solving is that problems be decomposed into pieces. If these pieces can be solved independently, then agents who are assigned these pieces need not coordinate with each other, thereby simplifying cooperation. More often, however, the subproblem tasks are interdependent, requiring communication among agents assigned those tasks (see later section on Distributed Awareness and Result Sharing).

In some application domains, problems are inherently decomposed. For example, in an advanced sensor grid, the physical distribution of sensors dictates what subproblems (areas to sense) each sensor is responsible for at any given time. In other application domains, a "problem" arises at one agent (for example, when a human user tasks that human's representative agent in the system with satisfying some objective). That agent needs to determine how to decompose the overall problem into subproblems, where the subproblems are likely to match the capabilities of other agents.

For the most part, techniques for problem decomposition have been rudimentary, assuming that an agent somehow just "knows" how to decompose problems properly. The representation is typically a mapping from a problem into a (possibly ordered) conjunction of subproblems. As agent-based systems become more advanced and the variety of agent capabilities grows, however, it is probable that there will be multiple decompositions possible for a given problem, leading to more complex decision-making on the part of the decomposer. Questions will arise such as: Which is the right decomposition for the current problem in the current circumstances? How do alternative decompositions strike different balances among objective criteria such as time to solution, quality of solution, computation and communication cost, flexibility to changing system demands, etc.? Are there new ways that the problem can be decomposed given the new capabilities of networked agents? How does past experience or other knowledge of the possible state of the network bias the decomposition decisions?

Matching Agents and Tasks

Assuming that an agent has subproblem tasks that it needs help with, it needs to find other agents to accomplish those tasks. A suitable agent for receiving an assigned task is both capable of completing it and available. The matching of tasks to agents is generally referred to as the "connection problem."

Contract Net Protocol

In an evolving, open environment, an agent cannot expect to always know the capabilities and availabilities of all other relevant agents, and thus will need to discover enough about these when needed to make a task assignment. The traditional mechanism for doing this is the *Contract Net Protocol* (C-Net) (Smith 1980; Davis 1983):

- An agent with a task that it needs to get done broadcasts a *task announcement* describing the task to be done, eligibility requirements (availability and other capabilities) for accepting the task, and a bid specification (what information is needed in a response to the announcement).
- Upon receipt of a task announcement, a recipient agent determines whether it is eligible to respond, and if so it sends back a *bid* that conveys how well it expects to be able to perform the task.
- After collecting bids, the agent with the task (the *manager*) evaluates the bids from the potential *contractors*, and picks one or more to assign the task to. It sends the winning contractor(s) an *award* message giving details of the task.
- A contractor sends to the manager a *report* message when the task is done, with information about the completed task. It might also send *interim report* messages along the way.

Using the C-Net, an agent with a task uses communication to dynamically build up models of eligible agents based on their bids. Of course, this assumes that agents have sufficiently similar semantics to understand descriptions of tasks and bids. This broadcasting approach can also introduce large communication costs and contention for bandwidth when tasks originate at multiple places at overlapping times. Strategies for focusing the contracting interactions attempt to reduce these costs. Another difficulty is designing appropriate bidding strategies. What should the response be from an agent that already has several outstanding (yet unrewarded) bids? Contracting works well in somewhat lightly loaded situations; in chronically overloaded situations the idea of announcing an agent looking for a task (rather than a task itself) can be used. In dynamic environments, systems may oscillate between task announcement and availability announcement modes.

Matchmaking

Of course, rather than having each agent with a task dynamically develop a limited (based on the task announcement) model of the relevant agents in the network, the dynamic model might be captured more centrally. Technologies for matchmaking (Sycara) take this approach. A *matchmaker* maintains a dynamically-updated database recording what agents are currently on the network and what their capabilities are. When an agent has a task that it needs to assign, it can contact the matchmaker, which can identify which agents have advertised capabilities that match the description of the task. Again, this assumes sufficient standardization in languages for advertising

agent capabilities and for describing tasks such that the "matches" found by the matchmaker are correct. For further discussions of languages for describing capabilities and mechanisms for matching agents, see the sections on Agent Infrastructures and Agent Architectures. Both matchmaking and contracting can be used together, with matchmaking used to find a set of appropriate agents to which to direct a call for bids.

Brokering

The C-Net supports both locating agents that can potentially accomplish a task and deciding which one(s) should be assigned the task based on what they bid and how those bids compare. Matchmaking emphasizes locating potential agents for accomplishing a task, though it can be augmented to prioritize the "matches" returned to suggest better matches. Pushed even farther, the matchmaking process could act like a *broker*. For example, it could determine whether potential agents for accomplishing the task exist, accept the task on their behalf, and then determine which agent to assign the task to based on criteria such as load balancing, cost minimization, etc. Various other combinations of matchmaking, brokering, and direct contracting are possible and combinations may be more robust than any one method alone (Decker et al. 1997)

Markets

Finally, when there are multiple similar tasks to be assigned and multiple agents capable of performing them, then making an efficient set of assignments can be done using *market-based mechanisms*. These techniques are most commonly captured using an auction, where agents that can supply a service and agents that need that service make (iterative) bids until the market clears, at which time the assignments that maximize the value (as captured in their bids) to the agents competing for tasks or services are enacted.

In market mechanisms, therefore, the idea is that the degree to which an agent that wants a particular agreement (for example, an agreement that assigns a particular resource to it) is more directly captured as the amount that each agent would be willing to pay for that agreement. The market typically goes through multiple rounds of agents bidding on resources, and then the resources (or the auctions associated with them) providing information reflecting the aggregate demand. In the simplest terms, the auction announces a price for the resource, and the agents indicate how much they are willing to buy for that amount. As the system iterates, it converges (under some conditions) to equilibrium where no agents change their resource demands given the current prices. At that point, an efficient (an in certain cases, for example Vickrey auctions, provably optimal) allocation of resources is achieved.

Distributed Awareness and Result Sharing

While a task-sharing strategy, exemplified in the Contract Net Protocol, can distribute tasks to agents, it is often the case that how one agent pursues its task should be dependent on what other agents are doing. That is, sometimes a larger problem cannot be decomposed into completely independent

subproblems. When dependencies exist, agents need to be sufficiently aware of the activities and partial task results of other agents in order to make the best local decisions about activities and results to pursue.

By increasing distributed awareness and sharing results, problem solvers can improve group performance in combinations of the following ways:

1. **Confidence**: Independently derived results/conclusions can be used for corroboration, yielding a collective result that has a higher confidence of being correct. For example, when studying for an exam, students might separately work out an exercise and then compare answers to increase confidence in their solutions.

2. **Completeness**: Each agent formulates results for whichever subtasks it can (or has been contracted to) accomplish, and these results altogether cover a more complete portion of the overall task. For example, in distributed vehicle monitoring, a more complete map of vehicle movements is possible when agents share their local maps.

3. **Precision**: To refine its own solution, an agent needs to know more about the solutions that others have formulated. For example, in a concurrent engineering application, each agent might separately come up with specifications for part of an artifact, but by sharing these, the specifications can be further honed to fit together more precisely.

4. **Timeliness**: Even if an agent could in principle solve a large task alone, solving subtasks in parallel can yield an overall solution faster.

Accruing the benefits of result sharing obviously means that agents need to share results. But making this work is not easy. First of all, agents need to know what to do with shared results: how should an agent assimilate results shared from others in with its own results? Second, given that assimilation might be non-trivial, that communicating large volumes of results can be costly, and that managing many assimilated results incurs overhead, agents should attempt to be as selective as possible about what they exchange. We now look at these issues.

Functionally Accurate Cooperation

One style of collective problem solving has been termed functionally-accurate (it gets the answer eventually, but with possibly many false starts) and cooperative (it requires iterative exchange) (Lesser and Corkill 1981). Functionally-accurate cooperation has been used extensively in distributed problem solving for tasks such as interpretation and design, where agents only discover the details of how their subproblem results interrelate through tentative formulation and iterative exchange. For this method to work well, participating agents need to treat the partial results they have formulated and received as tentative and therefore might have to entertain and contrast several competing partial hypotheses at once. A variety of agent architectures

can support this need; in particular, blackboard architectures (Lesser and Corkill 1981) have often been employed as semi-structured repositories for storing multiple competing hypotheses.

Exchanging tentative partial solutions can impact completeness, precision, and confidence. When agents can synthesize partial solutions into larger (possibly still partial) solutions, more of the overall problem is covered by the solution. When an agent uses a result from another to refine its own solutions, precision is increased. And when an agent combines confidence measures of two (corroborating or competing) partial solutions, the confidence it has in the solutions changes. In general, most distributed problem-solving systems assume similar representations of partial solutions (and their certainty measures), making combining them straightforward, although some researchers have considered challenges in crossing between representations, such as combining different uncertainty measurements (Zhang 1992).

In functionally accurate cooperation, the iterative exchange of partial results is expected to lead, eventually, to some agent having enough information to keep moving the overall problem solving forward. Given enough information exchange, therefore, the overall problem will be solved. Of course, without being tempered by some control decisions, this style of cooperative problem solving could incur dramatic amounts of communication overhead and wasted computation.

Distributed Constraint Satisfaction

One view of the iterative exchange of tentative solutions is as a distributed constraint satisfaction problem (DCSP). Each agent is trying to find values (solutions) for its variables (local subproblems) such that constraints between variables are satisfied (the local subproblem solutions "fit" together into a consistent global solution). It is typically assumed that agents whose variables have constraints between them know about this relationship, and therefore will communicate about their values with each other.

A variety of techniques for distributed constraint satisfaction have been developed over the years, and a complete description of them all is beyond the scope of this report (Yokoo et al. 1998; Yokoo and Ishida 1999). The main challenges in solving a distributed constraint satisfaction problem are ensuring termination and completeness. That is, in the most general sense, what agents in a DCSP are doing is passing around their current proposed assignments of values to their variables; when constraint violations between their variables are detected, then some of them need to make different assignments. There is thus a danger that agents will make assignments and reassignments such that they never finish. Similarly, local decisions about assignments and reassignments, without global oversight, could mean that some combinations of collective assignments might never be tried. Thus the DCSP search might be incomplete.

The solutions to these problems generally involve imposing some amount of global structure on the distributed search. Ensuring completeness is possible

if agents have consistent knowledge about which of them should try new values for variables while others hold their values constant. This permits a systematic search of the space of collective assignments, meaning both that the search is complete and that it is possible to detect when all combinations have been tried so that the search can terminate. At the extreme, however, this can lead to a fully sequentialized search among the agents, eliminating any potential benefits of parallelism among agent activities. Techniques have been developed for increasing parallelism through asynchronous activities on the parts of the agents, while still ensuring systematic search (Yokoo et al. 1998).

Shared Repositories and Negotiated Search

One strategy for reducing message passing is to instead concentrate tentative partial results in a single, shared repository. The blackboard architecture introduced above, for example, allows cooperating knowledge sources to exchange results and build off of them by communicating through a common, structured information space (i.e. the blackboard). This strategy has been adopted in a variety of distributed problem-solving approaches, including those for design applications (Lander and Lesser1993; Werkman 1992). In essence, using a shared repository can support search through alternative designs, where agents with different design criteria can revise and critique the alternatives. In many ways, this is a distributed constraint satisfaction problem, but it differs from traditional formulations in a few respects.

Two important differences are that agents are not assumed to know whose constraints might be affected by their design choices, and agents can relax constraints in a pinch. The first difference motivates the use of a shared repository, since agents would not know whom to notify of their decisions (as is assumed in typical DCSP formulations). The second difference motivates the need for heuristics to control the distributed search, since at any given time agents might need to choose between improving some solutions, rejecting some solutions, or relaxing expectations (thus making some solutions that were previously considered as rejected now acceptable). Some of these latter concerns have been introduced into more standard DCSP techniques to handle overconstrained problems (Hirayama and Yokoo 2000).

Distributed Constrained Heuristic Search

Constraint satisfaction problems in distributed environments also arise due to contention for resources. Rather than assuming a shared repository for tentative partial solutions, a search strategy that has been gainfully employed for distributed resource allocation problems has been to associate an "agent" with each resource, and have that agent process the contending demands for the resource. One form that this strategy takes is so-called market-oriented programming (Wellman 1993), where associated with resources are auctions that support the search for equilibrium in which resources are allocated efficiently. Market mechanisms are discussed later.

A second form that this strategy takes is to allow resources to compute their aggregate demands, which the agents can then take into account as they attack their constraint-satisfaction problem. For example, distributed constrained heuristic search (DCHS) uses aggregate demand to inform a heuristic search for solving a distributed constraint satisfaction problem (Sycara et al. 1991). The idea is that the aggregate demand can identify the more difficult contention issues, which can then be settled first. Much useless work is thereby avoided in settling the easier issues and then discovering that these fail to allow the hard issues to be settled. Difficulties can arise in highly dynamic environments, making aggregate demand itself dynamic and difficult to compute efficiently.

Organizational Structuring

When a shared repository cannot be supported or when problem-solving is not tantamount to resource scheduling, an alternative strategy for managing the sharing of results and convergence to collective awareness and decisions is to exploit the task decomposition structure, to the extent that it is known. This notion can be explicitly manifested in an organizational structure, which defines roles, responsibilities, and preferences for the agents within a cooperative society, and thus in turn defines control and communication patterns between them (Carley and Lin 1995; Corkill 1982; Gasser 1999; Ishida et al. 1992; Pattipati et al. 1998; Pattison et al. 1987; Prasad et al. 1996).[2] From a global view, the organizational structure associates with each agent the types of tasks that it can do and usually some prioritization over the types such that an agent that currently could do any of a number of tasks can identify the most important tasks as part of its organizational role. Allowing prioritization allows the structure to permit overlapping responsibilities (to increase the chances of success despite the loss of some of the agents) while still differentiating agents based on their primary roles.

An organizational structure provides the basis for deciding who might potentially be interested in a partial result. It also can dictate the degree to which an agent should believe and act on (versus remain skeptical about) a received result. For practical purposes, organizational structures are usually implemented in terms of stored pattern-response rules: when a partial result that matches the pattern is generated/received, then the response actions are taken (to transmit the partial result to a particular agent, or to act on it locally, or to decrement its importance, etc.). Note that a single partial result could trigger multiple actions.

Communication Strategies

[2] Hierarchical organizational structures are useful at description levels besides when describing agent-agent interactions. In the Agent Infrastructure section, hierarchical organizations support the definition of boundaries of security and control, for example, with regard to policies for managing lower-level resources. In the other direction, in the Agent-Human Interaction section, an organizational structure typically captures authority and information relationships within a human organization.

Organization structures, or similar knowledge, can provide static guidelines about who is generally interested in what results. But this ignores timing issues. When deciding whether to send a result, an agent really wants to know whether the potential recipient is likely to be interested in the result now (or soon). Sending a result that is potentially useful but that turns out to not be at best clutters up the memory of the recipient and at worst can distract the recipient away from the useful work that it otherwise would have done. On the other hand, refraining from sending a result for fear of these negative consequences can lead to delays in the pursuit of worthwhile results and even to the failure of the system to converge on reasonable solutions at all because some links in the solution chain were broken.

When cluttering memory is not terrible and when distracting garden paths are short, then the communication strategy can simply be to send all partial results. On the other hand, when it is likely that an exchange of a partial result will lead a subset of agents into redundant exploration of a part of the solution space, it is better to refrain, and only send a partial result when the agent that generated it has completed everything that it can do with it locally. Between the extremes of sending everything and sending only locally complete results are a variety of gradations (Durfee et al. 1987), including sending a small partial result early on (to potentially spur the recipient into pursuing useful related results earlier).

Task Structures

Up to this point, we have made intuitive appeals to why agents might need to communicate results. The TAEMS work of Decker and Lesser has investigated this question much more concretely (Decker and Lesser 1995). In their model, an agent's local problem solving can have non-local effects on the activity of other agents. Perhaps it is supplying a result that another agent must have to *enable* its problem-solving tasks. Or the result might *facilitate* the activities of the recipient, allowing it to generate better results and/or generate results faster. The opposites of these (*inhibit* and *hinder*, respectively) are among the other possible relationships.

By representing the problem decomposition structure explicitly and capturing within it these kinds of task relationships, we can employ a variety of coordination mechanisms. For example, an agent that provides an enabling result to another can use the task structure representation to detect this relationship and can then bias it's processing to provide this result earlier. In fact, it can use models of task quality versus time curves to make commitments to the recipient as to when it will generate a result with sufficiently high quality. In situations where there are complex networks of non-local task interrelationships, decisions of this kind of course get more difficult. Ultimately, relatively static organizational structures, relationships, and communication strategies can only go so far. Going farther means that the problem-solving agents need to monitor their current situation and follow collective plans for how they should interact to solve their problems.

Cooperative Distributed Planning

Cooperative distributed planning could involve centralized planning for distributed execution, distributed planning for centralized execution, or distributed planning for distributed execution.

Plans that are to be executed in a distributed fashion can nonetheless be formulated in a centralized manner. For example, a partial order planner can generate plans where there need not be a strict ordering between some actions, and in fact where those actions can be executed in parallel. A centralized coordinator agent with such a plan can break it into separate threads, possibly with some synchronization actions. These separate plan pieces can be passed (using task-passing technology) to agents that can execute them. If followed suitably, and under assumptions of correctness of knowledge and predictability of the world, the agents operating in parallel achieve a state of the world consistent with the goals of the plan.

Formulating a complex plan for a single execution system might require collaboration among a variety of cooperative planning specialists, just like generating the solution to any complex problem would. Thus, for complex planning in fields such as manufacturing and logistics, the process of planning could well be distributed among numerous agents, each of which contributes pieces to the plan, until an overarching plan is created. For some types of problems, the interactions among the planning specialists might be through the exchange of a partially-specified plan. For example, this model has been used in the manufacturing domain, where a general-purpose planner has been coupled with specialist planners for geometric reasoning and fixturing (Kambhampati et al. 1991). Similar techniques have been used for planning in domains such as mission planning for unmanned vehicles (Durfee et al. 1997a), for connectivity restoral plans in communication networks (Conry et al. 1991), and for logistics planning (Wilkins and Myers 1995).

Finally, the most challenging version of distributed planning is when both the planning process and its results are intended to be distributed. In this case, it might be unnecessary to ever have a multi-agent plan represented in its entirety anywhere in the system, and yet the distributed pieces of the plan should be compatible, which at a minimum means that the agents should not conflict with each other when executing the plans and preferably should cooperate with each other to achieve their plans when it would be rational to do so (e.g. when a helping agent is no worse off for its efforts), or in a cooperative situation when the whole group would be better off

The literature on this kind of cooperative distributed planning is relatively rich and varied. In what follows, we hit a few of the many possible techniques that can be useful.

Plan Merging

We begin by considering the problem of having multiple agents formulate plans for themselves as individuals and then having to ensure that their separate plans can be executed without conflict. Assume that the assignment of goals to agents has been done, either through task-sharing techniques or

because of the inherent distributed nature of the application domain (such as in advanced sensor networks). Now the challenge is to identify and resolve potential plan interactions. This could be done by a centralized coordinating agent that collects together these individual plans and analyzes them to discover what sequences of actions might lead to conflicts and to modify the plans to remove the conflicts. In general, the former problem amounts to a *reachability analysis* - given a set of possible initial states and a set of action sequences that can be executed asynchronously, enumerate all possible states of the world that can be reached. Of these, then, find the subset of worlds to avoid and insert constraints on the sequences to eliminate them (Georgeff 1983).

A host of approaches to dealing with more complex forms of this problem exist, but are beyond the scope of this treatment. We give the flavor of a few of these to illustrate some of the possibilities. When there are uncertainties about the time needs of tasks or of the possibility of arrival of new tasks, the distributed scheduling problem requires mechanisms to maximize expected performance and to make forecasts about future activities (Liu and Sycara 1996). When there might not be feasible schedules to satisfy all agents, issues arise about how agents should decide which plans to combine to maximize their global performance (Ephrati et al. 1995). More complex representations of reactive plans and techniques for coordinating them based on model-checking and Petri-net-based mechanisms have also been explored (Kabanza 1995; Lee 1997; Seghrouchni and Haddad1996).

Iterative Plan Formation

Plan merging is a powerful technique for increasing parallelism in the planning process as well as during execution. The synchronization and scheduling algorithms outlined above can be carried out in centralized and decentralized ways, where the flow is generally as follows (1) goals are assigned to agents; (2) agents formulate local plans; (3) local plans are exchanged and combined; (4) messaging and/or timing commitments are imposed to resolve negative plan interactions. The parallels between this method of planning and the task-sharing style of distributed problem-solving should be obvious. But sometimes, local decisions are dependent on the decisions of others; so local plans should be formulated with an eye on the coordination issues, rather than as if the agent could work alone.

One way of tempering proposed local plans based on global constraints is to require agents to search through larger spaces of plans rather than each proposing a single specific plan. Thus, each agent might construct the set of *all* feasible plans for accomplishing its own goals. The distributed planning process then consists of a search through how subsets of agents' plans can fit together. Ephrati and Rosenschein (Ephrati and Rosenschein 1994) have developed a plan combination search approach for doing this kind of search, where the emphasis is on beginning with encompassing sets of possible plans and refining these to converge on a nearly optimal subset. They avoid commitment to sequences of actions by specifying sets of propositions that hold as a result of action sequences instead. The agents engage in the search by proposing, given a particular set of propositions about the world, the

changes to that set that they each can make with a single action from their plans. These are all considered so as to generate candidate next sets of propositions about the world, and these candidates can be ranked using an A* heuristic (where each agent can use its plans to estimate the cost from the candidate to completing its own goals). The best candidate is chosen and the process repeats until no agent wants to propose any changes (each has accomplished its goal). Sometimes, an agent can inherently lean toward taking actions that help others (Martial 1992; Goldman and Rosenschein 1994), concentrating on strategies that agents can use to exploit "favor relations" among their goals, such as accomplishing a goal for another agent while pursuing its own goal.

An alternative to this approach instead exploits the hierarchical structure of a plan space to perform distributed hierarchical planning. By now, hierarchical planning is well-established in the AI literature. It has substantial advantages in that some interactions can be worked out in more abstract plan spaces, thereby pruning away large portions of the more detailed spaces. In the distributed planning literature, the advantages of hierarchical planning were first investigated by Corkill. A variation on the hierarchical distributed planning approach is to allow each agent to represent its local planned behaviors at multiple levels of abstraction, any of which can suffice to resolve all conflicts. In this hierarchical behavior-space search approach to distributed planning, the outer loop of the protocol identifies a particular level of abstraction to work with and decides whether conflicts should be resolved at this level or passed to more detailed levels. The inner loop of the protocol conducts what can be thought of as a distributed constraint satisfaction search to resolve the conflicts. Because the plans at various abstraction levels dictate the behaviors of agents to a particular degree, this approach has been characterized as search through hierarchical behavior space (Durfee and Montgomery 1991b).

Partial Global Planning

Partial Global Planning (Durfee 1988; Durfee and Lesser 1991a) assumes that agents will interleave actions with planning and that in turn planning and coordination will occur asynchronously. At any given time, an agent might be taking an action based on its most recently updated plan, but that plan in turn might still be in the process of being coordinated with the plans of other agents. Partial Global Planning starts with the premise that tasks are inherently decomposed -- or at least that they could be. An agent then develops an understanding of what goals it is trying to achieve and what actions it is likely to take to achieve them. Each agent thus initially formulates its plans based on its own local view. Since most agents will be concurrently concerned with multiple goals, local plans will most often be uncertain, involving branches of alternative actions depending on the results of previous actions and changes in the environmental context in carrying out the plan. In Partial Global Planning, for example, agents proceed "bottom-up" by taking their detailed local plans and forming abstractions of their plans to reveal only their major plan steps that could be of interest to other agents.

The knowledge to guide communication of plan information is contained in the *Meta-Level Organization* (MLO) (Durfee 1988, Durfee and Lesser 1991a), which specifies who needs to know the plans of a particular agent, and who has authority to impose new plans on an agent based on having a more global view. Local plans that can be seen as contributing to a single partial global goal are integrated into a partial global plan, which captures the planned concurrent activities (at the abstract plan step level) of the individuals. By analyzing these activities, an agent that has constructed the partial global plan can identify opportunities for improved coordination, such as facilitating task achievement of others by performing related tasks earlier and avoiding redundant task achievement. After reordering the major local plan steps of the participating agents so as to yield a more coordinated plan, interactions, in the form of communicating the results of tasks, are also planned. By examining the partial global plan, an agent can determine when a task will be completed by one agent that could be of interest to another agent and can explicitly plan the communication action to transmit the result. As local plans evolve during execution, this overall process is repeated.

As part of their work on TAEMS (described in the previous subsection on Task Structures), Decker and Lesser developed Generalized Partial Global Planning (GPGP) (Decker and Lesser 1995). That work built on Partial Global Planning's basic strategies of communicating and manipulating abstract representations of plans and extended those strategies to a broader class of domains and plan representations, as well as more complex relationships between plans. Remember that the TAEMS approach explicitly represents the structure of the agent's underlying tasks and their interrelationships and possible alternatives. In practice, this is typically done as a set of annotations on top of a hierarchical task network. This information includes task relationships that extend between the tasks of different agents. Such "coordination relationships" between agents represent opportunities for coordination.

In PGP, a fixed coordination mechanism is used to reorder the major local plan steps of the participating agents and to communicate this new plan. GPGP more flexibly views coordination mechanisms as creating and exchanging new constraints on tasks to be used by the existing agent scheduling component. Secondly, given any single coordination relationship, there could be a number of possible domain-independent coordination mechanisms. It therefore makes sense for an agent to be able to reason about and use any of them, and to choose (or learn) the appropriate mechanisms given the domain and the current situation. (Decker and Li 2000)

Distributed Continual Planning

The space of strategies for engaging in continual distributed planning (distributed planning interleaved with execution) among multiple agents was also the subject of a special issue of the AI Magazine (AI Magazine 1999). The work of Myers (1999) concentrates on a multiagent system for planning and execution, where the overall system forms, executes, monitors, and repairs a single plan through the cooperative interactions of agents that are dedicated to each of these (and other) aspects of planning. Similarly,

desJardins and Wolverton (1999) focus on how agents which are each experts at portions of the planning needs can work together and merge their results to form a coherent, complex plan. Boutilier (1999) describes means for formulating plans that maximize expected collective performance, taking into account the execution-time capabilities of agents to communicate and coordinate.

Other work emphasizes managing and executing plans in that are being conducted in a multiagent environment. Pollack and Horty (1999) concentrate on how an agent makes plans and manages commitments to plans during execution despite changes to the environment and in the multiagent context. Grosz and colleagues (1999), Tambe and Jung (1999), and Durfee (1999b) take a more collective "team" perspective, studying how agents that are supposed to be collaborating (and know they are) can decide on responsibilities, balance the need to support team commitments while still allowing some degree of extemporaneous execution, and formally ensure that their individual and group intentions interweave to accomplish their shared purpose.

Team Monitoring and Management

Following predefined agent-agent interaction protocols as agents individually pursue tasks presumes that the problem domain and the agents' objectives are sufficiently stable that roles, interactions and expectations are for the most part implicitly captured within the agents' knowledge and algorithms or can be discovered by comparing plans. When agents are faced with a broader variety of tasks, then explicitly representing and reasoning about teamwork among agents can not only support a wide range of applications for a particular agent system, but also permit interaction with the agent system at a more abstract (team-oriented) level.

Teamwork

The American Heritage Dictionary defines teamwork as *"Cooperative effort by members of a team to achieve a common goal."* Accordingly, research in teamwork has focused on enabling different autonomous entities (e.g., software agents or robots) to work together (cooperatively) toward a common goal. Researchers in this arena distinguish teamwork from coordination in general, even if coordination involves simultaneous actions, since such coordination need not be in service of a common goal and may not be cooperative. A famous example from the literature is that two cars driving in a convoy are essentially engaged in teamwork; but two cars in ordinary traffic, as coordinated by traffic signals, are not considered to be a team (Cohen and Levesque 1991).

Teamwork Theories and Models

While teamwork has been a topic of intense investigation in many different disciplines, in this section we will focus on the research efforts in the multiagents arena. Here, researchers have been attempting to fundamentally understand the nature of teamwork and develop practical techniques to rapidly construct robust agent teams. Early efforts in building specialized, small-scale teams have led to an emerging consensus that in complex,

uncertain environments, deriving multiagent coordination plans for tightly intertwined activities is highly problematic. These plans are formulated for a particular situation and are not expected to be reused across problem instances or domains. Thus, building teams becomes a very computation-intensive process, as team coordination decisions have to be derived again from scratch for each new domain. Furthermore, when agents coordinate their specific plans based on analyses of those plans, if circumstances unfold in unexpected ways during plan execution, it is not immediately clear to the agents the degree of flexibility they have to adapt their plans before they violate the previous analyses, because they lack a more general model of how they are operating as a team.

Teamwork Theories, Joint Intentions, Shared Plans

A new approach based on developing a general teamwork model, i.e., a general team coordination algorithm, appears to provide more promise (Tambe 1997; Jennings 1995; Rich and Sidner 1997). Fortunately, research in teamwork theories, such as the joint intentions theory (Cohen 1991), SharedPlans Theory (Grosz 1996a; Grosz 1996b), and others provides a solid foundation for building such teamwork models. Based in modal logics of beliefs, desires and intentions, these teamwork theories prescribe coordination behavior for agents to attempt to achieve "ideal" teamwork. For instance, the joint intentions theory suggests that agents must jointly commit to a team (common) goal. While joint commitment in turn implies several conditions on agents' mental states, one implication of such joint commitment is that if an agent privately believes that a team goal is achieved, unachievable or irrelevant, it must communicate with its teammates' to let them know that the goal has been achieved. By communicating this key information, teammates' do not waste effort to attain a goal that is already known to be achieved or unachievable or irrelevant. (The theory does not directly prescribe communication, which is a bit subtle, but we will work with this interpretation to simplify exposition).

Teamwork Models

The teamwork theories described above have led to the creation of teamwork models that enable agents to explicitly reason about commitments and responsibilities in teamwork and flexibly plan their own coordination from first principles. Furthermore, these models are domain independent and thus they are reusable across domains, aiding rapid construction of agent teams. A key outcome of this work is the notion of "team-oriented programming" to enable developers to program agent teams at a high-level, while the coordination is automatically generated at run time because of the teamwork model. Team-oriented programming thus implies that developers need not program in low-level coordination knowledge in building teams, as such coordination behavior is automatically generated. One example teamwork model is STEAM (Tambe 1997), which make operational the joint intentions and SharedPlans theories with a public domain implementation available in the Soar agent architecture. STEAM's reuse has been demonstrated in several different domains, ranging from helicopter pilot teams in battlefield simulations to soccer playing teams (in simulation), to software agent

assistant teams that assist researchers in their daily activities.

Identifying Teammates

While techniques for task allocation and team formation have already been discussed, the interfaces between agents in dynamic environments are contingent, negotiable, and dependent on certain kinds of social relationships. What this means for agent-agent interaction is that past action is not a determinant for future activity, requested behaviors (tasks, team roles, etc.) are negotiated and re-negotiable, and adoption of 'pro-attitudes' (attitude with respect to another specific agent) may be dependent on social relations like trust, reputation, authority, and so on.

For example, an agent that receives consistent task completion to a requested quality of service may be inclined to 'trust' the providing agent more than another. Similarly, a server that find itself over-committed and without sufficient resources to complete all its tasks, or has to pre-emptively service a higher-priority task, may have to select jobs to drop or delay which minimize damage to its 'reputation.'

Some of these interactions can be modeled as the iterated Prisoner's Dilemma game, as studied in Game Theory. In addition, there is a growing body of work in the agents literature that seeks to formalize and operationalize socio-cognitive theories of trust, autonomy and reputation in implementable computational models. For example, in the Castelfranchi and Falcone (1998a) socio-cognitive trust model, an agent *X* has a *trust disposition* towards agent *Y* with respect to a goal *g* which is positive if *X* has the beliefs:

- A competence belief, i.e. a sufficient evaluation of *Y*'s abilities to produce the expected result, and
- A willingness belief, that *Y* will actually do what *X* requires.

This model of trust can be embedded in a general theory of autonomy (Castelfranchi and Falcone 1998b), giving principles and reasons for bilateral delegation and autonomy adjustment. The practical utility of such theories is that agent interactions can be conditioned by precise and testable models that provide scientific foundations for teaming among autonomous agents.

Challenges in Cooperative Problem Solving

Realizing the promises of using cooperative agent-based systems in ways that highly improve the warfighters' capabilities requires exploiting aspects of the current state of the art, as well as meeting new research challenges. In this section, some of the most significant of these needs are discussed.

Distributed Situation Awareness

Proper decision-making, whether by human or computational agents, requires sufficient awareness about what is going on. The need for sufficient global awareness constructed through local, distributed observations arises in many relevant applications including advanced sensor grids, unmanned

autonomous systems, mobile operations, logistics, and coalition operations. The current state of the art provides rudimentary capabilities in this regard, but numerous challenges still exist.

Evolving relationships. Most research on distributed sensor networks has assumed that the positioning of sensors relative to each other is static, so that it is clear to a sensor which other sensor(s) to communicate with to ascertain information about a neighboring area. In applications such as advanced sensor grids and mobile operations, the relationships between the sensory systems can be continuously evolving. This compounds the distributed awareness problem, in those agents not only have to discover what is happening in nearby areas, but they also have to discover which agents can provide that information. This creates a higher-level problem of maintaining sufficient awareness of changing relationships in a distributed environment.

Active awareness. In any situation, the relevant parts of the situation about which to be aware will depend on what uncertainties are impacting the decision making of a human or computational agent. That is, details about the current situation that could change the actions that agents choose to take are important to discover and propagate, while details that do not affect decisions need not be discovered or conveyed. For applications such as advanced command posts, distributed situation awareness should thus be actively controlled based on the current distributed decision-making context. Unfortunately, just as it is difficult for a decision maker to be aware of a non-local sensor's observations, it is hard for a sensor to know what uncertainties are being faced by a non-local decision maker in an open, evolving environment. A challenge is to develop techniques for "distributed decision-making awareness" that complement techniques for distributed situation awareness, to improve both.

Selective communication. In covert operations such as unmanned autonomous systems and mobile operations where communication introduces risk, or in applications where bandwidth is restricted and delays are large such as in distributed sensor grids, the quality of distributed situation awareness must be traded off against the costs of attaining it. Deciding what local information is worth communicating is challenging because its impact can only be known once others receive it. Using awareness of (possibly evolving) agent relationships and possibly the decisions that other agents face, an agent can make inferences about the value of its information to others, and hence can make principled tradeoffs between the expected benefits and costs of communicating. These decisions are made even more complex by having to reason about whether the information that might be communicated will already be known to a recipient (through its local awareness or through some other communication) and whether future sensing actions will lead to better information that will subsume the current awareness, so the agent would be better off waiting to send better information later.

Non-episodic domains. Since awareness supports decision making and decisions can depend on not only the current state of the environment but also on its history, distributed situation awareness typically maintains

information that is time-stamped by when that information was believed to have been current. Most research to date has focused on episodic applications, where a distributed agent system would be faced with a task episode, would solve (or fail to solve) that episode, and then would be restarted from scratch for a new episode. Many realistic applications such as advanced command posts and coalition operations are continuously running, and when tasks are not compartmentalized into well-defined episodes, the agents need to determine for themselves when it is safe to treat information as obsolete and forget it. For single agents running in prolonged environments, this is hard enough; in a distributed agent case, it is harder because an agent will need to decide not only that old information is irrelevant for its own future reasoning, but also for the future reasoning of other agents.

Cooperative Action

Capability advertising. Cooperative problem solving assumes that agents can combine their capabilities to solve problems that none can solve alone. This presumes that each knows what it is capable of and can describe these capabilities in ways that other agents can understand. Furthermore, as agents become more sophisticated, and therefore more versatile in their capabilities, each will have to decide which of its capabilities it believes it should advertise, depending on what needs it anticipates other agents will have. An agent will need to revise how it describes itself as its model of the role(s) it can or should fill in the current society of agents evolves. A challenge is in devising description languages that can be commonly understood and are flexible enough for an agent to move among different descriptions of itself as the needs of other agents (and perhaps its understanding of those needs) change with time, in applications such as unmanned autonomous systems and mobile operations.

Teamwork modeling. Similarly, when seeking agents to work with, an agent will have a model of the nature of the cooperative endeavor, including how each agent is expected to contribute. As the types of tasks for which teamwork is needed expands to larger numbers of agents and mixed teams of computational and human agents, such as in coalition operations and logistics planning, richer models and representations of teamwork or more complex organizational models are needed. To avoid overwhelming knowledge-engineering efforts, a fundamental challenge will be in distilling general principles and methods of teamwork such that these can be reused across many domain instances, and the domain details can be automatically incorporated into basic teamwork models. Moreover, other situations where agents are ostensibly cooperating on shared goals but are part of separate, larger organizations may require team models to be extended to define what it means for larger organizations to cooperate on shared goals.

Organizational decision-making. In all but the most unforgiving of environments, it is usually the case that the same result can be achieved through a variety of different means. Distributed agents must therefore

decide among alternative means for achieving their objectives, where this decision is itself a distributed problem to solve. The decision will take into account the uncertainties about the distributed awareness of the alternative capabilities of the agents and the specific needs of the task domain to decide on an appropriate team plan. The challenge is that, in a dynamic open environment such as advanced command posts, the best decision can be difficult to come up with within reasonable coordination costs and will require explicitly reasoning about uncertainties in real-world domains. Practical teamwork models are, of course, forced to address such major shortcomings based on sound theoretical underpinnings to ensure that acceptable teamwork decisions are made, that remote agents are aware of these decisions and their roles in carrying them out, and that agents do not work at cross purposes or waste effort pursuing multiple alternative methods for achieving tasks.

Performance Competence. To count on cooperative agents to solve problems in complex environments, the agents and their methods for cooperation need to be amenable to analysis to verify the level of competence that can be expected of them. Techniques for assuring competent performance, including defining error bounds and degrees of fault tolerance conditioned on parameterized models of the task environment have, lagged behind the development of the protocols and heuristics for agent-agent interaction. A challenge for the introduction of these systems into open, dynamic, and critical operational systems like advanced command posts, advanced sensor grids, and unmanned autonomous systems is the ability to characterize in a formal way the capabilities and limitations of these systems and to create robust techniques for exception handling and failure recovery.

Distributed Resource Management

Distributed global welfare functions. When viewed from the perspective of cooperative problem solving, the emphasis is on having each of the agents making decisions that optimally contribute to the overall functionality of the collection of agents. That is, a system such as an advanced sensor grid or a coalition is being evaluated by how well it behaves as a whole. A challenge is in distributing this global evaluation function in a way that individual agents can locally make decisions that are most likely to be what they should have done had the global view been available. This again creates a meta-level decision problem: determining the joint decisions of the agents is itself a distributed problem for which sufficient shared awareness and coordinated actions are needed in order to generate a solution.

Dynamic domains. Given a specific set of tasks to pursue and available agents to pursue them, the exchange of information and tentative decisions of the agents can ultimately lead to a globally efficient allocation of resources. Unfortunately, for many applications, the dynamics of the domain make convergence to efficient decisions about managing resources impossible. The techniques developed to date have generally either assumed stability or have used "greedy" mechanisms that do the best they can with what they currently know. A challenge is in developing planning and resource management techniques that can take a longer term view, including

anticipating upcoming opportunities when managing resources while still being computationally tractable for applications like logistics planning and mobile operations.

Evolving preferences. When the demands placed on a system exceed its resources, tradeoffs are necessary. While strategies, described earlier, have been developed for making such tradeoffs based, for example, on economic models, the unpredictable nature of the kinds of application domains (such as mobile and coalition operations) for which we envision agent-based systems means that the preferences over how tradeoffs are made can change dramatically as circumstances unfold. A challenge, therefore, is in devising techniques by which the (typically human) attitudes toward tradeoffs can seamlessly percolate through the agent-based system and be manifested in the decisions that the system makes.

Interaction Among Self-Interested Agents

The emphasis of the previous discussion on cooperative problem solving was on how agents (and the capabilities they embody) can be marshaled to solve a problem. We now take a step back and consider what else might be going on while those agents are being marshaled or used. In particular, at the same time a problem emerging in one part of the environment is causing agents to coordinate over how to solve it, other problems can be emerging from other parts of the environmentThe confluence of different problems competing for attention within the system can lead, in the best case, to situations where we observe emergent behavior, and solutions to problems that the system was not designed to address, but, in the worst case, to situations where no problems are being solved because of how agents are divided up and are contending for resources.

The emphasis of this part of this section is thus on agent-agent interactions whose purpose is to identify and handle competing demands placed on a multi-agent system by agents that are not always inclined to cooperate. Now, while techniques for cooperative problem solving can potentially be used, they must be augmented to consider issues of whether the information or commitments of other agents can be trusted, what information to share with others, when to share it, and how much effort to devote to doing so.

Discovering Conflicts and Synergies

When tasks emanate from multiple loci in the agent network, the concurrent appearance of the tasks opens the door to conflicts: a task that could have been successfully pursued in isolation interferes with other tasks over what it is accomplishing or how it is accomplishing it (including the agents being recruited or resources being expended). It could also be the case that independently-emerging tasks could share subtasks that need not be replicated: the agents could mutually benefit (do less work) by exploiting such synergies between their planned activities.

A challenge arises in discovering these potential conflicts and synergies. In some cases, agents might have prior knowledge about which other agents they better coordinate with and so focus agent-agent interactions with these agents to ensure coordination. Alternatively, agents might have prior knowledge about what kinds of resources or attributes about the world might be contentious. These resources/attributes can thus act as connection points, where agents that are concerned with these find each other through some associated controller or broker or auction for the resource/ attribute. So long

as this prior knowledge exists, it can bootstrap the process of discovering conflicts and synergies.

When such prior knowledge is not available, agents need to engage in a search to discover whether their intended activities might interact and if so how. As with solving the connection problem, agents could potentially broadcast information to each other, or they could work though a more central clearinghouse that puts potentially interacting agents in touch with each other. Either way, the challenge an agent faces is deciding what aspects of its intended behavior it needs to share: sharing all of the relevant details can be overwhelming in computations and in communication, but leaving out some information might cause possible conflicts to be overlooked. A compromise can be to provide summarizing information and to permit a protocol that allows iterative exchanges to dig into the pertinent details (Clement and Durfee 1999).

Identifying Coordination Alternatives

Once a need for coordination hasbeen identified, the next step is to generate one or more candidate coordination solutions that resolve conflicts and possibly take advantage of synergies. Especially in the case of self-interested agents, it is unlikely that there is a universally acceptable coordination solution. For example, distributed constraint satisfaction techniques can be employed; however, unlike the cooperative case where any solution that satisfies constraints is acceptable to all agents, self-interested agents might have different preferences about which consistent solution should be adopted. Similarly, coordinating scarce resources can lead to a number of solutions where some agents wait for others to finish using a resource, but typically agents will disagree as to who should wait for whom.

On the other hand, there might be some coordination solutions that can be universally rejected because they are dominated by other solutions. In particular, concepts like Pareto optimality apply here: a solution *s* is Pareto optimal if there is no other solution *s'* such that some agent prefers to *s'* to *s* and no agent prefers *s* to *s'*. In other words, a solution is Pareto optimal unless there is another solution that someone likes better and no one likes worse. Of course, there could be multiple Pareto optimal solutions, but the set of such solutions is hopefully smaller than the set of all possible solutions.

Reaching Agreement on Coordination

Once alternative coordination solutions have been identified, the agents need to agree on which one to adopt. Obviously, the simplest case is when only one alternative is identified: then, either all agents must accept it, or the agents must accept the consequences of not being coordinated. In some cases, it might even be the case that the process by which alternatives are identified has been formulated such that the first possible solution found is the one that must be accepted. Control over that process then becomes important to self-interested agents.

Random Choice

Assuming all dominated solutions have been removed from consideration,

one simple mechanism for selecting one to agree upon is simply to pick one at random. If the random selection mechanism can be certified to be unbiased, then this approach has the advantages of being fair and easily implemented. Randomization is sometimes the most effective coordination solution, and has, in fact been exploited for problems such as coordinating access to a shared information channel (e.g., Ethernet).

Authority

A random selection gives all possible solutions equal likelihood of being picked, even if some agents feel very strongly about which solution is picked while others feel relatively indifferent. At the extreme, it could be the case that one agent has a preferences so overwhelming that it should have the authority to dictate which coordination solution will be selected. Authority structures for making coordination decisions are a tried-and-true technique that also occurs in human organizations such as corporations and militaries.

Voting

When no agent's preferences are overriding, then the challenge is to aggregate the preferences of the individual agents to converge on a group agreement. A standard way for doing this is through the use of voting mechanisms. In the simplest case, each agent votes (once) for the alternative it prefers, and the choice with the plurality of votes is selected. Of course, simple voting schemes like this are notorious for yielding suboptimal outcomes because they are susceptible to issues like the impact of irrelevant choices. Arrow's Impossibility Theorem proves that, even when agents reveal their preferences truthfully, there is no mechanism that can ensure an efficient outcome where no agent has dictatorial power (Arrow 1963, Sandholm 1999).

Reaching agreement becomes even more challenging if agents can choose to lie about their preferences, such as how much better one option is for them than another. A variety of ideas have been developed to encourage truthful revelation of preferences, such as taxing individuals based on how strongly they preferred an agreement that was in fact reached. In that case, an agent should not exaggerate its preferences, or else, if it gets its way, it will lose more in taxes than it really will gain. On the other hand, understating its preferences can lead to it not getting its preferred choice at all. Unfortunately, even these kinds of techniques are susceptible to being abused, especially when some agents work together to foil them.

Market Mechanisms

A conceptually simple way of ensuring that agreements are reached that strike a good balance is to adopt economic models such that an agent has to "put its money where its mouth is." There is burgeoning literature on market-oriented mechanisms (Wellman 1993; Sandholm 1999); the basic ideas were previously described.

Managing Ongoing Agent-Agent Interactions

So far, this survey of the state of the art has emphasized cases where coordination over interactions can be done prior to the agents acting in their environment. In many cases, the above techniques can be interleaved with execution, where ongoing agent activities are punctuated with periods of planning and coordination. However, in extremely open and fast-paced domains, this might not be possible.

One way to address such domains is to formulate policies that restrict agent activities to ensure some minimal degree of coordination no matter how their individual plans and activities evolve. An example of this approach is the work on *social laws* (Shoham and Tennenholtz 1992; Tennenholtz 1995). By analyzing the domain to determine states that must be avoided and their precursor states, techniques for devising social laws generate prohibitions on the actions of agents in those precursor states to ensure that undesirable states are never reached. If each agent incorporates the laws into its decision mechanisms, then a society of social-law-abiding agents is assured to be conflict free.

At the other extreme are approaches that tolerate the possibility of mis-coordination. Rather than bogging an agent down by forcing it to wait until there are agreements among all agents before it can proceed, an agent can unilaterally make decisions that it can justify in terms of past and expected agreements. While at any given time the agents might not all be acting in a coordinated fashion, eventually each will know enough about the past and expected future activities of other agents to take appropriately coordinated actions with them (Durfee and Lesser 1991a).

There are many other approaches to specifying and reasoning about dynamic multiagent systems, and in particular systems that are subject to unanticipated outcomes in their operation or dysfunctional behavior of their constituents. For example, there is work from philosophy and the study of legal systems that provides computational models of normative positions (Sergot 2001), institutional power (Jones and Sergot 1996) (the power to 'bring about' a state of affairs according to some institution), and commitments (Singh 1999). In addition, there is research that considers agents as computer processes with ascribed mental states: a system designer can reason about such states with use of, for example, epistemic logic (Fagin et al. 1995; Shoham and Tennenholtz 1994; Moses and Tennenholtz 1995). A third perspective uses computational models of organizational theory to study collective properties such as the assignment of tasks, distribution of knowledge, and organizational rules (Ferber and Gitknect 2000; Esteva et al. 2000; Zambonelli et al. 2001).

Elements of all three approaches can be draw together to provide a computational framework for the specification and animation of open agent-based systems. These involves identifying social constraints, roles, states and structures, and an 'institutionalized' communication language. For the social constraints, three levels of specification are defined:

- What kind of action count as valid actions: for example, when does a bid in an auction protocol constitute a valid bid. Distinguishing between valid and invalid actions facilitates the separation of

meaningful form meaningless activities in the society;

- What kind of action is permitted: for example, an agent may be empowered to make bids (i.e., when it signals a bid, it counts as a bid, as above), but may only be permitted to make bids at certain times in a protocol. This specification of permitted, prohibited and obliged actions allows agent behavior to be classified as legal or illegal, ethical or unethical, social or anti-social, depending on the society;
- What sanction or enforcement policies there are which deal with illegal, unethical or anti-social behavior.

More details of this framework can be found in (Artikis et al. 2002).

Challenges in Interaction Among Self-Interested Agents

Using agent-based systems to coordinate the plans and resource usages of multiple forces in a shared battle space requires extending the state of the art along each of the fronts mentioned above.

Discovering Conflicts and Synergies

Scaling. An important though usually implicit assumption in agent-based systems is that each agent has some degree of autonomy, meaning that agents are generally independent of most other agents. For example, in coalition operations, advanced command posts, and logistics applications, objectives and assets are divided and assigned so that how one of the many ongoing missions is carried out impacts only a handful of other missions. This decomposition into nearly-independent subtasks is a fundamental strategy for scaling to large multiagent systems. While known interdependencies can be reasoned about and coordinated, techniques are also needed for discovering unintended interactions between agents (as an extreme example, "friendly fire"). Rudimentary techniques for doing this have been proposed based on analyses of abstractions of agents' activities, but important challenges remain in developing robust distributed mechanisms and in exploiting built-in or dynamically-acquired knowledge to focus and streamline these activities.

Flexible execution. It is a given that an agent will not perform well if it does not have a plan when it enters a hostile environment and if it cannot revise its plan as circumstances unfold, such as in mobile operations and unmanned autonomous systems. Thus, discovering potential conflicts and synergies must be done despite inherent uncertainty in the details of how objectives will be met. One class of current strategies for dealing with this problem generally takes a "worst-case" perspective by treating plans as interacting if there is at least one possible combination of agent executions that could lead to an interaction, even if most executions will not. The other main strategy is to assume no interactions and then to "fix" problems when interactions do arise. A research challenge is to populate the space of mechanisms between these extremes to strike a more flexible balance between being overly restrictive (coordinating just in case) and overly permissive (coordinating only after problems arise).

Overhead costs. Coordination introduces a variety of costs, including the computation and communication resources that go into coordination as well as the delays that accrue from waiting until agents are coordinated before they concentrate on acting on their objectives. In particular, the process of discovering conflicts and synergies can be time- and resource-intensive, with uncertain benefits. In applications such as advanced sensor grids and mobile operations, where communication resources are scarce and time can be of the essence, the effort expended in discovering conflicts and synergies must be carefully considered. Important research challenges remain in studying principled techniques for reasoning about the tradeoffs between the costs and the benefits of coordination reasoning. It is for just these kinds of reasons that human organizations adopt policies or "standard operating procedures", although these approaches bring with them their own problems.

Reaching Agreement on Coordination

Behavioral constraints. In applications such as unmanned autonomous systems, users might want to impose adjustable constraints on autonomous operations (see next section on Agent-Human Interactions). The constraints imposed by the user, along with constraints associated with organizational roles and commitments to other agents, need to factor into an agent's decision-making about what actions it should take in the context of actions that other agents take. A challenge in agent-agent interaction is in deciding how and when to use constraints on agent behaviors in a multiagent context to focus the search for effective combinations of agent interactions without handcuffing agents.

Complex interdependencies. Distributed decision makers must sometimes construct alternative solutions that involve more complexity than simply being a combination of their individual decisions. For example, in an advanced sensor grid, a sensor that could contribute to monitoring an area must decide whether it is worthwhile doing so: for triangulation purposes, fewer than three sensors is insufficient, while more than three is wasteful. Therefore, the alternative joint decisions by the relevant sensors should only involve training three sensors on a region. However, distributed strategies for converging on solutions typically consider only pair wise constraints, and it is difficult to use current algorithms in a distributed manner to enforce trinary (or higher) constraints such as those needed for triangulation in advanced sensor grids. This is an example of the broad challenge in developing algorithms for the distributed formation of joint decisions involving complex constraints.

Nested modeling. To decide what its most appropriate action is, an agent should generally consider the larger context of what actions it expects other agents to take. When these are not explicitly communicated about (because of limitations in communication resources or privacy issues among parties that have self interests), such as could happen in coalition operations or in more adversarial domains, an agent should instead model the decision-making processes of the other agents. However, because those agents could in turn be modeling its own decision-making processes, it needs to model their models of itself, their models of its models of them, and so on. While techniques have been proposed for working with finitely-nested models and

for infinitely-nested models, substantial challenges remain in developing principles through which the models are created and tractable algorithms for determining which pieces of the models (which can exponentially expand at deeper levels) are worth reasoning about.

Reaching Agreement

Preference revelation. Given that agents have different preferences over the alternative agreements, mechanisms are needed to converge on one of the agreements. For example, in coalition operations, partners might have different preferences over how roles are apportioned, or in advanced sensor grids some sensors will prefer to conserve their own power by having other sensors take responsibility for an area. One challenge that has to date only been addressed to a limited extent is in developing means by which agents are assured of truthfully revealing exactly how strongly they prefer each of the alternatives.

Time-critical action. Sometimes, waiting for agreement can introduce unacceptable delays; an agent might need to act before there is time for all agents to reach agreement. Especially in application domains where communication channels can become compromised, the ability to unilaterally act prior to complete agreement can be crucial. The challenge, of course, is in deciding when to act and when to wait. Fundamental advances are needed in representing temporal information at both the domain and meta-level reasoning levels, as well as weighing such tradeoffs as agents are applied in environments such as mobile operations and unmanned autonomous systems where convergence to commonly held agreements becomes increasingly problematic.

Non-episodic domains. In non-episodic applications, agents will engage in a history of making and acting upon agreements. For example, particular carriers might be called upon multiple times during a prolonged logistics campaign, and coalition partners might work together in a series of missions. In these cases, the strategies for reaching agreements must account for past agreements (including the degree to which each participant fulfilled its part of the agreement) and for future agreements (encouraging evenhanded dealings that lead to future positive interactions). A challenge facing the development of agent-based systems that settle on coordination agreements in the context of most human activities is in broadening the techniques for reaching agreement to consider wider features of history and opportunity, as well as custom and culture.

Managing Ongoing Agent-Agent Interactions

Flexibility and fault tolerance. When operating in highly-dynamic environments, such as mobile operations and advanced command posts, where new interactions can arise in the midst of ongoing interactions, agents need the capability to forge agreements that apply across a span of possible futures. They also need to couple these with the ability to recognize when the evolution of the world deviates from that span, so that they can regroup. A challenge in developing techniques for coordinating the plans and resource usages of diverse agents is in equipping them with the ability to more broadly understand when they are operating within the envelope where their interactions are coordinated, and when changes to the environment or to the population of agents themselves has thrown them outside of the envelope.

Adaptation. Increasingly, it appears that military forces (and therefore the computational agents that accompany them), are facing unconventional missions for which standard doctrine does not exist. In such cases, such as rapidly-deployed coalitions and mobile operations, it is possible that agent-based systems will encounter situations for which they are underprepared at times where human consultation is not possible. It is important that agents monitor the outcomes of their decisions (even if their decisions are not to act) so as to augment their knowledge bases so that they can make better decisions in similar future situations. Self-adapting agents of this type hold the promise of improving performance and overall agent-based system capabilities over time, at the risk of having agents act in ways that are possibly unexpected from the perspective of human operators. Substantial challenges exist in developing agents that can adapt individually (see section on Agent Architectures) and collectively, and on developing policies for controlling these agent-based systems to harness the advantages of adaptation without incurring costs due to loss of predictability.

Agent Ecosystems. A final challenge is achieving the long-term extension of the idea of agent societies as being the synthesis of the complex information infrastructure as it is today (internet and WWW) with agent societies into a type of ecosystem: a rich, diverse, adaptive, and responsive environment. The ecosystem constantly scales up or down, evolves and adapts in order to best meet the changing demands of its vast and highly dynamic population (for which it is not unreasonable to think of in terms of millions). The benefit would be an environment that supports the dynamic creation of new types of relations and activities and, in doing so, creates value and degrees of scalability, sustainability and robustness that are well beyond what can be accomplished today. However, it is not unreasonable to envisage advanced Warfighter functionality in certain applications (sensor grids, logistics, etc.) as being realized in terms of an ecosystem. Particular features of such an ecosystem are emergent properties (aggregate or global properties arising from local computations), and intelligent systems evolution: seamless assimilation by newer, better, and 'smarter' agents of legacy functionality (of the component they've replaced) and coordination of the 'dumber' ones.

Risks

Reliance on automation is inherently risky because it depends on the automated systems to behave in intended ways even when operating in unforeseen circumstances. Agent-based systems, as an automation framework, in some ways increase these risks because agents are typically tasked at a more abstract level, leaving more room for unanticipated behavior. On the other hand, because they can be tasked with achieving objectives, agent-based systems also hold the promise of being more dependable and less brittle since they can make sophisticated choices that are expected to be most effective at achieving those objectives.

Achieving the promises while avoiding the pitfalls of agent-based systems, and doing so with well-defined degrees of assurance, requires the development of control and coordination technology for agents that are principled and verifiable. To date, only agent systems operating in very restricted environments and interacting through very limited protocols have been amenable to such analyses. It is expected that as agent technology is applied to increasingly complex problems (thus aggravating the risk), we will also grow in our understanding of coordination techniques and our ability to implement principled mechanisms (thus mitigating the risk). We need to move forward on the research front at least as rapidly as we move forward on the applications front.

An aggravating factor in agent-agent interaction, of course, is that there is no human in the loop for these interactions. At this point, it is still generally the case that humans are better at recognizing anomalous situations than agents are. As a result, whereas interactions with humans cannot proceed in counterproductive manners for long, agent-agent interactions could enter counterproductive states such as deadlock, and livelock that might not only render the agents themselves ineffective, but also could drag down the whole network and possibly even introduce security risks. Most likely, these risks will be mitigated by introducing these systems slowly with humans supervising their activities until sufficient trust has been built up about their operation.

There is also a risk that the choices that agents make when tradeoffs need to be made will not reflect the preferences of the human authorities because of incomplete or incorrect knowledge encoding. Again, human oversight is an important mitigating strategy, as are tools for automating the development of agents and their knowledge bases by working with experts so as to ensure complete and correct agent knowledge.

Forecast

There are many uncertainties about the future course of progress in the design and development of agent-agent interaction technologies. At the present time, there is an active and vigorous community working on these

issues, as well as substantial and growing support from the DoD and from the commercial sector for their development. The forecasts that follow assume that this growth trend in interest and support continues.

In addition, the forecasts presume that simultaneous progress is made on the agent infrastructures that undergird the various kinds of interactions between agents, such that agents can securely and understandably communicate with one another with some well-understood and achievable levels of reliability and timeliness. Advances in agent architectures to support advanced reasoning about complex situations, actions, and opportunities are also assumed.

Agent-Agent Interaction, continued

Technology element	Near term 2001-2003	Midterm 2004-2006	Long term 2007-2010
Cooperative Agent Interaction Technologies			
Distributed Situation Awareness	• Improved methods for selectively propagating relevant information • Models of roles of other agents for promoting awareness based on decision problems • Heuristic techniques for making communication tradeoff decisions	• Techniques for domain-specific revision of information relationships • Dynamic plan-based techniques for decision support information exchange • Myopic decision theoretic techniques for communication widely used	• Generic mechanisms for representing and updating relationship information • Practical mechanisms for representing non-local decision problems and making relevant communication decisions • Sequential communication decision techniques become practical
Cooperative Action	• Standards for agent-description languages emerge • Classes of teamwork models for particular operations become characterized and generic team behaviors for these defined • Separate measures of competence for agent-based systems become systematized	• Feedback mechanisms for matchmaking involving agents with disjunctive descriptions • More generic teamwork models formulated • Combined competence metrics become standardized and adopted	• Adaptive agent description techniques • Reusable teamwork models with automatic instantiation for task domains available • Techniques for formally evaluating competence along well-defined metrics lead to high-confidence systems
Distributed Resource Management	• Initial strategies for locally approximating global welfare functions are devised • Hard-coded, static models of dynamics support resource management with an eye to future opportunities • Human preferences over task/resource tradeoffs are hardwired	• Error bounds on decentralized global welfare function computations can be calculated in limited cases • Dynamic updates of anticipated future resource needs • Dynamic, situation-based revision of tradeoff preferences over tasks/resources	• A toolbox is available of techniques for approximating global performance to different degrees at various costs • Decision-theoretic methods for resource management over expected futures, costs, and rewards • Adaptive modification of preference tradeoffs based on assessing outcomes

Figure 1. *Agent-Agent Forecast Table (Part 1 of 3)*

Technology element	Near term 2001-2003	Midterm 2004-2006	Long term 2007-2010
Resource and Plan Coordination Technologies			
Discovering Conflicts and Synergies	• Scale-up strategies based on predefined partitions find wide usage • "Worst-case" and "best-case-and-recovery" techniques for flexible execution become more widespread • Overhead costs for discovering conflicts and synergies is measured	• Centralized mechanisms for discovery of partitions • Research into "expected-case" mechanisms begins to yield fruit • Decisions about expected gain of discovery and overhead of discovery influence choice of use of which, if any, discovery mechanism	• Distributed techniques for discovery of partitions • A gamut of methods for balancing execution-time flexibility with constraints supporting interaction are available • Generic methods for trading off costs and benefits of interaction discovery become widespread
Identifying Alternatives	• Variants of constraint-satisfaction methods broadly adopted • Strategies to widen the use of binary constraint techniques for more complex interactions • Characterization of agent modeling techniques based on levels of nesting	• Propagation of constraining information permits focused local search • More general n-ary constraint algorithms formulated • Prescriptive theories about appropriate levels of modeling for different application needs	• Incorporation of other contextual information (e.g., organizational) into alternatives search • More general n-ary techniques applied in distributed cases • Automated formation and adaptation of nested models by executing agent systems

Figure 1. *Agent-Agent Forecast Table (Part 2 of 3)*

Technology element	Near term 2001-2003	Midterm 2004-2006	Long term 2007-2010
Resource and Plan Coordination Technologies			
Reaching Agreement	• Standardized implementations of limited truth-revelation mechanisms (e.g., Vickrey auction, Clarke tax) • Better-defined heuristic methods for deciding between waiting for agreement or acting unilaterally • Agreement mechanisms based on single-shot decisions	• Extensions of truth-revelation techniques for more widespread usage • Research matures into principled methods for trading off rapid unilateral action with waiting for agreement • Techniques based on iterative interactions that factor in long-term effects of current decisions	• Development of monitoring mechanisms to ascertain and respond to honesty behaviors of agents • Well-defined technologies for developing agents that can act unilaterally or in concert depending on circumstances • Adaptive agreement methods based on past experiences and broader human factors
Managing Ongoing Agent-Agent Interactions	• Use of predefined feature patterns to identify situations outside of agents' capabilities • Logging of interaction experiences for after-the-fact analysis and knowledge extraction • Norm-governed interactions, trust and reputation mechanisms •	• Dynamic construction of state monitoring actions for detecting situations outside of agents' capabilities • Dynamic monitoring of situation evolution and action effects to revise internal models of actions and interactions • Open Agent Societies: inter-operation of separately designed, developed and deployed components according to specified rules •	• Generic exception handling techniques associated with coordination patterns • Adaptation of plans and strategies based on generalizing prior experiences and experimenting with alternative doctrine in simulated domains • Information Ecosystems: diverse, adaptive components with emergent functionality in robust, scalable systems •

Figure 1. *Agent-Agent Forecast Table (Part 3 of 3)*

Agent-Agent Interaction, continued

Summary and Recommendations

A fundamental promise of agent-based technologies is the degree to which these technologies can provide flexible and adaptive capabilities that can be teamed up to satisfy mission objectives. Realizing the promise requires that adaptation be embraced within agents, between agents, and by users of agent-based systems. While research into adaptation within agents has continued apace, adaptation between agents has received much less attention. Various threads of research already exist, including the development of agents that learn what to do in the context of others' actions, the formulation of belief revision techniques for modeling agents and users, and the study of mechanisms by which agents decide how to advertise their capabilities and define the organizational roles that they should play depending on the broader context of the society of human and computational agents with whom they interact. These threads need to be woven together into a coherent attack on the foundational issues of devising agent-based systems that conform to the contexts in which they are placed rather than requiring humans to tailor them to particular applications.

More broadly, for adaptation of agent-based systems to be embraced by the user community, fundamental advances are needed for prescribing policies that can control agent-agent interactions and for formally characterizing the competence and limitations of an agent-based system within an application task-environment. These advances will lead to the adoption of agent-based systems, which in turn imposes added pressures on the needs for adaptation within agent-based systems and the human organizations that use them. For example, while an initial implementation of an agent-based system in a military application might adhere to and even reinforce patterns of interaction that have become standard practice, the presence of agent-based systems holds promise for the judicious development of radically new and improved doctrine that is made possible by these systems, which might be discovered through adaptive mechanisms within the agent-based system and across the pertinent military unit.

Besides concerns about adaptation and consequently about formal competence characterization, a second key direction for more research is in the area of dealing with uncertainty in dynamic, multiagent domains. A core problem revolves around deciding on the balances that agents need to make between acting as individuals and acting as part of a collective enterprise. There are clearly times when the desires of the individual must be sacrificed for the group, while there are other times when the needs of the group are best met by allowing an individual the freedom to unilaterally act opportunistically without waiting for broader agreement. Precursor research into areas such as "adjustable autonomy" are a start in this direction, but must be extended to look to flexible means for making these adjustments and applied to autonomy between computational agents as well as between humans and their associated agents.

More broadly, making decisions about whether or not to act unilaterally and

for that matter on what the actions should be, inherently involves uncertainty. Uncertainty can arise from limited local awareness of the current situation and thus advances in distributed situation awareness must be attained to reduce this source of uncertainty. Uncertainty also arises over future courses of events, including future resource needs of other agents, future tasks that might need to be done, future information exchanges that might occur, and future changes to the environment that might cause some actions to fail and others to succeed. Models of uncertainty can be incorporated into agent reasoning techniques, but then the issue arises about how these models are populated. In particular, agents might modify their models based on past experience, agents might update each other's models based on locally-available information, and agents might bias their current and future decisions based on the uncertainty that they have or the uncertainties that they believe that others are facing. Given all of the sources of uncertainty and the competing responses to uncertainty in a dynamic multiagent setting, piecemeal solutions can emerge; the real challenge will be in developing, over time, a more encompassing framework for systematically addressing and resolving these problems.

References

AARIA. http://www.aaria.uc.edu/

AI Magazine 1999, *Special Issue on Distributed Continual Planning*, 20(4).

Arrow, K 1963. *Social Choice and Individual Values*. New Haven: Yale University Press.

Artikis, A., Pitt, J. and Sergot, M. 2002. Specification of Open Agent Societies". Proceedings International Conference on Autonomous Agents and Multi-Agent Systems AAMAS02. Forthcoming.

Bajcsy, R.; and Lieberman, L. 1988. *Active Perception*. Proceedings of the IEEE, 76(8): 996-1005.

Blum, A.; and Furst, M. 1997. *Fast planning through planning graph analysis*. Artificial Intelligence 90(1-2):281-300.

Boutilier, C. 1999a. *Multiagent Systems: Challenges and Opportunities for Decision-Theoretic Planning*. AI Magazine 20(4):35-43.

Boutilier, C., Dean, T.; and Hanks, S. 1999b. *Decision Theoretic Planning: Structural Assumptions and Computational Leverage*. Journal of AI Research.

Briggs, W.; and Cook, D. 1995. *Flexible social laws*. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95).

Carley, K.; and Lin, Z. 1995. *Organizational Designs Suited to High Performance under Stress*. IEEE Transactions on SMC 25(2):221-230.

Carver, N.; and Lesser, V. 1995. *The DRESUN Test bed for Research in FA/C Distributed Situation Assessment: Extensions to the Model of External Evidence*. Proceedings of the First International Conference on Multi-Agent Systems, 33-40, AAAI Press.

Castelfranchi, C.; and Falcone, R. 1998a. *Principles of Trust ofr MAS: Cognitive Anatomy, Social Importance and Quantification*. Proceedings International Conference on Multi-Agent Systems ICMAS98, 72-79, Paris, France.

Castelfranchi, C.; and Falcone, R. 1998b. *Towards a Theory of Delegation for Agent-Based Systems*. Robotics and Autonomous Systems, 24(3-4):141-157.

Chauhan, D. 1998. *JAFMAS*. http://www.ececs.uc.edu/~dchauhan/JAFMAS.html

Chiariglione, L. 1987. *Foundation for Intelligent Physical Agents*. http://drogo.cselt.stet.it/fipa

Clement, B.; and Durfee E. 1998. *Scheduling High-Level Tasks among Cooperative*

Agents. Proceedings of the Third International Conference on Multi-Agent Systems, 96-103.

Clement, B.; and Durfee, E. 1999. *Top-down search for the coordination of hierarchical plans of multiple agents*. Proceedings of Autonomous Agents '99.

Cohen P.; and Levesque, H. 1991. *Teamwork*. Technote 504, SRI International, Menlo Park, CA.

Conry, S., Kuwabara, K., Lesser, V.; and Meyer, R. 1991. *Multistage negotiation for distributed constraint satisfaction*. IEEE Transaction of Systems, Man, and Cybernetics SMC-21(6):1462-1477.

Corkill, D. 1992. *A Framework for Organizational Self-Design in Distributed Problem Solving Networks*. PhD thesis, University of Massachusetts.

Coury, B., Sadowsky, J., Schuster, P., Kurnow, M., Huber, M.; and Durfee, E. 1997. *Reducing the interaction burden of complex systems*. In the *Proceedings of the Human Factors and Ergonomics Society*. 41st Annual Meeting (Santa Monica, CA: HFES), 335-339.

Randall, D.; and Smith, R. 1983. *Negotiation as a metaphor for distributed problem solving*. Artificial Intelligence 20:63-109.

Decker, K.; and Li, J. 2000. *Coordinating mutually exclusive resources using GPGP*. *Autonomous Agents and Multi-Agent Systems*. Volume 3, Number 2, 133–158.

Decker, K.; and Lesser, V. 1995. *Designing a family of coordination mechanisms*. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 73-80.

Decker, K.; and Lesser, V. 1993. *A one-shot dynamic coordination algorithm for distributed sensor networks*. Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), 210-216,.

Decker, K., Sycara K.; and Williamson, M. 1997. *Middle-Agents for the Internet*. In Proceedings of the 15th International Joint Conference on Artificial Intelligence, 578–583, Nagoya, Japan.

desJardins, M.; and Wolverton, M. 1999. *Coordinating a Distributed Planning System*. AI Magazine 20(4):45-53.

Draper, D.,Hanks S.; and Weld D. 1994. *Probabilistic planning with information gathering and contingent execution*. Proceedings of the Second AI in Planning Systems Conference.

Durfee, E. 1999a. *Distributed Problem Solving and Planning*." In Gerhard Weiss (ed.). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge MA.

Durfee, E. 1999b. *Distributed Continual Planning for Unmanned Ground Vehicle Teams*. AI Magazine 20(4):55-61.

Durfee, E., Kenny, P.; and Kluge, K. 1998. *Integrated Permission Planning and Execution for Unmanned Ground Vehicles*. Autonomous Robots, 5:1-14.

Durfee, E., Huber, M., Kurnow, M.; and Lee, J. 1997a. *TAIPE: Tactical Assistants for Interaction Planning and Execution*. In Proceedings of the First International Conference on Autonomous Agents, 443-450.

Durfee, E., Kiskis, D.; and Birmingham, W. 1997b. *The Agent Architecture of the University of Michigan Digital Library*. IEE/British Computer Society Proceedings on Software Engineering (Special Issue on Intelligent Agents), 144(1): 61-71, February 1997. Also published in Readings in Agents (M. N. Huhns and M. P. Singh, editors), Morgan Kaufmann, 1998.

Durfee, E.;, and Lesser, V.. 1991a. *Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation*. IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Sensor Networks, SMC-21 (5):1167-1183.

Durfee, E.; and Montgomery, T. 1991b. *Coordination as Distributed Search in a Hierarchical Behavior Space*. IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Artificial Intelligence, SMC-21(6):1363-1378.

Durfee, E. 1988. *Coordination of Distributed Problem Solvers*, Kluwer Academic Press, Boston.

Durfee, E, Lesser, V.; and Corkill, D. 1987. *Cooperation Through Communication in a Distributed Problem-Solving Network*. Chapter 2 in Huhns, M. (ed.). *Distributed Artificial Intelligence,* Pitman.

Ephrati, E.; and Rosenschein, J. 1994. *Divide and conquer in multi-agent planning*. Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 375-380.

Ephrati, E., Pollack, M.; and Rosenschein, J. 1995. *A tractable heuristic that maximizes global utility through local plan combination*. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 94-101.

Erol, K., Hendler, J.; and Nau, D. 1994. *Semantics for Hierarchical Task Network Planning*. Technical Report CS-TR-3239, University of Maryland.

Esteva, M., Rodrigues, J., Sierra, C., Garcia, P.; and Arcos, J. 2001. *On the formal specifications of electronic institutions*. In F. Dignum and C. Sierra (eds.) Agent-mediated Electronic commerce (The European AgentLink Perspective, LNAI 1991, 126-147.

Fagin, R., Halpern, J., Moses, Y.; and Yardi, M. 1995. *Reasoning about Knowledge*. MIT Press.

Fennell, R.; and Lesser, V. 1977. *Parallelism in AI problem solving: A case study of HEARSAY-II*. IEEE Transaction on Computers C-26(2):98-111.

Fenster, M., Kraus, S.; and Rosenschein, J. 1995. *Coordination without communication: experimental validation of focal point techniques*. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 102-108.

Ferber, J. and Gitknect, O. 2000. *Operational semantics of Multi-Agent Organizations*. In N. Jennings and Y. Lesperance (eds.): Intelligent Agents VI, Agent Theories, Architecture and Languages. LNAI1757, 205-217.

Fikes, R.;. and Nilsson, N. 1972. *STRIPS: A new approach to the application of theorem proving to problem solving*. Artificial Intelligence 2(3-4):189-208.

Finin, T. 1997. *UMBC KQML Web*. http://www.cs.umbc.edu/kqml.

Firby, R. 1987. *An investigation into reactive planning in complex domains*. Proceedings of AAAI-87 202-206.

Fisher, M.; and Wooldridge, M. 1997. *Distributed problem-solving as concurrent theorem-proving*. Proceedings of MAAMAW'97, Lecture notes in Artificial Intelligence, Springer-Verlag.

Forbes, J., Huang T., KanazawaK.; and Russell S. 1995. *The BATmobile: Towards a Bayesian Automated Taxi*. Proceedings of Fourteenth International Joint Conference on Artificial Intelligence, Montreal, Canada.

Gallimore, R., Jennings, N., Lamba, H., Mason, C.; and Orenstein, B. 1998. *3D Scientific Data Interpretation using Cooperating Agents*. Proceedings of 3rd Internationl Conference on the Practical Applications of Agents and Multi-Agent Systems (PAAM-98), London, UK, 47-65.

Gasser, L.1999. *Computational Organization Theory. In Gerhard Weiss (ed.).* Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge, MA*.

Georgeff, M. 1983. *Communication and Interaction in multi-agent planning*. Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83), 125-129.

Georgeff, M.; and Ingrand, F. 1990. *Managing Deliberation and Reasoning in real-time Systems*. Proceedings of the DARPA Workshop on Innovative Approaches to Planning.

Goldman, C.; and Rosenschein, J. 1994. *Emergent coordination through the use of cooperative state-changing rules*. Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 408-413.

Goldman, R.; and Boddy, M. 1997. *Constraint-based scheduler for batch manufacturing*. IEEE Expert, v. 12(1):49-56.

Gray, R., Kotz, D., Cybenko, G.; and Rus, D. 1998. *D'Agents: Security in a multiple-*

language, mobile-agent system. In Giovanni Vigna, editor, *Mobile Agents and Security*, volume 1419 of Lecture Notes in Computer Science. Springer-Verlag.

Grosz, B. 1996a. *Collaborative Systems.* AI Magazine, 17(2):67-85.

Grosz, B.; and Kraus, S. 1996b. *Collaborative Plans for Complex Group Action*. Artificial Intelligence, 86(2):269-357.

Grosz, B., Hunsberger, L.; and Kraus, S. 1999. *Planning and Acting Together*. AI Magazine 20(4):23-34.

Hayes-Roth, B. 1995. *An architecture for adaptive intelligent systems*. Artificial Intelligence, 72:329-365.

Hewitt, C. 1991. *Open Information Systems: Semantics for Distributed Artificial Intelligence*. Artificial Intelligence, 47(1-3):79-106.

Hirayama, K; and Yokoo, M. 2000. *An Approach to Over-constrained Distributed Constraint Satisfaction Problems: Distributed hierarchical constraint satisfaction*. Proceedings of the Fourth International Conference on MultiAgent Systems *(ICMAS-2000),* 135-142.

Huber, M.; and Durfee, E. 1996. *An initial assessment of plan-recognition-based coordination for multi-agent teams*. Proceedings of the Second International Conf. on Multi-Agent Systems (ICMAS-96), 126-133.

Huhns, M.; and Bridgeland, D. 1991. *Multi-agent Truth Maintenance*. IEEE Transaction on Systems, Man, and Cybernetics, 21(6):1437-1445.

Ishida, T., Gasser, L.; and Yokoo, M. 1992. *Organization self-design of distributed production systems*, IEEE Transaction on Knowledge and Data System DKE4(2):123-134.

Jennings, N. 1995. *Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems using Joint Intentions*. *Artificial Intelligence*. 75 (2):195-240.

Jini. http://www.sun.com/jini/specs/

JMCAP. 1998. http://www.erg.sri.com/people/marie/papers/jmcap-summary.html

John, B.; and Kieras, D.. 2002. The GOMS family of user interfaces analysis techniques: Comparison and contrast. ACM Transaction on CHI. Forthcoming.

Jones, A.; and Sergot, M. 1996. *Institutionalized Power*. Journal of the IGPL, vol. 4, no. 3.

Kabanza, F. 1995. *Synchronizing multiagent plans using temporal logic specifications*. Proceedings of the First International Conf. on Multi-Agent Systems (ICMAS-95), 217-224.

Kaelbling, L., Littman, M.; and Cassandra, A. 1998. "*Planning and Acting in Partially*

Observable Stochastic Domains. Artificial Intelligence, Vol. 101.

Kambhampati, S., Cutkosky, M., Tenenbaum, M.; and Lee. S. 1991. *Combining specialized reasoners and general purpose planners: A case study*. Proceedings of the Ninth National Conference on Artificial Intelligence, 199-205.

Kautz, H.; and Selman, B.1998. *Blackbox: A new approach to the application of theorem proving to problem solving*." Proceedings of AIPS Workshop on Planning as Combinatorial Search, 58-60.

Kinney, D., Ljungberg, M., Rao, A., Sonenberg, *E., Tidhar, G.; and Werner, E. 1992. Planned Team Activity*. Preproceedings of the Fourth European Workshop on Modeling Autonomous Agents in a MultiAgent World.

Laird, J., Newell, A.; and Rosenbloom, P. 1987. *SOAR: An architecture for general intelligence*. Artificial Intelligence 33(1):1-64.

Lander, E.; and Lesser. V. 1993. *Understanding the role of negotiation in distributed search among heterogeneous agents*. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence *(IJCAI-93)*, 438-444.

Lansky, A. 1990. *Localized Search for Controlling Automated Reasoning*. Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, 115-125.

Lee, J., Huber. M., Durfee, E.; and Kenny, P. 1994. *UM-PRS: An Implementation of the Procedural Reasoning System for Multi-robot Applications*. Proceedings of the AAAI/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space, 842-849.

Lee, T.; and Wilkins, D. 1996. *Using SIPE-2 to integrate planning for military air campaigns*. IEEE Expert 11(6):11-12.

Lee, J. 1997. *An Explicit Semantics for Coordinated Multiagent Plan Execution*. PhD dissertation. University of Michigan.

Lesser, V.; and Corkill, D. 1981. *Functionally accurate, cooperative distributed systems*. IEEE Transaction on Systems, Man, and Cybernetics SMC-11(1):81-96.

Liu, J.; and Sycara, K. 1996. *Multiagent coordination in tightly coupled task scheduling*. Proceedings of the Second International Conf. on Multi-Agent Systems (ICMAS-96), 181-188.

Lowrance, J., Garvey, T.; and Strat, T. 1990. *A framework for evidential-reasoning systems*. In Shafer, G.; and Pearl, J. (eds.) *Uncertain Reasoning*, 611-618, San Mateo, CA: Morgan Kaufman Publishers, Inc.

MacIntosh, D., Conry, S.; and Meyer, R. 1991. *Distributed automated reasoning: Issues in coordination, cooperation, and performance*. IEEE Transaction on Systems, Man, and Cybernetics SMC-21, (6):1307-1316.

MacKenzie, D., Cameron, R.; and Arkin, R., 1997. *Specification and Execution of Multiagent Missions*. Autonomous Robots, Vol. 4, No. 1.

Martial, F. 1992. *Coordinating Plans of Autonomous Agents*. Lecture notes in Artificial Intelligence, Springer-Verlag.

Martin, D., Cheyer, A.; and Moran, D. 1999. *The open agent architecture: A framework for building distributed software systems*. Applied Artificial Intelligence, Vol. 13:91--128.

McCarthy, J. 1963. *Situations, actions, and causal laws*. Memo 2, Stanford AI Project.

Miyashita, K.; and Sycara, K. 1995. *CABINS: A Framework of Knowledge Acquisition and Iterative Revision for Schedule Improvement and Reactive Repair*. Artificial Intelligence, Vol. 76, (1-2):377-426.

Moses, Y.; and Tennenholtz, M. 1995. *Artificial Social Systems*. Computers and Artificial Intelligence, 533-562.

Musliner, D. , Durfee, E.; and Shin, K. 1995. *World Modeling for the Dynamic Construction of Real-Time Control Plans*. Artificial Intelligence. 74:83-127.

Myers, K. 1999. *CPEF: A Continuous Planning and Execution Framework*. AI Magazine 20(4):63-69.

Neches, R., Fikes, R., Finin, T., Gruber, R., Patil, R., Senator, T.; and Swartout, W. 1991. *Enabling Technology for Knowledge Sharing*. AI Magazine. 12(3):36-56.

Newell, A.; and Simon, H.(1963. *GPS, a program that simulates human thought*. Reprinted in Fiegenbaum and Feldman (eds.). *Computers and Thought*, McGraw Hill.

Pattipati, K., Pete, A., Kleinman, D.; and Levchuk, Y. 1998. *An Overview of Decision Networks and Organizations*. IEEE Trans. on Systems, Man and Cybernetics - Part C: Applications, Vol. 28(1):172-192.

Pattison, H., Corkill, D.; and Lesser, V. 1987. *Instantiating descriptions of organizational structures*. In Huhns, M. (ed.) *Distributed Artificial Intelligence*. London, Pittman.

Pollack, M.; and Horty, J. 1999. *There's More to Life Than Making Plans: Plan Management in Dynamic Multiagent Environments*. AI Magazine 20(4):71-83.

Prasad, N, Decker, K., Garvey, A.; and Lesser, V. 1996. *Exploring organizational designs with TAEMS: A case study of distributed data processing*. Proceedings of the Second International Conf. on Multi-Agent Systems (ICMAS-96), 283-290.

Rich, C.; and Sidner, C. 1997. COLLOGEN: *Applying Collaborative Discourse Theory*. AI Magazine.

Rosenschein, J.; and Breese, J. 1989. *Communication-free interactions among rational agents: A probabilistic approach*. In Gasser and Huhns (eds.) *Distributed Artificial Intelligence volume II*, 99-118, Morgan Kaufmann Publishers.

Rosu, D., Schwan, K., Yalamanchili, S.; and Jha, R. 1997. *On adaptive resource allocation for complex real-time applications*", In Proceedings of the 18th IEEE Real-Time Systems Symposium, San Francisco.

Sacerdoti, E. 1974. *Planning in a hierarchy of abstraction spaces*. Artificial Intelligence, 5(2):115-135.

Sacerdoti, E. 1977. A Structure for Plans and Behavior. Elsevier, New York.

Sandholm, T., 1993. *An implementation of the contract net protocol based on marginal cost calculations*. Proceedings of the National Conference on Artificial Intelligence, 256-262, Washington DC.

Sandholm, T. 1999. *Distributed Rational Decision Making*. In Gerhard Weiss (ed.). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge MA.

Schut, M.; and Wooldridge, M. 2001. *Principles of Intention Reconsideration.* Proceedings 5th International Conference on Autonomous Agents AA2001, 340-347, Montreal, Canada.

Seghrouchni, A.; and Haddad, S. 1996. *A recursive model for distributed planning*. Proceedings of the Second International Conf. on Multi-Agent Systems (ICMAS-96), 307-314.

Sen, S.; and Durfee, E. 1996. *A contracting model for flexible distributed scheduling*. Annals of Operations Research, vol. 65, 195-222.

Sergot, M. 2001. *A computational theory of normative positions*. ACM Transactions on Computational Logic.

Shaham, Y.; and Tennenholtz M. 1992. *On the synthesis of useful social laws for artificial agent societies*. Proceedings of the Tenth National Conf. on Artificial Intelligence (AAAI-92), 276-380.

Shoham, Y.; and Tennenholtz, M. 1994. *On social laws for artificial agent societies: off-line design*. Artificial Intelligence 73(1-2):231-252.

Singh, M. 1999. *An ontology for commitments in multi-agent systems*. Artificial Intelligence and Law, vol. 7, 97-113.

Smith, D., Parra E.; and Westfold S. 1996. *Synthesis of Planning and Scheduling Software*, In Tate, A. (ed.). *Advanced Planning Technology*, AAAI Press, Menlo Park, California, 226-234.

Smith, R. 1980. *The Contract Net Protocol: High level communication and control in a distributed problem solver*. IEEE Transaction on Computers, (12):1104-1113.

Smith, S. 1994. *Reactive Scheduling Systems*. Brown E.; and Scherer W. (eds.), *Intelligent Scheduling Systems*, Kluwer Publishing.

So, Y.; and Durfee, E. 1996. *Designing tree-structured organizations for computational agents*. Computational and Mathematical Organization Theory, 2(3):219-246.

Srinivas, S.; and Breese J. 1990. *IDEAL: A software package for analysis of influence diagrams*. Proceedings of the Sixth Conference on Uncertainty in AI, 212-219.

Stankovic, J., Ramamritham, K.; and Cheng, S. 1985. *Evaluation of a flexible task-scheduling algorithm for distributed hard real-time systems*. IEEE Transaction on Computers, C-34(12):1130-1143.

Sugawara, T.1995. *Reusing past plans in distributed planning*. Proceedings of the First International Conf. on Multi-Agent Systems (ICMAS-95), 360-367.

Sycara, K., Roth, S., Sadeh, N.; and Fox, M. 1991. *Distributed constrained heuristic search*. IEEE Transactions on Systems, Man, and Cybernetics SMC-21(6): 1446-1461.

Sycara, K. *Retsina*. http://www.cs.cmu.edu/~softagents/retsina.html.

Tambe, M. 1997. *Towards Flexible Teamwork*. Journal of Artificial Intelligence Research, 7:83-124.

Tambe, M.; and Jung, H. 1999. *The Benefits of Arguing in a Team*. AI Magazine, 20(4): 85-92.

Tennenholtz, M. 1995. *On computational social laws for dynamic non-homogeneous social structures*. Journal of Experimental and Theoretical Artificial Intelligence.

Weiss, G. (eds.) 1999. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge MA.

Weld, D. 1999. *Recent Advances in AI Planning*. AI Magazine.

Wellman, M. 1993. *A market-oriented programming environment and its application to distributed multicommodity flow problems*. Journal of Artificial Intelligence Research, 1:1-23.

Werkman, K. 1992. *Multiple agent cooperative design evaluation using negotiation*. Proceedings of the Second International Conference on Artificial Intelligence in Design, Pittsburgh PA.

Wexelblat, A.; and Maes, P. 1999. *Footprints: History-Rich Tools for Information Foraging*. CHI'99 Proceedings, ACM Press.

White, J. 1994. *Telescript Technology: The foundation for the electronic marketplace*. White paper, General Magic Inc.

Wilkins, D.; and Myers, K. 1995. *A common knowledge representation for plan generation and reactive execution*. Journal of Logic and Computation, 5(6): 731-761.

Wilkins, D.; and Myers K. 1998. *A Multiagent Planning Architecture*, Proceedings of AIPS-98, 154-162.

Wooldridge, M.; and Jennings, N. 1998. *Pitfalls of Agent-Oriented Development*. Proceedings 2nd International Conference On Autonomous Agents (Agents-98), Minneapolis, USA, 385-391.

Yokoo, M., Durfee, E., Ishida, T.; and Kuwabara, K. 1998. *Distributed Constraint Satisfaction Problem: Formalization and Algorithms*. IEEE Transactions on Knowledge and Data Engineering, TKDE10(5):673-685.

Yokoo, M.; and Ishida, T. 1999. *Search Algorithms for Agents*." In Gerhard Weiss (ed.). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, Cambridge MA.

Zambonelli, F., Jenning, N.; and Woolridge, M. 2001. *Organizational Abstractions for the analyss and design of multi-agent systems*. In R. Ciancarini and M. Woolridge (eds.) *Agent-Oriented Software Engineering*, Spinger-Verlag.

Zhang, C. 1992. *Cooperation under uncertainty in distributed expert systems*. Artificial Intelligence 56: 21-69.

Software Agents for the Warfighter
Part 2: Technology Components

Human-Agent Interaction

Brief Overview

Description

Future battlefield environments will involve complex interaction of many different entities, including human warfighters, decision-makers, coalition partners and their forces, sensors and sensor grids, software agents, UAVs and other unmanned vehicles, and all in highly dynamic, real-time environments. Within this space of complex interaction, the previous chapter focused on agent-agent interaction. This chapter will focus on human-agent interaction.

The basic problem in human-agent interaction can be summarized briefly: agents and humans are very different kinds of entities that exist in two very different kinds of worlds. For the foreseeable future there will be a fundamental asymmetry in their capabilities: the brightest agents will be limited in the generality if not the depth of their inferential, adaptive, and sensory capabilities; humans, though fallible, are functionally rich in reasoning strategies, their powers of observation and learning, their flexibility, and their sensitivity to context (Agnew and Brown 1989). Adapting to appropriate mutual roles that leverage the respective strengths of humans and agents, and crafting natural and effective modes of interaction are the key challenges that researchers are working to overcome.

A persistent misperception about all forms of automation is the notion that such assistance is a simple multiplier of human capability. In reality, however, help of whatever kind does not in reality simply enhance our ability to perform the task: it changes the nature of the task itself. As with all automation, the introduction of agents into human work practices must be done carefully to ensure that the cost of the coordination and monitoring demands on the human do not exceed the value of the agent assistance offered.

Researchers have specialized in various aspects of the human-agent interaction problem. Some of the key components intended to help address the challenges include:

- *Teamwork*: Teamwork has become the most widely-accepted metaphor for describing the nature of agent-agent interaction. However, human-agent teamwork must address much richer and more complex issues than today's agent-agent teamwork research. Individual differences among humans and indeed differences in both style and substance between humans and agents on nearly every front make true teamwork among humans and agents a challenge. Current approaches to human-agent teamwork are providing many lessons and some encouraging directions.

- *Observability and Interaction Style:* Effective teamwork requires that relevant aspects of the agents and the situation be observable at an appropriate level of abstraction. Although this is as much a requirement for agent-agent teamwork as it is for human-agent teamwork, the size of the representational gulfs separating humans from agents is much larger, and agent designers must find innovative ways to compensate for these limitations. Beyond the content of what team members need to know, we need to consider the form in which this information is exchanged and various styles of humans-agent interaction.

- *Adjustable Autonomy:* The goal of designing mixed-initiative systems with adjustable autonomy is to make sure that for any given context the agents are operating at an optimal boundary between the initiative of the human and that of the agent. To the extent we can adjust agent autonomy with reasonable dynamism (ideally allowing fine-grained handoffs of control to occur "anytime") and with a useful range of levels, teamwork mechanisms can flexibly renegotiate roles and tasks among humans and agents as needed when new opportunities arise or when breakdowns occur.

- *Safety and Privacy: ***

- *Trust: ***

- *Adaptability: ***

Relevance to the Warfighter

Agents can assist warfighters and decision-makers by providing them key information, taking initiative to offer assistance, and facilitating coordination among warfighters. In general, they can enhance warfighter capability by taking on "low level" tasks on their behalf, e.g., routine monitoring, but obtaining warfighter input in critical situations as needed. The difficult issue here is to make sure that warfighters are always in the control loop, ready to act appropriately, without overburdening them with low-level monitoring and decision responsibilities. Warfighters and agents must be able to continuously observe relevant aspects of the situation—and of each other—at an appropriate level of abstraction. This shared awareness can be enhanced through the use of mediating representations (Ford *et al.,* 1993) that provide means to visualize and manipulate relevant aspects of the situation. Furthermore, the interaction should take into account the role of the individual within formal organizations and informal social and technological networks consisting of warfighters, decision-makers, agents, sensors, UAVs and so forth, enabling more rapid response to national or international crises.

Risks

Up to now, most agent researchers have taken a technology-centric rather than a human-centric approach to the study of human-agent interaction, with more emphasis on logical than psychological issues. As agents increase in sophistication and richness of interaction with humans, there will be a critical requirement for better understanding of the novel cognitive, emotional, and social dimensions that will emerge. Sophisticated generic methods have been developed and used to analyze the details of human-computer interaction(e.g., Card, Moran & Newell, 1983). More recently, new approaches have been developed to take into account multi-agent

environments (e.g., Boy, 1998; Hutchins, 1995), and to use agents to model work activities against a rich background of organizational and situational contexts (Clancey *et al*. 1998; Sierhuis 2001). The eventual success of these still-maturing new approaches is a vital part of helping researchers understand and address important general social issues (e.g., safety, privacy, security) that will determine the acceptability of agents for various applications. (**Milind, feel free to add clarifications on technical vs. social aspects here)

Forecast

We assume in this forecast that the government will continue to sponsor major agent related research efforts that may then feed into industrial research. With this assumption we forecast the following. First, we may soon see real commercial products that exploit human-agent interaction technologies. Indeed, the presence of systems developed by industrial research labs illustrates that technology for development of small-scale human-agent interaction systems suitable for deployment in limited situations with warfighters is already here (2002-2004). We predict that agents will be able to support not only individuals but also increasingly complex organizations such as heterogeneous virtual organizations spanning multiple agencies and coalition forces. We also anticipate that human-agent interaction will increase in flexibility and adaptivity, as agents develop the capability of negotiating with humans and agents regarding tasks and resources. However, issues such as trust and safety may hinder actual deployment in the near term, requiring significant attention. An integrated approach that takes into account people and agents from the early start of the design and development process is required, i.e., human-centered development (HCD). It requires both cognitive simulations of the cognitive functions involved in the overall human-agent system, as well as an experimental set-up where people and agent prototypes are involved. This HCD process will be developed within the decade (2010). It is anticipated that acceptance of such virtual human-agent organizations may depend on non-technological organizational issues.

Summary and Recommend-ations

Human-agent interaction could not only significantly enhance individual user capabilities, but they may also provide tremendous benefits to large and small military units by enabling rapid coordinated response to crises and adaptive operations in changing circumstances. The presence of human-agent interaction tools from leading industrial research labs illustrate that this technology has matured significantly, at least in the realm of individual human-software agent interaction. Additional research should be devoted to issues in human-agent teamwork.. Furthermore, general issues such as safety, trust, privacy, and security, which might be at the core of acceptance of agent technology in large-scale human organizations, should be more fully investigated within the framework of human-agent systems. (**Milind, feel free to add clarifications on technical vs. social aspects here)

Relevance to the Warfighter

Advanced Sensor Grids

Advanced sensor grids are clearly highly relevant to providing warfighters a clear understanding of an evolving situation. However, this specific scenario does not involve human-agent interaction except insofar as agents are used to mediate controls and displays of this grid-derived information.

Unmanned Autonomous Systems

In this scenario, agents assist humans in monitoring and detection, keeping track of multiple targets and their level of threat, and aiding humans in making decisions under severe time pressure. This scenario emphasizes the need for adjustable autonomy in agents, whereby agents may assume control of monitoring and detection in routine situations, but in critical situations, revert control to humans.

Advanced Command Posts

In advanced command posts, agents should be designed to support operations within a human organization, including assisting humans in their collaboration. Several different types of "configurations" are possible to accomplish this goal. In the simplest configuration, an individual agent may assist an individual human user. This individual agent may then interact with other agents, including those that are assisting other users. In a more complex configuration, groups of agents may assist individual users. Such mixed human-agent groups may then interact with other human-agent groups, forming a more complex, large-scale organization. In both configurations, requiring agents and humans to act in a peer-to-peer team can provide the required robustness and flexibility of operations. Furthermore, these mixed human-agent teams may team up with other teams to scale-up towards large-scale teams, while providing organizational robustness and behavioral flexibility. Such teams of teams may be the key to forming large-scale virtual organizations.

Mobile Operations

Mobile operations for urban terrains (MOUT) are a representative scenario for mobile operations in general. Agents in MOUT scenarios clearly interact with humans as teammates, illustrating the need for human-agent teamwork. Furthermore, agents must often exhibit adjustable autonomy in their interaction. In particular, they may reduce their own level of autonomy, giving control of key decisions to humans, e.g., asking a human about contacting UAVs for assistance in surveillance or vice versa. The scenario also explicitly mentions the need for safety. Finally, humans in the scenario interact with agents via wearable, handheld or mobile devices.

Joint/Coalition Operations

This scenario again requires an organization of agents to assist a networked organization of humans. While the human-agent interaction requirement in this scenario overlaps in requirements with the command post scenario mentioned earlier, there is potentially a stronger emphasis on mixed-initiative planning in this case. In particular, agents plan tasks in a mixed-initiative manner with humans to avoid interference with coalition partners and humanitarian operations. Furthermore, this scenario also emphasizes safety and trust issues in human-agent interaction. For instance, a safety constraint on agent operations may be to avoid any problems in coordination regarding areas of operation with coalition forces, so as to avoid any potential friendly fire incidents.

Human-Agent Interaction, continued

Logistics

This scenario also requires an organization of agents to act in support of a human organization. While agents lower down in the command hierarchy interact with human users to ensure that appropriate information is sent to higher-levels of the command, agents at higher levels monitor deviations from plans and ensure appropriate notifications to human users. There are clear parallels with scenarios discussed above, such as the command post scenario, emphasizing the need for human-agent teamwork, and mixed-initiative and adjustable autonomy in agents to interact within such teams.

Information Assurance

Information assurance is clearly critical in delivering key information to human users. It is also critical to at least some aspects of trust between an agent and a user. However, issues involving human-agent interaction are not relevant in this case; rather issues of interaction among agents are relevant where information assurance is emphasized.

Technical Description

In this section we consider various aspects of human-agent interaction, especially those that make it so different from pure agent-agent interaction. We do this in two parts:

First, we consider the problem of coordination of human and agent roles, sketching out a spectrum of possibilities and attempting to dispel some of the myths that surround naïve approaches to the design of systems intended to provide intelligent assistance to humans. We explain some of the difficulties caused by what Norman (1992) calls the "gulf of execution" and the "gulf of evaluation."

Next, we consider in turn various components of successful human-agent interaction: teamwork, observability and interaction style, adjustable autonomy, safety and privacy, and trust.

Most of the work in human-agent interaction is relatively recent. This is in part because much of the early research was motivated by situations in which autonomous systems were envisioned to "replace" human participation. For example, unacceptable latency in ground-based control of deep-space satellites motivated the development of NASA's Remote Agent Architecture (RAA). RAA was designed to be used in situations where the length of round-trip control sequences from earth would have impaired the satellite's ability to respond to urgent problems or take advantage of unexpected science opportunities (Muscettola, Nayak, Pell and Williams 1998). In contrast to autonomous systems designed to take humans out of the loop, the autonomous capability of recent agent research efforts such as the Personal Satellite Assistant is specifically motivated by the need to support close human-agent interaction (Bradshaw *et al*. to appear). Because these topics are new within the agent research community, very few journal articles or major conference papers are available. Thus, the majority of our references will be either to agent symposia or workshop papers, or to relevant sources from outside disciplines that bear on the topic.

Challenges of Human-Agent Interaction

A basic premise of human-centered teamwork is that humans and agents[1] are two very different kinds of entities that exist in very different kinds of worlds. For the foreseeable future there will be a fundamental asymmetry in their capabilities: the brightest agents will be limited in the generality if not the depth of their inferential, adaptive, social, and sensory capabilities; humans, though fallible, are functionally rich in reasoning strategies and their powers of observation, learning, and sensitivity to context (Agnew and Brown, 1989). Moreover, agents interact directly and efficiently in cyberspace but indirectly and awkwardly in the material sphere; humans shine in the world of atoms but cannot juggle bits on their own. Adapting to appropriate mutual roles that take advantage of the respective strengths of

[1] Throughout this paper we use the term "agent" to refer to "artificial" (i.e., software or robotic) agents.

humans and agents, and crafting natural and effective modes of interaction are key challenges.

All this being said, we do not wish to extend here the long tradition of MABA-MABA (men-are-better-at/machines-are-better-at) lists that began with the classic report of Fitts (1951). The point is not to think so much about which tasks are best performed by humans and which by agents but rather how tasks can best be shared to be done by *both* humans and agents working in concert (Hancock and Scallen 1998). Licklider (1960) called this concept *man-computer symbiosis;* within DARPA this concept has fueled the creation of the Augmented Cognition Program (http://www.darpa.mil/ito/research/ac/). Ford describes such symbiosis in an ultimate form where human capabilities are transparently augmented by *cognitive prostheses*—computational systems that leverage and extend human intellectual, perceptual, and collaborative capacities, just as a steam shovel is a sort of muscular prosthesis or eyeglasses are a sort of visual prosthesis (Bradshaw *et al.,* 2002b; Ford *et al.,* 1997; Hamilton 2001).[2] (Englebart-human augmentation)

Coordination of Human and Agent Roles

Figure 1 depicts a range of possible roles that humans and agents may play with respect to one another with varied degrees of agent initiative present.[3] At the one extreme, traditional systems are designed to carry out the explicit commands of humans. At the other end of the spectrum is an imagined extreme in which agents would control humans.[4] Between these two extremes is the domain of today's agent systems, with most agents typically playing fixed roles as servants, assistants, associates, or guides. Although in practice many do not live up to their billing, the design goal of mixed-initiative systems is to allow agents to dynamically and flexibly assume a range of roles depending on the task to be performed and the current situation (Ferguson, Allen, and Miller 1996, Burstein and McDermott 1996). A challenge is to assure that the role and degree of autonomy is continuously and transparently adjusted to be context appropriate and within the bounds of policy—a topic discussed below under the heading of

---

[2] As a corollary to this line of thinking, Hoffman, *et al*. (2002) propose the "triples rule," which asserts that the basic unit of analysis for cognitive engineering and computer science is the triple of person, machine, and context. This same triple becomes the basic unit for the analysis of intelligence in joint human-machine systems.

[3] For a more fine-grained presentation of a continuum of control between humans and machines, see Hancock and Scallen's (1998) summary of Sheridan's (1980) ten-level formulation.

[4] Of course, in real systems, the relative degree of initiative that could be reasonably taken by an agent or human would not be a global property, but rather relative to particular functions that one or the other was currently assuming in some context of joint work (see Boy to appear; Hancock and Scallen 1998; Barber and Martin 1999; Goodrich *et al*. 2001).

adjustable autonomy. Moreover, the human must be aware of what the agent is doing, why it is doing it, and where it is on its current "agenda."

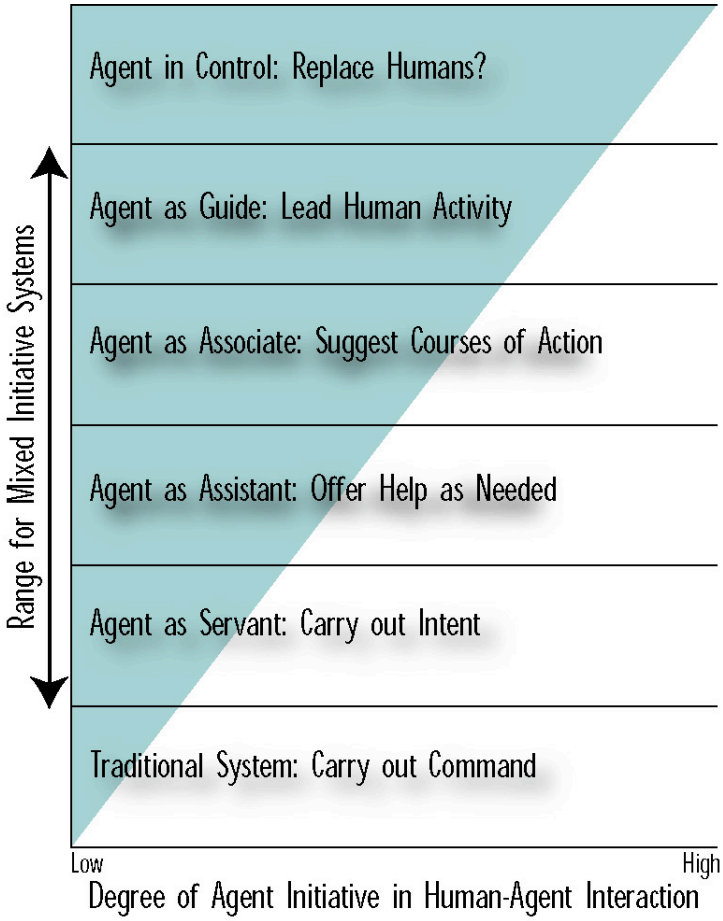


Figure 1. Spectrum of agent roles in human-agent interaction.

Agent as servant

An agent in the role of servant offers an advantage over traditional systems in that it would typically offer at least a minimal degree of flexibility in how it carries out the intent of the human. The limited autonomy of an agent on such a short leash means that it would frequently need to return to a human for help in the face of failure or unforeseen conditions, something which may or may not be desirable to the human. For instance, consider the mobile operations scenario in urban terrain, discussed earlier. Here, a human may order an agent controlling an Unmanned Ground Vehicle (UGV) to "*follow me*." Given unfavorable world conditions that prevent the UGV from being able to follow, the agent might simply fail and report its failure. A more helpful response for the agent would have been to suggest an alternative method to achieve the same goal, but in simple agents of this sort, such a capability is lacking. This is because these agents typically focus on the tasks they have been assigned, with little or no "awareness" of the broader context.

Though such agents may be incapable of acting as teammates with humans and other agents, they may be adequate for many types of routine tasks that

require simple inference and adaptation. Many interface agents, including web agents, bots, wizards, mail filtering agents, and so on, fall in this category. While the tasks they accomplish require significant domain-level expertise (which is very important), they have limited capability to ensure flexible interaction with humans.

Agent as assistant

To function as an assistant, an agent must have some awareness of the context in which the other humans and agents with which it interacts are functioning. It must understand when and how it is appropriate to assist and coordinate actions with others. In other words, it needs to be a teammate to a human in at least some minimal sense.

It should be noted that agent assistants can provide help in two different forms: one is in an advisory capacity, where the agent helps by simply informing the human about what can be done; the other is in an executive capacity where, like the agent servant, the agent assistant actually helps carry out some action on behalf of the user. As Lieberman and Selker (to appear) observe, keeping agents in an advisory mode has the advantage of avoiding "many of the problems of loss of responsibility feared by critics of agents" (p. **)

While developing agent assistants takes more work than developing agent servants, the increased autonomy of assistants can improve operational robustness since team members may provide each other mutual assistance in service of their common goal, including ensuring critical information is provided to teammates in a timely fashion. Consider the above "*follow me*" master-servant relationship where the agent simply failed. If instead, the human and the agent had formed a team to accomplish their joint goal, the agent may have recognized a threat to this joint goal and suggested alternative methods to accomplish the desired objective. For instance, an agent may suggest on its own that it will watch the flanks as it follows the human. Unlike traditional systems or agent servants, the agent assistant can continue to anticipate and possibly respond to newly-identified needs while the human is not actively attending to it.

Agent as associate or guide

Agents in the role of associates or guides extend the capabilities identified above in ways that allow them to automatically or semi-automatically fill-in for failed human or agent teammates, rapidly reorganizing team roles as necessary. This could lead to improved flexibility of operations. For instance, in the above "follow me" example, an agent associate or guide may suggest that following the human is not the best plan. Instead, it may suggest "let's first send in reconnaissance robots before moving forward." Indeed such an agent, if the policies given it by humans allow, may be permitted to take autonomous actions without necessarily consulting with the human, in the interest of achieving the joint goal.

The Substitution Myth

A persistent misperception about all forms of automation is the notion that such assistance is a simple multiplier of human capability. Such a view is natural because, from the point of view of an outsider observing the assisted human, it seems that—in successful cases at least—the person is able to

perform the task better or faster than he or she could without help. In reality, however, help of whatever kind does not simply enhance our ability to perform the task: it changes the nature of the task itself (Norman 1992). Those who have had a five-year-old child offer to help them with the dishes know this to be true—from the point of view of an adult, such "help" does not necessarily diminish the effort involved, it merely effects a transformation of the work from the physical action of washing the dishes to the cognitive task of monitoring the progress (and regress) of the child.

Ignorance of such considerations leads to what Wiener (1989) called "clumsy automation" and what Christoffersen and Woods (2002) term the "substitution myth": the erroneous notion that "automation activities simply can be substituted for human activities without otherwise affecting the operation of the system" (p. **). In refutation of the substitution myth, Table 1 contrasts the putative benefits of automated assistance with the results of empirical study.

Putative Benefit	**Real Complexity**[5]
Better results are obtained from "substitution" of machine activity for human activity.	Transforms practice; the roles of people change; old and sometimes beloved habits and familiar features are altered—the *envisioned world problem*.
Frees up human by offloading work to the machine.	Creates new kinds of cognitive work for the human, often at the wrong times.
Frees up limited attention by focusing human on the correct answer.	Creates more threads to track; makes it harder for people to remain aware of and integrate all of the activities and changes around them.
Less human knowledge is required.	New knowledge and skill demands are imposed on the human.
Agent will function autonomously.	Team play with people is critical to success.
Same amount and kind of feedback to human will be required as before.	New levels and types of feedback are needed to support peoples' new roles.
Agent enables more flexibility to the system in a generic way.	Resulting explosion of features, options, and modes creates new demands, types of errors, and paths toward failure—*automation surprises*.

[5] See Hoffman's (1997) discussion of the concept of "complexification."

Human errors are reduced.	Both agents and people are fallible; new problems are associated with human-agent coordination breakdowns.

Table 1 Putative benefits of automation vs. actual experience (adapted from Woods 1997).

Notwithstanding these challenges, adult humans and radically less-abled entities (e.g., children, dogs, video game characters) are capable of working together effectively in a variety of situations where a subjective experience of collaborative teaming is often maintained despite the magnitude of their differences. Generally this is due to the human's ability to rapidly size up and adapt to the limitations of their teammates in relatively short order. As with all automation, the introduction of agents into human work practices, particularly agents who do not yet generally exhibit the intelligence of a five-year old child, must be done carefully to ensure that the cost of the coordination and monitoring demands on the human do not exceed the value of the agent assistance offered. However, through the development of increasingly effective means to exploit synergies provided by appropriate combinations of human and agent capabilities, significant benefits can be accrued even from today's relatively simple agent systems. Thus research in human-agent interaction can powerfully leverage the value of nearly every other aspect of intelligent systems research.

The Gulfs of Evaluation and Execution

A further challenge to effective human-agent interaction is that the agents necessarily interpose a level of indirection between ourselves and actions in the world. This indirectness can often lead to situations where we are misled in our expectations about the state of the world and the effects of our actions (Figure 2):

> "The gulfs of execution and evaluation refer to the mismatch between our internal goals and expectations and the availability and representation of information about the state of the world and how it might be changed. The gulf of execution refers to the difficulty of acting upon the environment (and how well the [agent] supports those actions). The gulf of evaluation refers to the difficulty of assessing the state of the environment (and how well the [agent] supports the detection and interpretation of that state)…
>
> We can conceptualize the [agent] and its interface in this way. A person is a system with an active, internal representation. For an [agent] to be usable, the surface representation must correspond to something that is interpretable by the person, and the operations required to modify the information within the [agent] must be performable by the user. The interface serves to transform the

properties of the [agent's] representational system to those that match the properties of the person." (Norman 1992)[6]

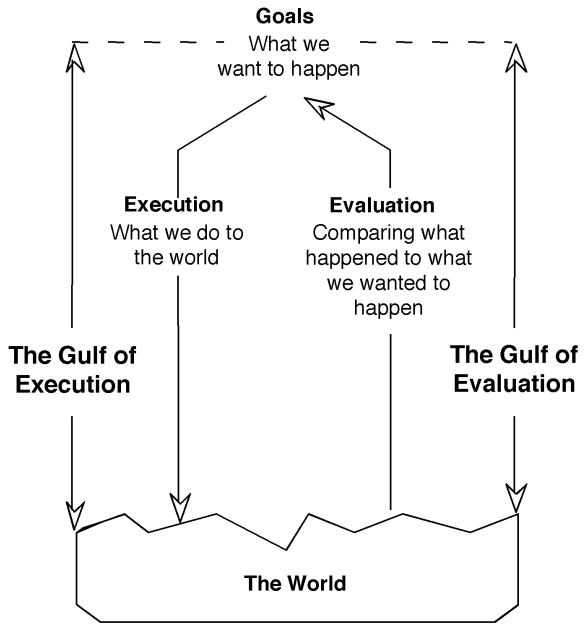


Figure 2. The gulfs of execution and evaluation (Norman 1992)

In brief, people need to *understand* what happened and why when a teammate alters its response; they need to be able to *control* the actions of an agent even when it does not always wait for the human's input before it makes a move; and they need to be able to reliably *predict* what will happen, even though the agent may change its responses over time (Erickson 1997). Compounding the challenge is the fact that sophisticated agents will also have similar needs to understand, guide, and predict the human.

Discovering means for humans to more effectively overcome these gulfs with agents—and vice versa—is a prime focus of current human-agent interaction research that we discuss in more detail below.

Components of Human-Agent Interaction

Billings (1997) describes a set of first principles of human-centered systems that are important to consider as a preface to a discussion of components of human-agent interaction:

"*Premise:* Humans are responsible for outcomes in human-machine systems.

Axiom: Humans must be in command of human-machine systems.

---

[6] We do not wish to imply that we are here taking a stance that the world is presented to us directly. Rather, as George Kelly elaborated in his principle of constructive alternativism, "'reality' does not reveal itself to us directly, but rather is subject to as many different constructions as we are able to invent" (Bradshaw *et al.* 1993, p. 288). Such considerations of the fluidity of meaning and interpretations are only recently being broadened within human-agent interaction research.

Corollary: Humans must be actively involved in the processes undertaken by these systems.

Corollary: Humans must be adequately informed of human-machine system processes.

Corollary: Humans must be able to monitor the machine components of the system.

Corollary: The activities of the machines must therefore be predictable.

Corollary: The machines must also be able to monitor the performance of the humans.

Corollary: Each intelligent agent in a human-machine system must have knowledge of the intent of the other agents."[7]

Note that Billings' main premise ("Humans are responsible for outcomes in human-machine systems") implicitly assumes a fundamental asymmetry between humans and today's agents. Notwithstanding this assumption, we expect the broad balance between human and agent initiative and responsibility to co-evolve commensurate with the degree of trust humans are willing (or required) to exercise in particular kinds of technology for specific contexts of use. Already we rely on technology to do things automatically for us that were unthinkable not too long ago.

The corollaries to this premise serve to underscore the importance of maintaining appropriate mutual awareness among team members. Each actor, both human and agent, must not only be able to realistically assess the overall situation and current the state of the other team members, but also to accurately ascertain intent and reliably predict future states.

Table 2 below relates the two challenges outlined above to various components of human-agent interaction described below that are intended to help address them. Each of these will be addressed in more detail in the appropriate subsections below.

	Coordination of Human and Agent Roles	**The Gulfs of Evaluation and Execution**

[7] Note that this applies both to humans and to software agents.

Teamwork	Develop general principles and mechanisms supporting human-machine teamwork that can be reused in many situations.	Provide agent services and tools to create, manage, and visualize teamwork-related aspects of the situation.
Observability and Interaction Style	Allow for the possibility of different modes and interaction styles for agents playing different kinds of roles.	Assure that relevant aspects of the agents and the situation are observable at an appropriate level of abstraction and in an appropriate interaction style. Provide negotiation capability and tools for shaping shared understanding.
Adjustable Autonomy	Implement means for fine-grained handoffs between human and agent control of tasks to take place at any time.	Provide tools to allow activity to be dynamically redirected.
Safety and Privacy	Ensure that appropriate bounds can be set and enforced on agent behavior and access to private information.	Provide feedback to establish confidence in the human that agent is behaving safely and respecting privacy.
Trust	Facilitate human acceptance of appropriate degree of agent autonomy.	Provide feedback to establish confidence in human that agent is performing predictably, reliably, and benevolently.
Adaptivity	Provide means for agents to tailor their coordination style.	Provide means for agents to tailor their behavior to situation, task, and individual and group differences.

Table 2. Challenges in human-computer interaction related to various components that are intended to help address them.

Teamwork

Teamwork has become the most widely-accepted metaphor for describing the nature of human-agent interaction:[8] "The overarching point from the

[8] Note that although teamwork is currently the most studied perspective and hence the one most explored in this chapter, this does not imply that teamwork is necessarily the *only* or the *most effective* way of looking at human-agent interaction in all situations.

research is that for any non-trivial level of automation to be successful, the key requirement is to design for fluent, coordinated interaction between the human and machine elements of the system. In other words, automation and intelligent systems must be designed to participate in team play" (Christoffersen and Woods 2002).

In most approaches to human-agent teamwork, as in typical agent-agent teamwork formulations, the key concept is that of shared knowledge, goals, or intentions, which functions as the glue that binds team members together (Cohen and Levesque, 1991). By virtue of a largely-reusable explicit formal model of shared intentions, general responsibilities and commitments that team members have to each other are managed in a coherent fashion that facilitates recovery when unanticipated problems arise. For example, a common occurrence in joint action is when one team member fails and can no longer perform in its role. The general teamwork model entails as a formal consequence that each team member be notified under appropriate conditions of the failure, and so does not require special-purpose exception handling mechanisms to do this for each possible failure mode.[9]

As described previously, however, human-agent teamwork must address more richer and more complex issues than today's agent-agent teamwork research. For example, even if the behavior of humans could someday be adequately *described* by today's simplistic agent teamwork models, the behavior of humans cannot be absolutely *prescribed* by such models the way that the behavior of computational agents can. Nor given the state of today's intelligent systems would we generally want them to be.[10] A further complication is the increased difficulty of maintaining appropriate situation awareness for all team members, when the most natural and effective representations of the situation for humans typically differ so greatly from those that are ideal for agents. Differing limits in computational and human cognitive resources (e.g., attention) must be continuously taken into account. Moreover, the range of coordination strategies possible for teams is very wide, and must be tailored to unique aspects of the situation, physical locations, and capabilities of the players. In short, individual differences among humans and, indeed, differences in both style and substance between humans and agents on nearly every front make true teamwork among humans and agents a challenge.

It is important to note at this juncture that researchers in human-agent teamwork have used the term in two broad ways: 1) as a conceptual analogy for heuristically directing research (e.g., to build systems that facilitate fluent, coordinated interaction between the human and agent elements of the system

---

[9] See the chapter on agent-agent interaction for more detail on general teamwork theories.

[10] This is consistent with Billings' (1997) well-known premise in his list of first principles of human-centered systems: "Humans are responsible for outcomes in human-machine systems."

as "team players") and 2) as the subject matter for research (e.g., to understand the nature of teamwork in people). The first activity focuses on practical engineering of useful systems through application of human-centered design principles, empirical studies of the use of these systems, and often a limited commitment to studying teamwork among people. The second activity is explicitly framed as a scientific study, and may have two angles: 1) providing information relevant to the design of successful human-agent systems, and 2) independent of application, understanding the nature of cognition in people and animals. The latter activity is seen by these researchers as essential for achieving the ultimate goals of artificial intelligence. An adequate approach to human-agent teamwork must reflect sensitivity to both research traditions: neither undervaluing the independent study of social and cognitive aspects of human teamwork, nor slavishly imitating superfluous aspects of natural systems in the development of artificial ones, like an engineer who insists that successful airplane designs must necessarily feature flapping wings because all birds do (Ford & Hayes,[11] 1998).

We will now briefly review a sampling of common types of approaches to human-agent teamwork.

Approaches Based on Collaboration Theory

Despite the asymmetry and heterogeneity in the human-agent relationship, enabling agents to explicitly represent their interaction with humans in terms of accomplishing their common joint goals is seen by many as key to flexible and robust human-agent interaction. Perhaps the strongest proponent of this view is Grosz (1996), but the advantages of such human-agent team relationships have also been discussed in Grosz and Sidner (1990), Rich and Sidner (1998), and others.

The Collagen system (Rich and Sidner 1998) is based on the SharedPlans (Grosz 1996; Grosz and Sidner 1990; Grosz and Kraus 1996) theory of collaboration. Collagen is a Collaborative Agent that engages users in a collaborative discourse regarding applications of interest. The collaborative discourse in this case is in service of the joint goal between the agent and the user to accomplish a task. For instance, Collagen may assist users via a collaborative dialogue in achieving their joint goal of setting up and programming a video cassette recorder, or in planning the user's air travel and so on.

Collagen exploits a key aspect of human-agent collaboration: the agent's use of plan-recognition to model a user's intentions (Lesh *et al.* 1999, Van Beek and Cohen 1991). Given such plan-recognition capabilities, a human is not burdened with communicating all its intentions explicitly to the agent at each instant, thus enabling more flexible human-agent teamwork. In particular, the agent attempts to infer the user's intentions autonomously and take

---

[11] Clancey (2002) eloquently argues that the term "collaboration" (as also the terms "intelligent," "teamwork," and "knowledge") should be reserved for discussions involving people rather than today's software or robotic agents. While his concerns about the lack of precision with which these terms are thrown about and the profound difference between these phenomena as they are manifest in humans vs. artificial systems have merit, in this chapter we will not attempt to swim against the tide.

appropriate helpful action. [12] With documented reuse across different applications and different users (Rich, Sidner, and Lesh 2001), Collagen illustrates that building human-agent teams (or at least human-agent pairs) for some kinds of tasks is within our reach.

The use of proxies

In some teamwork approaches, humans interact indirectly with agents through explicitly-represented proxies. Each proxy is capable of acting as a team member. An example of this approach is seen in the Teamcore system developed by Tambe (Tambe *et al*. 2000), where each proxy uses the STEAM system (Tambe 1997) as a model of teamwork.[13]

Several researchers and research groups have attempted to use software agents on a daily basis in routine activities. The first such report was from (Kautz et al 94), who focused on using personal assistant agents to work together to schedule talks and meetings at AT&T research labs. Later (Zeng and Sycara 96) report on a similar experiment with software agents at Carnegie Mellon University. Both of these research projects focused on *"visitor hosting"* as an application.

In 2000-2001, the *Electric Elves* system was deployed for a period of several months at University of Southern California's Information Sciences Institute (USC/ISI) (Chalupsky et al 2001). This is another research system that provides proxies for individual researchers and students at USC/ISI, as well as proxies for a variety of schedulers, matchmakers, and information agents. The resulting team of 15-20 agents helps to reschedule meetings, decide presenters for research meetings, track people and even order meals for the researchers. The agents interact with the human users via hand-held devices such as web enabled phones, palm pilots, as well as via speech, fax, and computer monitors.

In this example, we see most of the issues discussed brought forward, including teamwork among agents and humans via proxies, adjustable autonomy and mixed initiative in agents when they act on behalf of users, and safety. For instance, adjustable autonomy issues arise here, since an agent assistant must inform other agent proxies if a user will attend a meeting or not. However, the agent may be uncertain about the user's intentions or whereabouts. In such cases, it reduces its own autonomy to obtain the user's input.

---

[12] See the agent architectures and capabilities chapter for a discussion of user modeling approaches.

[13] See the agent-agent interaction chapter for more detail on Tambe's work.

Human-Agent Interaction, continued

Fig: Electric Elves: An Experiment in Deploying Agents in Human Organizations

Interaction models

Sierhuis has observed that the longstanding emphasis in teamwork research on "joint goals" and fixed task-specific "roles" seems to have overshadowed important aspects of teamwork in practice such as shared context, awareness, identity, and history (Bradshaw *et al.*, 2002c). Unlike the relatively rigid joint intentions of typical agent teamwork models, experience in work practice underscores the importance of conceptualizing agreements as things that are forever tentative and subject to ongoing negotiation. These realizations have been important in the design of Brahms, a language coupled with an agent modeling and simulation environment that can capture complexities of observation, communication, and collaboration in the context of group work (Sierhuis 2001).

Brahms is based on the idea of "situated action" (Suchman 1987; Clancey 1997) and offers to the researcher a tool to represent and study the richness of activity theory and "work practice" (Clancey in press). A traditional task or functional analysis of work leaves out informal logistics, especially how environmental conditions come to be detected and how problems are resolved. Without consideration of these factors, analysts cannot accurately model how work and information actually flow, nor can they properly design software agents that help automate human tasks or interact with people as their collaborators. For these goals, what is needed is a model that includes aspects of reasoning found in an information-processing model, plus aspects of geography, agent movement, and physical changes to the environment found in a multi-agent simulation –interruptions, coordination, impasses, and so on. A model of work practice focuses on informal, circumstantial, and located behaviors by which synchronization occurs (such that the task contributions of humans and machines flow together to accomplish goals) and allows the researcher to capture (at least part of) the richness of activity theory.

Among other things, Brahms has been used to model collaborative activities between astronauts and ground crews, to study team breakdowns and find ways of preventing such breakdowns through improved communications or modified work practice. Its focus is now expanding to include real-time capability that can operate in conjunction with KAoS policies and agent services (Acquisti *et al.* 2002, Bradshaw *et al.* 2002). This combined system is being applied to the design and implementation of software agents and robots that will work closely in teamwork fashion with humans. Such robots, and the software agents that implement their behavior, need to be aware of the detailed social and environmental context in which they are operating and of the shared understanding of the astronauts they are assisting.

Augmented cognition

For some researchers, the ultimate in human-agent teamwork is the notion of agents that can function as extensions of the human brain *(cognitive prostheses)* and body *(robotic prostheses).* A key feature of such extensions is that the human can make use of such extensions more or less transparently—even forgetting they are present—just as humans with myopia don't think constantly about the fact that they are wearing contact lenses but rather about the phenomena that they see more effectively through them. DARPA's Augmented Cognition Program (http://www.darpa.mil/ito/research/ac/) is an example of an early pioneering effort focused on appropriately exploiting and integrating all available channels of communication from agents to the human (e.g., visual, auditory, tactile) and conversely sensing and interpreting a wide range of physiological measures of the human in real-time so they can be used to tune agent behavior and thus enhance joint human-machine performance.[14] For example, sets of system sensor agents (e.g., joystick), human sensor agents (e.g., EEG, pupil tracking, arousal meter), and human display agents (e.g., visual, auditory, tactile) could work together with a pilot to promote stable and safe flight, sharing and adjusting aspects of control among the human and a virtual crew member agent while taking system failures and human attention and stress loads into account (Bradshaw *et al.* 2002b).

Scaling up

One of the long-term challenges to human-agent teamwork is representing and reasoning about organizations of humans and agents on a large scale. Although systems composed of large numbers of simple agents have been developed, few systems involving highly sophisticated agents have grown beyond the size of several dozen. Thus this subsection is more of a prediction on how such systems may evolve than a report of results.

Teams of teams

One approach to scaling up is constructing teams of teams. Such teams of teams may embody required structure and leadership. For example, if we

---

[14] A related program focused on similar issues with a robotics emphasis is NSF's Robotics and Human Augmentation (http://www.interact.nsf.gov/cise/descriptions.nsf/5b8c6c912ebf7f9b8525662c00723201/5e8661fa698fe674852565d9005985ef?OpenDocument). See also DARPA's Mobile Autonomous Robot Software (MARS) Robotic Vision 2020 Program (http://www.darpa.mil/ito/solicitations/FBO_02-15.html).

consider the advanced command-post scenario or the logistics scenario, human-agent teams would need to be organized into teams of teams that are hierarchically controlled in a fashion similar to the way the military is organized today. The key point is that although specific human-agent teams need not be hierarchically organized, groups of teams can be organized, and dynamically reorganized, in many ways to support the particular concept of operation.

Yet, given that even teams of agents constructed today are of limited size (a dozen to two dozen agents), such scalability in teams of teams has not been fully investigated. The scalability issues would likely only get more complex with human-agent teams. The key issue is that while a team of teams has a common goal, to what extent are individuals within individual teams first class members of that team of teams? The following provides one illustrative issue in such scaling. Within an individual team, monitoring of one's teammates is recognized as a critical responsibility of individual team members, so as to monitor team failures, assist failing teammates etc. (Grosz and Kraus 1996; Kaminka and Tambe 2000). Algorithms and techniques have been devised to ensure that such monitoring can be conducted without overburdening individuals. However, with scaling to teams of teams, should each individual agent expend the resources to monitor all other agents in all other teams? If it does, then it would likely overburden itself with monitoring; if it does not, then the individual agent cannot assist other failing members of its team of teams.

As teams grow in size and organizational structure, ad-hoc collaboration across sub-team boundaries will often become prevalent. In order to model and manage such collaborations, it will be useful to take some of the existing models of collaborative interaction, such as Brahms, and try to scale them up to larger organizations. In principle it should be possible to do this by restricting the scope of situational awareness of each participant in the organization to just that which is necessary. Humans and agents only have limited information about what individuals in other parts of the organization are doing, and this limits the opportunities for cross-organizational collaboration. Even more difficult is the important task of being able to see things from the point-of-view of another organization or individual. Nevertheless, such collaborations do occur, and it will be important to model such cross-organizational teamwork. One question is how to detect and manage the conflicts in understanding that can arise in such situations, by negotiation and by management of commitments.

Virtual organizations

Complex teams of teams, containing a heterogeneous mix of different types of agents, humans, intelligent sensors, and other computational resources, could form the basis of virtual organizations. Traditional virtual organizations effort has focused on pooling people from a variety of existing organizations into a virtual organization, aided by computers and information technology (Mowshowitz 1997). With the rise of human-agent interaction, agents could also be first class members of these virtual organizations, enabling/facilitating military units in rapidly responding to crises and dynamically adapting behaviors to meet the challenge of a changing environment.

Such virtual organizations are likely to be developed within advanced command post or joint/coalition operations applications. However, as discussed earlier, creating such virtual organizations is not just a matter of computational efficiency. Rather, difficult issues such as handoffs of tasks from humans to agents or vice versa, or authority relationships among agents and humans also need to be addressed. In addition, complexities such as the need for agents to belong to multiple organizations with possibly conflicting goals, and the coordination of redundant agents with similar goals in the same organization must be deliberated. In such situations the formation of ad-hoc cross-organizational teams will likely become the rule rather than the exception, as the virtual organization reorganizes and regroups in order to respond to changing circumstances and facilitate operational efficiency.Military coalitions are examples of large-scale multi-faceted virtual organizations, which sometimes need to be rapidly created and flexibly changed as circumstances alter. The international Coalition Agents eXperiment (CoAX, see Allsopp *et al*. 2002) aims to show that multi-agent systems are an effective way of dealing with the complexity of agile and robust Coalition operations and enabling interoperability between heterogeneous components including legacy and actual military systems. Integrating information across a Coalition is not just a matter of employing technology — it involves the creation of a coherent 'interoperability of the mind' at the human level as well, where many social and cultural factors come into play. The mapping between the human and technical worlds is accomplished through KAoS policy-based agent domain management services are intended to allow for the specification, management, conflict resolution, and enforcement of policies within the specific contexts established by complex military organizational structures (Bradshaw *et al*. 2001). These tools and services help humans to control and visualize interaction among coalition entities.

Observability and Interaction Style

As noted earlier, effective human-agent interaction requires each party to do its part to assure that its state and behavior are observable at an appropriate level of abstraction and in an appropriate interaction style. Although this is as much a requirement for agent-agent teamwork as it is for human-agent teamwork, the size of the representational gulfs separating humans from agents is much larger. Moreover, because the agent's ability to sense or infer information about the human environment and cognitive context is so limited, agent designers must find innovative ways to compensate for the fact that their agents are not situated in the human world. Brittleness of agent capabilities is difficult to avoid because only certain aspects of the human environment and cognitive context can be represented in the agent, and the representation that is made cannot be "general purpose" but must be optimized for the particular use scenarios the designer originally envisioned. Without sufficient basis for shared situation awareness and mutual feedback, coordination among team members simply cannot take place, and this need

for shared understanding and feedback increases as the size of the team and the degree of autonomy increase.[15]

Observability

For many years, Clark has written about the concept of "common ground" as a foundational concept for joint conversational or perceptual experiences among people. The idea is that two people "cannot talk successfully to each other without appealing to their common ground," i.e., "the sum of their mutual knowledge, mutual beliefs, and mutual suppositions" (Clark 1992, p. 3). A similar concept seems useful in trying to understand how to maintain coherent human-agent coordination. Christofferson and Woods (2002) find it useful to think of this "common ground" in two separate but interdependent parts: "(1) a shared representation of the problem state, and (2) representations of the activities of [humans and] agents" (p. 4). Table 3 below gives examples of the kinds of things that humans and agents need to be able to know in teamwork settings.

[15] Van de Velde (1995) provides a useful discussion of the three coupling mechanisms that can enable coordination among multiple agents: knowledge-level, symbol-level, and structural. Symbol-level coupling occurs when agents coordinate by exchange of symbol structures. Knowledge-level coupling occurs when an agent, through observation, "rationalizes the behavior of multiple agents by ascribing goals and knowledge to them that, assuming their rational behavior, explains their behavior." Structural coupling, as discussed extensively by Maturana and Varela (1992), occurs when two agents "coordinate without exchange of representation,… by being mutually adapted to the influences that they experience through their common environment… For example, a sidewalk… plays a coordinating role in the behavior of pedestrians and drivers… [and] the coordination of [soccer] players (within and across teams) is mediated primarily by… the ball." Similarly, as Clancey argues (1993), the usefulness of the blackboard metaphor is that it provides an external representation that regulates the coordination between multiple agents.

Questions About the Shared Representation of the Problem State	Questions About the Representation of the Activities of Humans and Agents
What type of problem is it?	How did I get into this state?
Is the problem routine or difficult?	What are they doing now?
Is the problem high or low priority?	Why are they doing it?
What types of solution strategies are appropriate?	Are they having difficulties? Why?
How is the problem state evolving?	What are they doing to cope with difficulties? Are they likely to fail?
	How long will they be busy?
	What will they do next?

Table 3. Examples of the kinds of things that humans and agents need to be able to know in teamwork settings (adapted in part from Christofferson and Woods 2002).

The current generation of agents is capable of providing very little in the way of the kind of feedback to humans addressed above, and humans are neither excited about the prospect of having to tell the agents all these things explicitly nor are they knowledgeable enough to have developed generic cognitive instrumentation that would provide that information effortlessly. Even at this early stage, however, we do suspect that future concepts for agent feedback ought to be:

> "*Event-based:* Representations will need to highlight changes and events in ways that the current generation of state-oriented display techniques do not.
>
> *Future-oriented:* In addition to historical information, new techniques will need to include explicit support for anticipatory reasoning, revealing information about what should/will happen next and when.
>
> *Pattern-based:* Operators must be able to quickly scan displays and pick up possible abnormalities or unexpected conditions at a glance rather than having to read and mentally integrate many individual pieces of data."[16] (Christofferson and Woods, 2002, p. 6)

[16] Chernoff faces, which leverage the efficiency of human facial recognition into the domain of statistical analysis, are an ideal exemplar of integrative pattern-based feedback

Interaction Style

Beyond the content of what team members need to know, we need to consider the form in which this information is exchanged and various styles of humans-agent interaction. Shared awareness can be enhanced through the use of mediating representations (Ford *et al.,* 1993) that provide means to visualize and manipulate relevant aspects of the situation.

The choice of representation can have an enormous effect on human problem solving performance (e.g., Larkin and Simon 1987; Glasgow, Narayanan, and Chandrasekaran 1995). As a simple example, consider the impact that representing information as binary numbers, Arabic numerals, or Roman numerals has on the ability of humans or agents to efficiently multiply. As another example, concrete or abstract diagrams can be a particularly powerful form of knowledge representation for humans because they allow the explicit depiction of relevant information and effective hiding of inessential features which otherwise would have to be done at the expense of large amounts of cognitive work. Moreover the appropriate mode of interaction needs to be considered in the context of the task, physical environment, and the individual preferences and capabilities of the human under varying conditions of cognitive load. When required, interruptions of the human by an agent must be done judiciously—with no more than the just the necessary degree of obtrusiveness. In addition, as the level of sophistication of communication between agents and humans increases, future human-agent collaboration must enable people to negotiate with agents and work with appropriate tools for shaping shared understanding.

Erickson (1997) argues that designers ought to take advantage of the ontological expectations that users bring with them when they interact with various portrayals of functionality in graphical user interfaces.[17] For example, specific computing functionality can be portrayed as an object or an agent, depending on what is most natural. The desktop metaphor takes advantage of users' previous knowledge that office artifacts are visible, are passive, have locations, and may contain things. "Objects stay where they are: nice, safe predictable things that just sit there and hold things." Ontological knowledge of a different sort comes into play when the agent metaphor is employed. Our common sense knowledge of what agents can do tells us that, unlike typical desktop objects, they can notice things, carry

---

(Chernoff 1973). Another outstanding example that is event-based, future-oriented, and pattern-based is the OZ cockpit display (Still and Temme 2002)

[17] This idea is related to Gibson's (1979) original concept of "affordances" and Norman's elaborations regarding perceived affordances and their relationship to cultural constraints and conventions (Norman, 1988). See also Pawson's (2000) general notion of "expressive systems" (Pawson 2000) where the expected behaviour of a software entity is reflected in how it 'expresses' it's capabilities to the user (if it looks like a mapping tool, it should 'behave' like a map).

out actions, know and learn things, and go places.[18] "Agents become the repositories for adaptive functionality." Erickson concludes that research "which focuses on the portrayal of adaptive functionality, rather than on the functionality itself, is a crucial need if we wish to design agents that interact gracefully with their users."

**add discussion of multimodality issues here. Focus on aspects of the problem that are unique to agent technology.

Reeves and Nass (1996) have studied the social dimension of human-machine interaction for many years, and have concluded that humans pervasively and unconsciously interact with technology in a social way. This tendency is even greater when people interact with agents that may frequently be designed to exhibit properties and an appearance that are more deliberately human-like than typical applications. Because people are expert in social interaction, it has been argued that agents should be designed in a way that leverages these human strengths.

The extreme form of this view would be that the ideal agent should present itself in the most believable anthropomorphic form possible. Not only can a large amount of information be subtly and efficiently conveyed through gesture, speech, gaze direction, facial expressions, and body movement, but when well done most people seem find interaction with animated agents an enjoyable experience. In the words of Laurel:

> "First, this form of representation makes optimal use of our ability to make accurate inferences about how a character is likely to think, decide, and act on the basis of its external traits. This marvelous cognitive shorthand is what makes plays and movies work… Second, the agent as character (whether humanoid, canine, cartoonish, or cybernetic) invites conversational interaction… [without necessarily requiring] elaborate natural language processing… Third, the metaphor of *character* successfully draws our attention to just those qualities that form the essential nature of an agent: responsiveness, competence, accessibility, and the capacity to perform actions on our behalf." (Laurel 1997)

How then do we explain the extremely negative reaction that people have in the presence of poorly-done animated agents? Reeves concludes that a key to effective human-agent interaction is expectancy: when the agent performs better than the human would expect it is very good; when the agent performs

---

[18] It is also easy for people to assume less tangible qualities about agents like that they are internally consistent, are rational, act in good faith, can introspect, can cooperate to achieve common goals, and have a persistent mental state. Obviously, agent designers need to work hard to make sure that human expectation accords with the stark facts of reality in each of these dimensions. Violated trust inevitably breeds user hostility.

worse than expectation it is very bad.[19] If a sophisticated animation is not balanced with an equally competent and veridical performance, humans will all too soon smell a fraud—and they will not long tolerate dressing up fluff with a good face. In evaluating whether to use an anthropomorphic presentation, Lieberman summarizes the central concern as being "whether we are attracting a person's attention to the things they are actually trying to do or distracting them. If the character is entertaining or supportive, then anthropomorphism can be helpful. If on the other hand it takes a person's attention away from the things that they are focusing on, then it's not" (Lieberman and Selker to appear).

The most comprehensive reference describing the technical complexities of realistic animated agents, with emphasis on their primary applications in education, training, and entertainment, is Johnson (to appear). An excellent survey and discussion of issues in interface agents can be found in Lieberman and Selker (to appear) and among two book collections of programming by demonstration (Cypher 1993) and programming by example (Lieberman 2001) applications.

Adjustable Autonomy

Many autonomous systems are designed with fixed assumptions about what level of autonomy is appropriate to their tasks. They execute their instructions without considering that the optimal level of autonomy may vary by task and over time, or that unforeseen events may prompt a need for either the human or the system to take more control.

At the limit of this extreme are "strong, silent systems" with only two modes: fully automatic and fully manual. In practice this leads to situations of human "underload," with the human having very little to do when things are going along as planned, followed by situations of human "overload," when extreme demands may be placed on the human in the case of automation breakdown.

The goal of designing mixed-initiative systems with adjustable autonomy is to make sure that for any given context the agents are operating at an optimal boundary between the initiative of the human and that of the agent. People want to maintain that boundary at the sweet spot in the tradeoff curve that minimizes their need to attend to interaction with the agent while providing them with a sufficient level of reassurance that nothing will go wrong. In principle, the actual adjustment of autonomy level could be performed either by a human, an agent, or some third party.

As an illustration of adjustable autonomy, consider the mobile operations scenario. Here, identifying and clearing out an enemy sniper is clearly a difficult challenge. A soldier's personal assistant agent may help by calling in UAVs and asking for UAV support to clear out snipers. Complete agent autonomy may be problematic, given uncertainties with misidentification, the fluid nature of the combat environment, and the potential for very costly

---

[19] Hence, it is not all bad that completely realistic animated human faces are currently beyond the reach of technology. Studies have shown that cartoon faces can effectively serve both purposes of conveying an anthropomorphic style of interaction while not implying human-like intelligence or capabilities (**ref).

errors if there are potential civilian casualties. Instead, the agent may obtain user input in key circumstances, to ensure that superior human decision-making capabilities and context are fully exploited.

To the extent we can adjust agent autonomy with reasonable dynamism (ideally allowing fine-grained handoffs of control to occur "anytime") and with a useful range of levels, teamwork mechanisms can flexibly renegotiate roles and tasks among humans and agents as needed when new opportunities arise or when breakdowns occur. It is important to note that the need for adjustments may cascade in complex fashion: interaction may be spread across many potentially-distributed agents and humans who act in multiply-connected interaction loops. For this reason, in problems of realistic scale, adjustable autonomy may involve not merely a simple shift in roles among a human-agent pair, but rather the distribution of dynamic demands across many coordinated actors.[20]

An agent's level of autonomy can be varied along several dimensions such as: 1) type or complexity of tasks or functions it is permitted to execute, 2) which of its functions or tasks may be autonomously controlled, 3) circumstances under which the agent will override manual control, 4) duration of autonomous operation, 5) the circumstances under which a human may be interrupted (or must be interrupted) in order to provide guidance (Dorais *et al.*1999). As long as the agent operates within the constraints specified as policy, it is otherwise free to act with complete autonomy. Policy-based constraints on behavior ideally can be imposed and removed at any time (Bradshaw *et al.* 2001). This coupling of autonomy with policy gives the agent maximum opportunity for local adaptation to unforeseen problems and opportunities while assuring humans that behavior is kept within desired bounds.

A key question in adjustable autonomy is how to effect graceful reductions in autonomy as agents reach limits of their competence in a given context. How to effect advance recognition and preparation for such handoffs is a major research challenge.[21] Several different approaches have been tried:

---

[20] As Hancock and Scallen (1998) rightfully observe, the problem of adaptive function allocation is not merely one of efficiency or technical elegance. Economic factors (e.g., can the task be more inexpensively performed by humans, agents, or some combination?), political and cultural factors (e.g., is it acceptable for agents to perform tasks traditionally assigned to humans?), or personal and moral considerations (e.g., is a given task enjoyable and challenging vs. boring and mind-numbing for the human?) are also essential considerations.

[21] Malin (1997) notes significant differences in flexibility between humans and today's agents: "If the team members are human, there is a broad range of possible flexibility in tasks, roles, and level of detain in communication, and support for tight and fast-paced task sharing. The computer team member of today is likely to be slower in give-and-take communication, less flexible and less likely to initiate a change in roles."

- *Uncertainty based*: This approach to adjustable autonomy transfers decision making control from an agent to a human when the agent determines that its decision making uncertainty is high (Gunderson 1999). Some systems exploit machine learning in this reasoning to gradually reduce uncertainty. In particular, an agent may observe human actions, and use inductive learning algorithms to determine decision-making rules. When the rules have sufficient confidence associated with them (e.g., they may predict user decisions with sufficient accuracy), the agent starts using these rules autonomously.

- *Safety based:* This approach transfers decision-making control from an agent to a human if the agent's decision can be harmful to the human or the mission (Dorais *et al*. 1998).

- *Capability based:* This approach transfers decision-making control by reasoning about capabilities of the agent and the human (Ferguson, Allen and Miller 95). In particular, a human may be more capable of performing particular tasks.

- *Decision theory based*: This approach essentially combines the uncertainty, safety, and capability based approaches. In particular, it carefully and explicitly evaluates the costs and benefits of asking a human, using decision-theoretic reasoning (Horvitz 1999a, 1999b). It compares the expected utility of asking a user with the expected utility of acting autonomously (where acting autonomously may potentially lead to errors given an uncertain environment).

All of these approaches focus on a "one-shot" decision, whereby the agent either makes the decision autonomously or transfers decision-making control to a human for his or her input. Once the control is transferred, the agent will wait as long as it takes to obtain human input. In some cases, decisions may be better made by humans when only they have the pertinent information, the necessary authority, or the key capabilities associated with the action, or when there is a significant moral or risk component to the action.

One key example of mixed-initiative and adjustable autonomy is the Lookout system from Microsoft (Horvitz 1999a, b). In this research application, an agent helps a person with their calendaring activities. The agent (a personal assistant) scans the person's email for meeting related activities, and decides whether a meeting should be scheduled. For instance, Figure 3 below shows an output from this system. The key issue that has been pulled out from this application concerns initiative on the part of the agent. In particular, at each step it must decide whether to do nothing, to perform a particular action for the person, or to dialogue with the person to understand better whether or not to perform the action.

The agent's adjustment of its own autonomy and hence its change initiative <what is "change initiative?> is based on decision-theoretic reasoning. In particular, here the agent automatically learns to model the utility of its three options as a function of the confidence it has that the person desires the particular action, and as a result to choose the option at any point in time that will maximize its expected utility (as a function of the confidence at that point in time). While Lookout is a research system from Microsoft, the

take away message is that decision analysis techniques are now maturing to the point that they can be applied and evaluated for questions of initiative in agent-human operations.

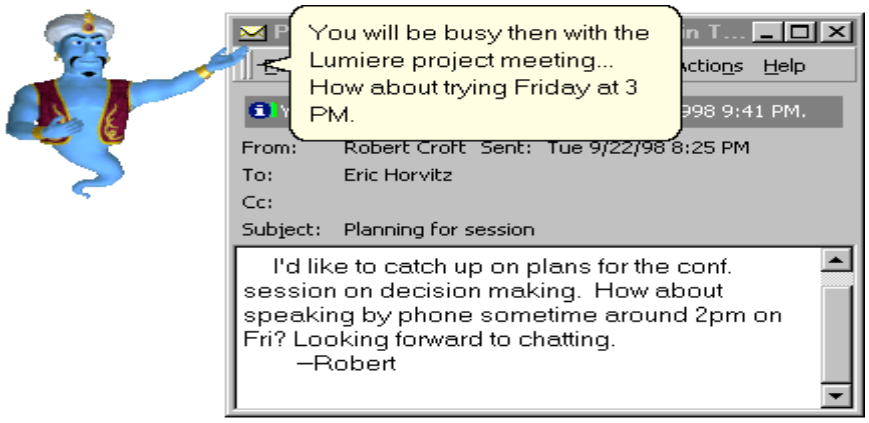


Figure: Output from Microsoft's Lookout system (Horvitz 99)

Research in adaptive function allocation—the dynamic assignment of tasks among humans and machines—also provides some useful lessons. Examples of the types of questions and adaptive responses that might be addressed in adaptive allocation are shown in Table 4 below.

Question	**Adaptive Response**
Who	IF: A human performs within predetermined criteria THEN: The human shall keep task control, otherwise the task is allocated to the agent.
What	IF: Only parts of tasks are being performed poorly THEN: Only these parts shall become available for dynamic allocation.
When	IF: Certain time periods are associated with increased demand, error, or loss of situation awareness THEN: These periods will be appropriate for dynamic allocation.
Where	IF: Particular environments or combinations of environmental variables are associated with increased task demand or error THEN: Encountering these environments triggers dynamic allocation.

Why	IF: Extended periods of allocation have detrimental effects (objective or subjective) THEN: Allocation shall both remove and return control.
How	IF: Human performance, environmental attributes, and psycho-physiological indexes are paramount for human-agent interaction THEN: All of these are inputs for allocation shift.

Table 4. Examples of the types of questions and adaptive responses to be addressed in adaptive allocation (adapted from Hancock and Scallen 1998, p. 526)

As mentioned above, the cost of interrupting humans can be very high, and thus should be carefully controlled. For example, in her pioneering research on interface agents, Maes (1997) allowed users to manually set two thresholds for the agent: one that provided a lower bound determining what level of confidence the agent would require before informing the human (the "tell-me" threshold) and another that provided a lower bound relative to whether it would perform an action autonomously (the "do-it" threshold). Models of human attention are currently being developed that can be used to estimate the costs and benefits of interrupting the human's current activities (e.g., Horvitz **).

Scalability

Most research on adjustable autonomy to date has focused on the interaction between a single agent-human pair. The requirements of teamwork and coordination, present in domains where human and agents act in teams, give rise to novel challenges not addressed by previous research. One key challenge is that in dynamic team environments agents must be judicious in asking for human intervention. In particular, although human input can prevent erroneous actions or lead to better decisions, an agent's inaction while waiting for a human response can lead to high team costs due to miscoordination and delays. Of course, if the agent were to act autonomously without human input, it could face significant errors in its action. For instance, in the mobile operations scenario, three personal assistants PA1, PA2 and PA3 of three different soldiers may work together to come up with a new route for advance. If PA1 transfers control to its human user to seek input, and the user is unable to provide input given the circumstance, then PA1 will significantly delay PA2 and PA3 (who may have already planned their parts of the route). If PA1 were to act autonomously without human input, it could cause significant errors. Hence, an agent must carefully weigh the cost of time lost due to delayed human input against the benefits of receiving that input.

Scerri *et al*. (2001) identify the "one shot" transfer of control techniques as culprits in this case. They suggest instead that agents engage in flexible, multiple transfers of control, even in service of a single decision. Thus, in the above example, if PA1 were not to obtain input from its human user, it could take the control back from the user, then take the action autonomously. Or, PA1 could take the control back, suggest to PA2 and PA3 that they delay the route generation, and then give control back to PA1's user, to give him/her more time to respond. Such flexible transfers of control can be planned via a

Markov-decision-process (MDP) based implementation. However, a significant amount of additional research is necessary as we scale up adjustable autonomy and mixed initiative systems to larger-scale multiple human and agent interaction. Furthermore methods for partial transfer of control need to be investigated, e.g. giving agents the ability to autonomously perform temporary stop-gap actions until human guidance becomes available.

Safety and Privacy

Automation safety issues have been extensively studied over decades in domains such as the patient safety movement and flightdeck automation. The basic principles should apply equally well to human-agent interaction.

Perrow's (1984) classic study convincingly argued that systems that are both highly-complex and highly-coupled (e.g., have highly interdependent components and rapid propagation of effects among components) are inevitably predisposed to disastrous cascading failures. Safety risks need to be understood holistically rather than as the failure of a single component.

Complex interaction: unknown sequences which are not immediately apparent, branching and feedback loops, failures that jump across subsystem boundaries

Tight coupling: time-dependent non-delayable processes, rigidly-ordered processes, single patch to successful result, little margin for error in time or resources.

Build in buffers, continous communication flow

Message passing, asynchrony, loose-coupling in complex cooperating systems.

Redundant pathways, backup systems, work arounds, loose coupling

Complexity is the enemy of comprehensibility; reduce when possible; learn from what people do; invent new methods where necessary. Analyze close calls.

Norman

Bounds of autonomy set by policy

Anticipate and plan for unexpected events and surprises

Fundamental asymmetry, but autonomy can't always be as fine-grained as we'd like; in crises, human may forget

Systems approach to safety—complex multi-dimensional factors rather than a single source of malfunction.

Highly-coupled systems vs. loosely coupled: Perrow-normal accidents

Asymmetry in human-agent interaction also gives rise to safety and trust issues. Within the human-agent team, when an agent is performing its own role autonomously, it could potentially take actions that harm human team members. For example, in the mobile operations scenario, the agent may ask for UAV support to neutralize a sniper, without realizing that such an action has a slight chance of accidentally harming the humans given their proximity. Even in our daily life, agents interacting with us could unintentionally cause harm, albeit not physically. For instance, agents' autonomous actions could cause harm financially (by undertaking a risky financial transaction), socially (by taking an inappropriate social action on our behalf), and so forth. Given even the potential for such harm, it becomes critical to address issues such as safety and trust in human-agent interaction. While such issues could potentially arise in interaction among agents, avoiding harm to other agents (e.g., robots) does not (at least as yet) rise to the same level of concern.

Safety

One key concern for researchers in human-agent interaction is to ensure that agents will not take actions that cause harm to the humans they interact with. In combat environments, the key problem may be any potential for physical harm caused directly or indirectly to the human users. For instance, in the joint/coalition operations scenario, we may need to ensure the safety of human users as well as any potential for such harm to coalition members. In general applications, such harm could be physical, financial (e.g., if the agent makes financial transactions), social (e.g., if the agent schedules the user's social interaction/meetings), etc.

Adjustable autonomy presents one approach to address this safety concern —agents may transfer decision-making control to users in potentially harmful situations. However, because of errors and mismatches in agents' world models with respect to a human user's environment, the agents' adaptiveness (whereby they may learn new knowledge via interaction), means that when agents do act autonomously, they may potentially still violate a human user's safety. This section focuses on some of the research conducted to ensure safety in autonomous agent operations based on notions of exploiting Asimov's laws as a source.

Asimov's laws and safety in agent planning

Interestingly, the early stages of this research on safety has been inspired by Asimov's laws of robotics (Asimov 42), i.e., from science fiction. For instance, Asimov's first law of robotics states that *"A robot may not injure a human being or through inaction allow a human being to come to harm"*. While stated in the context of robots, this law has inspired research in the agents arena, to ensure that agents avoid harmful actions when planning or when learning. Within planning, research in classical planning has focused on defining primitives that in a credible and computationally tractable manner make agents obey Asimov's first law. To that end, Weld and Etzioni (Weld and Etzioni, 94) introduced primitives "don't-disturb" and "restore" for safe planning using generative planning algorithms. "Don't-disturb" implies that some conditions are so harmful that an agent should never cause them through planning. For instance, in the coalition operations scenario, an agent must ensure that a coalition force is not in the line of fire. "Restore" is a weaker primitive: it implies that an agent may temporarily disturb a condition, as long as the condition is restored after the agent's operation.

Pynadath and Tambe (Pynadath and Tambe 2001), again explicitly citing Asimov's first law, provide such safety primitives in the context of planning in uncertain domains. In particular, they focus on an agent's generation of an optimal policy using Markov –decision processes (MDPs). If humans or other agents identify specific states or actions as *forbidden*, then the MDP policy ensures that those states or actions are never visited. Alternatively, some states or actions may be identified as necessary and must be visited. Pynadath and Tambe define an algorithm that ensures that such constraints are obeyed in the MDP policy generated (extending the value iteration algorithm for MDPs) and prove its correctness. A fortuitous outcome of defining such primitives is that planning is speeded up, as the agents can prune away unsafe paths from consideration.

Beyond planning, some researchers have focused on safety in plan execution in complex uncertain domains. One important system that uses UAVs as example domains, is the Cooperative Intelligent Real-time Control Architecture or CIRCA (Atkins et al. 97, Musliner et al. 95). In CIRCA, off-line compiled plans attempt to ensure safety in execution (e.g., ensure that the UAV does not crash). However, if, during execution, the agent reaches an unplanned-for state heralding failure, then techniques for quick on-line response ensure such failure avoidance in execution. These techniques range from invoking plans stored in a cache (if the unplanned-for state heralds immediate failure) to invoking significant amounts of replanning (if there is significant time available before failure).

Although these techniques can be used to avoid certain unsafe situations, there remain situations, particularly in military operations, where no action guarantees safety and freedom from risk. Adjustable autonomy can help by deferring such decisions to humans, but as explained above that can also incur costs and therefore risks. It is important when designing an agent application to examine the situations in which agents will be used, determine when safety cannot be guaranteed to a sufficient degree, and take steps of avoid deployment of agents in those situations. The techniques described above can help in this regard. For example, risky situations can be marked as forbidden, so that the agents can plan to avoid them. If it turns out to be impossible to avoid forbidden states, it is important to find this out as early as possible, certainly during mission execution but preferably earlier, either during the mission planning stages or the agent design stages.

Asimov's laws in Agent Learning

Research on agent safety in learning can be divided into two categories. The "pre-learning" category ensures that prior to learning, agent representations, learning methods, and so forth are appropriately constrained such that after learning, agents' behaviors remain "safe", e.g., they obey Asimov's laws. For instance, such an approach may ensure that some concepts are made un-learnable. The advantage of this approach is that it ensures minimal run-time effort, but sometimes it may require excessive sacrifice in what is learnable.

The post-learning category focuses on testing learned knowledge after learning. If the learned knowledge is harmful, then this knowledge is modified to ensure that it is safe. This approach could potentially require significant run-time effort and must suffer from run-time failures, but it does not sacrifice expressiveness or learning to accomplish safety. Gordon (Gordon, 2000) illustrates both the post-learning and pre-learning approaches in the context of agents based on finite-state machines. She illustrates that some types of learning operators that cause modifications to finite state machines are safe in that if some properties are true before learning, then those properties are preserved after learning. In contrast, other learning operators are not "safe", and thus require post-learning tests.

Prior work on the "utility problem" in machine learning, particularly, explanation-based learning (Minton, 90), could also be considered to have focused on the safety of agent learning. The utility problem suggested that sometimes learned knowledge can be harmful to agents, in that it may cause dramatic degradation in an agent's performance (e.g., a dramatic slowdown in speed of problem solving). The slowdown in this context came from the cost of indexing the learned knowledge i.e., the learned rules were very expensive to match. Investigations to attack the utility problem also have focused on both pre-learning approaches (Tambe et al. 90) and post-learning approaches (Minton 90).

Scalability in Safety

As with other research in human-software agent interaction, research in agent safety has traditionally focused on individual human-agent interaction. It is feasible that safety constraints imposed on two or more agents within a team may conflict, e.g., in a joint or coalition operation. Such situations require further research in agent safety to guarantee the safety of team operations, rather than individual operations. Fortunately, there appears to be increasing interest in safe learning agents (AAAI Spring Symposium 2002).

Privacy

Very short privacy biblio – A. Acquisti, April 2002

Technology: anonymous transactions, browsing, voting, sharing, etc.

Asokan, P., Janson, P., Steiner, M., and M. Waidner, (1999). ``State of the Art in Electronic Payment Systems". Appeared previously as: ``The State of the Art in Electronic Payment Systems". \QTR{em}{IEEE Computer}, pp. 28-35.

Chaum, D., (1983). ``Blind signatures for untraceable payments". In \QTR{em}{Advances in Cryptology. Proc. Crypto'82,} pp. 199--203. Plenum Press: New York.

Reiter, M. K., and A. D. Rubin, (1999). ``Anonymous Web Transactions with Crowds". \QTR{em}{Communications of the ACM}, 42(2), pp. 32--38.

Goldschlag, D., Reed, M. and P. Syverson, (1999). ``Onion routing for anonymous and private internet connections". \QTR{em}{Communications of the ACM,} 42(2), pp. 39--41.

Agre, P. E. and M. Rotenberg (eds) (1997). Technology and Privacy: The New Landscape. MIT Press: Cambridge MA.

Huberman, B. and E. Adar, (2000). "A Market for secretes". http://www.firstmonday.org/issues/issue6_8/adar/index.html#a1

Economics of privacy

Acquisti, A. and H. Varian, (2001). "Conditioning Prices on Purchase History", mimeo, UC Berkeley.

Calzolari, G. and A. Pavan, (2001). "Optimal Design of Privacy Policies", mimeo, MIT and University of Tulouse.

Spiekermann, S., Grossklags, J. and B. Berendt, (2001). "E-privacy in 2nd generation E-Commerce: privacy preferences versus actual behavior", Proceedings of EC'01: Third ACM Conference on Electronic Commerce, Association for Computing Machinery, Tampa, Florida, US, pp. 38-47.

Taylor, C. R., (2002). "Private Demands and Demands For Privacy: Dynamic Pricing and the Market for Customer Information", mimeo, Duke University.

Varian, H. R., (1996). "Economic Aspects of Personal Privacy", December 1996, UC Berkeley mimeo.

Legal issues

Samuelson, P. (2000). "Privacy as Intellectual Property?", UC Berkeley mimeo.

Regan, P. M. (1995). Legislating Privacy. The University of North Carolina: Chapel Hill.

Business overviews

Privacy, Consumers, and Costs: How The Lack of Privacy Costs Consumers and
Why Business Studies of Privacy Costs are Biased and Incomplete, by Robert Gellman, http://www.cdt.org/publications/dmfprivacy.pdf

Swire, P. P and R. E. Litan (1998). None of Your Business. Washington DC: Brookings Institution Press.

Federal Trade Commision, (2000). "Privacy Online: Fair Information Practices In The Electronic Marketplace", May 2000.

Laudon, (1996). Markets and Privacy", Communications of the ACM, 39 (9), 92-104.

Philosophical issues

Scoglio, S. (1998). Transforming Privacy: A Transpersonal Philosophy of Rights. Westport: Praeger.

Scott, G. G. (1995). Mind your Own Business. Insight Books: New York.

Trust

Norman, p. 51, providing reassurance

Because agents differ so significantly from the applications with which most people are familiar, we need to take into account the social issues no less than the technical ones:

"The technical aspect is to devise a computational structure that guarantees that from the technical standpoint, all is under control. This is not an easy task.

The social part of acceptability is to provide reassurance that all is working according to plan… This is [also] a non-trivial task" (Norman, p. 51)

Lieberman, p. 9

You can't cooperate with another agent if you assume they are incompetent (Woods)

An agent may accept risk from the actions of others. Furthermore, the agent may expect that the trusted agent will not take advantage of the adopted risk. This is the general conception of trust in multiagency. Steve Marsh produced one the earliest works on trust in multiagency [Marsh 1992, 1994]. Marsh clearly defines trust as a function of (a) a basic trust attitude toward another agent, and (b) the value of the object of trust. In his formulation, an agent A's trust of another agent B must be higher than a cooperation threshold before A begins cooperating with B. More recent

theories of trust in multiagency are found in [Castelfranchi, Falcone 2000, Birk 2000].

Since many of our scenarios involve dynamic teaming of agents (e.g., in joint or coalition operations, human and agents from different organizations may team up), trust between human and agents is critical. In particular, it is important to trust potential team members. Practical techniques for trust (as well as security) go beyond theory and suggest methods. For instance mobile agents research provides practical methods.

Trust models

We summarize the models presented in [Swarup and Fábrega, 1999]. Agents (called keys) are represented as nodes of a graph. Statements are also nodes of such a graph. One type of directed edge is between two agents (A → B) and it represents that A trusts B. Another type of directed edge is between an agent and a statement (A → P), and it represents that agent A states P. In a network of agents and statements, specific policies can be defined. For example a policy that insists on two independent corroborating trust paths per statement can be used to identify all agents that accept a given statement. The number of false positive and negative statements in the trust graph can be used in ratios of these numbers over the number of statements to compare policies.

Trust is also related to the concepts of commitment and joint intention. When an agent A makes a commitment to another agent B to help achieve a goal, B needs to trust that A will live up to its commitment. However changing circumstances and new opportunities might lead A to abandon its commitments. Current work by Das et al. (2002) is investigating intention reconciliation, the conditions under which agents might be motivated to abandon their group commitments. This could be coupled with a general model of safety in planning to avoid such situations, thus increasing the reliability of multi-agent planning.

Scalability and Trust

While these techniques are promising, they focus on interaction within small groups. Thus, unfortunately, there is no evidence that current trust models will scale. It is hard for humans to keep track of frequencies of encounter and how it enhances or deteriorates trust. Psychological experiments are needed to build methods that prove to be scalable.

Adaptivity

Negroponte: agent knows you and knows areas of expertise

Negroponte illustrates this by a compelling example: “At a recent dinner party I winked at my wife, and she knew all the paragraphs of information it would have taken me (otherwise) to explain the same to some stranger. The reason is quite obvious. A vast amount of shared experiences and robust models of each other make the epitome of communication be the lack of it” (p. 65) (Negroponte, N., Agents: From direct manipulation to delegation, In J.M. Bradshaw (Ed.), Software Agents, AAAI/MIT Press, 1997, pp. 57-66).

Without constant repair of the difference between agent representation of the world and human, lack of context, failure is inevitable.

Shorthand modes of communication among teammates that have the benefit of long mutual acquaintance.

Delegation cultural and individual differences: Milewski

Lieberman: personalization and user modeling, p. 7; p. 11-cognitive style; instructability

Flexibility, multiple perspectives (Feltovich, P. J., Spiro, R. J. & Coulson, R. L., 1997, Issues of expert flexibility in contexts characterized by complexity and change. In P.J. Feltovich, K. M. Ford, and R.R. Hoffman (Eds.), Expertise in Context, Cambridge, MA: AAAI/MIT Press, pp. 125-146.)

Machine Learning Approaches

Beginning with the initial research effort in the CAP system (Mitchell, 92) and later efforts such as (Maes 94) many researchers have focused on building assistants to human users by exploiting machine learning techniques. By observing and learning from human actions, agents can learn to filter users' e-mail, schedule appointments, filter newsgroups, and so on. Traditionally, this research has focused on assistants that often suggest actions rather than act autonomously. Furthermore, the focus is often on the agent capabilities rather than on the interaction with human users. Further work is thus required in this area.

Risks

There are two major risks associated with human-agent interaction.

Risk 1 Scalability: First, at least when compared to interaction among agents, much of the human-agent interaction research has focused on smaller scale human-agent groupings, sometimes involving only one agent and one human. With respect to scalability issues, larger numbers of agents need to interact with a large number of humans, e.g., if a large team of agents is to be embedded within a large-scale human organization, novel organizational issues will need to be addressed. For instance, large-scale human organizations may be hierarchical or may have embedded authority structures, and agents would need to conform to these human organizational structures and norms.

Aggravation or mitigation of Risk 1: Scalability will be increasingly critical for any realistic application, and thus, this risk is quite real. However, researchers are actively focusing on scaling up human-agent interaction, and thus there is a high likelihood that on-going research will continue to mitigate this risk.

Risk 2 non-acceptance of agents: A major risk relates to human users not accepting agents in their midst, as assistants, as peers, as entities acting autonomously with them or for them. The concerns could be due to the potential for such autonomous entities to knowingly or unknowingly cause harm to the users interests and goals, to violate the user's trust or to violate the user's privacy or security, e.g., by communicating private information to other agents. The concerns could also be that at least initially agents may require more work in maintenance and upkeep and may not provide significant benefits as they may not be fully integrated in the work practices.

However, the issues may not be purely technical and could relate to emotional or inter-personal issues (the agent may unknowingly act in a "rude" manner).

Aggravation or mitigation of Risk 2: Initial research in agent safety and trust covered in this chapter would need to make significant progress to address these concerns. The other two concerns related to privacy and security are issues that are covered in another chapter focusing on such security issues. However, many more careful studies in integrating human-agents are essential to mitigate this risk.

Forecast

Forecast Tables

We make the following key assumptions in all of our forecasts:

(i) Reasonable level of funding in agents research and in particular, human-agents research, e.g., as seen in the control of agent-based systems program from DARPA.

(ii) Investigation and resolution of issues related to actual deployment of agents in human organizations, where such issues may be social rather than being purely technical.

A word of caution about the forecast: While some areas such as human-agent interaction have had significant history and momentum, other areas of research such as safety have had very little research conducted to date. Thus, there is a higher risk associated with some of the forecast than others; and these high-risk forecasts are marked with a "*".

Human-Agent Interaction, continued

Technology element	Near term 2001-2003	Midterm 2004-2006	Long term 2007-2010
Core Agent Technologies			
Human-agent team interaction] Adjustable autonomy & Mixed initiative	• Implementations of human-agent peer-to-peer teams rather than master-servant type systems become increasingly standard in research systems • Reusable tools to enable human-agent teamwork begin to be available, with initial appearance of commercial tools • Research systems consisting of teams of a single human and multiple agents • Consensus on techniques to manage single-agent and single-human adjustable autonomy	• Individual human-agent teams commonly deployed with individual agents assisting individual users • Research prototypes involving mid-scale human-agent teams, involving 10 humans and 100 agents • Commercial reusable tools for human-agent teams more commonly available • Initial development of tools to enable scaability in human-agent teamwork • Consensus on techniques to manage multi human-agents adjustable autonomy (approx 1 user and 5-10 agents)	• Large-scale human-agent virtual organizations with about hundred humans and 1000 agents in prototype stage • Mid-size human-agent teams, deployed in small-scale human organizational units

Technology element	Near term 2001-2003	Midterm 2004-2006	Long term 2007-2010
Core Agent Technologies			
Adjustable autonomy & Mixed initiative	• Commercial tools exploiting mixed initiative/adjustable autonomy available for human-agent pairs • Initial research in adjustable autonomy for larger scale human-agent interaction (more than one agent and one human)	• Scalability towards the end of this period in the commercial tools, enabling larger scale human-agent interaction (single user and handful of agents)	• Scalability towards adjustable autonomy in virtual organizations and teams of teams (with 10s of agents and 10s of humans)
Safety* (predictions of "trust" related issues are very similar in nature)*	• Initial research in safety to formalize concepts • Initial computational techniques to manage safety, and its exploitation in research grade systems	• Safety addressed in regular implementations in single-agent single-human interaction • Investigations in scaability in safety, to manage safety of groups of human-agents • Experiments investigate impact of deploying agents in small scale human organizations, issues in such deployment addressed	• Safety addressed in mid-size implementations • Continued investigations of impact of deploying agents in large-scale organizations

Figure 1. Human-agent interaction Forecast Table (Part 1 of 1)

Summary and Recommendations

Human-agent interaction promises to significantly enhance individual use capabilities. Additionally, as seen in the advanced command post scenario, the joint/coalition operations scenario and others, this technology may provide tremendous benefits to large and small military units by enabling rapid coordinated response to crises and adaptive operations in changing

circumstances. Many of these benefits may accrue via formation of large-scale virtual organizations.

We make the following recommendations for further investment in research:

- **Scalability of human-agent interaction:** Research in single-agent and single-human interaction has reached significant level of maturity, as seen in research products from industrial laboratories. However, there is a significant technology gap in scalability toward large-scale human-agent organizations, which includes interaction between several human and agents. Investment in this area to accomplish such scale-up is essential. The issues in scale-up are not purely efficiency related, but more fundamental issues, such as coordination among multiple human-agents. These issues also differ from pure issues of agent coordination, given the asymmetry that humans bring to bear in a mixed human-agent organization. Specific incremental targets could be laid out for issues of teamwork, adjustable autonomy, and other aspects of human-agent collaboration in order to ensure incremental progress toward the goal of scalability:
    - *Near-term*: Scaling to robust operations of a single human and five agents. Agents in this case may be mostly software agents.
    - *Mid-term*: Scaling to robust operations of 10 human users and 100 agents in a team of teams. Agents in this case may include sensors, UAVs, and others that operate in the real-world.
    - *Long-term:* Scaling to robust operations of 100 human users and 1000 agents and operate in real-world environments, with all types of agents.
- **Deploying agents in human societies:** What is the impact of deploying large numbers of autonomous agents in human organizations or societies? It is unclear if such deployment will be welcomed by one and all. In this context, we recommend two types of investments:
    - *Investigation of safety, trust, and privacy issues in human-agent interaction:* These topics have not received much attention in the literature, although they could be at the core of agents' gaining acceptance from their human users in their day-to-day operations. In this context, investments are needed both in basic research in this arena and in practical technology.
    - *Social impact of deploying agents in human organizations:* We need a careful study of how deployment of agents in human organizations actually changes the organizational work practices, and of the best way to deploy such agents so as to improve rather than degrade organizational productivity.

As a simple example consider agents that detect computer activity on a user's terminal and then announce his/her presence in the office to other members of the organization. Such a system may have a negative impact on productivity if humans wish to protect their privacy. While this is a simple example, a careful investigation of such effects and potential remedies for negative effects should be performed.

References

Acquisti, A., Sierhuis, M., Clancey, W.J. and Bradshaw, J.M. 2002. Agent-based modeling of collaboration and work practices onboard the International Space Station. *Proceedings of the Eleventh Conference on Computer-Generated Forces and Behavior Representation*. Orlando, FL, to appear.

Agnew, N. M. and Brown, J. L. 1989. Foundations for a theory of knowing: II. Fallible but functional knowledge. *Canadian Psychology,* 30, 167-183.

Allsopp, D.N., Beautement, P., Bradshaw, J.M., Durfee, E.H., Kirton, M., Knoblock, C.A., Suri, N., Tate, A. and Thompson, C.W. 2002. Coalition agents experiment: Multi-Agent cooperation in an international coalition setting. *IEEE Intelligent Systems,* to appear.

Asimov, I. 1942. Runaround. *Astounding Science Fiction,* pp 94-103.

Atkins, E.M., Durfee, E.H. and Shin, K.G. 1997. Detecting and reacting to unplanned-for world states. In *Proceedings of AAAI'97.*

Barber, K.S. and Martin, C.E. 1999. Agent autonomy: Specification, measurement, and dynamic adjustment (PDF). "Autonomy control software" workshop at *International Conference on Autonomous Agents.*

Billings, C.E. 1997. Issues concerning human-centered intelligent systems: What's 'human-centered' and what's the problem? Plenary talk at the National Science Foundation *Workshop on Human-Centered Systems: Information, Interactivity, and Intelligence.* Arlington, VA, Feb.

Birk, A. 2000. Boosting cooperation by evolving trust. *Applied Artificial Intelligence Journal.*

Boy, G.A. (1998). Cognitive Function Analysis. Stamford, CT: Ablex Publishing.

Boy, G.A. (to appear). Human-centered design of artificial agents: A cognitive function analysis approach. In J.M. Bradshaw (Ed.), *Handbook of Software Agents*. MIT Press.

Bradshaw, J.M., Ford, K.M., Adams-Webber, J.R. and Boose, J.H. 1993. Beyond the repertory grid: New approaches to constructivist knowledge acquisition tool development. In K.M. Ford and J. M. Bradshaw (Eds.), *Knowledge Acquisition as Modeling,* pp. 287-333. NY: John Wiley.

Bradshaw, J.M., Suri, N., Cañas, A.J., Davis, R., Ford, K.M., Hoffman, R., Jeffers, R. and Reichherzer T. 2001. Terraforming cyberspace. *IEEE Computer,* 34, 7, 48-56. **add book reference.

Bradshaw, J.M. *et al.* 2002a. Brahms and KAoS agent framework support for human-robotic teamwork in practice. In H. Hexmoor, R. Falcone, and C. Castelfranchi (Eds.), *Agent Autonomy. Kluwer*, to appear.

Bradshaw, J.M. *et al.* 2002b. Toward a deliberative and reactive agent architecture for augmented cognition. *DARPA Augmented Cognition Program White Paper,* to appear.

Bradshaw, J.M. *et al.* 2002c. AAMAS Teamwork workshop paper.

Burstein, M.H and McDermott, D.V. 1996. Issues in the development of human-computer mixed-initiative planning. In B. Gorayaska and J.L.Mey (Eds.), *Cognitive Technology: In Search of a Humane Interface*. Elsevier Science.

Card, S.K., Moran, T.P. and Newell, A. 1983. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum.

Castelfranchi, C. and Falcone, R. 2000. Trust and control: A dialectic link. *Applied Artificial Intelligence Journal,* 14 (8), Special Issue on Trust in Agents, Part 1, C. Castelfranchi, R. Falcone, B. Firozabadi and Y. Tan (Eds.).

Chernoff, H. 1973. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association,* 68 (June), 361-368.

Christoffersen K. and Woods, D.D. 2002. How to make automated systems team players. To appear in E. Salas (Ed.), *Advances in Human Performance and Cognitive Engineering Research,* Vol. 2. JAI Press: Elsevier.

Clancey, W.J. 2002. Robot collaboration requires consciousness. Unpublished technical memo. Moffett Field, CA: NASA Ames Research Center, March 8.

Clancey, W.J. In press. Simulating activities: Relating motives, deliberation, and attentive coordination. *Cognitive Systems Review,* special issue on Situated and Embodied Cognition.

Clancey, W.J. 1997. *Situated Cognition: On Human Knowledge and Computer Representations*. Cambridge University Press, Cambridge.

Clancey, W.J. 1993. The knowledge level reinterpreted: Modeling socio-technical systems. In K.M. Ford and J. M. Bradshaw (Eds.), *Knowledge Acquisition as Modeling,* pp. 33-50. New York: Wiley.

Clancey, W.J., Sachs, P., Sierhuis, M. and van Hoof, R. 1998. Brahms: Simulating practice for work systems design. *Intl. Journal on Human-Computer Studies,* 49, 831-865.

Clark, H.H. 1992. *Arenas of Language Use*. Chicago, IL: University of Chicago Press.

Cohen, P. R., & Levesque, H. J. (1991). *Teamwork*. Technote 504. Menlo Park, CA: SRI International, March.

Cypher, A. (Ed.). 1993. *Watch What I Do: Programming By Demonstration*. Cambridge, MA: The MIT Press.

Das, S., Grosz, B. and Pfeiffer, A. 2002. Learning and decision-making for intention reconciliation. In *Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems*. Bologna, Italy. New York: ACM Press.

Dorais, G., Bonasso, R.P., Kortenkamp, D., Pell, B. and Schrekenghost, D. 1999. Adjustable autonomy for human-centered autonomous systems on Mars. In *Proceedings of the AAAI Spring Symposium on Agents with Adjustable Autonomy*. AAAI Technical Report SS-99-06. Menlo Park, CA: AAAI Press.

Erickson, T. 1997. Designing agents as if people mattered. In J.M. Bradshaw (Ed.), *Software Agent,* pp. 79-96. Cambridge, MA: AAAI Press/The MIT Press.

Ferguson, G., Allen, J. and Miller, B. 1996. TRAINS-95: Towards a Mixed-Initiative Planning Assistant. *Proc. Third Conference on Artificial Intelligence Planning Systems* (AIPS-96), pp. 70-77. Edinburgh, Scotland.

Fitts, P.M. (Ed.). 1951. *Human Engineering for an Effective Air Navigation and Traffic Control System*. Washington, DC: National Research Council.

Ford, K., Bradshaw, J.M., Adams-Webber, J.R. & Agnew, N.M. (1993). Knowledge acquisition as a constructive modeling activity. In K. Ford & J.M. Bradshaw (Eds.), *Knowledge Acquisition as Modeling* . New York: John Wiley.

Ford, K. M., Glymour, C., & Hayes, P. (1997). Cognitive prostheses. *AI Magazine*, 18(3), 104.

Ford, K. M., & Hayes, P. (1998). On computational wings: Rethinking the goals of Artificial Intelligence. *Scientific American. Special issue on "Exploring Intelligence"*, 9(4), 78-83.

Gibson, J. J. (1979). The Ecological Approach to Visual Perception. **

Glasgow, J., Narayanan, N.H. and Chandrasekaran B. (Eds.). 1995. *Diagrammatic Reasoning: Computational and Cognitive Perspectives*. Menlo Park, CA: AAAI/The MIT Press.

Goodrich, M.A., Olsen, Jr., D.R., Crandall, J.W. and Palmer, T.J. 2001. Experiments in adjustable autonomy. In *Proceedings of the IJCAI-01 Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents*. Seattle, WA, August. Menlo Park, CA: AAAI Press.

Gordon, D. 2000. Asimovian adaptive agents. *Journal of Artificial Intelligence Research,* 13.

Grosz, B.J. 1996. Collaborative systems. *AI Magazine*, 17, 2, 67-85.

Grosz, B.J. and Sidner, C.L. 1990. Plans for discourse. In P. Cohen, J. Morgan and M. Pollack, (Eds.), *Intentions in Communication,* pp. 417-444. Cambridge MA: Bradford.

Grosz, B.J. and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence,* 86(2), 269-357.

Gunderson, J. and Martin, W. 1999. Effects of uncertainty on variable autonomy in maintenence robots (postscript). "Autonomy control software" workshop at *International Conference on Autonomous Agents.*

Hamilton, S. 2001. Thinking outside the box at IHMC. *IEEE Computer,* 34, 1, 61-71.

Hancock, P.A. and Scallen, S.F. 1998. Allocating functions in human-machines systems. In R. Hoffman, M.F. Sherrick and J.S. Warm (Eds.),_*Viewing Psychology as a Whole,* pp. 509-540. Washington DC: American Psychological Association.

Hoffman, R.R. 1997. How to doom yourself to repeat the past: Some reflections on the history of cognitive technology. *Cognitive Technology,* 2:2, 4-15.

Hoffman, R.R., Hayes, P.J. Ford, K. M. & Hancock, P. (2002). The triples rule. **ref.

Horvitz, E. 1999a. Uncertainty, action, and interaction: In pursuit of mixed-initiative computing. *Intelligent Systems,* Sept./ October Issue, IEEE Computer Society.

Horvitz, E. 1999b. Principles of mixed-initiative user interfaces. In *Proceedings of CHI '99, ACM SIGCHI Conference on Human Factors Computing Systems*. Pittsburgh, PA, May.

Hutchins, E. 1995. How a cockpit remembers its speeds. *Cognitive Science*, 19, 265-288.

Johnson, L. In press. Natural interaction with animated agents. In J.M. Bradshaw (Ed.), *Handbook of Software Agents*. Cambridge, MA: AAAI Press/The MIT Press.

Kaminka, G. and Tambe, M. 2000. Robust agent teams via socially attentive monitoring. *Journal of Artificial Intelligence Research*, 12, 105-147.

Larkin, J.H. and Simon, H.A. 1987. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science,* 11, 65-99.

Laurel, B. 1997. Interface agents: Metaphors with character. In J. M. Bradshaw (Ed.), *Software Agents,* pp. 67-78. Cambridge, MA: AAAI Press/The MIT Press.

Lesh, N., Rich, C. and Sidner, C. 2001. Using plan recognition in human-computer collaboration. In *Proc. 7th International Conference on User Modeling*.

Licklider, J.C.R. 1960. Man-computer symbiosis. In *IRE Transactions in Electronics,* pp. 4-11. New York: Institute of Radio Engineers.

Lieberman, H. (Ed.). 2001. *Your Wish is My Command: Programming By Example*. San Francisco, CA: Morgan Kaufmann.

Lieberman, H. and Selker, T. 2002. Agents for the user interface. In J.M. Bradshaw (Ed.), *Handbook of Software Agents*. Cambridge, MA: AAAI Press/The MIT Press, to appear.

Maes, P. 1997. Agents that reduce work and information overload. In J.M. Bradshaw (Ed.), *Software Agents,* pp. 145-164. Cambridge, MA: AAAI Press/The MIT Press.

Malin, J. 1997. Designing highly-autonomous systems managers to communicate for teamwork. Position paper for the NSF Workshop on *Human-Centered Systems: Information, Interactivity, and Intelligence* (HCS), Arlington, VA, February 17-19.

Marsh, S. 1994. *Formalizing Trust as a Computational Concept*. Ph.D. Thesis, University of Stirling, April.

Marsh, S. 1992. Reliance in multi-agent systems: A preliminary report. In *MAAMAW'92, 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World,* Rome.

Maturana, H.R. and Varela, F.J. 1992. *The Tree of Knowledge: The Biological Roots of Human Understanding* (rev. ed.) Boston: Shambala.

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence, 42*(2-3): 363-391.

Mowshowitz, A. 1997. Virtual organization. *Communications of the ACM,* 40, 9.

Muscettola, N., Pandurang Nayak, P., Pell, B. and Williams, B. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence,* 103(1-2), 5-48.

Musliner, D.J., Durfee, E.H. and Shin, K.G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence,* 74, 83-127.

Norman, D. A. (1988). *The Psychology of Everyday Things*., New York: Basic Books

Norman, D.A. 1992. Cognitive artifacts. In J. M. Carroll (Ed.), *Designing Interaction: Psychology at the Human-Computer Interface,* pp. 17-38. Cambridge, England: Cambridge University Press.

Pawson R, 2000. *Expressive systems: A manifesto for Radical Business Software*.. http://www.expressive.systems.org/

Perrow, C. 1984. *Normal Accidents: Living With High-Risk Technologies*. New York: Basic Books.

Pynadath, D. and Tambe, M. 2001. Revisiting Asimov's First Law: A response to the call to arms. In *Intelligent Agents VIII, Proceedings of the International workshop on Agents, Theories, Architectures and Languages* (ATAL'01).

Pynadath, D., Tambe, M., Arens, Y., Chalupsky, H., et al. 2000. Electric elves: Immersing an agent organization in a human organization. In *Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents—The Human in the Loop*.

Reeves, B. and Nass, C. 1996. *The Media Equation*. New York: Cambridge University Press.

Rich, C. and Sidner, C. 1998. Collagen: A collaboration manager for software interface agents. *Technical report 97-21a*. Mitsubishi Electric Research Labs, Cambridge, MA

Rich, C., Sidner, C. and Lesh, N. 2001. COLLAGEN: Applying collaborative discourse theory. *AI Magazine*.

Scerri, P., Pynadath, D. and Tambe, M. 2001. Adjustable autonomy in real-world multi-agent environments. *International Conference on Autonomous Agents* (Agents'01).

Sheridan, T. 1980. Computer control and human alienation. *Technology Review,* 10, 61-73.

Sierhuis, M. 2001. *Modeling and simulating work practice; Brahms: A multi-agent modeling and simulation language for work system analysis and design*. Ph.D. thesis, Social Science and Informatics, University of Amsterdam, The Netherlands.

Still, D.L. and Temme, L.A. 2002. OZ: A human-centered cockpit display.

Suchman, L.A. 1987. *Plans and Situated Action: The Problem of Human Machine Communication*. Cambridge University Press: Cambridge, MA.

Swarup, V. and Fábrega, J.T. 1999. Trust: Benefits, models, and mechanisms. In J. Vitek and C. Jensen (Eds), *Secure Internet Programming,* pp. 3-18. Springer Verlag.

Sycara, K. and Zeng, D. 1996. Coordination of multiple intelligent software agents. *International Journal of Cooperative Information Systems,* 5, 2&3.

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research,* 7, 83-124.

Tambe, M., Newell A. and Rosenbloom, P. 1990. The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning,* 5, 3, 299-348.

Tambe, M., Pynadath, D., Chauvat, C., Das, A. and Kaminka, G. 2000. Adaptive agent architectures for heterogeneous team members. *Proceedings of the International Conference on Multi-agent Systems* (ICMAS).

Van Beek, P. and Cohen, R. 1991. Resolving plan ambiguity in cooperative response generation. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Van de Velde, W. 1995. Cognitive architectures—From knowledge level to structural coupling. In L. Steels (Ed.), *The Biology and Technology of Intelligent Autonomous Agents,* (pp. 197-221). Berlin: Springer Verlag.

Wiener, E.L. 1989. Human factors of advanced technology, "Glass Cockpit" transport aircraft. *NASA contractor report no. 177528*. Moffett Field, CA: NASA Ames Research Center.

Weld, D. and Etzioni, O. 1994. First Law of Robotics. *Proc. AAAI'94*.

Woods, D.D. 1997. Human-centered software agents: Lessons from clumsy automation. NSF workshop on *Human-Centered Systems: Information, Interactivity, and Intelligence*. Arlington, VA, February. http://www.ifp.uiuc.edu/nsfhcs/abstracts/woods.txt

Software Agents for the Warfighter
Part 2: Technology Components

Semantic Integration
Brief Overview

Description

To successfully perform their required tasks, intelligent information agents require accurate and meaningful communication and integration among other agents and information resources. However, the applications and infrastructure of information technology are rife with differences at many levels. This constitutes a major barrier to successful agent inter-operation but is also an unique opportunity for agent technology. In this chapter we address the specific problem of *semantic heterogeneity*, which is focused on the fact that different applications and databases and agents may ascribe different meanings to the same terms or use different terms in ways that are mutually incompatible (such as different level of abstractions or generality). Semantic issues are contrasted from syntactic or low-level structural issues, in that they are focused on the *meaning of the concepts* that are represented by terms, structures, and expressions that are exchanged among systems.

It has been recognized that the emerging technology concerned with the development and application of *ontologies* will play a central role in achieving *semantic integration (*Wache, et al. 2001; Stuckenschmidt and Visser 2000). An ontology is used by an agent, application, or other information resource to declare what terms it uses, and what the terms mean. By making this information available to agents that are potential users, it becomes possible for high fidelity communication to take place. Agents can communicate and share meaning with other agents, and agents can understand the meaning of information provided by applications, databases, and other information resources on the Web.

Two agents can be completely semantically integrated if they have a shared understanding of the terminology in their corresponding ontologies. If we ultimately want *automated* semantic integration, agents will need to communicate using their ontologies alone. This places a strong requirement on the adequacy of an ontology, and current technology can at best support semi-automated semantic integration (in which human intervention is needed to resolve semantic disagreements.

The over-arching challenge that we are addressing in this chapter is to *create networks of semantically integrated agent communities*. Meeting this over-arching challenge gives rise, in turn, to a wide variety of other challenges. Within such communities, agents will share and reuse their ontologies; however, in current practice, we still encounter difficulties in reusing and sharing the ontologies that we have designed. Given the tremendous variety of existing ontologies, the specification of and determination of the consistency of semantic mappings between multiple ontologies is essential.

Semantic Integration

We identify four major architectures that are being used within industry, government, and academia to support semantic integration. The differences between these architectures depend on the origins of the semantic mappings between the ontologies, the use of a mediating ontology, and the degree of agreement that exists among the anticipated community of interacting agents.

Within the context of these architectures, we review and evaluate six major technologies that address the challenges for semantic integration: formal languages used for specifying ontologies; the ontologies that are currently available to support semantic integration; techniques for generating and testing semantic mappings between ontologies; semantic markup; software support for the ontology lifecycle; and techniques for agent capability matching.

Relevance to the Warfighter

The challenge of semantic integration arises in any scenario involving agents that are designed by different teams or that are created for different domains. It is infeasible to assume that semantic integration problems will be solved by the adoption of a single common ontology by the U.S. military. Each branch of the U.S. armed forces (Army, Navy, Marines, and Air Force) will realistically use different ontologies that reflect the special nature of their battlefield environment. Further, there is a large number of defense contractors associated with each piece of military hardware and software, and each of these contractors themselves face problems of semantic integration within their own companies and supply chains. The context of modern military operations also means that missions can often involve international coalitions of disparate nations where prior agreements to share a single model are unlikely.

Risks

The promise of complete automated semantic integration is to support seamless exchange of data among computer systems that preserves the semantics intended by the communicating intelligent agents. If this were achieved, then software could potentially perform exactly as designed and there would be no risks. Until we achieve *complete* semantic integration, all risks are associated with incomplete integration. In such cases, we cannot guarantee that the exchange of information among agents preserves the intended semantics of the agents involved. In concrete terms, this can lead to a breakdown in communication that requires the intervention of humans to resolve semantic conflicts or to unintended behavior of the agents. The tradeoff between the incompleteness of semantic integration and the consequences of this incompleteness is a major research challenge.

Semantic Integration

Forecast

The long-term goal of self-integrating agents is many years away. In the short- and medium-term, we need to carefully examine the nature of semantics as we understand and use it today. At one extreme, the meaning of terms is implicit in the minds of the agent designers and in the behavior of their agents. At the other extreme, agent ontologies provide formal and explicit descriptions of the meaning of terms that are fully interpretable by other computational agents. We believe that progress toward semantic integration will take place by moving along this continuum. Only in the latter case is it possible to ensure semantic fidelity.

Awareness of this semantic continuum is important because the goal of achieving self-integrating agents in the *general case* is difficult. It is likely that it will be a research effort of 10+ years to develop practically useful robust self-integration in some limited contexts. Thus, we need to make assumptions and work in restricted domains to make progress. Not all tasks require guaranteed semantic fidelity. For these, it is possible to relax the requirement that the ontologies are formal. Taxonomies, thesauri, and various lexical resources such as WordNet can provide great value in achieving semantic integration in limited ways. It is also possible to relax the formality constraint if there is sufficient agreement about what terms mean that it does not need to be determined at agent-execution time.

Summary and Recommendations

A key factor in the development and widespread deployment of ontologies will be the extent to which ontologies will fulfill the promise of sharability and reusability. To some extent, these challenges will depend on social factors (such as the formation of community ontologies and the adoption of standard interlingua ontologies) as well as technical factors.

There is no easy way to map at the semantic level. In the short term, the emphasis is likely to be on tool support to enable humans to more quickly and accurately specify pre-defined mappings that are used at runtime to determine what a given term means with respect to a given agent's ontology. In the longer term, work will proceed which will enable agents to semi-automatically determine the meaning of new terms encountered through interactions with other agents.

There are several critical issues in semantic integration that can only be solved by empirical approaches. Progress in resolving these issues will only occur with the establishment of test bed environments that consist of multiple agents and ontologies within each of the integration architectures discussed in this chapter and that allow participants to perform experiments that test alternative approaches.

Relevance to the Warfighter

Semantic Integration

The challenge of semantic integration arises in any scenario involving agents that are designed by different teams or that are created for different domains. We cannot assume that semantic integration problems will be solved by the adoption of a single common ontology by the US military. Each branch of the US armed forces (Army, Navy, Marines, and Air Force) will realistically use different ontologies that reflect the special nature of their battlefield environment, and these ontologies must be integrated to support joint/coalition operations. Further, there is a large number of defense contractors associated with each piece of military hardware and software, and each of these contractors themselves face problems of semantic integration within their own companies and supply chains.

Advanced Sensor Grids

The use of advanced sensor grids by a coordinated team of intelligent agents strongly depends on the semantic integration of the agents and a host of heterogeneous information sources. Since many sensors are designed and implemented for a wide variety of purposes outside of military applications, it can be assumed that they will have very different ontologies. The use of ontologies enables an integration of entities operating on different level of abstractions. For instance an agent controlling simple sensor can operate with a sophisticated decision-making agent even that they use concepts at different levels of abstraction. The problem of sensor fusion consequently includes the problem of merging multiple ontologies to ensure the best feasible picture of the ongoing operational environment.

Unmanned Autonomous Systems

Within the context of unmanned autonomous systems, semantic heterogeneity arises in two situations—operational and design time. The first situation arises when different agents from the different branches of the Armed Forces need to communicate. Unmanned systems will operate in teams that must collectively achieve mission goals as specified by human operators. Autonomous systems that play soccer have the luxury of operating with total agreement and sharing of their (possibly implicit) ontologies. This is unrealistic when we have to combine autonomous systems that are operated by the different branches of the Armed Forces and were designed and implemented by different defense contractors. In the more realistic scenario, semantic integration is required to merge the ontologies of each autonomous system to ensure that they are able to communicate with each other and with central command.

The second situation in which issues of semantic integration arise for unmanned autonomous systems is in the design of the autonomous systems themselves. There is heterogeneity in the various ontologies that represent the world at different levels of abstraction. There is also heterogeneity in the representations for the perceptual and control subsystems within the agent. These also require semantic integration.

Advanced Command Posts

The lack of semantic integration between an advanced control post and the operational units in the field can lead to the wrong tactical decisions being made by commanders. Battle plans (at different levels of abstraction) must be communicated to the operational units, and all of the involved parties must be able to share the plans and updated tactical data in such a way that the semantics of the plans are preserved.

Mobile Operations

The dynamic nature of mobile operations requires the synthesis of tactical data from multiple heterogeneous information sources and the subsequent update of changes to an operational plan in response to new battlefield conditions. If all warfighter agents share a common ontology, then such problems can be avoided. However, if different operational units are in fact using different ontologies, then semantic integration is required to support the coordinated action of these units in a combat situation.

Joint/Coalition Operations

In many ways, the challenge of joint/coalition operations is the prototypical source of semantic integration problems. Any such operation not only involves the coordination of U.S. Army, Navy, Marine, and Air Force units, but also coordination with the armed forces of other countries, and it is likely that there will not be a common ontology that is shared by all of these operational units. To prevent coordination errors among coalition partners during both planning and execution of an operational plan, the seamless exchange of information is essential.

Logistics

The nature of materiel distribution networks and dynamic transportation plans means that multiple operational units are constantly exchanging logistics information at multiple levels of abstraction, both with respect to command structures and with respect to different time horizons (e.g. hourly, daily, longer term). Without common agreement on the semantics of resource requirements and operational constraints, real-time integration between both military units and suppliers cannot be achieved; the result is that the necessary supplies are not delivered to the units as planned, impeding the proper execution of battle plans.

Semantic Integration

Information Assurance

Semantic integration appears to have little relevance to issues of information assurance among agents.

Semantic Integration

Technical Description

Introduction

The Problem: Semantic Heterogeneity

When agents communicate with each other, there needs to be some way to ensure that the meaning of what one agent 'says' is accurately conveyed to the other agent. There are two extremes, in principle, for handling this problem. The simplest (and perhaps the most common) approach is to ignore the problem altogether and just assume that all agents are using the same terms to mean the same things. The assumption could be implicit and informal, or it could be an explicit agreement among all parties to commit to using the same terms in a pre-defined manner. This only works, however, when one has full control over what agents exist and what they might communicate. In reality, agents need to interact in a much wider world, where it cannot be assumed that other agents will use the same terms, or if they do, it cannot be assumed that the terms will mean the same thing.

The moment we accept the problem and grant that agents may not use the same terms to mean the same things, we need a way for an agent to discover what another agent means when it communicates. In order for this to happen, each agent will need to make available to the agents with which it communicates, declarations of exactly what terms it is using, and what those terms mean. This specification is commonly referred to as the agent's *ontology*. If we were supporting communication among human agents, the ontology could simply be a glossary. However, meaning must be accessible to other software agents. Thus, the ontology must be encoded in some kind of computer-interpretable formal language. Such an encoding will enable a given agent to use automated reasoning to accurately determine the meaning of other agents' terms. For example, if Alice sends a message to Bob, then along with this message is a pointer to Alice's ontology. Bob can use Alice's ontology to see what the terms mean, the message is successfully communicated, and Bob's task is successfully performed. For this to happen consistently, reliably and fully automatically in practice, there are a plethora of difficulties that must be addressed.

There are many languages for expressing ontologies—and their semantic properties vary in important ways. They may be based on different underlying paradigms, they may support different levels of expressiveness, and the formal properties may differ. Even if the exact same representation language is used, two different people given the task of creating and defining a set of terms and specifying their meaning for the same domain will typically produce two very different ontologies. Different terms may mean the same thing, the same term might mean different things, and some concepts will be included by one and ignored in the other. More fundamentally, the concepts that are identified as important may be encoded in very different ways in the same language. They may even be logically incompatible. Even if the same language is used, and even if there is substantial similarity in the choice of concepts and how they are encoded, *the*

reasoning required to determine whether two terms actually mean the same thing is, in general, intractable (Uschold 2001a).

It is the goal of this chapter to explore the myriad of issues and technologies related to the goal of overcoming these problems of semantic heterogeneity. For further reading on how semantic heterogeneity differs from other kinds of heterogeneity (e.g., syntactic, structural), see (Stuckenschmidt, et al. 2000). Much progress has been made in these other areas. Achieving integration at the semantic level is the most difficult and challenging goal.

Agents, Semantic Integration and the Semantic Web

Intelligent information agents require a computing infrastructure of some sort in which to operate. The infrastructure provides the medium for communication and interaction among agents, and for accessing applications, databases, documents, and any other resources that agents require to perform their tasks. The obvious infrastructure for agents to operate in is the Web—including intra-nets within companies or the Armed Services.

In the coming years, the Web is expected to evolve from a structure containing information resources that have little or no explicit semantics to a structure having a rich semantic infrastructure. The key-defining feature that is intended to distinguish the future Semantic Web from today's Web is that *the content of the Web will be usable by machines* (i.e., software agents). This will enable semantics-preserving communication between agents who advertise and/or require services to perform tasks on the Web, and it will enable agents to determine the meaning of the passive (i.e., non-agent) information on the Web that an agent requires to perform its tasks.

There is a great deal of excitement these days about this so-called 'Semantic Web.' There is a feature article in *Scientific American* (Berners-Lee, et al. 2001), there is a major DARPA program devoted to it (DAML 2001), and the World Wide Web Consortium (W3C) has embraced it as a formal activity (W3C 2001). There is much research activity devoted to developing a supporting technological infrastructure. However, it remains a vision, not a reality.

The main ideas and issues of semantic integration that we will discuss existed long before the Web. However, all of the technologies and approaches have the *potential* to be applied on the Web. We call attention to two main points about the relationship between agents, semantic integration, and the Semantic Web.

1. The pervasive use of the Web by people and the attempt to automate many tasks that people wish to do on the Web have brought the problems of semantic heterogeneity into much clearer focus.
2. The Semantic Web is an excellent testbed for exploring and demonstrating semantic integration of agents.

Sometimes, a technology we will be discussing most naturally arises in a Web context. In those cases, we will refer to the Semantic Web as if it is the assumed testbed for the idea. In other cases, the Web is really very much a side issue, so we will often not bring it up.

The Role of Ontologies

To address the challenges of semantic integration and reusability, various groups within industry, academia, and government have been developing sharable and reusable models known as ontologies. The most commonly quoted definition of an ontology is *"a formal, explicit specification of a shared conceptualization"* (Gruber 1993[1]). A *conceptualization*, in this context, refers to an abstract model of how people think about things in the world, usually restricted to a particular subject area. An *explicit specification* means that the concepts and relationships of the abstract model are given explicit names and definitions. The name is a term, and the definition is a specification of how the term necessarily relates to other terms. *Formal* means that the specification is encoded in a language whose formal properties are well understood—in practice, this almost always means logic-based languages. Formality is an important way to remove ambiguity that is prevalent in natural language; it also opens the door for automated machine processing of intended meaning.

All approaches agree that there are two essential components of any ontology:

1. a vocabulary of termsthat refer to the things of interest in a given domain,
2. some specification of meaning for the terms, grounded in some form of logic.

What distinguishes different approaches to ontologies is the degree and manner of specifying the necessary relationships among terms. This gives rise to a kind of continuum of kinds of ontologies. At one extreme, we have very lightweight ones that may consist of terms only, with little or no specification of relationships among the terms (the degenerate case of an ontology). At the other end of the spectrum, we have rigorously formalized logical theories, which comprise the ontologies (see Figure 1). As we move along the continuum, the number of necessary relationships specified increases (thus reducing ambiguity), the degree of formality increases, and there is increasing support for automated reasoning. (See Figure 7 for an example of an ontology that organizes its terminology as a taxonomy.)

[1] In a recent special issue of IEEE Intelligent Systems on the Semantic Web, leading workers in this even newer field of study also claim that the above definition best characterizes the essence of an ontology [Fensel *et al*. 2001].

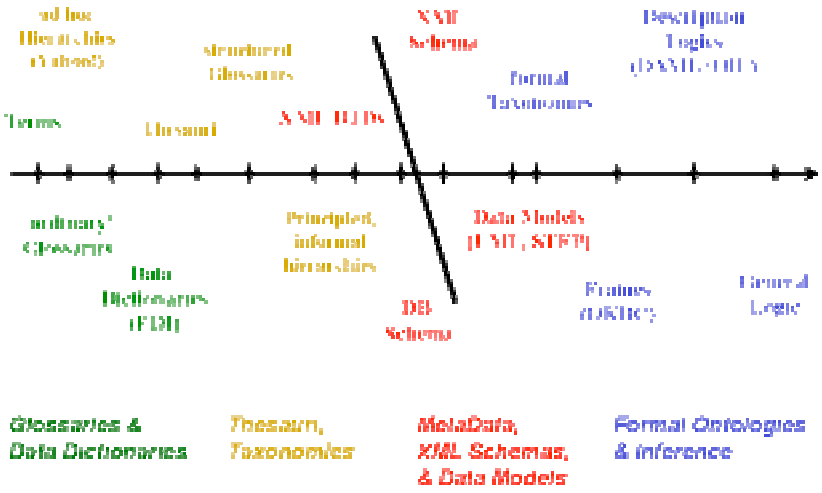


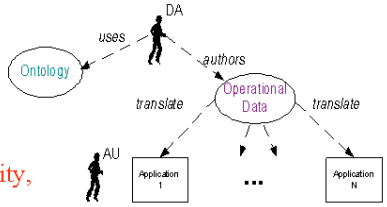
Figure 1: Kinds of Ontologies – *There are many kinds of things that people call ontologies. Moving to the right there is reduced ambiguity and increased number of necessary relationships, formality, and support for automated reasoning. Not everyone agrees on what is or what is not an ontology. We have drawn a line to the right of which there is quite broad agreement that they are ontologies. To the left, it is more controversial.*

How do ontologies help? For agents to successfully communicate, they must publicly declare what terms they are using and what those terms mean. Ontologies are the vehicle for doing this. The promise of ontologies is "*a shared and common understanding of a domain that can be communicated between people and application systems*" (Fensel 2001). Ontologies may be applied in a number of ways. Chief among these are (Jasper and Uschold 1999):

1. Neutral Authoring
2. Common Access to Information
3. Specification
4. Search

The key aspects of these are summarized in Figures 2 and 3. The primary focus of this paper will be using ontologies for common access of information, and to a lesser extent search. The others are relevant, but will not be addressed in any detail. Also note that there are many applications of ontologies that are covered by these four aspects. For example, the use of ontologies in team collaboration, as glossaries for enterprises, and in support of knowledge management are instances of common access to information. The application of ontologies to support repositories of design knowledge is an instance of neutral authoring.

- Neutral Authoring
  - artifact authored in single language, based on ontology
  - converted to multiple target formats
  - *Benefits*: knowledge reuse, maintainability, long term knowledge retention



- Ontology as Specification
  - build ontology for required domain
  - produce software consistent with ontology
    - manual or partially automated
  - *Benefits*: documentation, maintenance, reliability, knowledge (re)use

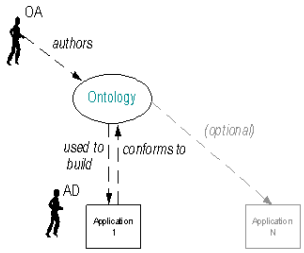
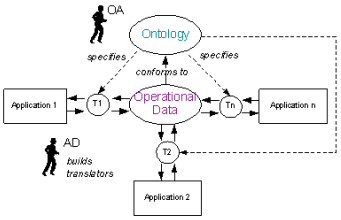


Figure 2: Ontology Application Scenarios – *An ontology supports reuse when it is used as a neutral authoring format; the ontology is built once, and then converted into multiple target formats. An ontology may also be used as a specification; all software must then be consistent with the ontology.*

- Common Access to Information
  - information required by multiple agents
  - expressed in wrong terms/format
  - ontology used as agreed standard, basis for converting/mapping
  - *Benefits*: interoperability, more effective use/reuse of knowledge



- Ontology-Based Search
  - Ontology used for concept-based structuring of information in a repository
  - *Benefits*: better information access

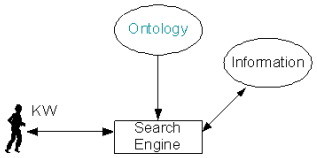


Figure 3: More Ontology Application Scenarios – *An ontology supports interoperability by providing common access to information. The ontology serves as an agreed standard that is used as the basis for mapping between agents. An ontology can also be used for concept-based structuring of information in a repository, leading to better searching techniques for information retrieval.*

Semantic Integration

What do we mean by 'semantics'?

The core meaning of the word 'semantics' is *meaning* itself. In the context of achieving successful agent communication, we are talking about the need for agents to understand the meaning of the information that they are exchanging and the meaning of the content of various information sources that agents require in order to perform their tasks. We focus attention on the questions of in what form is intended meaning expressed and how it is used. We identify a kind of semantic continuum ranging from the kind of semantics that exist on the Web today to a rich semantic infrastructure on the Semantic Web of the future. We claim that progress in overcoming problems of semantic heterogeneity will take place by moving along this continuum.

A Semantic Continuum

We ask three questions that give rise to a continuum of situations. At one extreme, there are no specifications of intended meaning at all, except those that are in the minds of the people who use the terms. At the other extreme, we have formal and explicit specifications of intended meaning that are fully automated.

1. *Are the semantics explicit or implicit?*
2. *Are the semantics expressed informally, or formally?*
3. *Are the semantics intended for automated processing?*

We describe four somewhat arbitrary points along this continuum (Figure 4)—there are many cases that are not clear cut and thus arguably may fall somewhere in between.

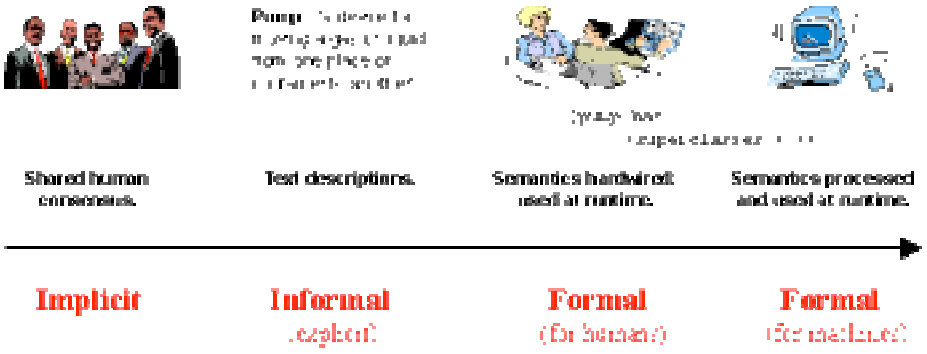


Figure 4: Semantic Continuum – *Semantics may be implicit, existing only in the minds of the humans who communicate and build systems. It may be explicit, and informal, for example in English. Formal semantics go a long way toward reducing ambiguity. They may serve a very useful role even if only used by humans. They also make possible automated inference, which can enable machines to automatically infer something about the meaning of terms.*

Implicit Semantics

In the simplest case, the semantics are *implicit* only. Meaning is conveyed based on a shared understanding derived from human consensus. A common example of this case is the typical use of XML tags, such as *price*, *address*, or *delivery date*. Nowhere in the XML document, nor anywhere else, does it say what these tags mean (Cover 98). However, if there is an implicit shared consensus about what the terms mean, then people can embed this implicit semantics in suitable wrappers. Online travel agents and booksellers routinely do this to find the best deals. From the perspective of mature commercial applications on the Web, this is the current state of the art. The disadvantage of implicit semantics is that they are rife with ambiguity. People often *do* disagree about the meaning of a term. For example, prices come in different currencies, may or may not include shipping and other services, etc.

Informal Semantics

At the next point on the continuum, the semantics are explicitly specified in an *informal* manner, often in a text specification document. Until the natural language processing problem is solved, only humans can make direct use of informally expressed semantics. Examples of informal semantics that are expressed in text specification documents are 1) the meaning of tags in HTML (e.g., <h2> means second level header); 2) the meaning of the `subClassOf` relationship in the original specification of the RDF Schema language (W3C 1999); and 3) the meaning of expressions in information modeling languages such as Dublin Core.

Typically, the semantics expressed in informal documents are embedded (i.e., hardwired) *by humans* in working software. For example, compiler writers use language definition specifications to write compilers. In the example in Figure 5, the ability of the agent to infer something about the meaning of *fuelpump* depends on the existence of a formal semantics of the underlying ontology language that is used to specify the ontology. However, although the semantics of the *expressions* in the language is machine-processible, the semantics of the *terms* is not – it is for humans only. People use the semantics of the expressions to write inference engines or other software to correctly interpret and manipulate expressions in the language. In this way they may embed in the code they write meanings of expressions that are not included in the formal semantics. . On the one hand, manually embedding the meaning of terms might be necessary; for example, there are no formal definitions for the terminology of Dublin Core, so that any agent system that conforms to Dublin Core has no other way of specifying its intended models. On the other hand, different implementations may be inconsistent.

The main disadvantage of informally specified semantics is that there is still much scope for ambiguity. This decreases one's confidence that two different implementations (say of RDF Schema or Dublin Core) will be consistent and compatible. Each implementation is different in sometimes subtle ways, and this can result in problems if interoperability is required or if implementations change.

Formal Semantics for Human Processing

In this case, we have explicit specifications of necessary relationships among terms (i.e., ontologies) expressed in a formal language. However, those specifications are intended for human processing only. They are, in effect, comments expressed in a formal language. Some examples of this are:

1. Modal logic is used to define the necessary relationships among ontological categories such as rigidity and identity (Guarino, et al. 1994). These are for the benefit of humans to reduce or eliminate ambiguity in what is meant by these notions.

2. Modal logic is used to define the necessary relationships among performatives such as *inform*, and *request* in agent communication languages (ACL; Smith, et al. 1998). Humans use the formal definitions to understand, evaluate and compare alternative ACLs, and/or to implement agent software systems that support these notions.

3. The axioms in many ontologies (such as the Enterprise Ontology; Uschold, et al. 1998) are created without the expectation that they would be automatically processed. Instead, the purpose of the axioms is to help communicate the intended meaning to people.

However, formal semantics for human processing do not directly support automated semantic integration of intelligent agents.

Formal Semantics for Automated Inference

Finally, there is the possibility of explicit specifications of necessary relationships among terms expressed in a formal language that are intended for automated inference. The idea is that when terms described in the specification are encountered, it is possible to automatically interpret something about their meaning and thus how to use them. Let us consider a simple example of how this can work (see Figure 5).

Suppose that an agent is tasked with discovering information about a variety of mechanical devices. It encounters a Web page with the term *fuel pump*, which it has never encountered before. Lacking natural language understanding capability, the term is so ambiguous that it could mean anything. We can reduce the ambiguity by associating the term *fuel-pump* with a formal definition of what the term means (in current approaches to the Semantic Web, this is called *semantic markup*). In this case, the definition refers to a term defined in an external Shared Hydraulics Ontology. The agent can determine from that ontology that a fuel pump is a subclass of *pump*, which in turn is a subclass of *mechanical device*. The agent is now able to return this document as being relevant to mechanical devices, even though it has never before heard of the term *fuel-pump*. It is possible to do this with today's technology, as evidenced by the various research tools that have been developed (Decker, et al. 1999; Jasper and Uschold 2001). There

are also attempts to commercialize this technology (e.g., Ontoprise 2001). Scale remains a huge barrier to commercial success.

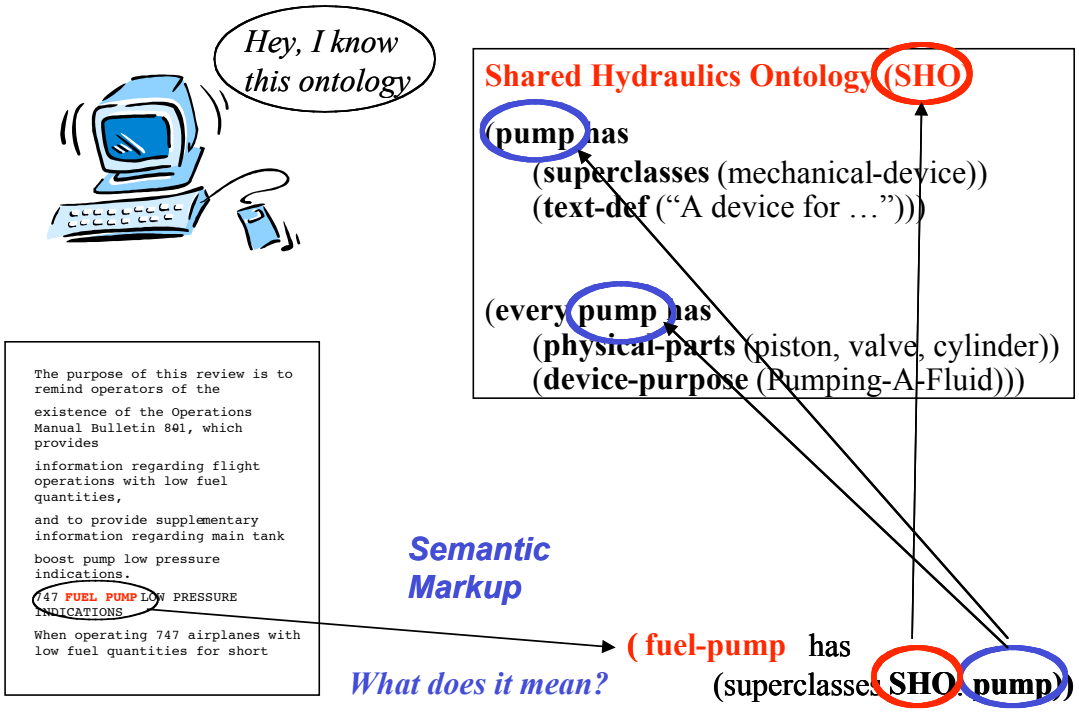


Figure 5 Automated Machine-Processible Semantics -- *An agent is searching for information on mechanical devices, as defined in a public ontology (SHO). A document contains the term 'fuel-pump,' which the agent has never encountered. Semantic markup reveals that it is a kind of 'pump' as defined in SHO, which is in turn defined in SHO as a kind of mechanical device. The agent infers that the document is relevant.*

This example illustrates the importance of semantic markup and the sharing of ontologies. It also demonstrates the importance of formal ontologies and automated inference. Inference engines can be used to derive new information for a wide variety of purposes; in particular, a formally specified ontology allows agents to use theorem proving and consistency checking techniques to determine implicit answers to queries (e.g., that 'fuel-pump' is relevant to 'mechanical-device') and whether or not they have agreement on the semantics of their terminology. Suppose the agent encounters the term *fuel-pump* and the agent's ontology includes the statement that a *fuel-pump is a subclass of pump*. Furthermore, *pump* is a term that the agent already knows about, e.g. via a public standard ontology. Even though the agent never encountered the term *fuel-pump* before, it is able to automatically infer something about the semantics of the term because it is explicitly mapped to a concept that is already included in its ontology. Specifically, it may infer that it is not a *typewriter*, or a *spaceship*, because it can infer from the

ontology that these things are not subclasses of *pump*. There is still ample scope for ambiguity. We don't know anything about how a fuel pump is different from any other kind of pump, but we do know some of the necessary properties of a fuel pump.

Formally specified ontologies use a logical language, such as DAML+OIL (Hendler and McGuinness 2001) and KIF (Knowledge Interchange Format; Genesereth and Fikes 1992; Hayes and Menzel 2001). A formal ontology consists of a set of sentences in this underlying logical language. Within mathematical logic[1], these sentences are also known as *axioms,* so that a formal ontology is also said to be *axiomatized.*

Challenges

Introduction

The over-arching challenge that we are addressing in this chapter may be stated as *How can we create a network of semantically integrated communities?* We can think of two theoretical extremes for semantic integration. In the first case there is *complete global agreement* on terms and their meaning. Here, issues in semantic integration do not arise. There is a single shared ontology, which need not even be explicit. At the other extreme, there is no agreement at all; we have *total semantic anarchy.*

There are many good social, economic, technical, and empirical reasons why we should never expect to achieve the former. People are reluctant to give up their familiar terms and concepts. Vendors are reluctant to give up their proprietary formats. Even if a standard is created and intended for use as a common interlingua, vendors often lack sufficient economic incentive to build and maintain robust translations to/from their proprietary formats. From a technical perspective, different choices of terms, definitions, and representation languages will suit different purposes—so adopting a global standard would inhibit progress for some. Finally, there is plenty of empirical evidence that a common global standard will never be adopted. Even at the level of a large enterprise, experience has shown that whole enterprise models do not succeed and can stifle innovation. Therefore we believe that any efforts to establish universal agreement on a single ontology are bound to fail. It should not be taken seriously as a goal.

---

[1] For an excellent introduction to logic, see [Barwise et. al. 2000].

[2] An interpretation consists of three parts:

- a set of elements (known as the *domain* or universe of discourse);
- a meaning function that associates symbols in the language with individual elements and sets of elements in the domain (intuitively this specifies what the symbols mean);
- a truth function that associates truth values with sentences in the language.

For an excellent introduction to logic, see [Barwise et al 2000].

The flip side of this is that the costs of semantic heterogeneity to companies and society are vast. The lack of standards inhibits progress for many. The rapid growth in research about ontologies is in large part a response to this fact. "The reason ontologies are becoming so popular is in large part due to what they promise: *a shared and common understanding of a domain that can be communicated between people and application systems*" (Fensel 2001).

In summary, the dream of global, or even enterprise-wide ontologies, data-models, controlled vocabularies, etc. is unrealistic. Yet, the cost of complete semantic anarchy is prohibitive. Therefore, we must move away from single-ontology views and see how to have multiple possibly conflicting ontologies.

We need to find some 'sweet spots' in between these two extremes. Locally, whenever the cost of semantic heterogeneity is too high, individuals and companies are motivated to form semantically homogenous communities. This entails developing standards to use locally within that community. These standards define the terms for the domain and thus may be viewed as (and can be formally represented as) ontologies. Local communities may now reap the benefits from standardization. A common occurrence is for another community to have already independently developed their own standard ontology for a different domain. There are various cases to consider.

1. The domains are completely different
2. The domains are different, with some important relationships between them
3. The domains are essentially the same.

In the first case, there is no problem, for the two communities need never interact. A good example of the second case is workflow and project management. Applications in both domains have the concepts of time, actions, and ordering of actions (e.g., before and after). Typically, such applications do not interoperate—yet there might be benefits if they were able to. If so, then these two communities would have to get together and hammer out their terminological differences and devise a way to interoperate. The third case is an extension of the second. It also requires the different communities to get together to create mappings that cross the terminological gap in order to achieve semantic integration.

In both the second and third cases, there is the option for both communities to merge their ontologies where there is direct overlap. This may be impractical if there are large repositories of legacy data that depend on these ontologies. An alternative is to devise ways to translate between the ontologies. This is itself a major challenge, which we consider in detail later in the chapter.

Barriers to Agent Communication

Another problem is that the ontologies may be specified in different representation languages. The syntactic differences among languages are becoming less and less of an issue with the growing use of XML. More significant is the fact that the different languages have different semantic frameworks (e.g., the Knowledge Interchange Format (KIF) vs. the Unified Modeling Language (UML)). In such cases, it can be difficult to distinguish between semantic disagreements within the ontologies and the different semantics of the underlying languages themselves. For a good case analysis which explores the semantic differences between two languages for representing ontologies and the problems this causes when trying to translate between them, see (Uschold, et al. 1998).

Even if we assume that the expressions encountered by the agents are from a language whose syntax and semantics are already known to the agent, there may still be *incompatible assumptions in the conceptualization* of the ontology. For example, there are many debates within the ontological community about the nature of objects, change, and identity (see Prior 1967). In one approach (known as endurantism), objects are 3-dimensional entities that endure through time and their identity does not change as they exist from moment to moment. In another approach (known as perdurantism), objects are 4-dimensional entities that have temporal "parts" existing at every moment within some temporal interval.

Even if we assume that the conceptualizations are compatible, it is still possible, indeed likely, that different people will build different ontologies for the same domain. Two different terms may have the same meaning; the same term may have two different meanings. The same concept may be modeled at different levels of detail. A given notion or concept may be modeled using different primitives in the language; for example, the concept of being red may be modeled by having the attribute *color* with value *Red*, or it may be modeled as the class *RedThings*. In determining whether or not translation is possible, the challenge is to distinguish between the superficial differences (such as *Red* vs. *RedThings*) from the more substantial differences that reflect radically different ontological commitments. In the former case, it is possible to specify semantic mappings between the two ontologies, but not in the latter case. Unfortunately, there are very few ontologies available that can be compared in this way, and we currently have no idea whether or not incompatible ontologies will be a widespread problem in practice.

Specific Challenges

In meeting the over-arching challenge of creating a network of semantically integrated communities, a variety of more specific challenges arises. Perhaps the most fundamental of these is in the area of *semantic mapping and translation*, which enables the terms in one agent's ontology to be translated into those in another agent's ontology. Crucial also is the fact that agents need to be able to locate the appropriate information resources, applications, etc. that enable them to perform their tasks. This requires that the content of these resources are associated with semantically well-defined concepts—we call this *semantic markup*. Semantically marking up things is a lot of work; people will not do it unless there is an immediate significant payback. A major challenge is to create semantic markup automatically.

Semantic Integration

A third major challenge is the difficulty of building good ontologies. It is very time-consuming, and difficult to do well. Once there are a lot of ontologies around, there is the challenge of reusing and sharing them to reduce the extent to which semantic heterogeneity issues arise.

Finally, we consider the challenge of fully automated self-integrating agents. This is a long-term goal. There are fundamental barriers in the way. One is the fact that the kind of inference required to achieve this fully automated agent integration is computationally intractable in the general case.

In the remainder of this section, we consider these major challenges, in turn:

1. Semantic Mapping and Translation
2. Semantic Markup
3. Difficulty of Building Ontologies
4. Reusing and Sharing Ontologies
5. Self-Integration

Semantic Mapping and Translation

We have started to see a proliferation of largely stand-alone ontologies that are useful in their local context. However, because they are independently developed, semantic interoperation among systems and applications that use different ontologies is greatly hampered.

The first step toward achieving semantic integration is for agents and information resources to publicly declare what terms they use and what they mean. However, given that the ontologies are often developed independently, and there is the need to express the meaning of a term from one ontology using terms from another ontology, *the burden of semantic integration rests on the need to translate between ontologies.* We address this fundamental challenge first.

The terms 'mapping' and 'translation' as applied to ontologies are not used in a consistent manner in the literature. To avoid confusion, we will use the terms in the following way:

Mapping: the specification of one or more links which say how to express the meaning of a term from one ontology in terms of the other ontology. A link may be simply from one term to another, or it may be a complex rule. A *mapping* from one ontology to another is a set of such links.

Creating a Mapping: the process of creating the individual links that comprise a mapping. It would normally make perfect sense to use the word mapping as a verb for this—however, we will refrain from doing so, to avoid term overloading.

Translation: the execution of a *mapping*. One may translate a whole ontology, a portion thereof, or just a single term.

The challenges of semantic mapping and translation are many. To recap briefly, the semantic foundations of the formal ontology representation

languages are often different; when the same language is used, people still conceptualize domains differently, and sometimes they will be incompatible. They will typically use different terms for the same concepts, and the concepts are often encoded using different language constructs, making it hard to translate back and forth. Even if none of these differences were present, it is not in general computationally feasible to automatically determine whether two terms actually mean the same thing.

Partial Translation – Any approach to semantic integration must face the challenge of partial translation—cases where there exist interpretations of terminology of the one agent's ontology that do not correspond to interpretations of the terminology of the other agent's ontology. Complete semantic integration is not possible in this case, since not all concepts can be exchanged while preserving the semantics; however, it is still possible to preserve the semantics of the concepts that the agents do share.

Partial translation arises primarily in three different scenarios. The agents' ontologies may use the same terminology for some sub-domains but not for others. For example, the agents may agree on the semantics of their process terminology but disagree on the semantics of their product terminology; the agents can therefore exchange any sentences that refer only to processes, but they cannot guarantee that semantics will be preserved if they exchange sentences that refer to product information.

The problem of partial translation often arises from the use of ontologies for generic domains (e.g., processes and products) together with agent-specific extensions to these generic ontologies. In practice, there will typically exist agent-specific extensions to any shared ontologies, so there may exist direct semantic mappings between agents that cannot be generated dynamically from the shared ontologies alone. However, if one agent draws a conclusion which is partly in a shared ontology and partly in its own private ontology, how does it communicate this conclusion to another agent? One approach is to restrict information exchange to terminology in shared ontologies. For example, there may exist a process ontology with a taxonomy containing the class *process* and two subclasses, *manufacturing_process* and *transport_process*. Suppose further that generic processes have duration and require resources, and that manufacturing processes transform input material into output material using a machine, whereas transport processes change location. An agent who knows only about manufacturing processes cannot share everything with an agent who knows only about transport processes; however, they can share those portions of process descriptions that use only the generic process ontology that they have in common.

Finally, partial translation arises in cases where two ontologies make different assumptions about the domain. For example, logistics management systems often make the assumption that time is discrete (e.g., there is no day between Monday and Tuesday); on the other hand, manufacturing scheduling systems often make the assumption that time is not discrete. These two systems cannot share all of their information, but they can share what they have in common, such as the ordering of time points.

Semantic Markup

In order for agents to understand the content of various information resources, that content must be marked up with semantic information. Consider the fuel pump example discussed earlier in the section on the semantic continuum. The fact that the term *fuel-pump* was associated with a formal definition was critical to this working. This is *semantic markup*. The challenge is to achieve semantic markup of Web resources on a large scale. It is difficult and time-consuming for people to provide semantic markup for preexisting documents. Authors who are creating content could do so much more easily, but it is still time-consuming. A further problem is that people will not take the trouble to markup their content unless they perceive a clear benefit. Hardly anyone bothers to use the non-semantic document metadata for Microsoft Word, for example. Even though this is a relatively minor inconveniencet, there is no widespread perception or experience that doing so is beneficial. Semantic markup will likely be much more time-consuming and difficult to automate than non-semantic markup, although there is much available by way of automated assistance.

The punch line is that for semantic markup to happen, the pain/gain ratio must be very low. Yet, it is completely critical to enabling semantic integration.

Difficulty of Building Ontologies

Building ontologies is difficult, time-consuming, and expensive, particularly if the goal is the design of an ontology that supports automated inference. One reason is that the task of building ontologies to support semantic integration is a microcosm of the integration problem itself. Semantic agreement between ontologies presupposes that there is consensus among the people who are building the ontologies, and the lack of consensus is one of the primary bottlenecks during ontology design.

Consensus is difficult to achieve because different design team members often have multiple perspectives on similar concepts. This is especially problematic when the team is multi-disciplinary, which arises, for example, in problems related to supply chain management. In such cases, the challenge is to identify which disagreements reflect different ontological commitments. In practice, the quest for consensus is dealt with in a variety of ways. At one extreme, small lightweight ontologies are developed by large numbers of people and then merged. At the other extreme, rigorous formal ontologies are developed by consortia and standards organizations. In the former case, there is a greater need for ontology mapping and merging, while the latter case requires better support for collaborative design and ontology analysis.

Another reason for the difficulty of ontology design is that ontologies are intimately bound to the problems of common sense reasoning within Artificial Intelligence (Hayes 1978, 1985). Such problems are notoriously difficult because seemingly trivial and obvious phenomena can in fact be quite challenging to deal with in knowledge representation languages and automated reasoning systems. To this extent, progress in ontology design will be heavily dependent on new research in common sense reasoning.

Finally, it is probably naive to expect that there will be a widespread development of ontology by the general public as it was Web sites using

HTML, even if the best ontology design lanaguage is standarized. Publishing Web pages has an important incentive for people. The web page is aimed for other people so its author can immediately get the attention of others. The ontologies are mainly dedicated for agents and are usually transparent for humans. Thus the author of ontology will not instantly get any attention or credit from others. Certainly, there will be ontologies developed when people are paid for it in areas, which requires rigorous quality and robustness as military or health-care applications. Today, there are many Web pages, which were developed by people who were paid for it. However, they constitute any some percentage of the Web. The real power of the Web is created by people developing pages not being paid for it. Probably, in order to achieve the same power of the Semantic Web the only way is to develop automatic ontology development tools based on natural language processing and causal reasoning[1] (Pearl 1997). Still, using the existing knowledge and tools it is possible to create islands of ontology adaptation in the areas such as military.

Reusing and Sharing Ontologies

Although ontologies came to prominence within Artificial Intelligence through the DARPA program for Sharable and Reusable Knowledge Bases (Neches, et al. 1991; Gruber 1995), there is still limited reuse and sharing of ontologies. It is difficult to determine why this is the case (Uschold, et al. 1998; Pinto 1999; Goldstein and Esterline 1995). The Ontolingua ontology library at the Knowledge Systems Laboratory contains almost 100 ontologies (http://www.ksl.stanford.edu/software/ontolingua), but there are limited links among most of them. There are cases of multiple independently constructed ontologies for the same domain (such as enterprise integration: Bernus, et al. 1996; Fillion, et al. 1995), yet there is no reuse or sharing of concepts between them. Within the context of semantic integration, this becomes the problem of how agents determine that they have overlapping sets of concepts and that they possibly share the semantics of their terminology.

Many ontologies originate as domain ontologies within different applications and scientific disciplines ontologies (Ashburner 2000; Cohn 2001; Dalianis and Persson 1997; Smith and Becker 1997). It may be argued that there are few domain concepts in common between physics and logistics and hence little reuse between ontologies for these domains. However, such domain ontologies often use very similar generic concepts; for example, both may contain a common ontology of time. The challenge of reuse and sharing involvesthe task of identifying the generic concepts within a domain ontology (see Figure 7). In fact, the goal of the Standard Upper Ontology project (Pease 2001) is to define a generic ontology that more domain-specific ontologies can reuse in this way. The Cyc ontology [Lenat 1995] also supports this organization.

[1] http://www.philosophypages.com/lg/e14.htm

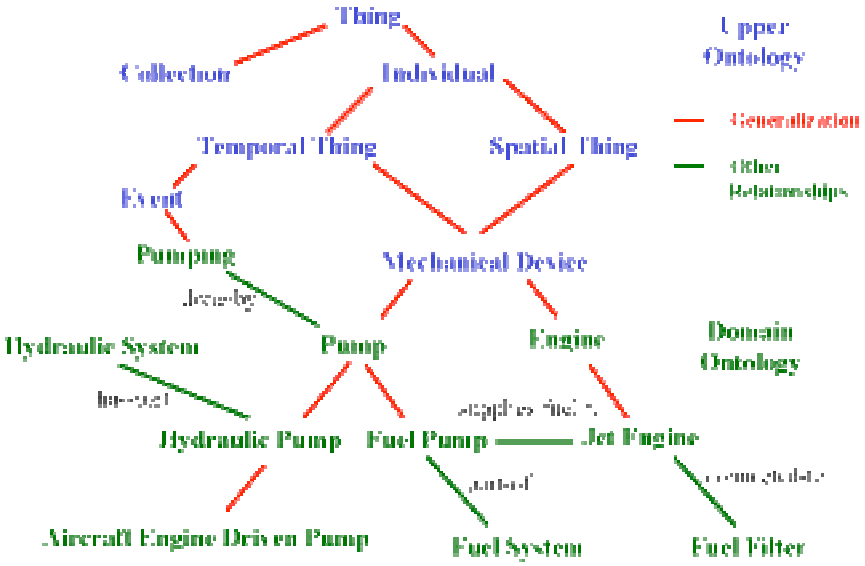


Figure 7: An Example Ontology –*In this case, we have a set of specific concepts for a particular domain attached to a set of more general concepts, which may be used as a shared basis for many other domain ontologies. The generic part is called an Upper Ontology; the specific part is called a Domain Ontology.*

Ontologies often have overlapping concepts, and these may cause problems with reuse. For example, the Standard for the Exchange of Product data (STEP; ISO10303) was designed for product modeling, and the Process Specification Language (PSL; Schlenof, et al. 1999) was designed for process modeling. However, both ontologies contain the concept *process-plan*, which is the sequence of activities that must be performed to manufacture a product according to its design specifications. Unfortunately, this concept is defined very differently in the two ontologies, preventing easy reuse between them.

Other key barriers to reuse were introduced earlier. One is the fact that ontologies are expressed in different languages. Even if the same language is used, there may be incompatible assumptions in the conceptualizations. Even if the conceptualizations are the same, people will still build different ontologies in the same domain. Terms will mean different things; different language constructs will be used to represent the same concepts.

Self-Integration

Automated semantic integration is the semantics-preserving exchange of information between intelligent agents with no human mediation during the agents' first encounter. Self-integration extends the automated semantic integration of a set of agents to include support of their cooperation in achieving some task. To achieve self-integration, agents must be self-describing—they need to advertise their capabilities, determine the capabilities of other agents from their advertisements and match the capabilities of other agents to achieve goals associated with the required task. In a self-describing system, each agent formally communicates the set of behaviors that it can exhibit as required by the actions of other agents. To

achieve self-integration, each agent is not only self-describing, but must also have some comprehension of its role within the task that the entire set of agents is achieving.

Meeting the Challenges

Many of the problems caused by semantic heterogeneity are inherent and will never go away. The challenge to the agent community and the ontology community is to discover where progress is possible and to move forward.

At least three main approaches exist. First, a lot of benefit can be obtained by increasing the degree of standardization, both in the languages and in the content of the actual ontologies. Second, where standardization is not possible, technologies need to be developed for mapping and translating between and among ontologies. Thirdly, when problems are known to be impossible, the challenge, in general, is to find ways to make simplifying assumptions that enable agents to do useful things in practical situations.

In the next two sections, we discuss ways to meet the above challenges. First we take a close look at four architectures for achieving semantic integration. This addresses the over-arching challenge of forming a network of semantically integrated communities. Following this, we take a closer look at a variety of specific technologies that can be used to address the more specific challenges discussed in the previous sections.

Architectures for Semantic Integration

In this section, we consider ways to achieve accurate communication between agents that cannot be assumed to be using the same ontologies. A semantics-preserving exchange of information means that there are mappings between logically equivalent concepts in each ontology. The challenge of semantic integration is therefore equivalent to the problem of generating such mappings, determining that they are correct, and providing a vehicle for executing the mappings, thus translating terms from one ontology into another.

We are considering the agents to be operating in an open environment. For simplicity, we will consider just two agents (Alice and Bob). They are attempting to communicate with each other, but have never interacted before. This is the environment in which most warfighter agents may be interacting, particularly in the case of joint coalition operations. It is also a more general case than a more closed scenario in which a fixed group of partners (such as a consortium, group of defense contractors or a defense alliance such as NATO) attempts to establish interoperability among their software *a priori*.

There are a variety of architectures that may be used to achieve semantic integration. The differences depend on the origins of the semantic mappings, whether there is a mediating ontology, and the degree of agreement that exists among the anticipated community of interacting agents. Different architectures can be distinguished and compared to one another by considering the following questions:

1. Who is generating and testing the semantic mapping?
    a. agent designer
    b. ontology designer, agents are reusing them
    c. agents themselves, dynamically at agent-interaction time
2. When are the mappings created that make the link between one agent's ontology and the other one's ontology?
    a. Mappings are pre-defined; the agents execute them to achieve translation between their ontologies;
    b. Mappings do not exist *a priori*; they are dynamically generated and executed to achieve translation.
3. What is the topology of the architecture?
    a. Mapping is done point-to-point between the agents;
    b. Mapping is mediated by a third ontology (or set of ontologies).

4. What is the degree of agreement between the agents?
    a. Single agreed-upon ontologies within a community, possibly merged from existing ones
    b. Alignment—could be loose or strong
    c. No *a priori* agreement

In this section, we present four types of architectures that are used to integrate agents. Each answers the above questions in different ways. The properties of these various architectures are introduced below and summarized in Table 1. We elaborate on each in subsequent sections.

Architecture	Who generates the mappings?	When is the agent-to-agent mapping defined?	Topology	Degree of Agreement
Manual mapping	Agent designers	Automatically generated when agents interact	Point-to-point	No *a priori* agreement
Interlingua ontologies	Agent designers	Defined before agents interact	Mediated	Shared ontologies
Community ontologies	Ontology designers	Defined before agents interact	Mediated	Alignment
Ontology negotiation	Agents themselves	Automatically generated when agents interact	Point-to-point	No *a priori* agreement

Table 1: Semantic integration architectures.

Status Quo

Current practice for integrating software applications relies on syntactic specifications with no explicit semantics; humans are required to construct translators between the applications based on the implicit semantics of their terminology (Bernus, et al. 1996; Fillion, et al. 1995; West and Fowler 1996). Ambiguity, unstated assumptions, and lack of agreement on the implicit semantics make integration an exceedingly difficult exercise.

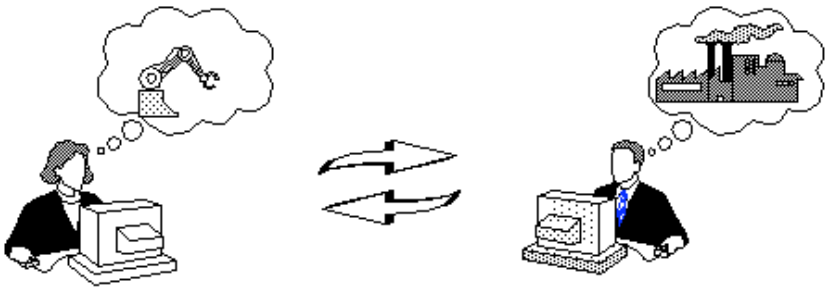


Figure 8: Status Quo – *The status quo in semantic integration does not use explicit ontologies. Ambiguity, unstated assumptions, and lack of agreement on the implicit semantics of the terminology used by software applications makes integration an exceedingly difficult exercise.*

Manual Mapping

The distinguishing feature of this architecture is that the agent-to-agent mapping is defined manually before the agents interact. The agents themselves do not generate the mappings; instead they execute them to achieve translation between the terms in their respective ontologies. The mappings are generated directly between the agent ontologies. There is no *a priori* agreement or sharing of ontologies assumed in this architecture.

Although insufficient to support self-integration, manual point-to-point translation between agent ontologies is at the leading edge of current technology. Humans generate hypothesized semantic mappings between the agent ontologies and then test these mappings to determine whether or not they actually preserve models of the ontologies (Obrst 2001; Fillion, et al. 1995).

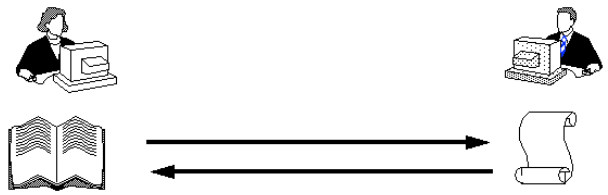


Figure 9: Manual Mapping -- *The thin arrows represent manually specified mappings between ontologies which are executed at agent-interaction time to translate between the agents' ontologies. There is no* a priori *agreement about semantics between the agents, and the mappings are point-to-point between the agents.*

The Manual Mapping architecture does address the major problem with the status quo, where the semantics are not explicitly captured in an ontology. Thus, integration often fails because each human user involved believes that their interpretation of the meaning of some term is consistent with the intended interpretation. There is no way of objectively determining whose interpretation is correct. By using explicit ontologies for software applications, it is easier to determine their correctness. Even if the ontologies are merely natural language glossaries or data dictionaries, this can help a human determine correct mappings. If they are in a formal language, this is even better, even if it is still just the human using the definitions. Automated inference may also assist in the verification of an ontology, which in turn can lead to complete semantic integration.

Although complete semantic integration can be achieved in this way, the obvious drawback of this approach is that it requires intensive human intervention and does not support the self-integration of agents since all of

the semantic mappings between the agents' ontologies have been specified prior to their first encounter.

Interlingua ontologies

The Interlingua architecture is a generalization of the Manual Mapping architecture. Its distinguishing feature is the existence of a mediating ontology that is independent of the agents' ontologies and which is used as a neutral interchange ontology (Ciociou, et al 2001). The semantic mappings between agent and interlingua ontologies are manually generated and verified prior to agent interaction time (Schlenof, et al. 1999). This process of creating the mapping between the agent ontology and the interlingua ontology is identical to the process of creating a mapping directly between two agent ontologies, as is done in the Manual Mapping approach. As such, we can consider the agent ontologies to be integrated with the interlingua ontology.

This architecture is much more powerful than Manual Mapping, particularly when there are many agents leading to the creation and maintenance of mappings becoming unmanageable. For example, we only need to specify one mapping for each agent ontology, whereas the Manual Mapping architecture requires a mapping for each pair of agent ontologies. The existence of the pre-defined mappings between the agent ontologies and the interlingua ontology enables the automatic generation of a point-to-point mapping between the agents' ontologies. If one agent's ontology changes, then only one mapping need be affected, rather than one for each agent. New agents can subscribe to the community of agents using this interlingua merely by creating a mapping to and from the interlingua. With no changes to their own mappings, all other agents now can translate between their ontology and the new agent's ontology. This was not possible in the manual mapping case because every point-to-point mapping has to be pre-specified.

Semantic mappings express the meaning of a term from one ontology in terms of the other ontology. Each such mapping rule may simply link one term to another or may specify a complex transformation. In the interlingua approach, there are two steps in translation: the execution of the mapping from the agent ontology to the interlingua and from the interlingua to the other agent's ontology. If the agent ontologies and the interlingua ontology are specified using the same logical language, then the translation can be accomplished by applying deduction to the axioms of the interlingua ontology and the formal mapping rules ([Ciocoiu 2002], [Ciocoiu & Nau 2000]). If these mapping rules have already been verified to preserve semantics between the agent and interlingua ontologies, we are guaranteed that translation between the agents also preserves semantics. In effect, a direct mapping rule from one agent's ontology to the other agent's ontology is inferred from the two separate rules. If run-time translation efficiency is important, then the point-to-point mapping rules could be cached as explicit rules; otherwise they need only be implicit. When they are implicit, then the otherwise distinct processes of creating and executing a mapping is conflated—it happens at the same time, rather than being two separate steps. See (Uschold, et al. 1999) for a more detailed discussion of the tradeoffs between the point-to-point and interlingua approaches.

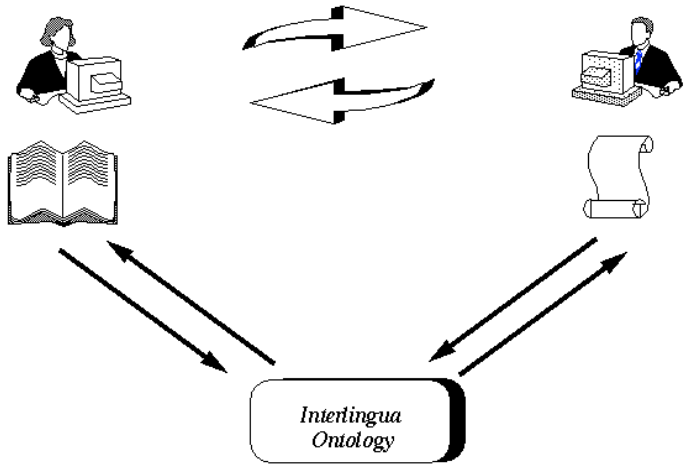


Figure 10: Interlingua Architecture -- *Alice's designer specifies the semantic mapping between Alice's ontology and a standard interchange ontology, and Bob's designer specifies the semantic mapping between Bob's ontology and the same interchange ontology. When Alice and Bob first interact, they use these previously specified mappings to automatically generate the semantic mappings between each other's ontologies. In this case, the interlingua ontology mediates the mapping between the agent ontologies. The agents that wish to participate in this architecture must agree a priori to use the interlingua ontology. The thin arrows represent manually generated mappings created by the agent designers prior to agent integration. The thick arrows represent the [possibly implicit] agent-to-agent mapping that is automatically generated.*

The interlingua need not be considered to be a *global ontology* that all agents are required to use directly. Rather, it should be considered to be a mediating ontology, as in Infomaster (Duschka and Genesereth 1997), in which all agents use their native ontology for their own reasoning and only use the interlingua ontology to communicate with each other. This perspective is particularly relevant if we consider the agents to be using heterogeneous information resources. Both the users' terminology and the native terminology of the information sources are mapped to a mediating ontology that can be thought of as specifying a reference terminology for the domain.

The Interlingua architecture is designed to work best in the context where there are a variety of different kinds of information sources, all pertaining to a common domain, rather than being an overall architecture for a multiplicity of domains. For example, planning information in a battle scenario would be exchanged between mobile operational units and the command posts via an interlingua process ontology. In practice, however, agents will use ontologies in multiple domains (e.g., process, product, resource, services) so that the architecture must be generalized to be "multi-hub", in which there may be a different interlingua ontology for each domain. In this case, an agent directly linked to one hub could be integrated with an agent in another hub if point-

to-point mappings between the hubs were provided. The critical challenge for such an approach is to ensure consistency among the set of overlapping concepts between each domain, a problem that motivates the Community integration architecture.

Community ontologies

Community ontologies are constructed by aligning existing ontologies through semantic mappings generated prior to agent interaction; agent ontologies use concepts from different modules of the community ontologies and generate direct semantic mappings from the alignment mappings. This approach is best exemplified by the following quote from Hendler (2001):

"*The Semantic Web, as I envision it evolving, will not be primarily comprised of nice neat ontologies that have been carefully constructed by expert Artificial Intelligence researchers. Rather, I envision a complex web of semantics ruled by the same sort of anarchy that currently rules the rest of the Web. Rather than a few large, complex, consistent ontologies, shared by great numbers of users, I envision a great number of small ontological components largely created of pointers to each other and developed by Web users in much the same way that Web content is currently created.*"

The Community architecture enables the reuse of predefined semantic mappings between existing agent ontologies, rather than generating new direct semantic mappings between agents at runtime. For example, suppose there is an ontology of resources (Onto1) and a manufacturing ontology (Onto2); within the ontology library, these ontologies are merged by a mapping in which the concept of "machine" in Onto2 is a subclass of the concept of "reusable resource" in Onto1. If Alice uses Onto1 and Bob uses Onto2, then they can use the predefined alignment mappings to map their concepts without generating any new semantic mappings.

In Figure 11, the community ontology consists of three previously defined and aligned ontology modules; each agent ontology uses two of these modules, although they only completely share one of these modules. Perhaps the best examples of this architecture are the Ontolingua library of ontologies (Farquhar, et al. 1996) and the DAML ontologies (www.daml.org/ontologies/).

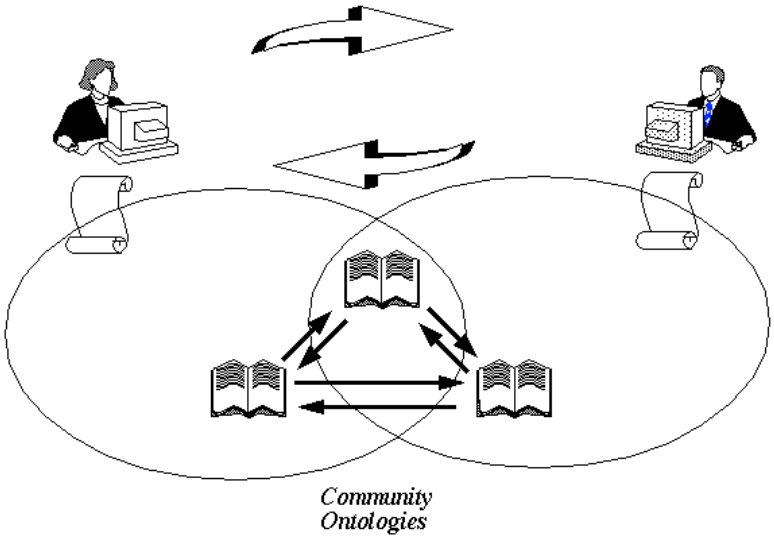


Figure 11: Community Ontologies -- *In this architecture, we assume the existence of a library of ontologies that has been built by aligning ontology modules developed by some user community. Alice's and Bob's designers use ontologies from this library, so that when Alice and Bob first interact, they reuse the relationships that were defined when the ontology modules were originally aligned in order to automatically generate the semantic mappings between each other's ontologies. Although the community ontology is also a mediating ontology, this architecture differs from the Interlingua approach insofar as it is the ontology designers who generate the mappings rather than the agent designers and the agents use the community ontologies internally rather than translating into them from some other internal ontology. Thin arrows represent alignment mappings between ontology modules generated manually by the ontology designers. Thick arrows represent automatically generated direct semantic mappings between the agents.*

The axioms in the agent ontologies are reused from the ontologies shared by the community. Since the shared ontologies are assembled "bottom-up" from independently designed ontologies, they attempt to alleviate one of the drawbacks of the Interlingua ontology, namely, the prohibitive cost of developing the interlingua ontologies.

The alignment mappings among the modules of the community ontologies that are used in an agent's ontology are included as axioms in the agent's ontology. Within the literature, these mappings have also been called articulation rules (Noy and Musen 2000). Note that the ontology designers are responsible for generating and testing the semantic mappings; the agent designers need only identify the ontology modules that they use.

Ontology negotiation

In this architecture, agents generate and test hypothesized semantic mappings at runtime without the use of any predefined mappings (Bailin 1999; Truszkowsi and Bailin 2001). In some sense, this architecture is the Holy Grail of semantic integration, since there is no human intervention in the agent interaction (except during the design of the agent ontologies).

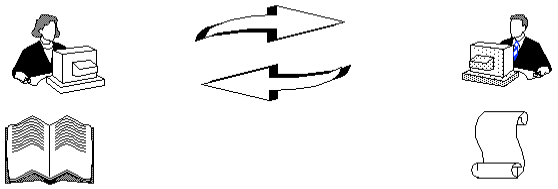


Figure 12: Ontology Negotiation -- *In the Ontology Negotiation architecture, Alice's and Bob's designers play no role to achieve semantic integration. Alice and Bob must somehow discover the semantic mappings between their ontologies on their own. Agents generate semantic mappings. Any semantic mappings between agents are generated automatically; there is no human intervention in the agent interaction (except during the design of the agent ontologies).*

In a sense, this is the direct automation of the Manual Mapping architecture. The Interlingua and Community architectures are semiautomatic approximations to Ontology Negotiation insofar as humans are still involved in some stage of the semantic mapping process.

Direct ontology negotiation may be unavoidable even within the Interlingua and Community architectures. There will always exist agent-specific extensions to any shared ontologies, so there may exist direct semantic mappings between agents that cannot be generated dynamically from the shared ontologies alone. In the limit, this leads to ontology negotiation for the agent-specific ontologies.

Human-Agent Integration

The challenge of semantic integration is to precisely determine the semantic intuitions of a human user. In all of the above architectures, complete semantic integration requires that these intuitions have been formally specified within the agent ontology. In the case of human-agent integration, this is not the case; the human is interacting directly with an intelligent agent without explicitly specifying his or her ontology prior to the interaction.

Semantic integration of humans and agents follows two directions. First, the human user needs to quickly browse and evaluate the agent's ontology to determine whether or not the agent conforms to the user's intended semantics for his or her terminology. In the other direction, the agent can learn the human user's implicit. If there is an ontological mismatch, the agent can extend its ontology through the acquisition of new terminology whose definitions are consistent with the agent's existing ontology.

This is a relatively new idea. Very little work within the research community has explored this architecture, and even what has been done has focused primarily on the related problem of collaborative ontology design (Farquhar, et al. 1996, 1997; Holsapple and Joshi 2001). It is a difficult problem for ontology designers to understand another designer's ontology and determine whether it is in fact consistent with their own and whether both designers are making the same assumptions, particularly when the ontologies are informal and the key information about the ontology is in natural language text comments.

Semantic Integration

Technologies

In the preceding section, we considered the challenges that the research community faces in building self-integrating agents. In this section, we consider the relevant technologies that will support this endeavor, as well as identifying areas in which more research is needed.

Ontology representation Languages

As we observed in the introduction, one of the sources of semantic heterogeneity lies in the languages that agents use to represent their world and the content of their communication acts. There have been several efforts within academia and industry to develop common languages that can be used as the basis for ontologies to support semantic integration; in this section, we will review those languages that are having the most impact on current practice.

A variety of languages are being used today to specify ontologies. We note six of the more important of these.

Knowledge Interchange Format and Conceptual Graphs (KIF/CG)

The Knowledge Interchange Format (KIF; Genesereth and Fikes 1992; Hayes and Menzel 2001) and Conceptual Graphs (CG; Sowa 2000) are languages designed to support the interchange of knowledge among heterogeneous computer systems. KIF includes a core language that has the expressiveness of first-order logic; its syntax and semantics are those of traditional first-order logic. Most recently, this has been extended to include extensions that allow sorted formulae for the specification of class hierarchies, and the specification of the metatheory of KIF within the language itself. Several inference tools are available for reasoning with KIF/CG (such as the SNARK theorem prover; Stickel, et al. 1994), although these have had limited use outside of the academic community.

Although defined separately, both KIF and CG have equivalent expressiveness and are being standardized together within the International Standards Organization.

DAML+OIL

The DARPA Agent Markup Language (DAML; Hendler and McGuinness 2001) is based on description logic (McGuinness and Patel-Schneider 1998; Broekstra, et al. 2000), which is a specialized formal logic that originated in the KL-ONE system of Brachman and Schmolze (1981). Description logic is a variation of first-order logic that arises from restrictions to support reasoning within class hierarchies and to assure decidability of inference. DAML and another description logic language, the Ontology Inference Layer (OIL; Fensel, et al. 2001), have been merged to create DAML+OIL (DAML 2001).

What distinguishes DAML+OIL from the other ontology representation languages is that it has been primarily designed for the Semantic Web (Broekstra, et al. 2000). It is intended to be compatible with emerging Web standards such as RDFS (Brickley and Guha 2000) to make it easier to use ontologies consistently across the Web. It is currently being proposed as a standard within W3C as the WebOnt project[1]. The charter of this project is to extend current Web languages to allow the specification of classes and subclasses, properties and subproperties (such as RDFS), but which extends these constructs to allow more complex relationships between entities. However, the community developing WebOnt is currently discussing the necessary changes and additions to DAML+OIL and consensus in this area still has to be achived[2].

CycL

The knowledge representation language CycL was developed for the specification of common sense ontologies (Lenat and Guha 1990). CycL incorporates features of first-order logic but also extends it with notions from second-order logic and nonmonotonic reasoning; a formal characterization of CycL has not been published in the research literature. Although a wide variety of software tools have been developed for inference and ontology design, the application of CycL has been restricted to the Cyc system and has not had widespread public use. However, this could change with the release in 2002 of a public version of the upper Cyc ontology.

OKBC

OKBC (Chaudhri, et al. 1998) is an API for knowledge servers designed to enable heterogeneous knowledge representation systems to provide a standard interface for use by client systems. The OKBC specification includes a frame language that is used as an ontology representation language in multiple systems, including Ontolingua (Farquhar, et al. 1996), which is described below. The semantics of the frame language is specified by providing a mapping into first-order logic. That mapping also enables ontologies represented in the frame language to be analyzed and used by theorem provers and other automatic reasoners.

EXPRESS

EXPRESS (Schenk and Wilson 1994) was initially designed to support information modeling, particularly the information required to design, build, and maintain products. Although EXPRESS generalizes earlier approaches such as IDEF1X (Menzel 1997), the major drawback for specifying ontologies for semantic integration is that EXPRESS does not have clear declarative semantics. This makes it difficult to verify ontologies that use EXPRESS and also makes it difficult to determine the consistency of semantic mappings between ontologies. There are also no automated

---

[1] http://www.w3.org/2001/sw/WebOnt/

[2] http://www.aifb.uni-karlsruhe.de/~sst/is/WebOntologyLanguage/

inference tools capable of reasoning with EXPRESS beyond checking for data integrity constraints.

The EXPRESS language has been accepted as an international standard (ISO 10303) and is widely used by other ISO standards, particularly the STEP standard for product data exchange.

Unified Modeling Language (UML)

The primary application of UML (Fowler and Scott 1999) for ontology design is in the specification of class diagrams for object-oriented software. However, UML does not have clearly specified declarative semantics, so it is not possible to determine whether an ontology is consistent or to determine the correctness of semantic mappings between ontologies. More recently, UML has been supplemented with the Object Constraint Language (OCL; Warmer and Kleppe 1999) that is closer to offering a semantics similar to first-order logic, and there is some research (Cranefield and Purvis 1999) on the suitability of OCL for more rigorous ontology representation.

F-Logic

Frame-Logic (Kifer, et al. 1995), like the OKBC frame language, provides a declarative language that captures the structural aspects of object-oriented and frame-based languages (such as complex objects, encapsulation, typing, and inheritance).

Language	**Declarative Semantics**	**Completeness**	**Expressiveness**	**Decidable**?
KIF/CG	Yes	Yes	First-order logic	No
DAML+OIL	Yes	Yes	Description logic	Yes
CycL	Yes	No	Beyond first-order logic	No
OKBC	Yes	No	Restriction of first-order logic	Yes
EXPRESS	No	N/A	N/A	N/A
UML	No	N/A	N/A	N/A
F-Logic	Yes	Yes	Restriction of first-order logic	Yes

Table 2: Ontology representation languages.

Criteria for Choosing Ontology representation Languages

Given this variety of languages, how can ontology designers select the appropriate language for their needs? What properties should a formal language possess if it is to be adequate for specifying ontologies to support semantic integration?

The language should have declarative semantics. This may seem to be stating the obvious given the earlier discussion, but the fact is that ontologies that fall along the informal end of the semantic continuum are written in languages that do not have declarative semantics. Consequently, there is no notion of satisfiability or consistency, and hypothesized semantic mappings cannot even be verified. Semantic integration in such cases becomes a subjective exercise validated only by the opinions of the human designers involved in the integration effort.

The language should be based on a sound and complete logic. A logic is sound if all of the inferences drawn from some theory using some proof procedure are sentences that are satisfied by all models of the theory. A logic is complete if any sentence that is satisfied by all models of a theory can be deduced from the theory using some proof procedure. First-order logic satisfies these properties, as do more exotic logics such as infinitary logic (that allows infinite formulas), modal logic, intuitionistic logic, and relevance logic. However, second-order logic[1] is incomplete—there exist second-order theories for which a sentence that is satisfied in all models of the theory cannot be deduced from the theory using a proof procedure. In such a case, automated inference cannot guarantee that the intended semantics of the ontology are preserved.

The ontology representation language should be expressive enough to capture the intended models corresponding to the human designer's intuitions. This criterion primarily addresses languages that are restrictions of first-order logic. Beyond first-order logic, there is a tradeoff between this criterion and the completeness of the language. There are concepts (e.g. connectedness of graphs, Peano arithmetic) that cannot be defined within first-order logic and require second-order logic. However, since second-order logic is incomplete, automated inference would in general be unable to reason with such concepts.

The decidability[2] of the ontology representation language is another important criterion. One of the motivations for description logics is to provide a knowledge representation language in which inference is guaranteed to be decidable (Horrocks, et al. 1999). First-order logic, on the other hand, is in general not decidable.

---

[1] In first-order logic, we can only quantify over elements in the domain of interest. In second-order logic, we can quantify over relations, functions, and any set of elements in the domain.

[2] A language is decidable if there does exista a universal algorithm to determine in all cases whether or not one sentence is a logical consequence of a set of sentences.

Applying the preceding criteria to the languages in Table 2, we can get a sense of future directions for research in ontology representation languages.

Expressiveness vs. Decidability

The inference required to determine whether or not the terms in two ontologies are semantically equivalent is in general not decidable. One approach is to restrict ontology design to the use of tractable representation languages rather than more expressive languages. This has been one of the primary motivations behind the specification of description logics, and more recently, of DAML+OIL. However, this approach leads to a tradeoff between tractable reasoning and allowing the existence of unintended models—expressive languages can eliminate unintended models, but such languages are in general undecidable. There is currently little research being done on identifying the expressiveness that is required in order to axiomatize the intended models in the problem domains of interest.

We will only be able to resolve the tradeoff between expressiveness and decidability empirically. We currently do not know what expressiveness is required in order to capture the semantic intuitions in the domains of interest. As we will see in the next section, there exist decidable first-order ontologies even though first-order logic is not decidable, and we do not yet have a clear idea of the ontologies that we will need.

The expressiveness of the ontology representation language can only be evaluated empirically. We do not know what expressiveness is required to axiomatize the concepts in ontologies for various domains. Testbed environments will be needed to evaluate the different ontology representation languages with respect to the intended applications.

Convergence to a single language

It is doubtful that there will be convergence to a single ontology representation language. The primary factor in language adoption will be influenced by the intended applications of the ontologies themselves. If the design and reuse of DAML ontologies becomes widespread on the Semantic Web, DAML+OIL will be the dominant ontology representation language for such applications. However, it is uncertain that this would lead to the adoption of DAML+OIL for domains such as manufacturing or logistics if the language is not expressive enough to capture the intended semantics of the relevant concepts. Related to this is the acceptance of ontologies within a particular community or industry sector; for example, so long as the CAD vendors and major automotive manufacturers use the STEP ontology, people will be using the EXPRESS language.

Will we need new languages?

Current ontology representation languages that meet the above criteria are all either equivalent to first-order logic or a restriction of first-order logic. It may be the case that the expressiveness criterion forces us to move beyond these languages. Possibilities include modal languages (to capture concepts such as knowledge, belief, and obligations) or even default logics to capture

concepts based on natural kinds and to enable ontologies to include descriptions of "typical" instances of a class. As with the expressiveness/decidability tradeoff, this can only be determined empirically through testbed implementations.

Translation among Languages

We can consider agent integration as having a syntactic and a semantic component. The semantic component maps concepts in one ontology to concepts that intuitively have the same meaning in another ontology. The syntactic component is concerned with the underlying representation language that is used to specify both the ontologies and the domain theories that use the ontologies. Even if the challenges of specifying semantic mappings between agent ontologies are addressed, there is still the problem of translating between the different ontology representation languages used by the agents (Wilson 1996). Some progress has been made for languages that do have an explicit model theory (such as KIF/CG, DAML+OIL, and CycL). However, for languages such as EXPRESS and UML that do not have an explicit model theory, the problem of translation is much more formidable. There are also problems in translating between two properly formal languages if their underlying model theories are different. Translating between Ontolingua, which is based on KIF, and Slang (Waldinger, et al. 1996), which is founded on category theory, gave rise to uncertainty even when performing the translation manually (Uschold, et al. 1998).

A distinction must also be made between the languages used to specify ontologies and the languages that are used in the implementation of an agent-based system. Regardless of whether the ontologies have been specified using KIF, OKBC, DAML, or CycL, the agents themselves may be using applications that use XML or a variety of database languages. Much of the work that has been done on the integration of heterogeneous information resources (e.g., InfoMaster; Duschka and Genesereth 1997) addresses this relationship between ontologies and applications. Likewise, many of the translator tools supported within the Ontolingua library were actually translators between KIF and the knowledge representation languages used within various artificial intelligence applications, such as LOOM and Prolog (Gruber 1991).

Ontologies for Semantic Integration

A wide variety of ontologies are being developed within industry, government, and academia. Many of these ontologies are being used as domain-specific agent ontologies (Ashlander 2000; Dalianis and Persson 1997; Smith and Becker 1997), often within rather narrow contexts. However, some of these ontologies are capable of supporting integration either as an interlingua ontology or as community ontologies. An overview of ontologies that are being designed within the artificial intelligence and information modelling research communities, together with some of their relevant characteristics, is given in Table 3.

Survey

Process Specification Language (PSL)

The Process Specification Language (PSL; Menzel and Gruninger 2001; Schlenof, et al. 1999; Cutting-Decelle, et al. 2000) has been designed to facilitate correct and complete exchange of process information among manufacturing and business software systems. Included in these applications are scheduling, process modeling, process planning, production planning, simulation, project management, workflow, and business process reengineering[1]. The PSL Ontology is modularly organized into PSL-Core and a partially ordered set of extensions to this core theory. The axioms of PSL-Core were directly incorporated from earlier work with the Process Interchange Format (PIF; Lee, et al. 1998).

PSL is a New Work Item (ISO 18629) within Joint Working Group 8 of Sub-committee 4 Industrial data and Sub-committee 5 Manufacturing integration of Technical committee ISO TC 184, Industrial automation systems and integration. Part 1 of the standard has been accepted as a Committee Draft (ISO18629-1). All theories within the PSL Ontology that are currently being standardized have been verified with respect to the intended semantics of their terminology.

Standard for the Exchange of Product data (STEP)

STEP (ISO 10303) has been standardized within the International Standards Organization to support interoperability among manufacturing product software (such as CAD systems and process planning software) throughout the entire product lifecycle. STEP provides standard data definitions for geometry (wire frame, surfaces and solid models), product identification, product structure, configuration and change management, materials, finite element analysis data, drafting, visual presentation, tolerances, kinematics, electrical properties, and process plans. STEP is currently being implemented in the aerospace, automotive, shipbuilding, building design, and electronics industries.

MANDATE

MANDATE (ISO 15531) is being standardized by the same group as STEP. Specified in EXPRESS, it is primarily concerned with manufacturing resource data.

Workflow Process Description Language (WPDL)

WPDL (Sharp and McDermott 2001) has been designed within the Workflow Management Coalition (Fischer 2001) as an interlingua among workflow software. It is specified using a syntax peculiar to the effort, and its semantics are informal.

---

[1] [Schlenoff et. al. 1999] describes an implementation of translators that used PSL to exchange process information between the process modeling tool ProCap (from KBSI) and the scheduling application ILOG Schedule.

Standard Upper Ontology (SUO)

The Standard Upper Ontology ([Pease et. al. 2001]) is limited to concepts that are generic, abstract and philosophical and therefore are general enough to address a broad range of domain areas. Concepts specific to given domains are not included; however, the SUO provides a structure and a set of general concepts upon which domain ontologies (e.g., medical, financial, engineering, etc.) can be constructed. Application developers can define new data elements in terms of the common generic ontology and thereby gain some degree of interoperability with other conformant systems. The SUO is currently a standards project within IEEE (http://suo.ieee.org); one proposal for an initial set of concepts is discussed in [Niles and Pease 2001].

DAML Ontologies

Ontologies to support the Semantic Web are being developed using the DARPA Agent Markup Language (DAML). A library of approximately 160 ontologies is available at www.daml.org/ontologies/. The most important of these ontologies is DAML-S (McIlraith, et al. 2001), which is an upper ontology for services that includes concepts for profiles, processes, and time. In this context, services refer to Web sites that do not merely provide static information but allow one to effect some action or change in the world, such as the sale of a product or the control of a physical device. Thus the DAML-S ontology must support automatic Web service discovery, invocation, composition, and interoperation.

Cyc

The Cyc ontology (Lenat and Guha 1990; Lenat 1995) has been the basis for the development of an extremely large knowledge base to capture commonsense knowledge. The range of concepts covered is quite broad, ranging from domain-specific knowledge to generic knowledge (such as objects and time). The Cyc ontology is divided into smaller modules known as micro theories that are able to support different ontologies that may be mutually inconsistent.

The Cyc ontology is proprietary, although more recently, the upper levels of its class hierarchy have become public. This upper level ontology was used in the DARPA project on High Performance Knowledge Bases (Cohen, et al. 1999).

Ontolingua

The Knowledge Systems Laboratory at Stanford University created an ontology development environment called Ontolingua (Farquhar, et al. 1996) that provides a library of modular reusable ontologies, such as (Gruber and Olsen 1994; Chaudri, et al. 1998). The ontology engineering tools in Ontolingua are oriented toward the design of ontologies by assembling and extending ontologies that have previously been submitted to the ontology library (Farquhar, et al. 1997). More recently, several tools have become available that more directly support the merging of multiple ontologies in the library (McGuinness, et al. 2000). Ontolingua provides an ontology

representation language that is a combination of the OKBC frame language and KIF.

Dublin Core

The Dublin Core (Weible and Miller 2000) began with the goal of developing extensible, consensus-built metadata standards for search and retrieval across different subject areas. The primary achievement to date is the Dublin Core Metadata Element Set (DCMES), a set of 15 terms for describing resources. These are presented in full at (DCMI 1999). The metadata elements include such things as title, creator (not author), subject, date, type, and format. They are defined using the existing standard for describing data elements (ISO/IEC 11179) (ISO11179). This effort has evolved beyond a basic element set to embrace new communities and subject areas. The Dublin Core Metadata Initiative has become a home for a broad spectrum of subject experts and metadata practitioners, built on community trust and open consensus building, and motivated by a desire to build a Web of greater coherence. In addition to providing international forums for the development of vendor-neutral vocabularies, the DCMI is promoting the development of tools and infrastructure to support high quality metadata applications on the Internet, including

- a semantic registry to store and search declared meanings and their relationships to other meanings; and
- an open source software repository to provide the tools for creating, editing, managing, and navigating metadata.

TOVE

The TOVE (TOronto Virtual Enterprise) project (Gruninger and Fox 1998; Gruninger 1997) created an integrated suite of ontologies to support enterprise engineering. Since this must be a shared terminology for the enterprise that every application can jointly understand and use, the ontologies span knowledge of activity, time, and causality (Fox, et al. 1995; Fadel, et al. 1994; Kim and Fox 1995; Tham, et al. 1994).

The TOVE ontologies were developed in cooperation with several companies and have been applied to the design and analysis of enterprise models within supply chain management, project management, and business process engineering. In particular (Atefi 1997) discusses the application of the TOVE ontologies to the analysis of customer relationship management processes within IBM Canada. In other work, the ontologies were used to model the supply chain of BHP Steel (Australia) and assist in the construction of supply chain management scenarios.

Enterprise Ontology

The Enterprise Project at the University of Edinburgh (Uschold, et al. 1998) supports an environment for integrating methods and tools for capturing and analyzing key aspects of an enterprise, based on an ontology for enterprise modeling. This ontology (the Enterprise Ontology) has five top-level classes

(Meta-Ontology, Activities and Processes, Organization, Strategy, and Marketing) for integrating the various aspects of an enterprise. The Enterprise Ontology is semi-formal—it provides a glossary of terms expressed in a restricted and structured form of natural language supplemented with a few formal axioms using the Ontolingua representation language. The Enterprise Ontology has been published through the Stanford University KSCl Ontolingua Server.

Lloyd's Register has used the Enterprise Ontology for more effective modeling and re-engineering of business processes for strategic planning. IBM UK intends to exploit the Enterprise Ontology in modeling its own internal organization as well as providing technical input via its Business Modeling Method BSDM (Business Systems Development Method). The Enterprise Ontology is an ongoing source of inspiration for projects both academic and commercial that require models of concepts in this domain. To our knowledge, it is never imported or translated into a target language in full. Rather, it is perused and picked over for ideas and concepts that may be useful in the new context.

Unified Enterprise Modeling Language (UEML)

This is a new project whose goal is to provide a common language suited for enterprise modeling (Kosanke and Nell 1997). It is intended to provide business users with a standard interface to software for enterprise modeling, analysis, and simulation. It also aims to provide a neutral language for enterprise model exchange.

Ontology	Semantic Continuum	Integration Architecture	Language
PSL	Formal semantics for automated inference	Interlingua for manufacturing process software	KIF
SUO	Formal semantics for automated inference	Community upper-level ontologies	KIF
STEP	Informal semantics	Interlingua for manufacturing product software	EXPRESS
Cyc	Formal semantics (although the public upper level is a taxonomy)	Community ontologies for commonsense reasoning	CycL
MANDATE	Informal semantics	Interlingua for manufacturing resource management	EXPRESS
Dublin Core	Implicit semantics	Interlingua for metadata	documentation
Ontolingua	Formal semantics for human	Community ontologies	OKBC + KIF

	processing		
DAML Ontologies	Formal semantics for automated inference	Community ontologies for Semantic Web	DAML+OIL
TOVE	Formal semantics for automated inference	Community ontologies for enterprise integration	KIF
Enterprise Ontology	Formal semantics for human processing	Community ontologies for enterprise integration	KIF
UEML	Informal semantics	Interlingua for enterprise modeling	documentation
OMG	Informal semantics	Community ontologies	UML
ISO 11179	Implicit semantics	Interlingua for metadata	documentation
WPDL	Implicit semantics	Interlingua for workflow software	documentation

Table 3: Summary of ontologies to support integration.

Analysis

However, there are still some shortcomings in current work:

Lack of coordination among ontology efforts

There is no coherent research program to coordinate the design, evaluation, and comparison of ontologies. An exception ot this is the community effort that went into the development of the Process Specification Language and its predecessor, the Process Interchange Format (Lee, et. al. 1998). Two of the best examples of collaborative ontology development (Ontolingua and the DAML ontologies) have been built by independent submission of ontologies by many different groups. Although this is desirable from the perspective of the users (who are building ontologies that they need), the result is typically a set of multiple overlapping ontologies for the same domain, with no rational way of comparing the advantages or disadvantages of each ontology. In the past, more emphasis has been placed on ontology mapping techniques rather than on the design of the ontologies themselves. However, we cannot assume that ontology mapping techniques alone will solve the semantic integration problem. If the agent ontologies are not rich enough to capture the semantic intuitions of the users, then no mapping technique will be able to supply these intuitions.

Ontologies are not being used within integration scenarios

Many agent-based systems are being built without explicit ontologies, making the semantic integration problem even more daunting. In particular, much of the work on ontology negotiation (Bailin 1999) is being developed in the absence of concrete agent ontologies. Algorithms for ontology

mapping are illustrated with toy examples, rather than using ontologies that have already been specified within the research community.

Current research with the Interlingua architecture is being done within the standards community using STEP, MANDATE, and ISO 11179; however, these standards are not based on explicit ontologies and do not provide an explicit semantics for their terminology. Consequently, when errors are discovered within implementations of translators for these standards, people cannot determine whether the problem is with the input files, the translator, or even the standard itself.

Existing ontologies are not being reused

Although the generic ontologies are more reusable, they may have limited utility in a given domain of application. On the other hand, domain-specific ontologies are often too specialized and cannot be reused by other agents. For example, the reuse of ontologies from the Ontolingua library has been limited. This lack of reusability is particularly problematic, since this has always been one of the promises of ontological engineering (Capellades 1999; Uschold, et al. 1998; Pinto 1999).

We need more ontologies!

There is also a lack of direction in terms of the domains that are being selected for ontology development. Ontology design needs to be "pulled" by applications such as logistics and autonomous mobile systems. Within the context of this report, more ontologies must be designed in the domains relevant to warfighter applications.

Nonclassical Ontologies

All ontologies being designed today (except possibly Cyc; Lenat and Guha 1990) are restricted to the classical logics, such as first- and second-order logic. There will also be problems with ontologies for concepts involving defaults (Reiter 1980) and natural kinds (Rosch 1973). In the case of defaults, it will be necessary to develop an entirely different methodology for verifying an ontology. For example, with ontologies that are based upon monotonic logics (e.g., first-order logic, modal logic, temporal logic), a set of axioms can be falsified by providing a single counterexample, that is, some sentence that is consistent with the axioms but which should not be satisfied by any intended model. However, in the case of a default theory, such a counterexample could be considered an exception. The design challenge then becomes one of distinguishing between exceptions and truly different intended semantics.

Decidability

Another issue is based on the distinction between the decidability of a logical language and the decidability of an ontology whose axioms are written using the language. First-order logic is not decidable, which means that there exist ontologies written in that language that are not decidable. However, it does not mean that all ontologies written in first-order logic are undecidable; in

fact, there exist decidable first-order theories, and hence there exist ontologies whose axioms are written in first-order logic, and inference using these theories is decidable. As with the preceding approach, there is a lack of research into the tractability of existing ontologies.

Semantic Markup

As noted above, semantics are an important and critical challenge that must be met to achieve semantic integration on a wide scale, especially in the Semantic Web context. There are many good statistical techniques from the Information Retrieval area for processing information in the absence of explicit semantic specifications, as well as much commercial activity being done by Web portal and search providers. These techniques can do an excellent job of putting semantically similar items in the same bucket. They may not attempt to define the semantics of a bucket, nor is there a fine-grained markup for individual terms or concepts. Much of the research published which uses semantic markup does not address the question of how is it created—typically it is the human expected to create it (Decker, et al. 1999; Jasper and Uschold 2001). Commercial tools (Voquette 2001) are available that support the automatic creation of semantic markup by using a pre-existing ontology.

One promising idea that only applies to newly authored content is described in (Hendler 2001): “markup for free”. A simple example would be if you were creating a PowerPoint presentation and needed some clipart. You select an icon of a computer. If the icon was already placed into a semantic category, then the presentation could automatically be marked up with that category. There are many challenges that must be addressed with this approach, but it deserves more research.

Version 5.0 of Adobe Acrobat includes an important new development in semantic markup: the eXtensible Metadata Platform (XMP). It is heralded as "the first major implementation of the ideas behind the Semantic Web, fully compliant with the specification and procedures developed by the World Wide Web Consortium” (Adobe 2002). It is an XML framework for Adobe products, providing the capability for storing document metadata. Other applications can access the metadata using an open-source license. The metadata is based on RDF and uses Dublin Core tags. It remains to be seen how significant this will be.

Ontology Mapping

It is rather obvious from the semantic integration architectures that ontology mapping plays a critical role—of comparable importance to the design of the ontologies themselves. In fact, much of the research within the ontology community has been devoted to ontology mapping, since the techniques can be applied to arbitrary ontologies. As with the characterization of the semantic integration architectures, different approaches to ontology mapping

can be distinguished by the degree of automation in the generation and testing of proposed semantic mappings.

Theoretical Foundations

In a sense, ontologies reduce the problem of semantic integration to the problem of ontology mapping—the preservation of semantics between agents is equivalent to specifying mappings between their ontologies. Preliminary work in the formal characterization of ontology mapping (Ciociou and Nau 2000; Euzanat 2001; Stuckenschmidt and Visser 2000; Wache, et al. 2001) has focused on generalizing results on relative interpretation from mathematical logic (Enderton 1972). On the one hand, this has the advantage of being general enough to be applied to any logical theory; however, the results are often too weak to be applied directly to current practice in ontology mapping. One notable exception is OntoMorph (Chapulsky 2000), in which the relationship between two specific constructs in different ontology languages is described with a rule that specifies the transformation from one to the other. Also, approaches based on description logic (Calvenese, et al. 1998) are closer to being implemented and tested against concrete ontologies, particularly in the context of the DAML program.

Another weakness of the current theoretical work is that it primarily provides a declarative characterization of semantic mappings and has not yet moved to the analysis of algorithms to generate or test hypothesized mappings. The algorithms that have been proposed for ontology alignment (such as SMART, Noy and Musen 1999; PROMPT, Noy and Musen 2000; Anchor-PROMPT, Noy and Musen 2001; and FCA-Merge, Stumme and Maedche 2001) have not been analyzed from a theoretical perspective (although Klein 2001 does give some informal analysis in the comparison of these various algorithms).

Alternative approaches to ontology mapping are based on the application of category theory (MacLane 1971), in which one specifies classes of objects and the operations among these objects that preserve their properties. Much of this work uses the notion of Information Flow from (Barwise and Seligman 1997) as the framework in which to specify semantic mappings between distributed agents. Of particular interest in this direction is the work of (Kent 2001), which applies category theory to the design of the Standard Upper Ontology. One drawback is that these approaches currently suffer from a lack of implementations; while there are many inference systems for first-order logic that can be used to implement the ideas in (Ciociou and Nau 2000) and (Euzanat 2001), there are few software tools available that support automated reasoning with category theoretic concepts.

Manual Ontology Mapping Tools

We earlier observed that the current state-of-the art is manual mapping among the ontologies that are implemented in various applications. The drawback of this approach, of course, is that it does not support *automated* semantic integration because the agents themselves are not generating any new semantic mappings; however, the same techniques can be applied to the problem of mapping agent ontologies to interlingua ontologies and specifying the translation definitions between agent and interlingua ontologies prior to agent interaction.

One family of approaches to mapping support uses concept repositories such as WordNet (Miller 1995; Fellbaum 1998), Boeing Thesaurus (Clark, et al. 2000) and MetaNet (Hunter 2001; Niles and Pease 2001). The semantics of the concepts in these and other similar resources are informally specified using either English text, or informally defined relationships (e.g., broader-than, narrower-than) between other concepts. The problem can be characterized as follows: There are two or more sets of terms where there is a substantial overlap in semantic content. The terms in one set need to be mapped to terms in the other sets. When such sets of terms are very large or it is too time-consuming for the human to do this, we require computer assistance in the form of mapping support tools. For a given term, the job of the mapping assistant is to find suggested mappings that the user may then confirm. This can work in various ways. One way is to use the concept set as a neutral intermediary between the different sets of terms. For simplicity, let us limit this discussion to the situation where there are only two sets of terms that need to be mapped. Auto-tagging technology is used to relate terms from each term set to one or more terms in the neutral term set which are likely candidates for being mapped to the given term. Suggested mappings are produced by finding terms that map to the same one or more terms in the neutral term set. WordNet could be used this way, as well as any thesaurus or taxonomy. If the neutral term set has good English definitions associated with each term, then it is easier for the human to make decisions.

The hypothesized semantic mappings generated in this way are not guaranteed to be complete, or even correct, but this is why they are semi-automatic—they allow the human designer to focus on the testing and evaluation of the mappings, while the tools propose possible mappings.

The other family of mapping support tools provides interactive environments that exploit the relationships within the ontologies themselves (McGuinness, et al. 2000). Rather than use the superficial names of relations, these tools consider subclass/superclass relation and domain/range of slot values for concepts within the ontologies. Such tools include Chimaera (McGuinness, et al. 2000) and PROMPT (Noy and Musen 2000).

More general approaches provide software tools that support the specification of rules for combining and integrating ontologies. Within SHOE (Heflin and Hendler 2000), for example, terminological differences can be mapped using simple equivalence rules, while scoping rules allow mapping a category in one ontology to the most specific category that subsumes it in the other ontology.

Semi-Automatic Ontology Mapping Algorithms

Semiautomatic approaches move beyond software environments by specifying algorithms that generate proposed mappings between ontologies (PROMPT, Noy and Musen 2000; Anchor-PROMPT, Noy and Musen 2001; FCA-Merge, Stumme and Maedche 2001; OBSERVER, Mena, et al. 1996, Kashyap and Sheth 1996). As with the manual approaches, no mechanisms are provided to test the consistency and correctness of any of the proposed mappings.

OBSERVER is an interesting system in that it uses a description logic-based reasoning engine to automate a kind of merging of two ontologies. It is semi-automatic, in that it requires a human to first specify hyponym (subclass), hypernym (superclass), and synonym links between the two ontologies. The reasoner merges the two, getting rid of redundant links and joining synonym concepts. This is similar to the approach used by Anchor-PROMPT. OBSERVER also is unique in that it is capable of generating arbitrary mapping links on the fly, rather than having to pre-specify them beforehand. Even though this work is a few years old, it seems to be right at the leading edge.

Similar techniques are used in the MetaNet project for generating semantic mappings; however, the representation they use is mainly term-based, whereas OBSERVER uses a richer attribute-based representation.

The models for many ontologies often have the same structure as various classes of graphs. Since semantic integration is based on the isomorphism among the models of the ontologies, graph isomorphism algorithms provide another technique for automatically postulating semantic mappings between ontologies.

Automatic Ontology Mapping

Ontology negotiation requires more automated support for ontology mapping, in which the agents must both generate *and* test hypothesized semantic mappings between ontologies. Consequently, inference systems play a key role in automatic ontology mapping. Within the Interlingua architecture, inference systems are needed to automatically generate semantic mappings directly between the agent ontologies. Within the Ontology Negotiation architecture, inference systems are required to automatically test the consistency of hypothesized semantic mappings. However, there has been little collaboration between the ontology community on the one hand, and the theorem proving and constraint satisfaction communities on the other. There exist a wide range of theorem provers (Stickel, et al. 1994) and constraint satisfaction systems (Marriott and Stuckey 1998), but for the most part they have been used on toy problems or strictly mathematical theories rather than using the axioms from designed and implemented ontologies. The limited application of theorem proving to ontologies has been in ontology verification (which we will examine in detail in the next section); there has been no work in applying theorem provers to the problems of generating semantic mappings in ontology translation.

Given the complexity of generating and testing semantic mappings, heuristic search strategies through the space of possible semantic mappings are required. One approach is to use informal or partially shared ontologies as initial hypotheses or as heuristics that prune the search tree by avoiding known semantic mismatches. Bailin (Bailin 1999; Truszkowski and Bailin 2001) uses algorithms based on constraint relaxation techniques to identify the potential mappings. The work of Yun Peng (Peng, et al. 2002) uses probabilistic reasoning to select the most likely mapping. Finally, there are approaches that approximate the semantic mapping (Ciociou, et al. 2001; Stuckenshmidt and Visser 2001).

Summary

There is no easy way to map at the semantic level. In the short term, the emphasis is likely to be on tool support to enable humans to more quickly and accurately specify pre-defined mappings that are used at runtime to determine what a given term means with respect to a given agent's ontology. In the longer term, work will proceed which will enable agents to semi-automatically determine the meaning of new terms encountered through interactions with other agents.

Supporting the Ontology Lifecycle

As we discussed earlier, building ontologies is difficult, time-consuming, and expensive,. Much research has been done within the ontology community to support the design, acquisition, verification, and deployment of ontologies. We are already beginning to see a proliferation of ontologies, and we will require sophisticated systems for managing them. This will include support tools for the whole ontology lifecycle.

Design Methodologies

There are several methodologies for building ontologies:

- OntoClean (Welty and Gaurino 2001; Gangemi, et al. 2001; Guarino and Welty 2000)
- Competency questions (Gruninger 1996; Gruninger and Fox 1994)
- "Middle-Out" design (Uschold and Gruninger 1996)
- Assembly and extension of multi-use ontology modules (Farquhar, et al. 1997)
- METHONTOLOGY (Gomez-Perez 1998; Fernandez, et al. 1997)

In addition, (Noy and Hafner 1997) gives a review of the methodologies implicit in the design of various ontologies such as Cyc and TOVE. However, there has been little rigorous evaluation of these methodologies beyond an informal summary of their advantages and disadvantages. Are different methodologies better suited to particular domains or applications? Do generic ontologies require different techniques than more domain-specific ones? What is the relationship between design methodologies and the ontology alignment/merging techniques discussed earlier?

There also have been no serious attempts to create all-encompassing methodologies that combine the best features of existing methodologies. One exception to this is an attempt to merge the methodologies used to develop the TOVE ontologies and the Enterprise Ontology (Uschold 1996). This was just a small first step. Much more needs to be done. There is a new project starting up called Quality-based Ontology Development (Horrocks 2001), which will attempt lay firmer theoretical and empirical foundations for work in this area.

More software support for the ontology design methodologies is needed. Early examples are tools from Stottler-Henke and Associates (that implements the informal phases of the methodology in Gruninger and Fox

1994) and OntologyWorks (that automates consistency checking of ontologies once their formal meta-properties have been expressed in the OntoClean methodology of Welty and Guarino 2001).

Ontology Acquisition and Learning

A new area has emerged in the past few years—using automated techniques to build ontologies.

Rather than acquiring the ontology of an agent, preliminary work is being done to support ontology learning by agents as one phase of ontology negotiation (Bailin 2001; Williams and Ren 2001). In this approach, agents automatically acquire the ontologies of other agents by generating example queries and then using inductive inference to build an ontology based on the answers to these queries. Other approaches include the use of text-processing techniques and/or machine learning techniques to automatically generate a 'first draft' of an ontology from a corpus of documents. Techniques include lexical entry and concept extraction, hierarchical concept clustering, dictionary parsing, and association rules. For good recent work in this area, and plenty of further references, see (Velardi, et al. 2001; Maedche and Staab 2001).

In the near-term, many agent systems will be extensions of legacy software, such as planning, scheduling, and logistics management systems. We will need software tools and methodologies to identify and axiomatize the implicit ontologies in existing agent systems.

Ontology Verification

Given the specification of ontologies within a logical language, one aspect of ontology verification rests upon consistency checking of the ontology's axioms using automated inference systems. As such, this task can be done with arbitrary inference engines (such as Snark; Stickel, et al. 1994), although several tools exist to perform more domain-specific checking (such as the tools associated with the Chimaera environment McGuinness, et al. 2000). There are also several tools based on description logics (Shiq and OilEd; Horrocks, et al. 2001) that perform consistency checks on the classification and implicit subsumption relationships within the ontology.

The evaluation and comparison of interlingua and community ontologies revolves around the identification of unintended models (interpretations that are not shared by all participating agents). This suggests an experimental or empirical approach to the evaluation of the ontology in which we attempt to find sentences that are true in all intended models of the ontology's axioms but false in an unintended model of the axioms. Such sentences were called competency questions in (Gruninger and Fox 1994) and (Gruninger 1996). If the ontology designer can find a model of the axioms that does not correspond to an intended model, then he or she has provided a counterexample to the axioms. In response, the designer can either redefine the scope of the class of intended models (i.e., we do not include the behavior within the characterization of the structures) or modify the axioms.

The distinction between intended and unintended models of an ontology is drawn only with respect to a particular domain; testbed implementations are therefore needed in order to evaluate ontologies with respect to their intended applications.

Even if all of the above technologies are used to build ontologies, we still face the challenge of correctly implementing agents that conform to these ontologies. In a sense, this is the converse of ontology design, since we are beginning with the ontology and using it to implement agents whose behavior is constrained by the ontology. For work that is being done to address this problem, see (Uschold, et al. 1998; Barley, et al. 1997), and (Tansley and Hayball 1993).

Assembling Ontologies

Ontology management systems are motivated by the opinion that global, or even enterprise-wide, ontologies may forever remain unrealized. Therefore we must move away from single-ontology views and use multiple (even conflicting) ontologies. In this view, modular ontologies should be built from merging and reuse of existing ontologies, and new or existing ontologies may need to be merged together or integrated with other ontologies.

The ability to support both the specialization of ontologies as well as the existence of mutually inconsistent ontologies is a critical requirement for an ontology management system. For example, the PSL Ontology is organized into PSL-Core (which incorporates the earlier work from the Process Interchange Format project) and a partially ordered set of extensions. These extensions may not be conservative (i.e., they may falsify sentences that were consistent with the Core theory), and the extensions may not be mutually consistent with each other (e.g., one extension may make a commitment to discrete time while another makes a commitment to dense time).

An alternative approach is to exploit the class hierarchy within the ontologies. The Process Handbook (Malone 1994; Lee and Malone 1990) and Process Interchange Format projects (Lee, et al. 1998; Polyak, et al. 1996) were among the first to introduce the notion of partially shared views, which support partial translation between ontologies by identifying the most specific class that is shared between the ontologies. This also supports the integration of generic ontologies with more domain-specific ontologies.

Commercial Tools

Commercial ontology tools are just starting to arrive. Unicorn is focusing on industrial strength ontologies for enterprise usage. Their product, Unicorn Coherence™, in its beta test phase in early 2002, is a platform for authoring full ontological models and applying the models to deriving transformations between disparate data formats (such as relational schema, XML Schema, and APIs). Key features include

- enterprise quality support for access control, version control, and collaboration in authoring ontological models
- ability to evolve a model with guaranteed backward compatibility
- mapping ontology model to relational, XML and LDAP schemata
- automatic export of queries/transformations to map between relational, XML, and LDAP records/documents.

Another product offering is from Sandpiper Software. They have partnered with Rational Software, and are in the process of building a Rational Rose add-in to support ontology modeling. It extends UML/Rose to provide constructs that support visual modeling of ontology elements, essentially equivalent to Ontolingua or OKBC style definitions. It will have an axiom editor that will support a simplified version of KIF. They also plan to export OKBC-compliant knowledge bases out of Rose, as well as to generate XML schemas and other constructs as required by their customers.[1]

Agent Capability Matching

To support self-integration, agents must identify potential collaborating agents and then automatically determine whether semantic integration is possible with these agents. To achieve self-integration, agents must therefore advertise their capabilities and match the capabilities of other agents to achieve goals associated with the required task.

Review

In self-integration, each agent also has some comprehension of its role within the task that the entire set of agents is achieving. There seem to be two approaches to this problem. In the first approach (McIlraith, et al. 2001; McDermott, et al. 2001), agent capabilities are associated with the services that the agent can provide. This implies that there is an underlying problem-solving ontology that is shared among all of the agents; for example, each agent must share the semantics of concepts such as *goal* and *plan*.

In the second approach to self-integration, agent capabilities are associated with the inference problems that the agents perform (Ivezic, et al. 2000). For example, a supply chain planning agent may be characterized as generating the process plans that are necessary to manufacture some product, while a scheduling agent may be characterized by the satisfaction of temporal and resources constraints to achieve some goal with a given set of process plans. Integrating these agents would require the scheduling agent to know that the planning agent can provide the process plans. This approach is most often seen in domains where agents are "wrapped" around legacy software within an enterprise. Note that this is also distinct from the preceding approach since constraint satisfaction is an inference mechanism and not a plan that is achieved by the actions of the agent.

Work on self-describing agents is primarily being done in the context of the Semantic Web (Sycara, et al. 1998, 1999; McDermott, et al. 2001). Sycara has proposed a new language LARKS (Language for Advertisement and Request of Knowledge Services) to enable matching of agent capabilities; further, ontologies are specified in a specially designed concept language (ITL). McDermott's work is based on DAML+OIL, together with concepts from the Planning Domain Definition Language.

[1] Personal communication with Elisa Kendall, Chairman & CEO of Sandpiper Software.

Summary

In some ways, agent capability matching can be considered to be an application of ontologies. For example, this approach presumes that there is a common shared understanding of *capabilities* themselves—are they specified as a set of functions, goals, or some other constraint on intended behavior? The capture of intuitions about capabilities may require the use of ontologies for processes, services, goals, obligations, and problem solving in general. To this extent, this work will depend on the ontologies that are being designed within the research community.

Risks

The challenge of semantic integration is to support seamless exchange of data among computer systems that preserves the intended semantics of the communicating intelligent agents. If this were achieved, then software would perform exactly as designed and there would be no risks. However, we will not achieve *complete* semantic integration on a large scale even within the long-term. We must therefore examine the risks associated with incomplete integration.

With incomplete semantic integration, we cannot guarantee that the exchange of information among agents preserves the intended semantics of the agents involved. In concrete terms, this can lead to scenarios with two kinds of problems.

1. The agents cannot understand each other, and no information is exchanged.

2. Two agents misunderstand each other, so that one agent uses a different set of constraints for inference than the original agent intended.

The risk in the first scenario is a breakdown in communication that requires the intervention of humans to resolve semantic conflicts. This is, of course, undesirable in dynamic environments, since manual mapping is required and no communication can take place before the mapping between the agents' ontologies has been specified.

The second scenario has more dangerous consequences, since an incorrect semantic mapping leads to unintended behavior. The problems that arise in this case require a cost analysis – what is the cost of a communication breakdown arising from incomplete semantic integration? Within the context of the warfighter, unintended behavior may include

- friendly fire and the mistaken shooting of civilian targets (when different operational units have different definitions of the concept of "target");

- breakdown of logistics plans that arise from different definitions of delivery dates or resource requirements. For example, consider a materiel distribution network and suppose that one agent understands *delivery date* to be the date on which the supplier delivers the order, while the supply agent understands *delivery date* to be the date that they ship the product. If the shipping delays arising from these unintended models are not too costly, then there is no need to change the ontology.

- lack of coordination in battle plans among members of a joint/coalition operation.

An actual case of a problem arising from a lack of semantic integration occurred when NASA lost its Mars Pathfinder probe in 2000 because of the

incompatible measurement ontologies used by two different teams within the mission.

Realistically, we can only minimize the risks associated with incorrect semantic mappings and unintended interpretations of the agents' ontologies. This tradeoff between the incompleteness of semantic integration and the consequences of incompleteness is a major research challenge.

Forecast

The problem facing the development of networks of semantically integrated agent communities is to achieve integration in the face of pervasive semantic heterogeneity (e.g., different languages, different terms, different conceptualizations). We have to accept that there will never be global standards on all these things. Therefore, key research areas are semantic mapping, translation, and interoperability. However, because the advantages of standards are so great, the goal should be to work toward homogeneity wherever it is possible to obtain and to develop mapping and translation capabilities where heterogeneity remains.

We envision that progress in semantic integration will proceed by moving along the semantic continuum described earlier. Initially, we need to make a large number of simplifying assumptions (e.g., about homogeneity). Then they should be relaxed, one by one, moving ever closer to the goal of a semantically integrated Web. A wide variety of challenge problems should be major drivers for choosing which assumptions to relax and thus in which areas to make progress first.

In particular, immediate applications of ontologies will occur in areas that use only implicit semantics, such as many existing Web-based applications. For example, travel and bookseller agents automatically access Web pages looking for good deals. We claim that

the **less** the following things are true:

- *there is widespread agreement about the meaning of a term, and the syntax for expressing it, and*
- *all the software is built by humans who correctly embed the agreed meaning of the term, and*
- *all the databases and Web pages and applications use the term in the agreed way,*

then the **more** necessary it is to

- *have an explicit formal declarative semantics of the term that the machine can process to interpret the meaning of that term.*

The import of this is that we should direct our attention to identifying those circumstances where it is going to be worth the effort to represent the semantics of terms. For example, most of the proposed applications of the Semantic Web being discussed in the literature do not emphasize or discuss any need for using inference; there are some exceptions (Decker, et al.

1999), but there is much work to be done to meet performance and scale requirements of the World Wide Web[1].

Because of the risks involved with incomplete semantic integration, many of the requirements for automated semantic integration that arise from the Warfighter scenarios will ultimately require more formal semantics to guarantee the correctness of semantic mappings between agent ontologies. On the other hand, in the near term, we will see more pragmatic approaches that will rely heavily on informal semantics.

Within five years, we will begin to see networks of semantically integrated communities in domains in which there is the most economic incentive to support reuse and sharing of ontologies. This will likely be a hybrid of the semantic integration architectures. There will be growing amounts of standardization that locally removes semantic heterogeneity for the participants. The community of ontologies approach will have the most impact in the medium term, linking up the various local groups' standard ontologies. Interlingua approaches using formally axiomatized ontologies will begin to be applied to non-trivial domains in research environments.

In the long term, networks of semantically integrated communities will become more widespread. A thorough understanding of the benefits and tradeoffs of the different architectures and the different levels of semantic richness will be achieved. Complete semantic integration will be implemented for a small but important segment of the semantic integration market, perhaps 5-10%. This could be more, if there are major breakthroughs in inference technology and in ontology development and verification technology.

Near Term

Standardization of ontology representation languages

It is doubtful that there will be convergence to a single ontology representation language. However, ontology development will likely be dominated by a handful of languages, in particular, DAML+OIL (WebOnt), KIF/CG, and possibly UML. All three of these languages are being standardized within different international organizations. Once standardized, translators will be developed within the academic and industrial research communities to allow the sharing of ontologies between the logical languages KIF/CG and DAML+OIL (WebOnt). Since UML does not have a formal declarative semantics, translators between UML and the logical languages will be more difficult and probably will not be developed within the near term.

Translators between ontology representation languages and the various knowledge representation and information modeling languages that are used

[1] For further discussion of inference on the Semantic Web, see [Jasper & Tyler 2001].

to implement agents will be more difficult to build. In the near term, translators such as these will probably not be sophisticated at all: they won't be complete or sound except in fairly simple instances, and at best will be semiautomatic.

Agents are implemented using explicit ontologies

In current practice, most agents are still being designed and implemented without explicit ontologies[1]. In the near term, designers will increasingly either extend existing ontologies or develop their own ontologies while they are implementing their agents. This will happen most often in projects that integrate teams from multiple problem domains (such as supply chain management).

Development of interlingua ontologies

One interlingua ontology (the Process Specification Languages) is already being developed for industrial applications. However, it has only been implemented on limited integration problems. In the near term, ontologies such as PSL will be deployed in larger integration problems that span the entire supply chain both within an enterprise and also between enterprises.

There are also several new interlingua ontologies that are at an early stage of development (SUO and UEML). It is premature to determine the applicability and success of these projects on industrial practice.

Development of loosely aligned community ontologies

There are already several examples of community ontologies (the DAML ontologies and Ontolingua). One drawback has been that these communities are too loosely aligned so that reuse and sharing has been hampered. Near-term developments in ontology design and mapping will support better alignment of ontologies within the communities.

The only ontologies to have much commercial impact in the short term may be lightweight 'ontologies,' such as the Yahoo! subject taxonomy and de facto standard lexicons used by industry consortia, such as e-STEEL (esteel 2000).

Implementation of testbeds for evaluating ontologies and mapping methods

There are several critical issues in semantic integration that can only be solved by empirical approaches. These include the expressiveness/decidability tradeoff for ontology representation languages, , the evaluation of different mapping techniques, and determining whether the lack of ontology reuse is due to superficial or deep ontological commitments. We will see the establishment of academic and industrial testbeds that consist of multiple agents and ontologies within each of the

[1] Note that Cougaar [Cougaar 2001] agents have an explicit ontology, but it isn't expressed in a formal way.

integration architectures discussed earlier in this report. Participants will carry out experiments within these environments to test the critical issues presented above.

Use of semiautomatic tools for ontology mapping

Tools for ontology mapping have already been developed within the academic community. We will see the application of these tools to industrial problems, particularly in the growth of semantic communities and in areas such as electronic commerce, where a wide array of different ontologies is needed.

Ontology Management Systems

In the next five years, we will see the commercial deployment of integrated environments to support ontology design and verification. In particular, such systems will have much the same functionality as collaborative computer-assisted software engineering tools, such as version management.

Application of theorem proving and constraint satisfaction techniques to ontology verification

There are many automated inference tools available that perform theorem proving and constraint satisfaction. As more sophisticated ontologies are developed, we will see increasing application of these tools to support ontology mapping as well as ontology verification.

Semantic Markup

In the near term, we will see learning techniques merging with traditional statistics-based techniques for significant scale semi-automatic semantic markup for light- and medium-weight ontologies. Information extraction techniques will be available to automatically add semantic markup to text documents representing significant portions of the document's information content. Authoring tools will do semantic markup for free, with little or no author input.

Midterm

Widespread sharing of ontologies

Experience gained from the deployment of interlingua ontologies and the development of loosely aligned community ontologies will lead to the sharing of ontologies, much the way we see the sharing of code within the open source community today. Another medium for the dissemination of ontologies will be through the standards communities if ontologies in certain domains actually become international standards. A mitigating factor is the degree to which proprietary ontologies (such as Cyc) will play a role in semantic integration.

Alignment of existing interlingua ontologies

The development of interlingua ontologies will initially be restricted to particular domains, such as processes, products, and resources. This will create islands of integrated systems that will require alignment for overlapping concepts among the different interlingua. This alignment will not be the merging of standards, but merely the specification of semantic mappings between overlapping concepts in different interlingua ontologies. We will also begin to see multi-hub approaches being seriously tested in research environments.

Emergence of ontology-based standards within industry

Industry standards are developed through consensus; although this would seem to benefit from the application of ontologies, standards are not currently developed in conjunction with explicit ontologies. While short-term developments will see the "retro-fitting" of ontologies onto existing standards, in the midterm we will see standards being built from the ontologies themselves.

Semiautomatic integration via Ontology Negotiation

The Ontology Negotiation architecture will in general be too difficult to achieve. However, heuristics for ontology mapping will be widely used to assist in the implementations of this approach. Agents will automatically generate semantic mappings, but humans will still be needed to determine that the mappings are correct. Manual mapping will be increasingly replaced by other architectures

Computer-assisted Ontological Engineering

We will see the emergence of sophisticated user interfaces for ontology development; this will enable end-users (rather than experts in artificial intelligence and mathematical logic) to directly design new ontologies and modify existing ones to suit their purposes.

Open Source Ontology Development

The widespread sharing and reuse of ontologies, coupled with computer-assisted ontological engineering tools, will give rise to a large body of ontology developers. This will alleviate the expense of building ontologies and reduce the time it takes to develop an ontology throughout its lifecycle.

Integration of Legacy Systems

In the next five years, we will see the application of ontologies to the problem of integrating legacy systems, such as enterprise resource management, logistics, and operational planning. Initially, this will occur through the Manual Mapping architecture, particularly in domains where no acceptable interlingua ontologies exist.

Semantic Markup

Within five years, we will see the large-scale deployment of semi-automatic techniques and small-scale deployment of fully automatic techniques in

limited situations. Work will begin on sophisticated tools for semi-automatic markup with formal ontologies.

Self-describing Agents

We will see the appearance of relatively simple agents that are able to fully specify their capabilities and advertise capabilities over the Semantic Web.

Long Term

Coordinated development of new ontologies that are aligned with current ontologies

Through the application of ontology mapping tools, alignment will occur as ontologies are designed, rather than as an afterthought.

Limited self-integration of agents via the Interlingua and Community architectures

Self-integration will occur within industry sectors, where agents will be wrapped around legacy systems such as enterprise resource planning, supply chain management, and business process engineering. The interlingua ontologies will have been developed from earlier ontology-based standards and will be restricted to particular domains such as processes, products, and resources.

Limited self-integration of agents via the Ontology Negotiation architecture

Within this architecture, self-integration will be in restricted academic contexts or using simple ontologies

Ontologies that incorporate defaults and natural kinds

Although there has been much research in nonmonotonic reasoning over the past 25 years, it has not been applied to ontology design. In the next 5 to 10 years, we will see ontologies that include commonsense concepts (such as natural kinds) that are axiomatized using defaults and probabilistic knowledge representation.

We will also begin to see ontologies that include modal concepts, such as knowledge, belief, and obligations. This will require extensions to ontology representation languages to incorporate concepts that use "nonclassical" axioms. Any such efforts must first address the extremely severe computational problems and paradoxes that arise with these modalities.

Ontology Learning

In the long term, we will see the application of machine learning algorithms to the problem of ontology acquisition. In particular, initial versions of ontologies will be developed automatically from restricted natural language texts (such as technical manuals).

Forecast Tables

Technology element	Near term 2001-2003	Midterm 2004-2006	Long term 2007-2010
· Semantic Integration Technologies			
Ontology representation Languages	· Standardization of ontology representation languages · Availability of commercial translators between major languages	· Small number of well-established standard languages to suit variety of requirements.	· Extensions to languages to incorporate nonclassical and nonmonotonic logics
Ontologies for Semantic Integration	· Agents are implemented using explicit ontologies · Merging of domain-specific ontologies to create generic community ontologies	· Widespread sharing of ontologies · Emergence of ontology-based standards within industry. · Small scale success of complete semantic integration.	· Coordinated development of new ontologies · Ontologies that incorporate defaults and natural kinds. · Commercially significant deployment of complete semantic integration for niche applications.
Semantic Markup	· Ontology learning techniques merging with traditional statistics-based techniques for significant scale semi-automatic semantic markup for light and medium-weight ontologies. · Authoring tools do semantic markup for free, with little or no author input.	· Large-scale deployment of semi-automatic techniques, and small-scale deployment of fully automatic techniques in limited situations. · Early work in sophisticated tools for semi-automatic markup with formal ontologies.	· Semi-automatic semantic markup via ontology learning
Ontology Mapping	· Use of semi-automatic tools for ontology mapping	· Semi-automatic integration via the Ontology Negotiation Architecture	· Primitive self-integration via Ontology Negotiation architecture for simple ontologies

Table 4: *Forecast table for semantic integration (Part 1 of 2).*

Technology Element	Near Term 2001-2003	Midterm 2004-2006	Long term 2007-2010
Ontology Lifecycle	· Implementation of testbeds for evaluating ontologies and mapping methods · Commercially available ontology management systems · Application of theorem proving and constraint satisfaction to ontology verification	· Computer-assisted ontology engineering · Open source ontology development. · Widespread use of research tools providing limited support for complete semantic integration.	· Limited commercial support tools for complete semantic integration. · Ontology learning in restricted technical domains
Agent Capability Matching	· Harmonization of languages for ontology representation and capability description	· Self-describing agents	· Simple self-integrating agents for limited Web services
Semantic Integration Architectures	· Deployment of complete interlingua ontologies · Development of loosely aligned community ontologies	· Alignment of existing interlingua ontologies · Community ontologies support semiautomatic alignment · Integration of legacy systems	· Limited self-integration via Interlingua and Community architectures

Table 5. *Forecast table for semantic integration (Part 2 of 2)*

Summary and Recommendations

There are two theoretical extremes for environments in which intelligent agents operate. In the first case there is *complete global agreement* on terms and their meaning. In this case, issues in semantic integration do not arise; there is a single shared ontology, which need not even be explicit. At the other extreme, there is no agreement at all; we have *complete semantic anarchy*. There are many good social, economic, technical, and empirical reasons why we should never expect to achieve the former. People are reluctant to give up their familiar terms and concepts and vendors are reluctant to give up their proprietary formats. On the other hand, the costs of semantic heterogeneity to companies and society are prohibitive. The key to progress will be to move away from single ontology views, and to achieve semantic integration with multiple, and possibly conflicting, ontologies.

Consequently, we believe that there are two key elements of a research strategy that will take place in the coming decade. First, the formation of semantically homogeneous communities can only be achieved by addressing the balance between the need for agreement and standardization, on the one hand, and the fact that different people and groups will have different needs on the other. Within the group that abides by an agreement (i.e., by committing to a single ontology), the problems of heterogeneity do not arise. If inter-operation is necessary between groups who cannot agree, then semantic mapping will be required.

The second major element will address the balance between using semantically lightweight representations versus semantically rich representations with formal axiomatizations. The tradeoff here is between computational cost and flexibility and powerful reasoning capabilities. We believe that there will always be a wide range of solutions that have more or less agreement and which use lightweight or richer representations. To a large extent it will depend on the type of the problem. Problems that require maximum flexibility and powerful reasoning capabilities will be driving the advance of the semantic integration technology development.

Six major technologies are currently being developed to address the challenges for automated semantic integration:

- formal languages used for specifying ontologies,
- ontologies that are currently available to support semantic integration,
- techniques for generating and testing semantic mappings between ontologies,
- semantic markup,
- software support for the ontology lifecycle, and
- techniques for agent capability matching.

A key factor in the development and widespread deployment of ontologies will be the extent to which ontologies will fulfill the promise of sharability

and reusability. Recent developments in the standardization of ontology representation languages will provide firm foundations for sharing ontologies. To some extent, these challenges will depend on social factors (such as the formation of community ontologies and the adoption of standard interlingua ontologies) as well as technical factors.

There is no easy way to map at the semantic level. In the short term, the emphasis is likely to be on tool support to enable humans to more quickly and accurately specify pre-defined mappings that are used at runtime to determine what a given term means with respect to a given agent's ontology. In the longer term, work will proceed which will enable agents to semi-automatically determine the meaning of new terms encountered through interactions with other agents.

There are several critical issues in semantic integration that can only be solved by empirical approaches. These include the expressiveness/ decidability tradeoff for ontology representation languages, the evaluation of different mapping techniques, and determining whether the lack of ontology reuse is due to superficial or deep ontological commitments. Progress in resolving these issues will only occur with the establishment of testbed environments that consist of multiple agents and ontologies within each of the integration architectures discussed earlier in this report and that allow participants to carry out experiments that test alternative approaches.

Notes

[1] In a recent special issue of IEEE Intelligent Systems on the Semantic Web, leading workers in this even newer field of study also claim that the above definition best characterizes the essence of an ontology (Fensel, et al. 2001).

[1] http://www.w3.org/2001/sw/WebOnt/

[1] In first-order logic, we can only quantify over elements in the domain of interest. In second-order logic, we can quantify over relations, functions, and any set of elements in the domain.

[1] Personal communication with Elisa Kendall, Chairman and CEO of Sandpiper Software.

[1] For further discussion of inference on the Semantic Web, see (Jasper and Tyler 2001).

[1] Note that Cougaar (Cougaar 2001) agents have an explicit ontology, but it isn't expressed in a formal way.

References

Adobe Inc. 2002. A Manager's Introduction to Adobe eXtensible Metadata Platform, The Adobe XML Metadata Framework; http://www.adobe.com/products/xmp/pdfs/whitepaper.pdf

Ashburner, M. 2000. Gene ontology: Tool for the unification of biology. *Nature Genetics* 25:25-29.

Atefi, K. 1997. Formalization of Business Process Re-engineering Heuristics", M.A.Sc. Thesis, Mechanical and Industrial Engineering Dept., University of Toronto.

Bailin, S. 1999. *Ontology Negotiation in a Community of Agent*s. Technical Report KEI-99-01, Knowledge Evolution, Inc.

Barley, M., Clark, P., Williamson, K. and Woods, S. 1997. The Neutral Representation Project, *Ontological Engineering*, AAAI-97 Spring Symposium Series, Stanford.

Barwise, J., Etchemendy, J., Allwein, G. and Barker-Plummer, D. 2000. *Language, Proof, and Logic*. Seven Bridges Press.

Barwise, J. and Seligman, J. 1997. *Information Flow: The Logic of Distributed Systems*. Cambridge University Press.

Berners-Lee, T., Hendler, J. and Lassila, O. 2001. The Semantic Web, Scientific American, May 2001. See http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html

Bernus, P., Nemes, L. and Williams, T.J. 1996. *Architectures for Enterprise Integration.* Chapman and Hall: London.

Brickley, D. and Guha, V.R. 2000. *Resource Description Framework Schema Specification 1.0*. W3C Candidate Recommendation. (http://www.w3.org/TR/rdf-schema)

Broekstra, J., Klein, M., Fensel, D. and Horrocks, I. 2000. Adding formal semantics to the Web: Building on top of RDF Schema. *Proceedings of the ECDL 2000 Workshop on the Semantic Web*.

Calvenese, D., De Giacomo, G., Lenzerini, M., Nardi, D. and Rosati, R. 1998. Description logic framework for information integration. *Proceedings of the Sixth International Conference on Knowledge Representation and Reasoning*, 2-13.

Campbell, A.E. and Schapiro, S.C. 1995. Ontologic Mediation: An Overview. *Proceedings of the IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing*, Menlo Park CA: AAAI Press.

Capellades, M.A. 1999. Assessment of the reusability of ontologies: A practical example, Workshop on *Ontology Management*, AAAI-99, Orlando.

Chapulsky, H. 2000. OntoMorph: A translation system for symbolic knowledge. *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning*, Breckenridge, Colorado.

Chaudri, V.K., Farquhar, A., Fikes, R., Karp, P.D. and Rice, J.P. 1998. OKBC: A programmatic foundation for knowledge base interoperability, *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisoconsin.

[Ciocoiu, M. 2002 *Ontology-based Semantics*, Ph.D. thesis, Department of Computer Science, University of Maryland, College Park.

Ciocoiu, M., Gruninger M. and Nau, D. 2001. Ontologies for integrating engineering applications, *Journal of Computing and Information Science in Engineering,*1:45-60.

Ciociou, M. and Nau, D. 2000. Ontology-based semantics. *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning*, Breckenridge, Colorado, 539-560.

Clark, P., Thompson, J., Holmback, H. and Duncan, L. 2000. Exploiting a Thesaurus-Based Semantic Net for Knowledge-Based Search. In Proc. 12th Conf on Innovative Applications of AI (AAAI/IAAI'2000), pages 988-995.

Cohen, P., Schrag, R., Jones, E., Pease, A., Lin, A., Starr, B., Gunning, D. and Burke, M. 1999. The DARPA High Performance Knowledge Bases Project. *AI Magazine* 19:25-49.

Cohn, A. 2001. Formalizing biospatial knowledge, *Formal Ontology in Information Systems*. Ogunquit, Maine.

Cognitive Agent Architecture (Cougaar) 2001. Open Source Website, http://www.cougaar.org

Cover, R. 1998. XML and Semantic Transparency, The XML Cover Pages http://www.oasis-open.org/cover/xmlAndSemantics.html

Cranefield, S. and Purvis, M. 1999. UML as an ontology modeling language. In *Proceedings of the Workshop on Intelligent Information Integration*, Sixteenth International Joint Conference on Artificial Intelligence.

Cutting-Decelle, A.F., Anumba, C.J., Baldwin, A.N. and Gruninger, M. 2000. Towards a unified specification of construction process information: The PSL approach, in *Product and Process Modelling in Building and Construction*, Steiger-Garcao and Scherer (eds), 199-207.

Daliannis, H. and Persson, F. 1997. Reuse of an ontology in an electrical distribution network domain, *Ontological Engineering*, AAAI-97 Spring Symposium Series, Stanford.

DAML. 2001. See http://www.daml.org/.

DCMI. 2001. Dublin Core Metadata Initiative; DCMI Frequently Asked Questions (FAQ) http://dublincore.org/resources/faq/

DCMI. 1999. Dublin Core Metadata Element Set, Version 1.1: Reference Description. http://dublincore.org/documents/1999/07/02/dces/

Decker, S., Erdmann, M., Fensel, D. and Studer, R. 1999. Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al. (eds.): Semantic Issues in Multimedia Systems. Proceedings of DS-8. Kluwer Academic Publisher, Boston, 351-369. See also: http://ontobroker.aifb.uni-karlsruhe.de/index_ob.html.

Dennett, D. 1989. *The Intentional Stance*. MIT Press.

Duschka, O.M. and Genesereth, M.R. 1997. Infomaster - an information integration tool. In Proceedings of the International Workshop on Intelligent Information Integration. Freiburg, Germany.

Enderton, H. 1972. *A Mathematical Introduction to Logic*. Academic Press.

Erdmann, M. and Studer, R. 2001. How to Structure and Access XML Documents with Ontologies. *DKE* 36(3): 317—335 http://citeseer.nj.nec.com/erdmann00how.html

[esteel 2000] NewView Technologies. http://www.e-steel.com

Euzanat, J. 2001. Towards a principled approach to semantic interoperability. *Workshop on Ontologies and Information Sharing*, AAAI, Seattle.

Fadel, F., Fox, M.S. and Gruninger, M. 1994. A resource ontology for enterprise modelling. Third Workshop on Enabling Technologies-Infrastructures for Collaborative Enterprises, West Virginia University, pp. 117-128.

Farquhar, A., Fikes, R. and Rice, J. 1997. Tools for assembling modular ontologies in Ontolingua, *Proceedings AAAI-97* (Portland), 436-441.

Farquhar, A., Fikes, R. and Rice, J. 1996. The Ontolingua server: A tool for collaborative ontology construction. *Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada.

Fellbaum, C. (ed.) 1998. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge.

Fensel, D. 2001. Ontologies: Dynamic Networks of Formally Represented Meaning" http://www.cs.vu.nl/~dieter/ontologies.pdf (paper) http://www.cs.vu.nl/~dieter/ftp/slides/ibrow.05.2001.pdf (slide presentation)

Fensel, D. 2000. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag.

Fensel, D., Hendler, J., Lieberman, H. and Wahlster, W. (eds.) 2002. *Semantic Web Technology*. MIT Press, Boston, to appear

Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D.L. and Patel-Schneider, P. F. 2001. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems,* 16(2):38-44. http://citeseer.nj.nec.com/fensel01oil.html

Fernandez, M., Gomez-Perez, A. and Juristo, N. 1997. METHONTOLOGY: From ontological art towards ontological engineering, *Ontological Engineering*, AAAI-97 Spring Symposium Series, Stanford.

Fillion, F., Menzel, C., Blinn, T. and Mayer, R. 1995. An Ontology-Based Environment for Enterprise Model Integration. *Proceedings of the IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing*.

Fischer, L. (ed.) 2001. *Workflow Handbook 2001*. Future Strategies Publishing.

Fowler, M. and Scott, K. 1999. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley.

Fox, M.S., Barbuceanu, M. and Gruninger, M. 1995. An Organisation Ontology for Enterprise Modelling: Preliminary Concepts for Linking Structure and Behaviour, *Computers in Industry*, Vol. 29, pp. 123-134.

Gangemi, A, Guarino, N., Masolo, C. and Oltramari, A. 2001. Understanding top-level ontological distinctions. *Workshop on the IEEE Standard Upper Ontology*, IJCAI, Seattle.

Gangemi, A., Pisanello, D. and Steve, G. 1998. Ontology integration: Experiences with medical terminology. In Guarino (ed.) *Formal Ontology in Information Systems*, 163-178. IOS Press.

Genesereth, M.R. and Fikes, R. 1992. *Knowledge Interchange Format 3.0.*Technical Report KSL-92-01, Knowledge Systems Laboratory, Stanford University.

Goldstein, D. and Esterline, A. 1995. Methods for Building Sharable Ontologies. In *Proceedings of the IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing*. Menlo Park CA: AAAI Press.

Gomez-Perez, A. 1998. Knowledge sharing and reuse. In Liebowicz (ed.) *Handbook of Applied Expert Systems*. CRC Press.

Gruber, T.R. 1995. Towards principles for the design of ontologies for knowledge sharing. *International Journal of Human Computer Studies* 43:907-928.

Gruber, T.R. 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition* 5:199-220.

Gruber, T.R. 1991. *Ontolingua: A Mechanism to Support Portable Ontologies*. Knowledge Systems Laboratory, Technical Report KSL-91-66.

Gruber, T.R. and Olsen, G.R. 1994. An ontology for engineering mathematics, *Proceedings of the Third Conference on Knowledge Representation and Reasoning*, 258-269.

Gruninger, M. 1997. Ontologies for Enterprise Engineering, *Enterprise Engineering and Integration: Building International Consensus*, Springer-Verlag.

Gruninger, M. 1996. Designing Generic Ontologies, *Workshop on Ontological Engineerin*g, European Conference on Artificial Intelligence, Budapest.

Gruninger, M. and Fox, M.S. 1998. Enterprise Modelling. *AI Magazine*, AAAI Press

Gruninger, M. and Fox, M.S. 1994. The Role of Competency Questions in Enterprise Engineering. *Benchmarking - Theory and Practic*e. Chapman Press.

Guarino, N., Carrara, M. and Giaretta, P. 1994. An Ontology of Meta-Level Categories. In E. Sandewall and P. Torasso (eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR94*). Morgan Kaufmann, San Mateo, CA: 270-280.

Guarino, N. and Welty, C. 2000. Identity, unity, and individuality: Towards a formal toolkit for ontological analysis. *Proceedings of the Fourteenth European Conference on Artificial Intelligence*, Berlin.

Hayes, P. 1996. *A Catalog of Temporal Theories*. Technical Report UIUC-BI-AI-96-01. Beckmann Institute for Advanced Study and Technology, University of Illinois at Urbana Champaign.

Hayes, P. 1985. The second naïve physics manifesto. In Brachman, P. and Levesque, H. (eds.) *Readings in Knowledge Representation*, Palo Alto: Morgan Kaufmann.

Hayes, P. 1978. The naïve physics manifesto, in Ritchie, D. (ed.) *Expert Systems in the Microelectronics Age*, Edinburgh University Press.

Hayes, P. and Menzel, C. 2001. A semantics for the Knowledge Interchange Format, *Workshop on the IEEE Standard Upper Ontology*, IJCAI, Seattle.

Hefflin, J. and Hendler, J. 2000. Semantic interoperability on the Web, *Proceedings of Extreme Markup Languages 2000.*

Hendler, J. 2001. Agents on the Semantic Web. *IEEE Intelligent Systems,* 16, 2, March/April.

Hendler, J. and McGuinness, D. 2001. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, January.

Horrocks, I. 2001. QUOD: **QU**ality-based **O**ntology **D**evelopment (project proposal) http://www.cs.man.ac.uk/~horrocks/QUOD/

Horrocks, I., Sattler, U. and Tobies, S. 1999. Practical reasoning for expressive description logics. In *Proceedings of the International Conference on Logic Programming and Automated Reasoning*, 161-180.

Hunter, J. 2001. MetaNet—A Metadata Term Thesaurus to Enable Semantic Interoperability Between Metadata Domains. *Journal of Digital information,* 1(8) (February) http://jodi.ecs.soton.ac.uk/Articles/v01/i08/Hunter/

ISO 11179: Specification and Standardization of Data Elements http://www.diffuse.org/meta.html#ISO11179

Ivezic, N., Libes, D. and Nell, J.G. 2000. *A Vision for Self-Integrating Systems: Improving Process Interoperability*. Technical Report NISTIR-317, National Institute of Standards and Technology, Gaithersburg, Maryland.

Jasper, R. and Tyler, A. 2001. The Role of Semantics and Inference in the Semantic Web: A Commercial Challenge. In *Proceedings of the International Semantic Web Working Symposium* held Stanford University, July 30 - August 1. http://www.semanticweb.org/SWWS/program/position/soi-jasper.pdf

Jasper, R. and Uschold, M. 2001. Enabling Task-Centered Knowledge Support through Semantic Metadata. In *Semantic Web Technolog*y, D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster (eds.): , MIT Press, Boston, to appear.

Jasper, R. and Uschold, M. 1999. A Framework for Understanding and Classifying Ontology Applications. In Twelfth Workshop on Knowledge Acquisition Modeling and Management KAW'99,. http://sern.ucalgary.ca/KSI/KAW/KAW99/

Kashyap, V. and Sheth, A. 1996. Semantic Heterogeneity in Global Information System: The Role of Metadata, Context and Ontologies, in Cooperative Information Systems: Current Trends and Directions, M. Papazoglou and G. Schlageter, Eds. London: Academic Press, pp. 139-178.

Kent, R. 2001. A SUO-KIF Formalization for the IFF Category Theory Ontology. *Workshop on the IEEE Standard Upper Ontology*, IJCAI, Seattle.

Kifer, M., Lausen, G. and Wu, J. 1995. Logical foundations of object-oriented and frame-based languages, *Journal of the Association for Computing Machinery*, May.

Kim, H. and Fox, M.S. 1995. An Ontology of Quality for Enterprise Modelling, Fourth Workshop on Enabling Technologies-Infrastructures for Collaborative Enterprises, West Virginia University.

Klein, M. 2001. Combining and relating ontologies: An analysis of problems and solutions, *Workshop on Ontologies and Information Sharing*, AAAI, Seattle.

Kosanke, K. and Nell, J. (eds.) 1997. *Enterprise Engineering and Integration: Building International Consensus*. Springer-Verlag.

Lakoff, G. 1987. *Women, Fire, and Dangerous Things*. University of Chicago Press.

Lee, J., Gruninger, M., Jin, Y., Malone, T., Tate, A. and Yost, G. 1998. The PIF Process Interchange Format and Framework, *Knowledge Engineering Review*, 2:1-30.

Lee, J. and Malone, T. 1990. Partially shared views: A scheme for communicating among groups using different hierarchies, *ACM Transactions on Information Systems*.

Lenat, D.B. 1995. CYC: A large-scale investment in knowledge infrastructure, *Communications of the ACM* 38:33-38.

Lenat, D.B. and Guha, R.V. 1990. *Building Large Knowledge Based Systems: Representation and Inference in the CYC Project*. Addison Wesley.

MacLane, S. 1971. *Categories for the Working Mathemat*ician. Springer-Verlag, New York.

Maedche, A. and Staab, S. 2001. *Learning Ontologies for the Semantic Web*. http://citeseer.nj.nec.com/maedche01learning.html

Mariott, K. and Stuckey, P. 1998. *Programming with Constraints*. MIT Press.

McDermott, D., Burstein, M. and Smith, D. 2001. Overcoming ontology mismatches in transactions with self-describing service agents.

McGuinness, D.L., Fikes, R., Rice, J. and Wilder, S. 2000. An environment for merging and testing large ontologies. *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning*, Breckenridge, Colorado.

McGuinness, D.L. and Patel-Schneider, P.F. 1998. Usability issues in description logic systems, *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin.

McIlraith, S. 2001. DAML-S: Semantic markup for web services.

McIlraith, S., Son, T.C. and Zeng, H. 2001. Semantic web service, *IEEE Intelligent Systems,* 16:46-53.

Mena, E., Kashyap, V., Sheth, A. and Illarramendi, A. 1996. OBSERVER: An approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. In *Proceedings of the 1st IFCIS International Conference on*

Cooperative Information Systems (CoopIS '96), Brussels, Belgium, June. http://citeseer.nj.nec.com/mena96observer.html

Menzel, M. 1997. Modeling method ontologies: A formal foundation for enterprise model integration, *Workshop on Ontological Engineering*, AAAI-97 Spring Symposium, Stanford.

Menzel, C. and Gruninger M. 2001. A formal foundation for process modeling, *Formal Ontology in Information Systems 2001*, Ogunquit, Maine.

Miller, G.A. 1995. WordNet: A lexical database for English, *Communications of the ACM* 38:39-41.

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. and Swartout, W. 1991. Enabling technology for knowledge sharing. *AI Magazine,* 12, 3, 36-56.

Niles, I. and Pease, A. 2001. Origins of the IEEE Standard Upper Ontology, *Formal Ontology in Information Systems 2001*, Ogunquit Maine.

Noy, N. and Hafner, C. 1997. The State of the Art in Ontology Design: A Survey and Comparative Review, *AI Magazine*, 18, 3, 53-74.

Noy, N. and Musen, M. 2001. Anchor-PROMPT: Using non-local context for semantic matching, *Workshop on Ontologies and Information Sharing*, IJCAI-2001, Seattle.

Noy, N. and Musen, M. 2000. PROMPT: Algorithm and tool for automated ontology merging and alignment, Proceedings of AAAI-2000.

Noy, N. and Musen, M. 1999. SMART: Automated support for ontology merging and alignment, *Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*. Banff, Canada.

Ontoprise, Inc. 2001. *Semantics for the Web*. http://www.ontoprise.de/com/

Obrst, L., Wray, R. and Liu, H. 2001. Ontology engineering for ecommerce: A real B2B example *Formal Ontology in Information Systems 2001*, Ogunquit, Maine.

Pearl, J., 1997. *Graphical Models for Probabilistic and Causal Reasoning*. In Gabbay, D.; and Smets, P. (eds.). *Handbook of Defeasible Reasoning and Uncertainty Management Systems*. Kluwer Academic Publishers.

Pease, A., Menzel, C., Uschold, M., and Obrst, L. (eds.) 2001. *Proceedings of the IEEE Standard Upper Ontology Workshop*, IJCAI, Seattle.

Peng, Y., Zou, Y., Luan, X., Ivezic, N., Gruninger, M. and Jones, A. 2002. Semantic resolution for e-commerce, to appear in NASA Workshop on Radical Agent Concepts, Tysons Corner, Virginia, January.

Pinto, H.S. 1999. Towards ontology reuse. *Workshop on Ontology Management*, AAAI-99, Orlando.

Pinto, H.S. and Martins, J.P. 2001a . Ontology integration: How to perform the process, *Workshop on Ontologies and Information Sharing*, IJCAI, Seattle.

Pinto, H.S. and Martins, J.P. 2001b. Revising and extending the Units-of-Measure subontology, *Workshop on the IEEE Standard Upper Ontology*, IJCAI, Seattle.

Polyak, S., Lee, J., Gruninger, M. and Menzel, C. 1996. Applying the Process Interchange Format to a supply chain process interoperability scenario, *Proceedings of Workshop on Ontological Engineering*, ECAI '96. Budapest, Hungary.

Prior, A.N. 1967. Past, Present, and Future. Clarendon Press, Oxford.

Reiter, R. 1980. A logic for default reasoning, *Artificial Intelligence* 13:48-87.

Rosch, E. 1973. Natural categories, *Cognitive Psychology* 4:328-350.

Schlenoff, C., Gruninger, M. and Ciocoiu, M. 1999. The Essence of the Process Specification Language. *Transactions of the Society for Computer Simulation,* 16, 4 (December) 204-216.

Schenk, D.A. and Wilson, P.R. 1994. *Information Modeling the EXPRESS Way*. Oxford University Press.

Sharp, A. and McDermott, P. 2001. *Workflow Modeling: Tools for Process Improvement and Application Development*. Artech House Publishing.

Smith, I., Cohen, P., Bradshaw, J., Greaves, M. and Holmback, H. 1998. Designing Conversation Policies using Joint Intention Theory. In *Proceedings of the Third International Conference on Multi Agent Systems* (ICMAS-98), 3 - 7 July, Paris, France, IEEE Press, pp. 269-276.

Smith, S. and Becker, M. 1997. An Ontology for Constructing Scheduling Systems. *Ontological Engineering*, AAAI-97 Spring Symposium Series, Stanford.

Sowa, J.F. 2000. *Knowledge Representation: Logical, Philosophical and Computational Foundations*. Brooks/Cole.

Stevens, R., Horrocks, I., Goble, C. and Bechhofer, S. 2001. Building a reasonable bioinformatics ontology using OIL, *Workshop on Ontologies and Information Sharing*, IJCAI, Seattle.

Stickel, M., Waldinger, R., Lowry, M., Pressburger, T. and Underwood, I. 1994. Deductive composition of astronomical software from subroutine libraries*, Proceedings of the Twelfth International Conference on Automated Deduction*, Nancy, France, 341-355.

Stuckenschmidt, H., Wache, H., Vogele, T. and Visser, U. 2000. Enabling Technologies for Interoperability. In eds. U. Visser and H. Pundt, *Workshop on the 14th International Symposium of Computer Science for Environmental Protection,* pp. 35—46. Bonn, Germany. TZI, University of Bremen. http://citeseer.nj.nec.com/459286.html

Stuckenschmidt, H. and Visser, U. 2000. Semantic Translation Based on Approximate Reclassification. In *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning*, Breckenridge, Colorado.

Stumme, G. and Maedche, A. 2001. Ontology merging for federated ontologies on the semantic web, *Workshop on Ontologies and Information Sharing*, IJCAI-2001, Seattle.

Sycara, K., Klusch, M., Widoff, S. and Lu, J. 1999. Dynamic Service Matchmaking Among Agents in Open Information Environments. *ACM Special Interests Group on Management of Data,* 28, 1, 47-53, March.

Sycara, K., Lu, J. and Klusch, M. 1998. *Interoperability Among Heterogeneous Software Agents on the Internet*. Technical Report CMU-RI-TR-98-22, Robotics Institute, Carnegie Mellon University.

Tham, D., Fox, M.S. and Gruninger, M. 1994. A Cost Ontology for Enterprise Modelling. *Third Workshop on Enabling Technologies-Infrastructures for Collaborative Enterprises,* West Virginia University.

Truszkowski, W. and Bailin, S. 2001. Ontology Negotiation Between Strange Agents. *Fifth International Conference on Autonomous Agents*, Montreal.

Uschold, M. 2001a. Barriers to Effective Agent Communication. In *Proceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents*. Montreal, Canada, May 29. (http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-52/oas01-uschold.pdf)

Uschold, M. 2001b. *Where is the Semantics on the Semantic Web?* Invited presentation at the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents. Montreal, Canada, May 29. (http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-52/oas01-uschold-presentation.pdf)

Uschold, M. 2000. Creating, integrating and maintaining local and global ontologies. In *Applications of Ontologies and Problem-Solving Methods*. ECAI, August 20-25.

Uschold, M. 1996. Building ontologies: Towards a unified methodology. In *16th Annual Conf. of the British Computer Society Specialist Group on Expert Systems*. Cambridge, UK. http://citeseer.nj.nec.com/uschold96building.html

Uschold, M. and Gruninger, M. 1996. Ontologies: Principles, Methods, and Applications. *Knowledge Engineering Review*, 1:96-137.

Uschold, M., Healy, M., Williamson, K., Clark, P. and Woods, S. 1998. Ontology Reuse and Application. In *Proceedings of the International Conference on Formal Ontology and Information Systems* - FOIS'98 (Frontiers in AI and Applications v46), pp. 179-192, Ed: N. Guarino. Amsterdam: IOS Press.

Uschold, M., Jasper, R. and Clark, P. 1999. Three Approaches for Knowledge Sharing: A Comparative Analysis. In *Proc. 12th Workshop on Knowledge Acquisition, Modeling, and Management* (KAW'99).

Uschold, M., King, M., Moralee, S. and Zorgios, Y. 1998. The Enterprise Ontology. *The Knowledge Engineering Review,* Vol. 13, Special Issue on Putting Ontologies to Use.

Velardi, P., Missikoff, M. and Fabriani, P. 2001. Using Text Processing Techniques to Automatically Enrich a Domain Ontology. In *Proceedings of the Second Conference on Formal Ontology in Information Systems* (FOIS-01), October, Maine.

Voquette Inc. 2001. *The Power of Relevant Information*. Technical White Paper. Available from: http://www.voquette.com/ContentEnhancement.jsp

Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H. and Huebner, S. 2001. Ontology-Based Integration of Information - A Survey of Existing Approaches. In *Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing,* Seattle, WA, August 4-5 (pp 108-118). http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-47/wache.pdf

Waldinger, R., Srinivas, Y.V., Goldberg, A. and Jullig, R. 1996. *Specware Language Manual*.

Warmer, J. and Kleppe, A. 1999. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley.

Weibel, S. and Miller, E. 2000. *An Introduction to Dublin Core*. http://www.xml.com/pub/a/2000/10/25/dublincore/

World Wide Web Consortium. 2001. *Semantic Web Activity Statement*. http://www.w3.org/2001/sw/Activity

World Wide Web Consortium. 1999. *Resource Description Framework (RDF) Schema Specification*. Eds. D. Brickly and R. Guha. http://www.w3.org/TR/1998/WD-rdf-schema/

Welty, C. and Guarino, N. 2001. Evaluating Ontological Decisions with OntoClean. To appear in *Communications of the ACM*.

West, M. and Fowler, J. 1996. *Developing High Quality Data Models*. Available from http://www.stepcom.ncl.ac.uk.

Williams, A. and Ren, Z. 2001. Agents Teaching Agents to Share Meaning. *Fifth International Conference on Autonomous Agents*.

Wilson, P. 1996. *On the Translation of KIF/Frame Ontologies to EXPRESS*. NISTIR 5927, National Institute of Standards and Technology, Gaithersburg, MD.

Software Agents for the Warfighter
Part 2: Technology Components

Mobile Agents

Brief Overview

Description

Mobile Agents are programs that, with varying degrees of autonomy, can move between hosts across a network. Mobile agents combine the notions of mobile code, mobile computation, and mobile state. They are location aware and can move to new network locations through explicit mobility operations. The technologies surveyed in this chapter include mobile-agent and related middleware, as well as coordination mechanisms for loosely coupled applications, which allow mobile agents to handle the *physical* mobility of devices more effectively.

Mobile Agents are relevant to warfighter scenarios for several reasons. First, mobile agents can relocate themselves during execution, an essential property for reactive and adaptive systems that must respond to changing execution environments. As soon as a change in operating conditions is detected, a mobile-agent application can reconfigure itself to relocate its computations away from a physical attack on the network or closer to a critical database after a drop in network bandwidth. Mobility also applies to the application's data itself, enabling new styles of pro-active applications where system wide actions have to be performed even before any clients are around. An example is the pre-caching of maps to remote geographical locations to ensure instant access to essential information once warfighters enter that region.

Mobile Agents also support disconnected operations. In a mobile system, a client can send agents to a server before disconnecting. The agents can then perform their task while the client is unreachable and communicate results back whenever connectivity is regained. Similarly, a server might send a component to a handheld or other portable device to further reduce connectivity requirements.

Even if the network is stable, mobility still allows bandwidth conservation and latency reduction. For example, if a client application needs to perform a complex multi-step query, it can send the query code to the network location of the databases, avoiding the transmission of intermediate results across the network. Although the database developers could add a database operation that performed the complex query, it is unreasonable to expect that developers can predict and address *every* client need in advance. Mobile agents allow a client application to make efficient use of network resources even when the available services expose low-level, application-independent interfaces.

In all four cases – changing network conditions, disconnected operation, bandwidth conservation and latency reduction – the common thread is dynamic deployment and reconfiguration. Traditional programming languages constrain designers to commit to a particular system structure at build time. The choice whether a particular service is implemented on the client or server side must be made early and can not be revisited if some of the initial assumptions about the application turn out to be invalid. Mobile agents, on the other hand, decouple system design from system deployment, and turn control over deployment to the applications themselves, allowing much more flexible design patterns. An application can deploy its components to the most attractive network locations and redeploy those components when network conditions change, leading to more efficient use of available resources and faster completion times.

Mobility does raise several technical issues, the most important of which is security. Mobile-agent systems have dual security requirements, the need to secure the infrastructure from rogue agents, and the need to protect agents from compromised hosts. Commercially available systems still do not provide sufficient security guarantees to protect the infrastructure from all attacks, this chapter will describe some promising technologies for addressing this problem.

Relevance to the Warfighter

Mobile agent technologies provide capabilities that can significantly enhance military systems. The following sub-sections describe potential applications of mobile agents in various warfighter-relevant scenarios.

Advanced Sensor Grids

Mobile agents can migrate to key locations in a network of autonomous sensors and then filter the collected sensor data to reduce bandwidth requirements and implement local management policies. Mobile agents are particularly useful when it is not possible to pre-install stationary agents on all of the sensors. In particular, the filtering algorithm, which can be of arbitrary complexity, may depend on the phenomenon being observed and change with time. For example, different filtering agents can be used to achieve different levels of accuracy. For high-end sensors, onboard processing is possible, and agents can be sent to the sensors themselves; in other cases, agents can be sent to routers or gateway machines within and at the edge of the sensor field. The agents can be relocated on the fly, allowing the sensor application to optimize the placement of its management and filtering code with respect to current network loads.

By using automated monitoring algorithms, mobile agents can act as customized monitors that wait for phenomena of interest to be observed at distributed sensor locations and then notify human operators. Such a capability can significantly reduce the workload of human operators.

In addition to acting as on-line filters, mobile agents can also help perform data mining on off-line sensor data. In particular, mobile agents can efficiently correlate information across multiple sensors in order to classify observed phenomenon to provide actionable information to warfighters.

Unmanned Autonomous Systems

Mobile agents allow the dynamic configuration of the software deployed on unmanned autonomous systems. New capabilities such as new algorithms, missions, and functions that were not anticipated when the system was originally designed and deployed can be dynamically deployed while vehicles and sensors are in the field. This functionality is already used in outer space exploration projects where communication latencies and physical inaccessibility to devices require mobile-agent like approaches. Fully embracing mobile agent architectures will enable finer granularity and improve ease of use. Mobile agent technologies further allow exchange of functions between platforms, providing a very efficient and localized software update mechanism. Department of Defense programs such as Future Combat Systems are exploring the design of hardware vehicles that support reconfiguration in the field. Mobile agents, taking advantage of mobile code, provide reconfiguration at the software level, complementing the flexibility provided by hardware reconfiguration.

Certain kinds of autonomous vehicles such as undersea vehicles lose network connectivity while submerged. The capability of mobile agents to support disconnected operation is important under these circumstances.

Space exploration vehicles such as the Mars rover present a variation of the network disconnection problem – extremely long network latencies. Mobile agents, by moving themselves to the remote nodes, overcome latency problems.

Mobile Operations

Mobile operations are characterized by low and intermittent connections as the warfighters may not be able to communicate or may be forced to communicate in short bursts to avoid detection. In this setting, mobile-agent technology eases the task of developing software systems. A mobile agent that is capable of performing some task in its entirety can be sent to or from the warfighters to avoid the need for continuous data transmission. This agent can continue its task even if the network link becomes completely unavailable, either due to mission requirements for "radio" silence or due to physical interference, physical separation, or hardware problems. Moreover, the agent can change its behavior or relocate itself as mission and network conditions change. For example, if radio silence becomes essential, an agent running on a remote sensor platform might relocate itself to a less powerful sensor on one of the warfighters themselves, avoiding all radio emissions at the expense of lower sensor resolution. Finally, the entire communications infrastructure, not just application-level tasks, can be implemented with mobile agents. As conditions change, new communication agents can be distributed to all the warfighters in a particular unit. For example, if radio silence has become important, the unit leader's device might deploy communication agents that compress and buffer all data messages, so that the messages can be sent in short bursts at mission-appropriate times.

Joint/Coalition Operations

Joint/Coalition Operations are characterized by ad-hoc network infrastructures connecting information systems operating in different (computer) languages and with different data formats. The particular policies and requirements of a coalition depend on its members and the mission at hand. It would be difficult, if not impossible, to pre-install code for all

possible mission needs on every computer of every coalition partner. Mobile-agent technology allows dissemination of software components that act as "translators" in order to provide interoperability among the different network infrastructures and data formats of the coalition partners, and to perform the tasks related to the mission at hand. Moreover, in addition to supporting interoperability, mobile agents can be deployed to setup flexible information feeds between coalition partners and to enforce policies regarding information release.

Mobility is essential to the rapid deployment of the needed functionality across the large coalition infrastructure. Without mobility, the mission could be significantly delayed as computer personnel installed the needed software in more traditional (and less automated) ways. Moreover, mobility allows the deployment of new capabilities as the mission evolves, not just the deployment of initial functionality.

Rapidly Formed Mission Teams

Similarly, any rapidly formed mission team, whether it is under the control of a coalition or a single nation's military, might not have all needed software pre-installed on their portable devices. Mobile agents allow the rapid deployment of the necessary software components to the warfighters and eliminate the need for a separate and time-consuming software preparation phase. New software components can be deployed as mission conditions change.

Logistics and Other "No Downtime" Systems

Many large-scale distributed systems, such as the logistics systems that integrate military units and commercial suppliers, must be able to evolve, but cannot tolerate downtime. Mobile agents provide a technology for dynamically upgrading such distributed systems with new procedures without interrupting their operation. For example, a logistics system can be upgraded without interrupting the supply chain. Furthermore, mobile agents can provide additional fault tolerance, since an agent can be duplicated at any point of its computation and stored on secondary storage or sent to another platform. For example, a logistics system can create replicates of any software node in the supply hierarchy, according to current network conditions and the changing importance of particular supply sources or sinks.

Mobile-agent technology provides an attractive way to implement distributed monitoring tools that enforce non-local security policies over large-scale networks. Such non-local monitoring can detect distributed denial of service (DDOS) and other large-scale attacks and help pinpoint the source of those attacks. Through the use of mobile agents, this monitoring functionality can be deployed dynamically in response to previous attacks. For example, if a military network is under extensive DDOS attack during a particular mission, the DDOS monitoring code can be dispatched on the fly to a larger number of machines. Similarly, if a military network comes under a previously unknown attack, mobile agents can be implemented and deployed to detect that particular attack while the military mission is still underway. The ability to dynamically distribute existing and new monitoring code makes the military networks more robust to cyber-attack, including attacks that are encountered for the first time.

Information Retrieval and Management

Mobile agents allow bandwidth conservation and latency reduction in many information-retrieval and management applications. For example, in a low-bandwidth environment, a mobile agent that performs a multi-step query against multiple databases can be dispatched close to the location of the databases, avoiding the transmission of intermediate results across the network. Similarly, in an unreliable network environment, the same mobile agent can continue its query task even if the network goes down temporarily. In the reverse direction, code that provides high-level access to a particular database or service can be dynamically dispatched to a warfighter's machine, further reducing the warfighter's reliance on the network. For example, if a warfighter is directly or indirectly making heavy use of a particular database, code to cache query results can be dynamically dispatched to the warfighter's machine. Queries that can be answered from the cached results will never be transmitted across the network, even though the warfighter had no pre-installed query caching capabilities.

Risks

The main risks of mobile-agent technology are the security issues associated with untrusted mobile code and machines. Existing mobile-agent systems do provide significant agent and machine protection, such as encryption for a mobile agent in transit, digital signatures to authenticate the owner and origin of a mobile agent, and "sandboxing" and other restricted execution techniques to prevent a mobile agent from taking destructive action, either maliciously or accidentally. This protection is sufficient to allow focused use of mobile agents in many military applications, particularly non-coalition operations in which all mobile agents originate from inside a single country's military. As of this writing, however, guaranteeing the security of a mobile-agent application running in a mixed network in which some agents and some hosts may be compromised remains an open problem. Integrity of mobile-agent systems may be compromised through a range of cyber-attacks, leading to issues such as denial of service for critical tasks and loss of privileged information. While these risks are serious, we point out that current technologies for Internet programming have essentially the same problems, and thus mobile agents, rather than allowing fundamentally new security exploits, are simply forcing us to confront security issues inherent to wide-area network programming.

Summary and Recommendations

Mobility technologies are a focus of many commercial software development environments, but widespread adoption awaits consensus on communication and mobility standards, as well as assurances of scalability, security, control, and robustness that will be provided by mature community infrastructure capabilities. We recommend focused use of interim non-standardized mobile-agent technologies until commercial standards and infrastructure allow these to be adopted in a more universal fashion. For example, using mobile agents to deploy new functionality from a United States headquarters to a United States warfighting team is efficient and secure with current systems, while building an infrastructure for coalition operations based entirely on mobile agents requires further basic research

and development. Dynamic languages such as Java and C#, which provide dynamic code loading, language-based access control, and introspection, will likely remain the key technical element for building product-grade agent systems, and require further research and development to expand their security, introspection and state-capture features. In addition, it is essential to develop freely available, open source, mobile-agent platforms to support future research in mobility technologies and to become the basis of commercial systems. Open source systems are essential for scientific experimentation and provide opportunities for the kind of external scrutiny that will reveal security weaknesses and, in the end, give rise to mature warfighter mobile-agent standards.

Technical Description

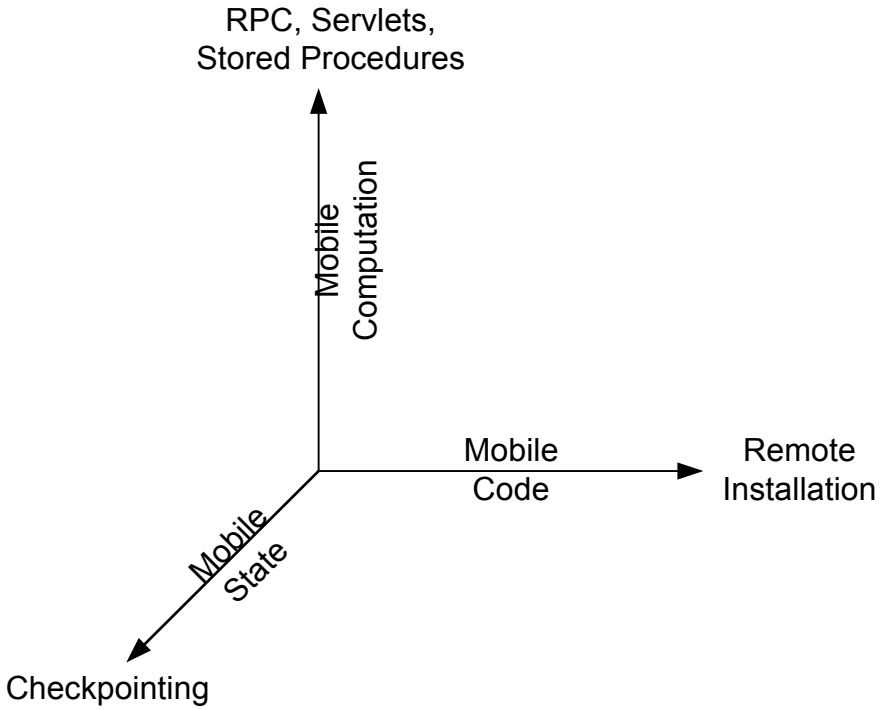
Advances in computer communications and computing power have changed the landscape of computing: computing devices ranging from the smallest embedded sensors to the largest servers are routinely interconnected and must interoperate. Their connections often are set up over untrusted and untrustworthy networks, with limited connectivity and dynamic topologies. The computational capacities of the devices as well as the communications bandwidth between the devices are in a state of constant change and users expect computer systems to dynamically adapt to such changes. Systems should opportunistically take advantage of new resources while at the same time transparently compensating for failures of systems and communication links. Moreover, the characteristics of the applications running on those devices often are quite dynamic, with new software added to the system at run time.

While many of the characteristics of distributed systems have changed, the tools for developing distributed software have not evolved. The majority of distributed programming is still being done in languages and environments that were designed either for uni-processor hardware systems or for static software systems in which the locations and functionality of all clients and servers can be specified a priori. This chapter will look at different approaches to mobile agents, a new paradigm that eases the task of developing modern distributed systems. In addition, the chapter will look at programming languages and middleware designed to support mobile agents, as well as the coordination languages and security mechanisms required by those languages and infrastructures.

Mobile agents are software agents with the additional capability to move between computers across a network connection. By movement, we mean that the running program that constitutes an agent moves from one system to another, taking with the agent the code that constitutes the agent as well as the state information of the agent. The movement of agents may be user-directed or self-directed (i.e. autonomous). In the case of user-directed movement, agents are configured with an itinerary that dictates the movement of the agents. In the case of self-directed movement, agents may move in order to better optimize their operation. Mobility may also be a combination of user- and self-directedness.

Classification of Mobile Agent Capabilities

Mobile agents encompass three basic capabilities: mobile code, mobile computation, and mobile state. These three capabilities are shown in the figure below. Each of the capabilities is an evolution of previously developed capabilities. The following sections describe each capability.



Mobile Computation

Mobile computation involves moving a computation from one system to another. This capability is an evolution of remote computation, which allows a system to tap into the computational resources of another system over a network connection. One of the original mechanisms for remote computation was Remote Procedure Call (RPC). Java Remote Method Invocation (RMI) is another example of remote computation as are servlets and stored procedures.

The difference between mobile and remote computation is that mobile computation supports network disconnection. In a traditional remote computation model, the system requesting the service (the client) must remain connected to the system providing the service (the server) for the duration of the remote computation operation. Additionally, depending on the interface exposed by the server, an interaction can require an arbitrary number of messages between client and server. If network connectivity is lost, the remote computation will become an orphaned computation that will either be terminated or whose results will be discarded. A mobile computation, on the other hand, is an autonomous entity. Once the computation moves from the first system (which may nominally be called the client) to the second system (the server), the computation continues to

execute on the server even if the client becomes disconnected. The agent returns to the client with the results of the computation when (and if) the connectivity is recovered.

Mobile Code

Mobile Code is the ability to move code from one system to another. The code may be either source code that is compiled or interpreted or binary code. Binary code may further be either machine dependent or be some intermediate, machine-independent form.

Mobile code is used in other contexts besides mobile agents. For example, system administrators use mobile code in order to remotely install or upgrade software on client systems. Similarly, a web browser uses mobile code to pull an applet or script to execute as part of a web page.

Code may be mobile in two different ways: push and pull. In the push model, the system sending the code originates the code transfer operation whereas in the pull model, the system receiving the code originates the code transfer operation. An example of the pull model is a Web browser downloading components such as applets or scripts. Remote installation is an example of the push model. Mobile agent systems use the push model of code mobility.

Pull mobility is often considered to be more secure and trustworthy because the host receiving the code is the one that requested the code. Usually, the origin of the request lies in some action carried out by a user of the system and hence pull mobility is superficially more secure. Push mobility on the other hand allows a system to send code to the receiving system at unexpected or unmonitored times. Hence push mobility is less trustworthy from a user's point of view. In practice the overwhelming majority of security exploits encountered in distributed systems originates in careless user actions such as running mail attachments.

Mobile code allows systems to be extremely flexible. New capabilities can be downloaded to systems on the fly thereby dynamically adding features or upgrading existing features. Moreover, if capabilities can be downloaded on demand, temporarily unused capabilities can also be discarded. Swapping capabilities on an as-needed basis allows systems to support small memory constrained devices. Discarding capabilities after use can also help improve system security[1].

Mobile State

Mobile state is an evolution of state capture, which allows the execution state of a process to be captured. State capture has been traditionally used for checkpointing systems to protect against unexpected system failure. In the event of a failure, the execution of a process can be restarted from the last

---

[1] Networks allow system capabilities to be exploited by remote systems. Some researchers suggest that not having capabilities present at all times reduces the opportunities for exploitation. It is not clear that this is a solution to many of the security concerns raised by agent technologies but it is a direction worth evaluating further.

checkpointed state thereby not wasting time by starting from the very beginning. Checkpointing is thus very useful for long-running processes. Operating system research has investigate capturing entire process states, a variant of checkpointing, for load balancing purposes in the early 1980s, but that avenue of research proved to be a dead-end due to coarse granularity of process and semantics problem due to the impossibility of capturing operating system resources such as open file descriptors.

Mobile state allows the movement of the execution state of an agent to another system for continued execution. The key advantage provided by mobile state is that the execution of the agent does not need to restart after the agent moves to a new host. Instead, the execution continues at the very next instruction in the agent.

Not all mobile agent systems provide support for state mobility [Fuggetta, Picco, Vigna 1998]. The term strong mobility is used to describe systems that can capture and move execution state with the agent. Operationally, strong mobility guarantees that all variables will have identical values and the program counter will be at the same position. Weakly mobile-agent systems, on the other hand, usually support the capture of most of a program's data, but restart the program from a predefined program point and thus require some programmer involvement at each migration. The advantage of strong mobility is that the result of migrating is well defined and easier to understand, but its disadvantage is that it is much more complex to implement efficiently. The languages that support strong mobility are Telescript [Whi96], D'Agents, NOMADS, and Ara, while weak mobility is supported by a large number of mobile agent frameworks [Suri, Bradshaw, Breedy, Groth, Hill, Jeffers 2000; Suri, Bradshaw, Breedy, Groth, Hill, Jeffers, Mitrovich, Pouliot, Smith 2000; Baumann, Hohl, Rothermel, Straßer, Mole 1998; Sekiguchi, Masuhara, Yonezawa 1999; Binder 2001; Truyen, Robben, Vanhaute, Coninx, Jossen, Verbaeten 2000 Funfrocken 1998; Bouchenak 1999]. Results by Bettini and De Nicola suggest that strong mobility can be translated into weak mobility without affecting the application semantics [Bettini, De Nicola 2001]. The result is partial as it only works for single-threaded agents. Research is needed to be able to translate multi-threaded strongly mobile agents.

The most important advantage provided by strong mobility is the ability to support external asynchronous migration requests (also known as forced mobility). This allows entities other than the agent (such as other system components, an administrator, or the owner) to request that an agent be moved. Forced mobility is useful for survivability, load-balancing, forced isolation, and replication for fault-tolerance.

Classification of Mobile Systems

The classification of mobile systems due to Picco and Vigna distinguishes three broad approaches to mobility and summarizes the previous description of mobility technologies:

1. *Remote evaluation* – Remote evaluation technologies provide means for an application to invoke services on another node by specifying

the code, as well as the input data, necessary to invoke the service. The code and input data are sent to the remote node, and the remote node then executes the code and sends the output data back to the client.

2. *Code on Demand* – This approach supports software components with dynamically loaded behavior. In this approach, code fragments are requested as they are needed, and dynamically compiled (if needed), verified and linked into a running system.

3. *Mobile Agents* – Mobile agents[2] strengthen code-on-demand with support for moving running computations. Rather than simply moving code (and possibly input data), mobile agents view a computation as a single entity and support the migration of a complete program to another node. This transfer is often seamless, so that the computation can proceed without disruption.

Remote Evaluation

Remote evaluation is doubtlessly the simplest way to achieve mobility. This approach is often used for system administration tasks in which small programs written in a scripting language are submitted to hosts on a secure network. Stamos coined the name [Stamos 1986] to describe a technique where one computer sends another computer a request in the form of a program. The receiving computer executes the program in the request and returns the result back to the sending computer. A number of papers investigated this approach in the early 90's [Stamos, Gifford 1990; Stamos, Gifford 1990b; Segal 1991], but the only noteworthy infrastructure supporting this approach today is the SafeTCL scripting language [Borenstein 1994; OWL97]. The main drawback of scripting languages is that they are not suited to the development of large and reliable software systems because they often lack the basic software engineering features (e.g. encapsulation and data hiding) needed in large systems. Furthermore, the remote evaluation paradigm is confined to classical client/server settings and does not support detached operations well.

Code on Demand

Code on demand is one of the main innovations of Sun Microsystems's Java programming language [GLS97]. This approach allows applications to be delivered piecemeal. The Java execution environment, called a Java Virtual Machine (JVM), is able to find and load any missing components at run time. These components are dynamically linked into the running system, and components that are never needed for a particular application run are never sent across the network, conserving network bandwidth. While dynamic loading is not a novel concept, the idea of allowing potentially untrusted content to be integrated into a running execution environment popularized the concept of 'safe programming languages'. The subsection on security will look in more detail at the requirement for safety. The success of Java

---

[2] The term 'mobile agent' is slightly misleading, as mobility is not restricted to agents, but can be used with any software component or program. Unfortunately, the literature does not differentiate between 'mobile agents' and 'mobile programs'.

owes as much to the safety features that were installed to ensure security as to its dynamic nature. Microsoft's dotNet infrastructure, and in particular the Common Language Runtime provide a similar functionality, but that technology remains, as of this writing, untested and may not yet provide a comparable level of security as Java, though in the long run it is almost certainly going to play a major role. To summarize, the advantage of code-on-demand over remote evaluation is that a language such as Java is a general-purpose programming language with several mature implementations suited for building complex systems. The disadvantage of code on demand approaches is that software is not location-aware; in other words the code running in a Java system can not know where it is located nor is it able to trigger its own migration. Standards such as Java remote method invocation (RMI) extend the pure code-on-demand approach with the means to transfer data along with code under program control; they can thus be used as a basis to implement mobile agent systems but do not provide all of the functionality of a mobile agent system.

Mobile Agents

Mobile software agents improve on previous approaches by bundling code and data into computational entities that can control their own mobility. Mobile agents programs can thus control their own deployment, perform load balancing, and program distributed applications. Mobile-agent infrastructures can be implemented as extensions to code-on-demand systems [Suri, Bradshaw, Breedy, Groth, Hill, Jeffers 2000; Lange, Oshima 1998; Bryce, Vitek 1999; Baumann, Hohl, Rothermel, Straßer, Mole 1998; Sekiguchi, Masuhara, Yonezawa 1999; Binder 2001; Truyen, Robben, Vanhaute, Coninx, Jossen, Verbaeten 2000; Funfrocken 1998; Bouchenak 1999], as extensions to remote evaluation systems, or as new programming languages [White 1997; Tschudin 1994]. The advantage of building on an existing language such as Java is that the existing technology can be leveraged, but this comes at the price of some conceptual complexity, since the system designer must deal with inherent limitations of Java. For example, Java does not provide adequate resource management and process isolation facilities to allow untrusted computations to execute on a trusted machine. Another advantage of mobile-agent systems is that the infrastructure, not the programmer, is in charge of migrating the state and code needed by the computation. Finally, since computations are first-class entities in mobile-agent systems, they naturally can be associated with authority, access rights, and resources.

It is important to realize that, at a basic level, all of these approaches are equally powerful [Puliato, Riccobene, Scarpa 1999]. There is no distributed application that can be implemented with one technology and not any other [Bettini, De Nicola 2001]. Just as we now recognize that high-level programming languages increase programmer productivity, high-level mobility abstractions increase productivity even further. The goal of mobile-agent research is to find programming abstractions that are well suited to the tasks at hand, and provide well-engineered, efficient linguistic constructs to support these abstractions. In the long run, mobile-agent languages must be supplemented with automated tools for reasoning about programs and validating their properties (by static analysis, abstract interpretation or model checking). The goal of research in this arena is not only to provide the

verification technology but also to design languages that are amenable to verification. Just as it is much easier to verify Java code than assembly, it is easier to verify programs that use mobility explicitly than systems in which mobility is implicit. In the long run we expect verification technologies to play an essential role to ensure correctness and provide the kind of security guarantees expected in critical warfighter information systems.

Foundations of Mobility

Understanding the semantics of mobile computation is essential for reasoning about mobile agents. Reasoning about mobility can, in turn, yield guarantees about the correctness of mission critical software. Researchers have explored a number of theoretical models based on process calculi with encouraging results. Two of the approaches that have been studied in this direction are Cardelli and Gordon's ambient calculus [Cardelli, Ghelli, Gordon 2000c] and Vitek and Castagna's Seal calculus [Vitek, Castagna 1999]. These models abstract both:

- *logical mobility* (mobility of programs) and
- *physical mobility* (mobility of nodes, such as handheld devices).

The results obtained thus far include a number of type systems for controlling agent mobility [Cardelli, Gordon 1999; Cgc99; Cardelli, Ghelli, Gordon 2000; Cardelli, Ghelli, Gordon 2000a], and a logic for stating properties of agent programs [Cardelli, Ghelli, Gordon 2000a]. These formalizations have wider applicability as shown by an application of the ambient calculus as a query language for XML [Cardelli, Gordon 2001]. The research on foundations of mobility is actively continuing; in the long run theoretical results can be expected to feed back into languages and infrastructures.

While the theoretical models can accommodate physical mobility, none of the mobile-agent infrastructures currently available provide support for migratory hosts. Host migration and ad-hoc networking are being addressed by work done on coordination languages, which are discussed later.

A major difference between the theoretical models and implementations is the distinction between *strong mobility* and *weak mobility* [Fuggetta, Picco, Vigna 1998]. Most foundational models assume strong mobility whereas most implementations support weak mobility. This may not be a major problem since, as mentioned earlier, strong mobility can be translated into weak mobility, but it will make reasoning about program properties less straightforward.

Requirements Addressed by Mobile Agents

A paradigm is as much defined by the features that are excluded as the features that are included. We now present a list of the key required features as well as some features that we explicitly do not expect mobile agent systems to support. Mobile agents, like most modern distributed systems, have to address five issues that require infrastructure support:

No Global State The physical size of the network and the number of hosts that can participate in a distributed computation must scale to global networks of millions of nodes (of which tens of thousands or more might be participating in a single distributed computation). At this size, there can be no assumption of shared state in the programming model, nor can there be algorithms that require synchronization, or that rely on up to-date information.

Explicit Localities Fluctuations in bandwidth, latency and reliability are so common that they cannot be hidden. Location of resources and of the computation that accesses those resources must be explicit in the programming model.

Restricted Connectivity Failures of machines and communication links can occur without warning or detection. Some of these failures may be temporary as machines may be restarted and connections reestablished, but others may be permanent. Thus, at any time, a computation may communicate with only a subset of the network, or be fully disconnected.

Dynamic Configuration The network topology, both in the physical sense and in the logical sense of available services, changes over time. New hosts and communication links appear with no advance notice, while other hosts disappear and reappear under a new name and address. Applications should be able to adapt to these changes and dynamically reconfigure themselves. Without the ability to dynamically reconfigure its components, applications will have difficulty adapting to changes in locality and connectivity.

Security Since security requirements vary from application to application, basic security mechanisms must be provided by the underlying infrastructure (type safe programming language, checked array access, access control mechanisms) and must be extended with tools for automatic validation of security properties (model checking, program analysis, proof-carrying code).

These five requirements – absence of global state, explicit localities, restricted connectivity, dynamic configuration, and security – drive many of the design decision behind current mobile agent systems. Mobile agent architectures support the above requirements. Mobile agents do not define any notion of shared or global state. Furthermore, mobile agents do not require that host be connected while the agent execute, in fact mobile agents have been repeatedly advocated for disconnected operations (restricted connectivity). Finally, mobile agents, through the use of mobile code, support dynamic configuration. Security, the last issue, remains a challenge that must be addressed through infrastructural tools and services.

Layers of Mobile Code Mobile software agents are just one element of mobile code technology that will be used in a warfighter's computational environment. While mobile agents cover the application layer, we see mobile code also being used for

active middleware (see the section on physical mobility and coordination below) down to active networks and finally software radios and morphable CPUs at the physical layer. Taken individually, each use of mobile code allows customizing a layer's functioning. Taken together, one can start integrating configuration issues across layer boundaries and further optimize a system's overall performance.

Summary

To summarize this portion of the report, we now review some of the main mobile-agent systems and classify them according to the following characteristics: the type of mobility supported by the language or system, the language(s) in which agents are written, whether host mobility (mobile devices) is supported, and the quality, availability, and current level of support of the implementation. The quality of implementation column discriminates products from research prototypes. The availability column indicates which systems can be freely used, and which require a license. Finally, the level of support column indicates whether the project is still active, and whether assistance is forthcoming.

The general conclusions that emerge are that weak mobility is by far the predominant approach. The only commercial strongly mobile system (Telescript) was discontinued several years ago, and the other two strongly mobile system (D'Agents and NOMADS) are university research prototypes. Java is the most popular agent implementation language, due to both the popularity of the language and its support for dynamic loading and advanced security features. Most available systems are research prototypes, but they have the advantage of being open source and thus can be used as a starting point for further development. A number of these projects have an active developer community, an important factor for the adoption of an infrastructure, although it is important to note that most open-source systems do not enjoy the same kind of support as commercial products.

Agent System	Mobility support	Base language	Host mobility	Quality of implement.	Availability	Level of support
Telescript	Mobile agent (strong)	Telescript	No	Product	Discontinued	None
NOMADS	Mobile agent (strong)	Java	No	Prototype	Free	Medium

Java	Code on demand	Java	No	Product	Commercial	High
SafeTCL	Remote evaluation	Tcl	No	Product	Open source	High
D'agents	Mobile agent (strong)	Java, Tcl, Scheme	No	Prototype	Open source	Medium
JavaSeal	Mobile agent (weak)	Java	No	Prototype	Open source	Low
Mole	Mobile agent (weak)	Java	No	Prototype	Open source	Low
Aglets	Mobile agent (weak)	Java	No	Product	Open source	Medium
Lime	Mobile agent (weak)	Java	Yes	Prototype	Open source	Medium
Messenger	Mobile code (weak)	M0	No	Prototype	Open source	Low

Table: Summary of mobile-agent systems.

While the above table is not an exhaustive list of existing mobile-agent systems (over 100 such systems have been implemented in the last five years), it provides a good overview of the most influential systems. The main conclusion to draw is that Java is the common thread in most current mobile-agent implementations. Thus a research emphasis should be directed on evolving mainstream Java implementations to meet the needs of agent systems. The most critical need is additional security features, which will be addressed in the next section, but an important secondary need is support for capturing thread state. Furthermore, since many military scenarios involve limited capacity devices, research on real-time Java should bring benefit to military applications. Another emerging technology is Microsoft's dotNet with the Common Language Runtime. Like Java, dotNet is based on a virtual machine and supports the dynamic loading of programs. The suitability of dotNet to mobile-agent applications has not yet been evaluated, more research is required in that direction as well.

Physical Mobility and Coordination

While mobile agent systems mostly advocate moving code as the only building block for distributed computation, a family of so-called **coordination languages**[3] is emerging as an extension to mobile-agent systems. Coordination languages provide higher-level communication protocols such as generative communication and publish/subscribe models.

While the theoretical models for "general mobility" can accommodate physical mobility, none of the mobile agent system infrastructures currently available provides support for migratory hosts. Host migration and ad hoc networking are being addressed by work done on coordination.

This new research direction on "coordination languages" was started in the late 80's by Gelernter with a programming model named Linda [Carriero, Gelernter 1989]. The common theme underlying most coordination languages is a form of associative communication in which a process can register an interest and another can offer a service. The infrastructure (often called a *tuplespace*) then matches interests with offers. Communication is thus uncoupled since no explicit link needs to be established between communicating partners; a message may be read at any time and by any process interested in it. These properties make it straightforward to provide resource discovery protocols that match up clients with servers based on their respective offers, to program in an event-driven style and to dynamically configure running systems. There is a vital research community focusing on coordination from the programming language aspect with a yearly conference (Coordination), and a growing influence: even Sun Microsystems's Jini technology incorporates a coordination language called JavaSpaces.

Traditional computational models assume that all devices and software components are deployed before an application starts executing, and that once deployed, configurations are static. In the field of wireless computing, however, in which Personal Digital Assistants (PDAs) and other portable devices can establish ad-hoc network connections, these assumptions do not hold. Instead new computational models are needed to ease the task of developing applications for such fluid environments. Mobile agents provide part of the solution since they provide computation mobility, which allows an application to reduce its dependence on an unreliable ad-hoc network. The mobility of devices presents other challenges, however. In particular, current mobile-agent infrastructures do not provide high-level communication primitives suited for physically mobile systems. Experience with a medium-sized mobile-agent application suggests that well over half of the application's code deals with communication in some way. This code is both tedious to write and the source of many of errors.

---

[3] In this context coordination languages provide a programming model for coordination of a number of parallel and distributed tasks. They operate at a much higher level than agent-to-agent interaction languages.

Designing communication mechanisms for mobile environments is a challenging task. Communication in a physically mobile system is:

- *Transient and Opportunistic*: Communication patterns must fit into an environment where hosts are intermittently connected to the network and agents can leave a host at any time. Communication thus tends to be opportunistic, with applications taking advantage of whatever network resources happen to be available at a particular time, but not relying on their continued availability. The underlying communication protocols must accommodate long latencies and/or timeouts caused by the sudden departure of an interlocutor or the disconnection of the agent itself.
- *Unnamed and Untrusted:* Communication in mobile systems is often based on the services being offered, rather than the identity of the entity providing those services. As long as the needed services are provided, agents do not necessarily have to know each other's names and locations to interact. The corollary of anonymity is that interlocutors do not necessarily trust each other, which implies that the communication infrastructure must provide the mechanisms needed to implement secure communication protocols.

Coordination languages describe a family of programming languages and infrastructures that provide communications mechanisms with the two characteristics above. While the early coordination languages were based on centralized systems, a new generation of languages targeted at ad-hoc networks is emerging. The most widely known is Picco and Murphy's Lime language [Picco, Murphy, Roman 1999]. Lime is a decentralized coordination language implemented in Java in which the central tuplespace of Linda is replaced by a multitude of application-specific tuplespaces owned by mobile agents. Whenever two mobile agents are located close to one another, their tuplespaces are seamlessly merged to form a transiently shared data structure [Murphy, Pietro, Roman 2001]. The advantage of systems like Lime is that they provide simple ways to program resource-discovery protocols and other common communication patterns in agent systems. Limitations of the original Lime model have been partially addressed in [Carbunar,Valente,Vitek 2001], but this area still requires attention from both the formal and implementation sides. The most challenging problem is, as usual, how to provide security guarantees for applications using a coordination language in untrusted networks. Once these problems are addressed, however, coordination languages will provide a powerful way for mobile agents to communicate when operating inside ad-hoc wireless networks. Moreover, the mobile agent will contain very little communication code itself, since the necessary (and complex) functionality will be encapsulated inside the coordination-language infrastructure.

Ad hoc networking in combination with mobile code is also investigated in the field of active networking, where active packets, a kind of miniature

mobile software agents, are used to deploy and execute customized routing algorithms [Tschudin 2000].

Security

The main technical, and social, obstacle to approaches based on mobile software agents is security. Not only must researchers devise technical solutions, but also users and organizations must become confident enough in those solutions to permit foreign programs to migrate to and execute on their machines [Farmer, Guttman, Swarup 1996; Tschudin 1999]. If the department in charge of a military database is not convinced of the quality of the security mechanisms, the department will never allow mobile agents to visit the database. Although a mobile-agent application can still function (by having its agents access the database from across the network), the application will use more network bandwidth and suffer higher latencies.

Thus the security and containment of untrusted mobile code, and the objective analysis of proposed security solutions, is a critical research area. When a host receives mobile code, it ideally should evaluate the security implications of executing that particular code, but at the least, it must determine the trustworthiness of the agent's sender (and programmer). Failure to properly contain mobile code may result in serious damage to the host or in the leakage of restricted information. Such damage can be malicious (e.g. espionage or vandalism) or unintentional (programming failures or unexpected interactions with other system components). Other consequences of failing to contain mobile code include denial-of-service attacks and the infiltration of privileged networks through a downloaded Trojan horse or virus [Tschudin 1999; Volpano, Smith 1998; Necula, Lee 1998; Jeager 1999].

The symmetry of mobile-agent security concerns is remarkable as both the agent and the environment in which it executes must be protected from each other. Through purposeful engineering on the part of its developer, an agent may seek to obtain restricted data from the host on which it is running on or damage the host in some way. On the other hand, a host may seek to steal data from or corrupt the agents that migrate to it [Sander, Tschudin 1997]. In the civilian world, a (dishonest) company might gain an economic advantage over a competitor via a malicious agent or host, while in the military world, an adversary might gain a strategic or tactical advantage during an armed conflict.

To relate the security issues to Java programming, we compare an agent to an Applet, a Java program "embedded" inside a Web page and downloaded and executed on a user's machine whenever that user browses the web page. The applet runs within an environment composed of several layers, the first layer is the Java Development Kit (JDK) and its class libraries, the second layer consists of the Java Virtual Machine, the third layer is the operating system, and the last layer is the host device itself. The distinction between these layers is important, since some layers may be easier to subvert than others. For instance, an application may trust a server that belongs to a known organization, but may not trust the libraries found on that server. In Java, this trust mismatch can occur if some of the classes against which an applet is linked have been downloaded from the network [Gong, Mueller,

Prafallchyandra, Schemers 1997; Wallach 1999]. There are two different threats that must be considered when attempting to secure mobile-agent applications:

- **Exogenous threats**: attacks occurring outside of the mobile-agent system. For example, if a host is running both a mobile-agent system and a Web server, an adversary might attack the host via a "standard" Web server exploit, and gain access without every attacking the mobile-agent system itself.
- **Endogenous threats**: threats specific to a mobile-agent system.
  - *Horizontal hostility* (malicious agents): Attacks between agents running on the same host in which an agent tries to disrupt the execution of other co-located agents.
  - *Vertical hostility* (malicious agents & malicious hosts): Attacks against an agent by the execution environment, as well as attacks against the environment by an agent.

In the remainder, we consider only endogenous threats, as they are specific to mobile agents. There are two different viewpoints to take into account:

- For a host, it is necessary to provide protection mechanisms so that agents cannot attack each other (horizontal protection) or the host itself (vertical protection);
- For an agent, it may be necessary to protect it from attacks initiated by the host (hostile host) and other agents (horizontal).

We now consider each issue in turn.

Malicious Agents

A number of techniques have been used in the past to place protection boundaries between so-called "untrusted code" moved to a host and the remainder of the software running on that host. Traditional operating systems use virtual memory to enforce protection between processes [Colusa 1995]. A process cannot read or write another processes' memory and communication between processes requires traps to the kernel. By limiting the traps an untrusted process can invoke, it can be isolated to varying degrees from other processes on the host. However there is usually little point in sending a computation to a host if the computation cannot interact with other computations there, load balancing being the only exception [Malkhi, Reiter, Rubin 1998]. In the context of mobile agent systems, an attractive alternative to operating system protection mechanisms is to use language-based protection mechanisms [Volpano, Smith 1998]. The attraction of language-based protection is twofold: precision of protection and performance. Language-based mechanisms allow access rights to be placed with more precision than traditional virtual-memory systems, and the cost of cross-protection boundaries can often be reduced to zero, since checking is moved from runtime to the language compiler [Jensen, Metayer, Thorn 1998; Goldberg 1998].

Safe Languages. The requirement for language-based security is, first and foremost, safe languages. A safe language is a language that enforces memory safety and type safety. In other words, a safe language does not permit arbitrary memory modifications, and carefully constrains how data of one type is transformed into data of another type. The objective of a safe language is to permit reasoning about security properties of programs at the source level in a compositional manner. It should be possible to check the code with automatic tools to obtain the guarantee that the mobile agent is not malicious. For this to be the true, it is necessary to produce assured implementations of safe languages, that is, implementations that do not contain hidden security vulnerabilities. Well-known examples of safe languages include Java and SafeTCL [Gra97; Gong, Mueller, Prafallchyandra, Schemers 1997]. The Telescript agent language is a case of a safe language explicitly designed for secure mobile code [Tardo, Valenta 1996]. Traditional languages, such as C, do not ensure memory or type safety, and thus, it is much more difficult to obtain trust in agents written in those languages. Even in safe language, there are many opportunities for security exploits. After many years, the research community is closer to producing assured implementations of the Java programming language, but much work remains. The survey by Moreau and Hartel lists several hundreds papers on formalizing aspects of Java [Hartel, Moreau 2001]. The difficulty in obtaining a clear specification of all aspects of Java underscores the need for research in semantics and formal techniques without which there can be no hope of obtaining any assurance.

Sandboxing. Protection against vertical attacks is achieved by enforcing a separation between the user code and the system, a technique popularized by the well-known Java-sandbox security model; related approaches have been used in operating systems [Gong, Mueller, Prafallchyandra, Schemers 1997; Grimm, Bershad 1999; Jones 1999]. In this model user code runs with restricted access right within the same address space as the system code. Security relies on type safety, language access control mechanisms and dynamic checks. Over the years, a number of faults were discovered and fixed in this model [Wallach 1999; Wallach, Balfanz, Dean, Felten 1997; Jensen, Metayer, Thorn 1998; Goldberg 1998]. The sandbox model is a basis for building more powerful security architectures that are suited to agent systems [GMS97]. Sandboxing alone does not provide protection against horizontal attacks. For this, it is necessary to extend the protection model to include protection domains, which constrain how one agent can interact with another. Protection domains can be constructed in a safe language by providing a separate namespace for each component. Fully disjoint namespaces are not desirable as they result in disjoint applications [Malkhi, Reiter, Rubin 1998]. Instead, if mobile agents must interact, some degree of sharing among namespaces is necessary. Several research systems have tried to provide better isolation of Java applications [Von Eicken, Chang, Czajkowski, Hawblitzel 1999; Bryce, Vitek 2002], but these attempts achieved limited success due to the constraint of working above commercial Java Virtual Machines (which allow only certain kinds of extensions to Java's basic security mechanisms). In the future, protection domains must be integrated into the Java Virtual Machine definition.

Denial of Service. Mobile agents can mount denial of service attacks by using an immoderate amount of CPU or memory. An environment for mobile agents therefore must provide support for tracking memory and CPU usage, as well as support for *termination*. Termination implies stopping all threads of an agent and reclaiming its memory. Current Java systems fail to protect the Java Virtual Machine against denial of service attacks, since they support neither resource accounting nor full agent termination. Providing efficient accounting and termination support in a language-based system remains an open research problem.

Beyond Safe Languages. Safe languages must ensure that an agent's code obeys certain well-formedness rules. In the case of Java, this assurance is obtained by verifying the bytecode of incoming agents with a complex data flow analysis algorithm [Goldberg 1998] and by imposing some constraints on how programs may be linked [Jensen, Metayer, Thorn 1998]. A large body of research on proof-carrying code [Necula, Lee 1998] is trying to broaden the set of agent languages to traditional unsafe languages such as C. Proof-carrying code associates a security proof with each program. The host need only check that the proof matches the program to determine whether the program obeys the desired security properties. Checking a proof against a program is computationally much easier than analyzing the code directly to generate the proof, making proof-carrying code an attractive approach. This direction of research is encouraging, as it may allow the expression of complex security properties, and verification of a program's compliance with those properties.

Malicious Hosts

In mobile-agent computing, an agent's owner must be able to trust that it is not subverted when visiting a series of servers, some of which may have been compromised and made capable of malicious action against the agent [Sander, Tschudin 1997]. Malicious servers are a particularly difficult problem, since the server must have access to all of the agent's code in order to execute it. A small body of research has attempted to solve this problem. The solutions fall into the following categories:

1. code signing,
2. replication,
3. partial result authentication codes,
4. trace validation,
5. secure coprocessors, and
6. encrypted functions.

We will assess each approach in the following paragraphs.

Code Signing can be used to protect agents from malicious hosts by attaching digital signatures to the code of the mobile agent. Code signing is being used by Sun and Microsoft to provide guarantees of authenticity for downloaded code [GS98]. The technology can be used to ensure that a server has not altered the code of an agent while in transit. Code signing does not protect the agent's data from being modified, nor does it prevent the server from accessing the information contained in the agent, but it does

provide a basic level of assurance that it is essential for warfighter applications. Furthermore in a network in which servers can not be compromised and agents come from a single source code signing may be the best solution to security. This scenario fits many of the warfighter requirements. With the exception of coalition operations most warfighter applications are expected to rely on code signing.

Replication was studied as a general method for mobile agent computation security, marrying some ideas from the fields of fault tolerance and cryptography [Vogler, Moschgath, Kunkelmann 1997; Roth 1999]. The approach relies on the replication of agents and servers. The same agent computation is performed on several servers. Voting then can be used to move from one phase of a distributed computation to the next. While replication enjoys some pleasing theoretical properties, it is heavily restricted in practice. It supposes that computations are deterministic, and that several servers with the same resources are available. The connectivity assumptions also are not appropriate in warfighter scenarios, making it unlikely that replication can play a large role in warfighter applications.

Partial Result Authentication Codes are very similar to message authentication codes (MAC). Instead of authenticating the origins of a message, however, they authenticate the correctness of an intermediate agent state or partial result. For example, if we consider the values of selected program variables at some point during execution, we can determine whether those values could possibly have arisen from normal program execution. If not, the program has been altered in some way. PRACSs are computationally cheaper than digital signatures and have slightly different security properties (forward integrity): if an agent visits n servers and some server in m ($m<n$) is malicious, the results of servers 1 to m-1 cannot be falsified. In the warfighter scenarios, mobile agents often do not have intermediate results. Nevertheless, this approach can be used to ensure that results of a disconnected query are truthful [Hohl 1997].

Proof Verification is an approach in which a digitally signed trace of an agent's computation is returned along with the result. This trace can then be validated – a malicious host would affect the agent by changing the results and thus producing a trace that does not correspond to a valid computation [Vigna 1998]. Although techniques for producing compact traces have been developed, the size and complexity of the trace remains an issue. It is unlikely that this approach will be practical in warfighter applications.

Code obfuscation aims at protecting a mobile agent's functioning by making it very hard to divert the agent's execution in a meaningful way. This is achieved by transforming the code such that automated reverse engineering cannot be applied. This prevents a malicious host from locating the places in the code that should be modified or that should be executed in a non-conformant way. Also, variables can be split all over the program in order to make simple read-outs impossible. Although there are many tools and also commercial products that use code obfuscation, especially in the field of digital right management, recent theoretical results point at the impossibility to achieve perfect obfuscation [Barak 2001].

Secure Coprocessors involve building a trusted execution environment for agents within a secure coprocessor [Wilhelm, Staamann, Buttyan 1999; Yee 1999]. This approach is based on tamper-proof hardware and public key infrastructures. Some experimental systems have been designed, but not validated. Secure co-processors have the potential of providing appropriate security for mobile-agent programs, but at the cost of upgrading to more expensive hardware. Secure coprocessors will be useful in some warfighter applications, but not in all (or even the majority). However, trusted coprocessors might be the only hard security anchor available today for securing mobile agent applications.

Encrypted functions that can be executed in their encrypted form are a software-only cryptographic approach to the malicious host problem. If available, this would be the ideal way to protect any mobile agent and its payload. The approach of computing with encrypted functions was demonstrated in [Sanders, Tschudin 1998], another system was proposed in [Loureiro 2001]. The conclusion is that for special functions it is possible to let a mobile agent protect itself from a malicious host without having to rely on trusted hardware or on-line help from remote agents. However, the solutions proposed seem to be impractical for today's standards and no implementation has been reported so far.

Summary

Mobile agents systems require extensive research to ensure that agents can execute securely. The majority of the research effort has been devoted to developing secure language-based systems. With the exception of the work of Gray [Gray, Kotz, Cybenko, Rus 1998] and Peine [Peine 1998], mobile-agent systems are single-language environments. The majority of mobile-agent systems today use Java and its security model as a starting point. Java provides adequate security to protect a host from the agents that are running on it, but still lacks mechanisms for protecting agents from one another and for accounting for resource usage and enforcing termination. These technical issues must be addressed to obtain robust mobile-agent systems. The state of research on the protection of agents against malicious hosts does suggests that solutions are not immediately forthcoming, but this problem is one that is shared with any distributed system, as nodes may be compromised and then the results of any request invalidated.

Risks

The main risks associated with mobile-agent technology are related to security. Although existing systems provide sufficient host and agent protection for many applications, as of this writing, it is not possible to guarantee the security of a mobile-agent application running in a mixed network in which some agents and some hosts may be compromised. The integrity of mobile-agent hosts may be compromised through a range of cyber-attacks, leading to issues such as denial of service for critical tasks and loss of privileged information.

Aggravation of risks. Once a mobile-agent host is compromised, malicious mobile agents can be sent easily to any other mobile-agent host, unless

complex security mechanisms are in place. Thus, an organization's own mobile-agent system could become a weapon against it, as an attacker launched a flood of malicious agents from a single compromised host. Up to now, these threats have been mostly theoretical, but a widely deployed mobile-agent system would represent an attractive target and tool for hackers and cyber-terrorists. The threats range from the difficulty to contain information to general service disruption. Furthermore, the existence of a standardized common space where mobile agents can persist over long periods of time also creates risk clusters and potentially enables new forms of long term stealth attacks.

Mitigation of risks. The progress made in the last few years in the fields of computer security and formal verification suggests that automated checking of many security properties is possible. Moreover, other technologies for Internet programming suffer from the same or similar security. Even if mobile agents are not accepted, the same security issues must still be addressed. Mobile code also enables new forms of active and extremely rapid responses in case of attacks on the computer infrastructure, which would not be possible with the current model of static software deployment.

Forecast

Forecast Tables

Our forecasts assume a reasonable level of funding in mobile agents research and in related technologies such as the PCES program from DARPA which supports the development of Real-Time Java (RTJ). RTJ is a key enabling technology for mobile agent technologies in many of the warfighter scenarios.

Technology element	Near term 2001-2003	Midterm 2004-2006	Long term 2007-2010
Mobility Technologies			
Mobile Agents	• Experiments with MA systems with strong mobility deployed in medium scale mixed networks (scale to1K agents). • Development of MA-based management tools for network monitoring and debugging. • Open source MA toolkit built on top of a customizable virtual machine framework allows researchers and companies to experiment with extensions and new implementation technologies. • Research in foundations of MA systems produce the first definition of complex MA language.	• Experiments with interoperability of agents systems developed in different MA languages and systems (scale to 1000K agents). • Development of MA-based network security infrastructure. • Production-grade strong mobility implementations are released. • Embedded, Real-Time MA system implemented based on Real-Time Java. • Program verification technology for MA systems obtains first results on real-life systems.	• Experiments with MA systems supporting physical mobility as well as logical mobility (UAV). • First embedded devices with software entirely composed by mobile and hot-swappable code. • Widespread deployment of MA network management and security infrastructure. • High-efficiency MA systems become widespread, performance equivalent to native, statically compiled, code. • Mobile-agent platforms included in commercial operating systems. Standardization of MA language and interaction protocol.
Coordination languages	• Research prototypes for coordination in small-scale ad-hoc networks with host mobility. • Intelligent resource discovery prototype for wireless devices.	• Adoption of an XML-based coordination language for small devices. • Semantics of coordination languages codified.	Adoption of standardized coordination protocols for wireless devices. • Integration of MA technology with coordination languages.
Mobile agent security	• Resource accounting and termination support. • Verification of agent compliance with complex security properties	• Protection domains and non-interference guarantees widely available.	• Proof-carrying techniques for unsafe languages widely deployed. • Offline policy tools for protecting against denial of service attacks.

Summary and Recommendations

Mobile agent and coordination technologies are a focus of many commercial software development environments, but widespread adoption awaits consensus on communication and mobility standards, as well as assurances of scalability, security, control, and robustness that will be provided by mature community infrastructure capabilities. We recommend focused, limited use of interim non-standardized mobility until commercial standards and infrastructure allow it to be adopted in a more universal fashion. For example, using mobile agents to deploy new functionality from a United States headquarters to a United States warfighting team is efficient and secure with current systems, but building an infrastructure for Coalition operations based entirely on mobile agents must await further research and development.

We make the following recommendations for further investment in research:

- **Dynamic languages:** Dynamic programming languages will likely remain the key technical element for building product-grade agent systems. Developing dynamic languages for embedded and real-time systems will be particularly important for many warfighter applications. Research in foundation and implementation of mobile languages is thus essential to further develop our understanding of the issues and realize the possibilities offered this paradigm. In the long run, the interworking of mobile agents written in different languages and working across mobile code layers should also be studied.
- **Open source platforms:** Freely available infrastructures are essential to obtain reliable and secure mobile technologies. Current proprietary systems are a roadblock to research and development of innovative agent solutions, furthermore they present some serious security risks as their implementation is shrouded in secrecy and has not been subjected to the kind of rigorous scrutiny that has strengthened many open source systems. One of our most important recommendations is to support the development of several open source mobile-agent technologies. These technologies will form the basis for research and commercial products, and the most successful technologies will be standardized.
- **Hands-on Experiences:** Large-scale experiments are required to better separate what is technically feasible from how mobile agents are most useable. Especially the problem of automated management of a mobile agent infrastructure should be investigated, most prominently with a mobile agent approach itself in order to benefit from the extensibility provided by mobile code.

Mobile Agents, continued

- **Security:** Information assurance remains an important issue in mixed networks, if mobile agents are to be deployed in untrusted settings. A commitment to research on a broad spectrum of security techniques for mobile code and their integration is needed to provide the necessary security guarantees. Because mobile code is used at many more places than just mobile agents these results will be of general applicability to wide-area programming.

References

[Stamos, Gifford 1990] J. Stamos and David K. Gifford. Remote evaluation. *ACM Transactions on Programming Languages and Systems*, 12(4):537--565, October 1990.

[Stamos, Gifford 1990b] J. W. Stamos and D. K. Gifford, "*Implementing Remote Evaluation*," IEEE Transactions on Software Engineering, Volume 16, Number 7, July 1990.

[Volpano 1996] D. Volpano. *Provably-secure programming languages for remote evaluation*. ACM Computing Surveys, 28A(2):electronic, December 1996.

[Puliato, Riccobene, Scarpa 1999] B. Puliato, S. Riccobene, and M. Scarpa. *An Analytical Comparison of the ClientSever*, Remote Evaluation, and Mobile Agents Paradigms. In Proc. ASA/MA'99, pages 278-292, October 1999.

[Stamos 1986] J. W. Stamos. Remote Evaluation. PhD thesis, Laboratory for Computer Science, *Massachusetts Institute of Technology*, January 1986. Technical report MIT/LCS/TR-354.

[Segal 1991] M. E. Segal. "*Extending dynamic program updating systems to support distributed systems that communicate via remote evaluation*". In Proc. International Workshop on Configurable Distributed Systems, pages 188-- 199, 1991.

[Borenstein 1994] BORENSTEIN, N. E-*mail with a mind of its own: The SafeTcl language for enabled mail*. In Proceedings of IFIP International Conference (Barcelona, Spain, 1994).

[Ousterhout, Levy, Welch 1997] J. K. Ousterhout, J. Y. Levy, and B. B. Welch. The Safe-Tcl Security Model. *Sun Microsystems Laboratories*, March 1997.

[Funfrocken 1998] Stefan Funfrocken, Transparent Migration of Java-Based Mobile Agents: Capturing and Reestablishing the State of Java Programs, In Proceedings of the Second International Workshop on Mobile Agents, Lecture Notes in Computer Science, No. 1477, Springer-Verlag, pp. 26-37, Stuttgart, September 1998.

[Bouchenak 1999] S. Bouchenak. Pickling threads state in the Java system. Third European Research Seminar on Advances in Distributed Systems (ERSADS'99), Madeira Island, Portugal, April 1999.

[Baumann, Hohl, Rothermel, Straßer, Mole 1998] Baumann J., Hohl F., Rothermel K., Straßer M., Mole - Concepts of a Mobile Agent System, to appear in: WWW Journal, Special issue on Applications and Techniques of Web Agents, 1998.

[Bettini, De Nicola 2001] Lorenzo Bettini and Rocco De Nicola, Translating Strong Mobility into Weak Mobility, In Mobile Agents, 2001.

[Truyen, Robben, Vanhaute, Coninx, Jossen, Verbaeten 2000] Eddy Truyen, Bert Robben, Bart Vanhaute, Tim Coninx, Wouter Joosen, and Pieere Verbaeten. Portable Support for Transparent Thread Migration in Java. In Proceedings of the Joint Symposium on Agent Systems and Applications / Mobile Agents (ASA/MA), pages 29--43, September 2000.

[Sekiguchi, Masuhara, Yonezawa 1999] Tatsurou Sekiguchi, Hidehiko Masuhara, and Akinori Yonezawa. A Simple Extension of Java Language for Controllable Transparent Migration and its Portable Implementation. In Coordination Languages and Models, Lecture Notes in Computer Science, pages 211--226, 1999.

[Binder 2001] W. Binder. Design and implementation of the J-SEAL2 mobile agent kernel. In The 2001 Symposium on Applications and the Internet (SAINT-2001), San Diego, CA, USA, Jan. 2001.

[White 1997] White, J. E. (1997b) Telescript, in Mobile Agents (eds. Cockayne, W. R. and Zyda, M.), Manning Publ., Greenwich, pp. 37--57.

[Lange, Oshima 1998] Lange D. and Oshima M. *Programming and Deploying Java Mobile Agents with Aglets*, Addison Wesley, 1998

[Bryce, Vitek 1999] C. Bryce and J. Vitek. *The javaseal mobile agent kernel*. In D. Milojevic, editor, Proceedings of the 1st International Symposium on Agent Systems and Applications, Third International Symposium on Mobile Agents (ASAMA'99), pages 176--189, Palm Springs, May 9--13, 1999. ACM Press.

[Cardelli, Ghelli, Gordon 2000] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon, Types for the Ambient Calculus,In I&C special issue on TCS'2000.

[Cardelli, Ghelli, Gordon 2000b] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon, Ambient Groups and Mobility Types, Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses, Takayasu Ito (Eds.). Theoretical Computer Science. 2000.

[Cardelli, Ghelli, Gordon 2000a] Luca Cardelli and Andrew D. Gordon, Anytime, Anywhere. Modal Logics for Mobile Ambients. Proceedings of the 27th ACM Symposium on Principles of Programming Languages, 2000. pp 365-377.

[Cardelli, Ghelli, Gordon 1999] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon, Mobility Types for Mobile Ambients, Jiri Wiedermann, Peter van Emde Boas, and Mogens Nielsen, Editors. Automata, Languagese and Programming, 26th International Colloquium, ICALP'99 Proceedings. Lecture Notes in Computer Science, Vol. 1644, Springer, 1999.

[Cardelli, Gordon 1999] Luca Cardelli and Andrew D. Gordon, Types for Mobile Ambients. Proceedings of the 26th ACM Symposium on Principles of Programming Languages, 1999. pp 79-92.

[Cardelli, Ghelli, Gordon 2000c] Luca Cardelli and Andrew D. Gordon, Mobile Ambients. TCS special issue on Coordination, D. Le Métayer, July 2000.

[Cardelli, Gordon 2001] Luca Cardelli, Giorgio Ghelli, A Query Language Based on the Ambient Logic. Programming Languages and Systems, 10th European Symposium on Programming, ESOP 2001.

[Vitek, Castagna 1999] J. Vitek and G. Castagna. *Seal: A framework for secure mobile computations*. In H. E. Bal, B. Belkhouche, and L. Cardelli, editors, Internet Programming Languages, volume 1686 of Lecture Notes in Computer Science. Springer-Verlag, 1999.

[Fuggetta, Picco, Vigna 1998] A. Fuggetta, G. P. Picco, and G. Vigna. *Understanding Code Mobility*. IEEE Trans. on Software Engineering, May 1998.

[Carriero, Gelernter 1989] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444--458, April 1989.

[Picco, Murphy, Roman 1999] G.P. Picco, A.L. Murphy, and G.-C. Roman. *Lime: Linda Meets Mobility*. In D. Garlan, editor, Proceedings of the 21 st International Conference on Software Engineering, May 1999.

[Murphy, Pietro, Roman 2001] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. *Lime: A Middleware for Physical and Logical Mobility*. In Proceedings of the 21 st International Conference on Distributed Computing Systems (ICDCS-21), May 2001.

[Tardo, Valenta 1996] Joseph TARDO and Luis VALENTA. Mobile Agent Security and Telescript, In Proceedings of IEEE COMPCON, February 1996.

[Tschudin 1999] C. F. Tschudin. *Mobile agent security*. In M. Klusch, editor, Intelligent Information Agents. Springer, July 1999.

[Sander, Tschudin 1997] Tomas Sander and Christian F.Tschudin. "*Protecting Mobile Agents Against Malicious Hosts*," LNCS on "Mobile Agent Security", 1997.

[Vogler, Moschgath, Kunkelmann 1997] Vogler H., Moschgath M.-L., Kunkelmann T., *An Approach for Mobile Agent Security and Fault Tolerance using Distributed Transactions*, Darmstadt Univ. of Technology, ITO, to appear in the Proc. of ICPADS'97

[Volpano, Smith 1998] D. Volpano and G. Smith. *Language Issues in Mobile Program Security*. In Giovanni Vigna, editor, Mobile Agent Security, Lecture Notes in Computer Science No. 1419, pages 25--43. Springer-Verlag, 1998.

[Hohl 1997] Fritz Hohl, "*Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts*," LNCS on "Mobile Agent Security", 1997.

[Gray 1996] R. S. Gray, *Agent Tcl: A Flexible and Secure Mobile-Agent System*, Proceedings of the 4th Annual Tcl/Tk Workshop (TCL 96), July 1996, pp. 9-23.

[Necula, Lee 1998] George C. Necula and Peter Lee. Safe, *untrusted agents using using proof-carrying code*. In Giovanni Vigna, editor, Special Issue on Mobile Agent Security, volume 1419 of Lect. Notes in Computer Science, pages 61--91. Springer-Verlag, June 1998.

[Gray, Kotz, Cybenko, Rus 1998] R.Gray, David Kotz, George Cybenko and Daniela Rus. "*Security in a multiple-language mobile-agent system*," In Giovanni Vigna, editor, Lecture Notes in Computer Science: Mobile Agents and Security, 1998.

[Vigna 1998] Vigna J., *Cryptographic Traces for Mobile Agents*, in: Giovanni Vigna (Ed.), Mobile Agent Security, LNCS 1419, 1998, Springer, pp 137-153

[Gong, Schemers, Singing, Sealing 1998] Gong L., Schemers R., Signing, Sealing, *and Guarding Java Objects*, in: Giovanni Vigna (Ed.), Mobile Agent Security, LNCS 1420, 1998, Springer, pp 206-216

[Colusa 1995] Colusa Software. *Omniware: a Universal Substrate for Mobile Code*. White paper, Colusa Software. 1995. http://www.colusa.com.

[Jones 1999] Michael B. Jones. *Interposition agents: Transparently interposing user code at the system interface*. In Secure Internet Programming: Security Issues for Distributed and Mobile Objects (J. Vitek, C. Jensen, Eds.), LNCS 1603, pp. 117-146, Springer, 1999.

[Grimm, Bershad 1999] R. Grimm and B. N. Bershad. *Providing Policy-Neutral and Transparent Access Control in Extensible Systems*. In Secure Internet Programming: Security Issues for Distributed and Mobile Objects (J. Vitek, C. Jensen, Eds.), LNCS 1603, pp. 117-146, Springer, 1999.

[Wilhelm, Staamann, Buttyn 1999] Wilhelm, U.G.; Staamann, S.; Buttyn, L.: *Introducing trusted third parties to the mobile agent paradigm*. In Secure Internet Programming: Security Issues for Distributed and Mobile Objects (J. Vitek, C. Jensen, Eds.), LNCS 1603, pp. 117-146, Springer, 1999.

[Yee 1999] Yee B., *A sanctuarity for mobile agents*, In Secure Internet Programming: Security Issues for Distributed and Mobile Objects (J. Vitek, C. Jensen, Eds.), LNCS 1603, pp. 117-146, Springer, 1999.

[Roth 1999] V. Roth. Mutual protection of cooperating agents. . In Secure Internet Programming: Security Issues for Distributed and Mobile Objects (J. Vitek, C. Jensen, Eds.), LNCS 1603, pp. 117-146, Springer, 1999.

[Jeager 1999] T. Jaeger. *Access control in configurable systems*. In Secure Internet Programming: Security Issues for Distributed and Mobile Objects (J. Vitek, C. Jensen, Eds.), LNCS 1603, pp. 117-146, Springer, 1999.

[Farmer, Guttman, Swarup 1996] W. M. Farmer, J. D. Guttman, and V. Swarup. *Security for mobile agents: Issues and requirements*. In National Information Systems Security Conference. National Institute of Standards and Technology, October 1996.

[Wallach 1999] Dan S. Wallach. *A New Approach to Mobile Code Security*. PhD Thesis, Princeton University. January 1999.

[Wallach, Balfanz, Dean, Felten 1997] D. S. Wallach, D. Balfanz, D. Dean, and E. W. Felten. *Extensible security architectures for Java*. Technical report 546-97, Department of Computer Science, Princeton University, Apr. 1997.

[Gong, Mueller, Prafallchyandra, Schemers 1997] Li GONG, Marianne MUELLER, Hemma PRAFULLCHANDRA AND, Roland SCHEMERS. *Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit* 1.2, In Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California, December 1997.

[McGraw, Felten 1997] G. McGraw and E. W. Felten. *Java Security: Hostile Applets*, Holes, and Antidotes, John Wiley & Sons, 1997.

[Malkhi, Reiter, Rubin 1998] Malkhi, M. K. Reiter, and A. D. Rubin. *Secure execution of Java applets using a remote playground*. In Proc. of the 1998 IEEE Symp. on Security and Privacy, pp. 40--51, Oakland, CA, May 1998.

[Jensen, Metayer, Thorn 1998] T. Jensen, D. Le Metayer, and T. Thorn. *Security and Dynamic Class Loading in Java: A Formalization*. In Proceedings of the 1998 IEEE International Conference on Computer Languages, pages 4--15, May 1998.

[Goldberg 1998] Allen Goldberg. *A specification of java loading and bytecode verification*. In Proceedings of the Fifth ACM Conference on Computer and Communications Security, 1998.

[Hartel, Moreau 2001] P. H. Hartel and L. A. V. Moreau. Formalizing the Safety of Java, the Java Virtual Machine and Java Card. *ACM Computing Surveys*, to appear, 2001.

[Von Eicken, Chang, Czajkowski, Hawblitzel 1999] T. Von Eicken, C.-C. Chang, G. Czajkowski, and C. Hawblitzel. J-*Kernel: A Capability-Based Operating System for Java.* Lecture Notes in Computer Science, 1603:369--394, 1999.

[Bryce, Vitek 2002] C. Bryce and J. Vitek. The JavaSeal mobile agent kernel. *Autonomous Agents and MultiAgent Systems*, 2002.

[Suri, Bradshaw, Breedy, Groth, Hill, Jeffers 2000] Suri, N., Bradshaw, J.M., Breedy, M.R., Groth, P.T., Hill, G.A., and Jeffers, R. 2000. *Strong Mobiling and Fine-Grained Resource Control in NOMADS*. Agent Systems, Mobile Agents, and Applications. LNCS 1882. Berlin: Springer-Verlag.

[Suri, Bradshaw, Breedy, Groth, Hill, Jeffers, Mitrovich, Pouliot, Smith 2000] Suri, N., Bradshaw, J. M., Breedy, M. R., Groth, P. T., Hill, G. A., Jeffers, R., Mitrovich, T. R., Pouliot, B. R., & Smith, D. S. 2000. ***NOMADS****: Toward an environment for strong and safe agent mobility*. Proceedings of Autonomous Agents '2000, Barcelona, Spain, New York: ACM Press.

[Tschudin 1997] Tschudin C. The Messenger Environment M0 – A Condensed Description. In J. Vitek and C. Tschudin (eds): Mobile Object Systems. LNCS 1222, Springer-Verlag.

[Tschudin 1994] Tschudin C. An Introduction to the M0 Messenger System. Technical report 86 (Cahier du CUI), University of Geneva, 1994.

[Loureiro 2001] S. Loureiro, Mobile Code Protection. PhD thesis, January 2001. Sophia Antipolis, France.

[Tschudin 2000] C. Tschudin, H. Lundgren and H. Gulbrandsen: Active Routing for Ad Hoc Networks. IEEE Communications Magazin, Apr 2000

[Barak 2001] Barak B., et al: On the (Im)possibility of Obfuscating. *CRYPTO `01,* vol. 2139 LNCS, pp. 1-18, Santa Barbara, CA, August 19-23, 2001.

Software Agents for the Warfighter
Part 2: Technology Components

Infrastructure for Software Agents

Brief Overview

Description

This chapter describes the infrastructure required to support the lifecycle of software agents. The infrastructure components can be subclassed into two distinct groups: a set of requisite core services and a set of management services and tools. This chapter differs from the agent architecture section – whose focus is on the architecture internal to software agents. In contrast, the focus of this chapter is the architecture of components external to software agents.

There are several research and standardization programs underway to model, standardize and develop infrastructure for agent-based and other distributed systems. These include the DARPA CoABS (Control of Agent-Based Systems) Grid [Global InfoTek, Inc. 2002], the FIPA (Foundation for Intelligent Physical Agents) [Foundation for Intelligent Physical Agents 2002], Java™ Agent Services [McCabe 2002], the Grid Computing / Global Information Grid efforts [Global InfoTek, Inc. 2002], as well as peer-to-peer networking architectures [Barkai 2002, Miller 2001].

The Technical Description section identifies the challenges faced in developing infrastructure for software agents and outlines the broad set of requirements for such an infrastructure. The subsequent technology survey reviews the programs mentioned above.

Relevance to the Warfighter

Unlike the other chapters that describe internal agent technology components, this section does not have a section on the relevance to the warfighter.

Risks

Using infrastructure does not explicitly introduce any risks into software agent systems. Well-designed infrastructure can actually mitigate risks that other agent technologies might introduce.

Forecast

Improvement in scalability is the major factor that we expect to change in the next few years. Scalability is defined in two dimensions with the first dimension being the number of agents that interoperate. We expect that infrastructure can support 10s to 100s of agents now, 1000s to 10000s of agents in a few years, and upto millions of agents in 10 years. The second dimension may be

described as the "breadth" or "spread" of the agents i.e., the number of different organizations, administrative domains, and countries that may be spanned by agents. Currently, few agent systems span multiple organizations and administrative domains and multiple countries. We expect to see a progression where agent systems will truly be global in scope in 10 years.

Summary and Recommendations

Infrastructure needs to address challenges such as security, scalability, robustness/fault tolerance, efficiency, accessibility, and interoperability. While somewhat similar to distributed systems, software agents have a number of characteristics that are unique. Software agent infrastructure needs to address agent specific requirements although it can build upon existing research and implementations of distributed system infrastructure.

Our recommendation in this area is that research should be continued in both the distributed systems infrastructure area as well as the software agents infrastructure area. Key challenges such as scalability, distributed security, administration and management tools, and deployment must be addressed.

Many of the issues discussed in this chapter are the subject of ongoing work. This needs to be both encouraged and made use of where possible. Other areas currently have a lower commercial priority, such as high-level security and deployment, and as such will require more extensive investigation and development.

Technical Description

Software agents require a set of external (to the agent) capabilities in order to function – these are collectively referred to as the infrastructure. The infrastructure will typically consist of a several services and management components that are normally provided by operating systems, middleware, or other frameworks. This chapter describes these infrastructural components and services, classifying them as either core services or support services. Core services are compulsory regardless of application domain and defined as those without which an agent cannot effectively exist. Examples include message transport, registration and lookup, and identity-generation. Optional services include authentication, encryption, resource management, persistence, logging, debugging, visualization, and deployment. Support services also include management tools used by parties other than the agent, such as the owner of the agent or system, for administration purposes.

Sometimes, the choice as to whether a capability belongs at the agent level or at the infrastructure level may seem ambiguous. One deciding factor is the scope of the capability or service. If the scope is narrow and specialized, the capability may belong at the agent level. Often, as capabilities mature, become generalized, and start to be used by other components, they migrate to the framework, middleware, or operating system layers.

Comparison with Distributed System Infrastructure

Agent-based systems have many characteristics that overlap with distributed systems and hence have similar infrastructure requirements. However, there are several important differences. Distributed systems usually have distinct service providers (services) and service consumers (clients). A service provider makes itself available through a relatively static advertisement. Short-lived consumers discover the service, interact with it (typically in a transactional manner), and then disappear. While services may maintain some long-term state (e.g. a database), the client-service relationship is bounded as a "session". Finally, the execution model is essentially a client-driven transaction.

On the other hand, agent-based systems are often peer-to-peer. That is, agents both provide and consume services of other agents (this does not imply that there cannot be specialized agents that are only service providers and vice-versa). Agents are uniquely identified long-lived entities. Furthermore, they are "social" entities, in that they may use a model of other agents – identities, capabilities, states, maybe reputations – in determining how to cooperatively solve problems. The lifetime of an agent, or of its social context, is not restricted to a simple session. The lifetime of an agent may exceed that of the system that hosts it, or indeed the software components from which it is implemented. Therefore, the infrastructure needs to provide mechanisms that allow the hosts (and even the infrastructure itself) to be updated while preserving the state of an agent.

Another important difference is concurrency. In distributed systems, clients request services through remote procedure calls (RPC) or equivalent mechanism and usually wait for the service request to be satisfied. Only the services need to handle concurrency, which arises in the form of multiple overlapping requests. Agent-based systems are more complex because agents are computationally autonomous with their own threads of execution. While inter-agent messages may stimulate processing, concurrent execution is the norm, with all of the usual attendant complexities. Moreover, the concurrency goes beyond servicing multiple overlapping requests to handling asynchronous messages that interrupt and affect the concurrent processing activities of the agent.

Finally, software agents may be mobile, allowing them to move (or be moved) from one host to another. Mobility introduces additional requirements in all aspects of infrastructure. Mobility may also help provide infrastructure-level capabilities such as long-term survivability.

Despite the above differences, software agent infrastructure is often built as a layer on top of distributed systems infrastructure. For example, the CoABS Grid [Global InfoTek, Inc. 2002] is built on top of Jini [Oaks 2000, Sun Microsystems, Inc. 2002, Sun Microsystems, Inc. 1999]. In such cases, the constraints of the underlying *de facto* or *de jure* distributed computing infrastructure may affect the realization of the software agent infrastructure.[1]

Challenges

Designers of software agent infrastructure face a number of challenges. The majority of these challenges arise from the simple fact that agent systems are a superset of distributed computing systems, and distributed computing is hard. [Woolridge 1998]. Since these challenges are not unique to software agents, we present an overview here but do not discuss them in detail. For more detailed discussion of these issues, consult [Birman 1996, Mullender 1993, Tanenbaum 2002]. However the characteristics of agent systems offer new challenges and change the perspective on the traditional challenges.

Security

Security is an important challenge and applies not only to providing necessary security services (such as authentication and encryption), but making sure that the services provided by the infrastructure are secure. For example, one of the services provided by the infrastructure is registration and lookup. In order for this service to be secure, mechanisms must be present to protect one agent from changing/deleting registry information belonging to another agent (which could lead to a large set of security problems). Another challenge for the infrastructure is to protect against external attacks, such as denial of service, on the services. Using the example of the registry again, guards should be in place to prevent any entity from repeatedly performing operations on the registry, whether maliciously or not, that are detrimentally affecting the service.

---

[1] A simple example: the original FIPA model allowed the use of an arbitrary predicate in agent lookup. However, agent registry systems based on, e.g., LDAP, or Jini lookup only support simple single-level attribute matching. Should the implementation try to provide full FIPA semantics, or be limited to what commercial systems can support?

New security challenges arise given that agents are autonomous entities that represent and act on behalf of humans. A key challenge is establishing distributed trust among social agents.[2] Preserving long-term identity, protecting identity from theft, and non-repudiation are challenges unique to software agents. Some of these challenges are considered in this infrastructure section while others are addressed in the architecture section.

Scalability

Infrastructure components must be carefully designed to support the construction of large-scale agent-based systems. Scalability can be improved through open interoperability and the application of well-known design principles such as decentralization, replication, and parallelism. Open interoperability implies that a greater range of components can be employed by agents, thereby aiding scalable growth. Designing for scalability inevitably means introducing new mechanisms – coordination, data synchronization, and so forth. Distributed approaches are almost always more complex than centralized algorithms. Therefore, it is important not to "over design" systems which are intended for small-scale deployments.

Robustness / Fault Tolerance

Infrastructure must be reliable in order to support long-running agent systems that are able to continuously execute on the order of several years. Heavily-loaded systems must robust and continue to operate as designed. Moreover, the system should be fault-tolerant i.e. reliable under extreme, abnormal or unexpected conditions. Extreme or abnormal behavior occurs when a system is subject to situations beyond expected operational and performance related conditions. Examples of unexpected conditions include failures of systems and information warfare and kinetic (i.e., physical) attacks. Persistence services and techniques such as replication can help improve the robustness of the infrastructure.

Agents place differing requirements on the availability of services depending upon their importance to the agent lifecycle. For example, a transport service is considered ubiquitous and as such is *required* to be available at all times for inter-agent communication. The failure of such a service would have a direct impact on the performance of the overall system. Other, non-essential services may only need to be available as a function of the agent's transient task or behavior.

Efficiency

Efficiency is a measure of the overhead introduced by the infrastructure. The amount of overhead depends on the "heavyness" of the infrastructure. An infrastructure that tries to address challenges such as security, reliability, robustness, and interoperability often incurs higher overhead and is therefore less efficient. For example, robustness may be achieved through replication, which in turn adds significant overhead. Another example is using XML encoding, which is more verbose (and hence adds overhead) but greatly enhances interoperability.

Accessibility

Accessibility is sometimes not considered as a challenge for infrastructure. By

---

[2] A reputation server is an example of a service that allows agents to decide on the level of trust to be placed in other agents.

accessible, we mean that the infrastructure should be easily obtainable and useable for all the required platforms (mainframes to personal computers to cell phones). Infrastructure can be free at the source level (as in open-source products), free at the binary level, or be a commercial product.

Interoperability

Even if a particular architecture is freely available, the possibility that that one architecture will be used in all agent systems is unlikely. In reality agent systems are often heterogeneous in nature. As such, an infrastructure must be designed to manage a flexible collection of services according to the requirements of the agent. Therefore, interoperability is an important consideration. Typically, interoperability is achieved through common standards [Object Management Group 2002, Foundation for Intelligent Physical Agents 2002, McCabe 2002]. In addition, one should assume that there will always be competing standards and therefore it is important to embrace heterogeneity. CoABS is an example of an approach to not define common standards but to build mechanisms to interconnect the different architectures [Global InfoTek, Inc. 2002].

This caught me out – the next section heading is at level 3, not level 4, but it's visually indistinguishable from the level 4 headings.

Requirements

The requirements for software agent infrastructure fall into two categories: tools and services. Tools support developers to create and debug agent-based software and help users to deploy and manage agents. Services, on the other hand, are used by agents at runtime. The availability of services simplifies the task of agent developers. This section describes both tools and services that are part of and/or provided by the infrastructure for software agents.

Services themselves can be further categorized into core services and optional services. Core services include specifically, although not exclusively: Message Transport, Registration and Lookup, and Identity Generation. These are the minimum set of infrastructure services required to support the agent lifecycle. The following sections discuss the core services followed by the optional services and conclude with discussions of tools.

Core Service: Message Transport

Message transport is the communication mechanism used by agents to exchange messages. All agent architectures rely on the existence of infrastructure that provides a message transport service. The type of the messaging service can be characterized in two dimensions: synchronous –vs- asynchronous and one-way –vs- sequenced (e.g., request reply). Synchronous messaging blocks the sender of the message while the message is being transmitted whereas asynchronous messaging queues the outgoing message in a buffer and allows the sender to continue immediately. In the case of synchronous messaging, the duration for which the sender is blocked can vary. In the case of asynchronous messaging, the send operation may fail if the buffers are not able to contain the message

One-way messaging implies that the sender does not wait for a reply. Usually, one-way messaging is used with asynchronous messaging. When required, one-way messaging is combined with synchronous messaging to provide the sender with delivery confirmation.

Sequenced communication implies that multiple messages are transferred in a given unit of communication – typically identified as a conversation fragment. A state transition diagram or interaction protocol can be used to map the relationship between the messages that make up the sequence. The simplest form of sequenced communication is request-reply messaging, where the sender sends a message, which is received by the receiver, which then generates a reply that is sent back to the sender. Mechanisms such as RPC [Bloomer 1992] and RMI [Remote Method Invocation 2002] provide request-reply semantics. More sophisticated sequenced communication is provided by toolkits such as KAoS [Graves 2002, Bradshaw 1997], which provides support for arbitrary state-transition diagrams that can enforce the sequencing of messages.

Message transport may be implemented on a variety of communication mechanisms such as Sockets over TCP or UDP [Stevens 1999], RPC [Bloomer 1992], RMI [Remote Method Invocation 2002], or SOAP [Simple Object Access Protocol 2000]. It may also use higher-level messaging services, such as IBM MQSeries [IBM 2002] or various implementations of the Java Messaging Service [Sun Microsystems Inc. 2002] standard. The message transport service may, in fact, use a variety of communication mechanisms transparently to the agents, using transcoding gateways to interconnect the dissimilar technologies. (This is particularly common if the agent system includes wired and wireless platforms.)

Message delivery to the receiver may be accomplished through polling or through a callback mechanism. With polling, the message transport service queues messages in a mailbox; the receiver then has to periodically check the mailbox for any new messages. With callbacks, on the other hand, the message transport service invokes a callback function to notify the receiver of the availability of the message (or to just deliver the message).

Mobile agents place additional demands on message transport. The challenge arises from fast moving agents that leave a host before a message destined for the agent arrives at the host. Forwarding of the message by the first host to the new host of the agent may not work if the agent continues to move through the network. Different approaches including the use of home agents as in Mobile IP [Perkins 1998] and proxies [Tan 2002] have been explored. The worst-case scenario is that the message continues to follows the agent but never reaches the agent but [Murphy 1999] discusses an approach to placing constraints on mobility to provide guaranteed message delivery.

Core Service: Registration and Lookup

Registration and lookup services allow agents to advertise their existence and properties, primarily so that other agents may locate them. A registry is essentially a directory that stores meta-information in the form of agent descriptions. Operations typically include registration, deregistration, modification and querying of registry entries.

The content of an agent description stored in the registry should at the very minimum include the agent's unique identity and one or more communication addresses. Additional attributes may include a user-friendly name, the communication languages and ontologies supported by the agent, and

information about the owner and author of the agent. The physical location of the agent may also be stored if that is an important attribute. Additional application-specific information may also be included, for example if an agent charges for for some service it delivers, the registry can include usage and charging specific information. Finally, the registry may also include the public keys (or digital signatures) of agents in which case the registry acts as a key store or a certificate authority.

Registries need to address similar scalability issues to those of directory services in distributed systems. Specifically, scalability is a concern depending on the number of agents and the frequency and nature of queries and updates. Often this is solved with some form of federation. Some recently successful models include the registry services used by messaging clients such as ICQ [ICQ 2002], AOL Instant Messenger [America On-Line 2002], and Windows Messenger [Microsoft Corp 2002b].

Another challenging issue is keeping the registry up to date. Agents that rapidly change their attributes make it difficult for the registry to accurately reflect the state of the system. An example is storing the current location of a mobile agent. If the mobile agent moves quickly from host to host, keeping the registry updated with the current location of the host is difficult. When necessary, a registry can offer an indication of how recently an entry was updated, although realistically no assumption can be made by a querying agent, other than that the received information is current and correct.

Long-lived agents present a different kind of problem: how to ensure that registry information can survive changes to the registry infrastructure itself.

Query based lookup involves searching the registry for key or template based matches. The simplest form of lookup would be a single key match, such as the agent identity. Typically the returned results will consist of a set of agent descriptions that fit the partial or complete search criteria. For a large agent population, the volume of information may be large, and it may take a significant time to perform the query. It is therefore wise and often useful to apply search constraints, such as a limit on the number of returned results and depth in searches across federated registries.

The type of registry dictates the complexity of the search query. For example, some registries may accept predicate based matching criteria. The predicate is usually a logical modifier that enhances the expressivity of the registry matching function.

An example of a predicate-based lookup is finding agents that satisfy certain geometrical relationships in terms of their physical location. An agent on a camera device facing northwards might need to contact an agent within the same vicinity that is facing eastward. In such a case, the first agent can construct a predicate that takes into account the position of the first agent and evaluates the relative position and orientation of another agent. This predicate can then be given to the lookup service to obtain a list of candidate agents.

The power and flexibility of a predicate-based approach comes at a price. Not only must the client agent be capable of constructing and transmitting an

arbitrary predicate; the registry service must be able to process it safely and securely. This is likely to be computationally expensive. More seriously, the use of any complex query system means that one cannot take advantage of existing mechanisms such as LDAP or Jini . As usual, there are costs and benefits to be weighed in making a design decision.

A registry provides a single point of control and authorization for a multi agent system. (This is true even if the actual implementation of the registry is distributed to enhance performance and availability.) If a multi-agent system spans organizational boundaries, each participating organization may insist on operating their own registry that implements their particular policies. In this case the registry systems must support some form of federation model, so that registrations and search requests can operate across multiple registries.

Federation of registries is a complex issue but essentially implies interconnectivity of registries to form a distributed resource [Gnutella 2002, Clip2 2002]. In its simplest form registries will reference one another to form a web-like structure. Obvious problems with this type of structure arise when considering such issues as consistency of multiple registrations for single agents. Additionally, federation is very much a technology-specific feature of individual registries, such as LDAP [Howes 1997] or UDDI [UDDI 2002, UDDI 2000].

Federation may affect the programmatic interface to the registry service, to provide control over the scope of registrations and searches. It will also affect the security aspects of the registry. As was noted earlier, it is essential that registry information should not be changeable by unauthorized parties, in order to prevent denial of service or impersonation. In a federated scheme, each system must be able to trust the access control security of all other registries.

It should be noted that infrastructure services may also be published in a separate, but complimentary registry. This provides a known location where agents can seek required services. Ideally, agents and infrastructure services should not be referenced in the same registry as they have different operational semantics.

Core Service: Identity Generation

Identity generation or naming is the last of the core services that must be available to any agent at startup. This service has but one function, to provide (globally) unique, semantic free, non repudiable identifiers for application as agent names. Uniqueness can be assured by generating a sufficiently complex identity, such as a 128 bit sequence. To be semantically free, the name should have no additional meaning attached to it other than the fact it is some nebulous and arbitrary identifier.

Authentication Service

Authentication allows entities in an agent-based system to securely identify themselves to each other by presenting credentials. Once identification is successful, trust relationships can be established between the entities. Authentication is required between agents and humans, agents and agents,

agents and their host platforms, as well as agents and other systems.

Passwords, Kerberos [Kerberos 2002], and Public Key Cryptography [Schneier 1996] are traditional approaches to authentication. Password authentication relies on the exchange of a secret word that was previously configured or agreed upon. Kerberos provides a more secure form of authentication by not sending passwords over network connections [Kerberos 2002]. Public Key Cryptography [Schneier 1996] (also referred to as Asymmetric Key Cryptography and PKI – Public Key Infrastructure) is a different approach to authentication. PKI uses a pair of keys in conjunction so that one key is used for encryption and the second key is used for decryption. Authentication can be provided by keeping the encryption key private whereas secure transmission can be provided by keeping the decryption key private. Two pairs of keys can be used in conjunction to achieve both authentication and secure transmission.

In PKI, keys may be stored in a variety of ways. Keys that are published are often stored in a certificate authority (CA). Entities must be able to retrieves the keys of other entities reliably for the CA. Spoofing (returning the wrong public key) results in a breakdown of the security offered by PKI. Private keys must be stored in a secure manner and never released to other entities. One approach to storing the private key is to rely on a physical device such as a smart card.

Mobile agents complicate PKI key security. For a mobile agent to be able to authenticate, the agent would need to carry a private key as it moves to various hosts. In a multi-hop scenario, an agent would have to authenticate to a remote host while executing on a foreign host. However, a mobile agent running inside a foreign execution environment is completely at the mercy of the foreign system. Currently, no mechanisms exist to protect the mobile agent from inspection (and consequently, extraction of the private keys). [Jansen 1999].

The WWW has resulted in the introduction of new authentication methods such as Microsoft's .NET Passport [Microsoft Corp. 2002]. In this approach, an entity authenticates with a central server, which is then trusted by third parties. Once the initial authentication is completed, no further authentication is necessary with each individual web site.

Authentication can be implicit under some circumstances. For example, a multi-user operating system authenticates users when they login into the system. If that user were to then execute or interact with an agent running on that system, the agent can implicitly authenticate the user based on the credentials already presented by the user to the operating system. Depending on established trust relationships, the implicit authentication can extend to other systems. However, multiple overlapping trust relationships complicate authentication rules and could lead to unexpected results.

Agent systems introduce some new complications into the area of authentication. In most distributed computing systems, it is reasonable to perform authentication once at the beginning of a session, trusting that the credentials thus obtained will remain valid throughout the session. For a long-lived agent, no such assumption is warranted. It becomes necessary to consider the issues of certificate expiry or revocation for the agent, the human principal

on whose behalf the agent is operating, or even the agent platform.

Encryption Service

Encryption mechanisms are a critical part of the infrastructure. While they may not be used directly by agents, several other infrastructure services such as authentication, message transport, persistence, and logging.

Two broad categories of encryption services exist: public key and secret (or private) key. Numerous secret key algorithms exist such as DES, 3DES. The advantage of secret key algorithms is efficiency in terms of encryption and decryption speed. The disadvantage of secret key algorithms is key-propagation. If the encryption and decryption need to be performed at different sites (as in the case of encrypted messaging), then the secret key needs to be communicated between the encrypting and decrypting endpoints. If the key is communicated through an insecure communication channel, the security of the messaging can be compromised. [Schneier 1996]

Public key algorithms use an asymmetric encryption approach where the encryption and decryption keys are different and while they are related, it is computationally hard to derive one key from the other. Given these two properties, public key algorithms make it convenient to publish one key (referred to as the public key) while keeping the other key secret (referred to as the private key). Public key cryptography may be used for secure transmission of data or for authenticating the source of the data (digital signature). [Schneier 1996]

One of the disadvantages of public key algorithms is high computational complexity resulting in poor performance. A second disadvantage of public key cryptography is the infrastructure necessary to make the public key available. The public keys are published in a directory called the Certificate Authority (CA). Public key cryptography breaks down if the CA can be subverted or spoofed thereby providing false public keys for entities. The term Public Key Infrastructure (PKI) is often used to refer to the set of APIs, tools, and services that provide public key cryptography,

Hybrid approaches combine public and secret key cryptography. For example, the Secure Sockets Layer (SSL) protocol [Stallings 1998] uses public key cryptography to securely exchange a secret key (referred to as the session key) during the connection establishment phase. Once the session key has been exchanged, a more efficient secret key encryption algorithm is used for the actual data transmitted over the communications channel.

Resource Management Service

Resource management encompasses allocation of resources to software agents as well as ensuring that agents do not exceed their allocated bounds (often referred to as resource control). Resource management is not a requirement that is unique to software agents. However, agents often operate autonomously and continuously without direct human supervision or intervention, making resource management is more critical for agents. In particular, resource management is essential to providing predictable behavior of and for agents.

Note that resource allocation is often controlled through policies specified through administration. Policy-based control for resource usage has been explored by KAoS/NOMADS [Bradshaw 2001, Suri 2000] and Ponder/SoMA [Damianou 2000, Bellavista 1999].

The primary resources used by agents are CPU time, memory, storage, and network traffic. Resource allocation is usually specified in terms of a limit on how much a resource can be used. Note that the network resource has a wider scope than the rest, which are all part of a single system. The network resource is often shared across several computers and hence requires more careful management. Particular environments may necessitate additional limits (for example, the number of files that might be opened concurrently, the number of processes or threads that may be created, etc.)

Limits placed on resource usage fall into two categories: rate based and quantity based. Rate based implies that the limit is specified in terms of a unit of time. Examples are a network write rate limit and a disk read rate limit expressed in bytes/sec. Quantity limits do not depend on time. Examples are the amount of memory/disk space used and the area of the screen that has been occupied.

Approaches to resource allocation range from none to static to priority-based approaches (like operating systems) to market-based. Frequently, resource allocation is completely ignored. Agents are executed just like other computer programs. Such an approach may be adequate in situations where agents can consume all available resources on a system and allocation of the resources between the agents is unnecessary (or there is only one agent running per system).

In the static approach, administrators specify the allocation of resources through policies. These allocations can be changed through subsequent changes to the policies that govern the agents. The static approach is useful is situations where the number and priorities of agents do not change very often. An advantage of the static approach is simplicity and deterministic behavior.

Priority-based approaches allow administrators to specify the relative priorities of agents on a system. The exact allocation of resources is then left up to a software component. This approach is similar to that provided by most operating systems and offers a good mix of administrator control and automatic management.

Finally, market-based approaches are the most dynamic. In this approach, agents are provided (artificial) currency, which they can use to get access to resources. The cost of resources can vary based on availability and administrators can provide additional currency to agents as needed. Unlike the previous approaches, no external entity allocates resources to agents. Agents are expected to negotiate for the resources they require. The resources that may be given to agents will depend on the cost of the resource as determined by the market demand. Current market-based approaches do not seem well suited to direct administrator control of individual agent behavior and do not offer deterministic behavior. However, for very large numbers of agents, market-based approaches may be the only viable approach to resource allocation.

The second major component of resource management is resource control, which is responsible for making sure that agents access resources within the limits that are allocated to them. Resource control is important for the safe execution of agents unless agents are isolated and completely trusted.

Resource control has been recognized as an important requirement but has largely been ignored by the research community. Agent researchers often view resource control as a service that needs to be provided by the operating system or language environment. Implementing resource control at the agent level is not useful. The agent has to be trusted to self-impose the limits and even with the best intentions may have a faulty implementation. In situations where the agents cannot be trusted (such as with mobile agents), the infrastructure must provide the necessary resource controls and safeguards.

Recently, work has started on implementing resource control mechanisms for the Java environment. This is an important step because a (increasingly) large percentage of agent software operates in the Java environment.

Persistence Service

Persistence services allow the storing of information on some form of secondary, semi-permanent storage mechanism (i.e., one that can survive the restarting of a system). Agents may use persistence services for two different purposes: to store the state of an agent or to store data.

One of the key capabilities of agents is to be able to adapt to the environment and learn the preferences of users over time. The persistence service allows agents to store information regarding the preferences of users. Agents may also store other parameters regarding the system state. Therefore, persistence is key to helping the learning characteristics of agents.

Agent state is used in two different contexts: data state and execution state. The data state of an agent consists of the information held by the agent. In object-oriented implementations, the data state consists of the values of the variables of the objects that represent the agent. The execution state on the other hand consists of the state of the thread or threads handling the execution of the agent. A persistence service may support persistence of data state only or persistence of data and execution state (persisting only execution state without the data state would be meaningless). Providing support for persistence of execution state is difficult due to two reasons. If an agent is interacting with another agent, then the semantics of capturing the execution state of the first agent and not the second agent are not well defined. First, capturing the execution state requires support from the underlying execution platform, which is either a virtual machine (such as a Java VM) or the operating system [Atkinson 2000, Suri 2002, Suri 2000, Sakamoto 2000, Truyen 2000]. Second, while it may be possible to capture the state of an agent, restoring the state may be problematical. (For example, the agent may have been in the middle of an interaction with another agent which no longer exists.) In general, it is important to design agents to be robust in the face of such situations.

Storing the state of an agent is important for survivability of long-running agents. Computer systems often restart for various reasons such as faults,

power failures, or reconfiguration. A long-running agent must be able to survive beyond one execution of a computer. Therefore, agents can use persistence services to save their state so that they can recover in the event of a restart.

State persistence can also be used to improve scalability. If a system is overloaded, persistence can be used to swap agents to temporary storage. The persisted agents may then be reloaded at a later point in time or migrated to a different system for execution (load balancing). [Uszok 2001].

Logging Service

Logging services record operations performed by software agents. The operations can vary from transactions performed by agents to message traffic between agents. Logging services help in non-repudiation by keeping record of actions performed by agents. Also, logging of message traffic between agents can assist in debugging. In some cases, the message transport service may offer automatic logging or non-repudiation services.

Logging may be centralized or distributed. A centralized logging facility uses a single service that handles all logging traffic. Centralization allows logging to be ordered (sequenced in time)[3] but introduces a bottleneck in the performance as well as a single point of failure. Distributed logging uses several logging services scattered through the network (but not necessarily one per host executing agents). A distributed logging service scales well but introduces a problem when multiple logs need to be merged to obtain a global view. The problem can be mapped to the more general problem of gathering global state information in distributed systems [Babaoglu 1993].

Logging of mobile agents is further complicated for two reasons. The first problem arises if the logging facility changes as the agent moves around the network. Therefore, even the log for a single agent may need to be assembled from portions at different logging services. The second problem arises with logging for non-repudiation of transactions, which requires that the transaction be signed by a private key. However, if a mobile agent carries a private key with the agent, then the key information may be extracted by a malicious host [Jansen 1999]. While some solutions have been proposed [Jansen 1999, Sander 1997, Ferreira 2002], none of them solve the problem effectively.

Logging of message traffic between agents raises another challenge with encrypted communications. Messages are usually encrypted so that only the receiver of the message can decrypt the message. Therefore, it would be a security violation if the logging service needs to be able to decrypt the messages in order to log them. One possibility is to log the messages in their original encrypted form. If the message traffic needs to be analyzed, then the necessary decryption keys can be obtained through human intervention[4].

---

[3] Note that the sequencing will be based on the arrival of the messages or events at the logging service. Due to varying latencies in the network and the impossibility of a synchronized global clock, even a centralized logging service cannot guarantee globally ordered logging of messages and events [Tanenbaum 2002].

[4] This has an interesting analog in the real world where an entity wanting to monitor

Logging services may also serve as a solution to protect agents from being tampered by malicious hosts. Solutions proposed by [Sander 1997, Tan 2002, Vigna 1998] rely on tracing the execution of an agent. The execution trace is then logged either at the execution host or at a secondary location. In case the execution of the agent needs to be checked (to detect tampering), the agent can be reexecuted and the new trace compared with the original logged trace[5].

Reputation Service

This is a trusted service that agents can use to monitor the compliance of other agents to publicly endorsed policy agreements. Reputation services are one of the few mechanisms that are able to enforce obligations; since obligations cannot be prevented but only required.

A typical use of a reputation service is for all parties to an agreement to `escrow' their agreement with the reputation service. If one of the parties determines that another party has defaulted on an obligation it may lodge a complaint with the reputation service.

In software systems the concept of a legal remedy may seem moot; however, simply recording instances of default and offering that information to others querying the service may be a powerful deterrence mechanism. If an agent defaults on an obligation, other agents and services may become more reluctant to offer it facilities if they are able to query a reputation service.

Note: This service can be extended to encompass auditing functions, which are linked to the logging tool described below.

Coordination Services and Languages

A family of so-called **coordination languages**[6] is emerging as an extension to mobile-agent systems. Coordination languages provide higher-level communication protocols such as generative communication and publish/subscribe models.

While the theoretical models for "general mobility" can accommodate physical mobility, none of the mobile agent system infrastructures currently available provides support for migratory hosts. Host migration and ad hoc networking are being addressed by work done on coordination.

This new research direction on "coordination languages" was started in the late 80's by Gelernter with a programming model named Linda [Carriero, Gelernter 1989]. The common theme underlying most coordination languages is a form

---

communications of individuals has to show just cause and obtain a warrant.

[5] While several solutions have been proposed to protect agents from malicious hosts, most (including execution tracing) are not practical and hence do not have any usable implementations.

[6] In this context coordination languages provide a programming model for coordination of a number of parallel and distributed tasks. They operate at a much higher level than agent-to-agent interaction languages.

of associative communication in which a process can register an interest and another can offer a service. The infrastructure (often called a *tuplespace*) then matches interests with offers. Communication is thus uncoupled since no explicit link needs to be established between communicating partners; a message may be read at any time and by any process interested in it. These properties make it straightforward to provide resource discovery protocols that match up clients with servers based on their respective offers, to program in an event-driven style and to dynamically configure running systems. There is a vital research community focusing on coordination from the programming language aspect with a yearly conference (Coordination), and a growing influence: even Sun Microsystems's Jini technology incorporates a coordination language called JavaSpaces.

Traditional computational models assume that all devices and software components are deployed before an application starts executing, and that once deployed, configurations are static. In the field of wireless computing, however, in which Personal Digital Assistants (PDAs) and other portable devices can establish ad-hoc network connections, these assumptions do not hold. Instead new computational models are needed to ease the task of developing applications for such fluid environments. Mobile agents provide part of the solution since they provide computation mobility, which allows an application to reduce its dependence on an unreliable ad-hoc network. The mobility of devices presents other challenges, however. In particular, current mobile-agent infrastructures do not provide high-level communication primitives suited for physically mobile systems. Experience with a medium-sized mobile-agent application suggests that well over half of the application's code deals with communication in some way. This code is both tedious to write and the source of many of errors.

Designing communication mechanisms for mobile environments is a challenging task. Communication in a physically mobile system is:

- *Transient and Opportunistic*: Communication patterns must fit into an environment where hosts are intermittently connected to the network and agents can leave a host at any time. Communication thus tends to be opportunistic, with applications taking advantage of whatever network resources happen to be available at a particular time, but not relying on their continued availability. The underlying communication protocols must accommodate long latencies and/or timeouts caused by the sudden departure of an interlocutor or the disconnection of the agent itself.
- *Unnamed and Untrusted:* Communication in mobile systems is often based on the services being offered, rather than the identity of the entity providing those services. As long as the needed services are provided, agents do not necessarily have to know each other's names and locations to interact. The corollary of anonymity is that interlocutors do not necessarily trust each other, which implies that the communication infrastructure must provide the mechanisms needed to implement secure communication protocols.

Coordination languages describe a family of programming languages and

infrastructures that provide communications mechanisms with the two characteristics above. While the early coordination languages were based on centralized systems, a new generation of languages targeted at ad-hoc networks is emerging. The most widely known is Picco and Murphy's Lime language [Picco, Murphy, Roman 1999]. Lime is a decentralized coordination language implemented in Java in which the central tuplespace of Linda is replaced by a multitude of application-specific tuplespaces owned by mobile agents. Whenever two mobile agents are located close to one another, their tuplespaces are seamlessly merged to form a transiently shared data structure [Murphy, Pietro, Roman 2001]. The advantage of systems like Lime is that they provide simple ways to program resource-discovery protocols and other common communication patterns in agent systems. Limitations of the original Lime model have been partially addressed in [Carbunar,Valente,Vitek 2001], but this area still requires attention from both the formal and implementation sides. The most challenging problem is, as usual, how to provide security guarantees for applications using a coordination language in untrusted networks. Once these problems are addressed, however, coordination languages will provide a powerful way for mobile agents to communicate when operating inside ad-hoc wireless networks. Moreover, the mobile agent will contain very little communication code itself, since the necessary (and complex) functionality will be encapsulated inside the coordination-language infrastructure.

Ad hoc networking in combination with mobile code is also investigated in the field of active networking, where active packets, a kind of miniature mobile software agents, are used to deploy and execute customized routing algorithms [Tschudin 2000].

Administration and Management Tools

As software agent systems increase in complexity, it becomes increasingly important to provide the appropriate tools to deploy and manage them. Among the factors leading to increased complexity are the number of agents, the "sophistication" (i.e. unpredictability) of the agents, and the degree of interconnectedness. The developers and administrators of these systems need tools that provide convenient and easy to use abstractions for configuring agents and for monitoring and controlling their execution.

It is worth noting that agent management is in most respects the same as any other kind of system management. There are a variety of commercial management systems which support discovery, topology management, visualization, aggregation, and virtualization of managed resources, from disk drives to web servers. While agent management introduces some new issues, such as treating software components as security principals, the basic patterns work quite well.

One promising approach to management is to organize agents into groups (also referred to as domains) [Bradshaw 2001, Bradshaw 1999]. The groups may then be nested to support a hierarchical organization. Groups may also overlap (that is, agents may belong to more than one group). Hierarchy and overlap allows construction of groups that can reflect human organizations. Agents may be grouped for ownership, functional, or organizational purposes. Overlapping groups allow modeling of agents that participate in multiple roles. Grouping is

essential to keep agent systems manageable as the number of agents increase.

Controlling the execution of agents can be achieved through the use of policies [Bradshaw 2001, Bradshaw 1999, Damianou 2000]. There are fundamentally two kinds of policy statements: those relating to **permissions** and those relating to **obligations.** Permission-style policy statements include authorizations (i.e., granting agents the right to perform an action) and negative authorizations (i.e., denying agents the right to perform an action). Obligation-style policy statements include obligations (i.e., requiring agents to perform some an action), and waivers (i.e., giving agents the option to not perform some action).

Policies may be expressed in a variety of languages. At one extreme they may be written in some propositional or constraint language. There are a wide variety of simpler schemes, each of which gives up some types of expressivity. The choice of language for a particular application will be affected by considerations of composability, computability, efficiency, expressivity, and amenability to the detection of equivalence and the discovery of conflicts.

Policies may apply at a domain level (i.e., a group of agents), at the system level, an execution environment level, or at an individual agent level. This means that an agent may be subject to multiple, possibly conflicting, policies, and the policy management tools should be capable of exposing and resolving these conflicts.

The use of a rich, fine-grained policy model such as this may have far-reaching implications. In particular, one must decide whether the system is "enforced" or "cooperative". In an "enforced" system, every action by an agent which is subject to policy control, is automatically checked, and may be disallowed. Enforcement may be achieved through support from the platform or execution environment (e.g., the Java VM) or by using proxy objects that are under control of the policy system. In the latter case, agents perform actions on the proxy objects, which consult the policy system before allowing the request to proceed.[7] This may be expensive and is not easily applied to legacy code. The alternative is a "cooperative" approach (analogous to "cooperative multitasking" on some ancient operating systems), in which agents explicitly check each proposed action against the policy system. This is an unsatisfactory way of handling security, to put it mildly. In either case, the policy engine may become a bottleneck or single-point-of-failure, particularly where policies are context-sensitive.

Debugging Tools

Debugging support is an important tool for managing agent systems due to their asynchronous and distributed nature. The problems described in the section on logging services apply to debugging also. The varying network latencies and the impossibility of a global clock (or exactly synchronized clocks on individual systems) make it difficult to obtain an accurate global view of the system. Good visualization and logging tools will help in debugging problems at the level of the infrastructure.

---

[7] Proxies of this type are also called privileged code wrappers.

Visualization Tools

Visualization tools allow the evolving state of the agent system to be observed. Visualization may be tailored for different perspectives – administrators, developers, and users – and is a necessary part of effective administration and management tools.

Agent systems require visualization of several different aspects such as agent life-cycle, errors, messaging, and operational compliance. Visualization of the life-cycle shows agent startup and shutdown. Visualization of errors includes premature termination of agents, non-terminating errors reported by agents, and synchronization problems such as deadlock and livelock. Deadlock arises when two or more agents are waiting on each other in a cycle and therefore will remain waiting indefinitely. Livelock occurs when two or more agents repeatedly try to perform the same operation at the same time resulting in all of them aborting.

Visualization of message traffic is another important requirement. This visualization is useful to discover both missing communication as well as unintentional communication between agents (which may have resulted due to an oversight in specifying security policies).

Visualization of resource consumption allows administrators to properly allocate available resources amongst competing agents. Visualization may also reveal resource abuse by certain agents (which could be malicious agents or buggy agents).

Finally, mobility needs to be visualized. Mobility includes both mobility of physical platforms that host agents as well as mobility of agents across platforms. The difficulty here is keeping the visualization up to date with respect to the state of the world – especially with fast-moving systems or agents.

The key challenge faced by visualization in agent systems is the same as in distributed systems. The asynchronous nature of these systems implies that the state information to be visualized is scattered across multiple systems that are operating independently and concurrently. Therefore, the state information is gathered via message-passing. However, transmitting the state information across a network consumes a variable amount of time during which the system continues to change. If the asynchronous nature is not taken into consideration, the visualization will show artifacts (such as agents being located at incorrect places or in multiple places, agents terminating before they appear to start, agents receiving messages in a different order, etc.) [Birman 1996, Mullender 1993, Tanenbaum 2002]

Deployment Tools

Deployment is a difficult challenge often ignored by researchers. Currently, deployment of agent software (especially multi-agent and mobile agent) software is complicated and requires expert installation. A multi-agent system may include agents from several different developers that need to be installed separately on multiple systems. Even at the level of a single agent, several separately installed components may be required. Versioning is an added

concern when multiple agents share components.

Security and administration requirements further complicate deployment. Properly configuring security is tedious and error-prone. When an agent system is installed, security configuration might be too restrictive (or misconfigured) causing the agent to fail. A common reaction is to weaken the security until the agent is able to operate (and in the process weaken the security of the overall system). Another possibility is that the initial security configuration might be too weak but is never tightened since the agent operates successfully. A good solution to the administration and management tools will help in security weaknesses as described above. In particular, systems administrators will understandably resist the proliferation of security systems, and tend to prefer approaches that reuse existing mechanisms.

Mobile agents introduce a more challenging deployment problem. Mobile agents derive many of their advantages by moving to systems other than the initially installed host. However, mobile agents also require an execution environment to receive and execute agents. Without the presence of the execution environment on the target host, an agent will not be able to move to the target host. Therefore, mobile agents require that execution environments be deployed onto all the hosts to which agents might wish to move.

A further complicating factor is that today's mobile agent systems do not interoperate well. An agent written using a particular mobile agent toolkit cannot execute on a different toolkit's execution environment. If a system utilizes agents based on different mobile agent toolkits, then all the necessary execution environments must be deployed onto the target systems [Pinsdorf 2002, Magnin 2002, Grimstrup 2001]

One of the capabilities of mobile agents is to handle unexpected scenarios by dynamically moving capabilities onto systems. In order for this capability to be realized to its full potential, execution environments must become ubiquitous. Given this capability, mobile agents can actually help solve the deployment problem by moving and installing components on systems.

Examples of Prior Work on Infrastructure

CoABS Grid

The CoABS Grid [Global InfoTek, Inc. 2002] was developed by Global InfoTek and ISX Corporation as part of the DARPA-sponsored Control of Agent-Based Systems (CoABS) [Global InfoTek, Inc. 2002]. The primary motivation for the Grid was to create an integration platform for legacy stove-piped systems. The Grid is Java-based and built on top of the Jini architecture [Oaks 2000, Sun Microsystems, Inc. 2002, Sun Microsystems, Inc. 1999] from Sun Microsystems (although it can support agents in other languages through proxies).

The key capabilities of the Grid are registration, lookup, and messaging. The

Grid supports both agents and services (with services nominally being passive entities that are tasked by agents). Agents are modeled as a specialization of services. We will describe only agent-relevant aspects in the remainder of this section.

The Grid provides a registry for agents and services based on the Jini Lookup Service (LUS). Agents can register meta-information with the Grid. Agents also store a proxy to themselves in the registry. Therefore, other entities can search for an agent and retrieve a reference to the target agent's proxy, which allows messages to be sent to the target agent. The Grid defines a standard template for agent meta-information that can be filled-in by an agent when it wishes to register with the Grid. This template allows an agent to identify the name of the agent, the owner, the architecture, the agent communication languages (ACLs), the messages content languages, and the ontologies supported by the agent. Additional domain-specific attributes can also be added to the registry.

Agents can search for other agents on the Grid through the registry. Searches can be performed based on the agent meta-information template described above. Predicate-based searches are also supported, which allows agents to perform more sophisticated searches matching with complicated matches on several attributes. The result of a lookup is a set of matching agent proxies – objects that represent the agents that have been found. At a minimum, a proxy allows messages to be sent to the agent. The Grid allows agents to create custom proxies with additional specialized operations.

Messaging in the Grid is asynchronous. The transmitting agent sends a message to the recipient by adding the message to the recipient's message queue (done via the recipient's proxy). Each message contains the sender's proxy so that the recipient can reply easily. More sophisticated forms of messaging can be provided via custom agent proxies. Secure communication is provided through encryption and includes both authentication of the sender as well as secure transmission of the message on the wire. The grid also provides a logging service that can be enabled via a central control. Logging allows the message traffic between agents to be recorded for later analysis or debugging.

A set of GUI tools for administration and monitoring round out the Grid software. The GUI tools allow configuration of the grid, startup, and shutdown operations. A simple visualization of the registry allows administrators to view the current set of registered agents.

Since the Grid is built on top of the Jini environment, all of the capabilities of Jini are also available within the Grid environment. For example, the Grid takes advantage of Jini's ability to have replicated lookup services. Similarly, the Grid takes advantage of Jini's proxies and mobile code capabilities. The designers of the Grid expose the lower Jini layer and encourage users to tap into the capabilities of Jini.

The main contribution of the Grid is providing an integrating platform for heterogeneous agent systems. Prior approaches have relied on specifying standards to which agent software must be written. The Grid's approach is to take different agent systems and make them interoperable through runtime support. However, a key requirement to make this a reality is semantic

interoperability (which is addressed in its own section in this report).

FIPA

FIPA (Foundation for Intelligent Physical Agents) [Foundation for Intelligent Physical Agents 2002] is a standards organization that specifies standards for software agent systems. FIPA began in 1996 and published the first set of specifications in 1997 (referred to as FIPA 97) and a revised set in 1998 (FIPA 98). FIPA's goal was to provide interoperability between different agents and agent systems.

Recently, FIPA has adopted a more flexible approach by developing a specification for an abstract architecture [Foundation for Intelligent Physical Agents 2001]. The abstract architecture allows FIPA-compliant agent systems to be realized using a variety of existing industry standard services and technologies. The earlier standards published by FIPA fit within the umbrella defined by the abstract architecture.

The main abstraction defined by FIPA is that of an Agent Platform (AP). The Agent Platform contains a number of services the most important of which is the Agent Management System (AMS) [Foundation for Intelligent Physical Agents 2001b]. The AMS handles the creation, registration, lookup, communication, migration, and termination of agents. Two other mandatory services that must be provided by an AP are a Message Transport Service [Foundation for Intelligent Physical Agents 2001c] that handles the encoding and transmission of messages between agents and a Directory Service [Foundation for Intelligent Physical Agents 2001] that handles the lookup of agents.

FIPA agents communicate through message passing that is handled by the Message Transport Service. FIPA also defines the format of the messages as part of the Agent Communication Language (ACL) [Foundation for Intelligent Physical Agents 2001d] specification as well as a representation in XML [Foundation for Intelligent Physical Agents 2001e].

FIPA specifies but does not mandate the use of a number of Content Languages, which define the content of the ACL messages. Examples of Content Languages include KIF (Knowledge Interchange Format) [Foundation for Intelligent Physical Agents 2001f, Knowledge Systems Laboratory 2002] and RDF (Resource Description Framework) [Foundation for Intelligent Physical Agents 2001g, World-Wide-Web Consortium 2002]. In addition to Content Languages, FIPA also defines a Communicative Act Library [Foundation for Intelligent Physical Agents 2001h] that defines common communicative acts that can be included in agent messages.

Another communication-related set of specifications covers interaction protocols [Foundation for Intelligent Physical Agents 2001i]. Interaction protocols go beyond one-way messaging and specify sequences for messaging. FIPA defines a number of interaction protocols for requests, queries, auctions, etc.

A number of other specifications cover areas such as ontologies [Foundation for Intelligent Physical Agents 2001j], policies and domains [Foundation for

Intelligent Physical Agents 2001k], and application-specific agents.

A number of FIPA compliant agent systems have been developed. Examples are Jade [Bellifemine 2001, JADE 2002], April [McCabe 1994, April 2002], and FIPA-OS [FIPA-OS 2002]. The Agentcities project is a FIPA-related effort to establish a large network of interconnected FIPA-compliant agent platforms throughout the world [Agentcities 2002].

JAS

The Java Agent Services (JAS) project is an initiative to define an industry standard API specification for delivering software agent infrastructures in the Java environment [McCabe 2002]. The JAS implements the FIPA Abstract Agent Architecture [Foundation for Intelligent Physical Agents 2001] within the auspices of the Java Community Process (JCP) [Sun Microsystems Inc. 2002]. As of this writing, the JAS specification has been released for public review.

The JAS API defines a set of Java interfaces that deliver a layer of open interoperability between agents through the core services of message transport, agent directory (registry) services and agent naming services. Message transport includes support for message composition, encoding and transmission between agents using arbitrary application transport protocols.

The API also specifies a Service Root construct which is used to deliver references to infrastructure services to an agent at startup.

The JAS widely employs a collection called the JasBean, modeled on the JavaBean concept, that defines a collection of key-value attribute pairs. Many of the interfaces defined by JAS extend this collection, providing setter and getter methods to set and retrieve the key-value tuples. Using a tuple-based interface to data structures increases flexibility and implementers are free to add additional key-value pairs as needed and to store the parameters in any suitable manner.

The JAS project also delivers a Reference Implementation of the API, which includes a Service Provider Interface (SPI) model and a message representation, called the Abstract Content Representation (ACR), in addition to several exemplar services.

In the SPI model instances of new services are generated using service factories; a given factory being responsible for creating a given type of service. Using the standard Factory Pattern a framework can be built that supports the dynamic addition of services through the SPI interfaces.

The ACR is a UML based model of abstract elements that defines the syntactic attributes of Agent Communication Language (ACL) encoded messages and arbitrary content languages. The motivation for this is to abstract the common features of such languages into a single semantically neutral representation that can be supplied to transport codecs.

The ACR essentially maps the predications, terms, propositions, actions, quantifiers, variables, constants and connectives of content languages, and the attributes of ACL expressions into a set of syntactic elements and relations. This avoids the necessity for n-to-n mappings typically required between

different content languages (such as KIF, RDF, etc.) and message encoding schemes (such as XML, Java RMI, etc.). Instead, ACR acts as the intermediate representation so that a message using any content language can be encoded/decoded with reduced complexity.

Example mappings of the core services available through the API are:

Message Transport Service can map to RMI [Remote Method Invocation 2002], TCP/IP [Stevens 1994], SMTP [Stevens 1994], HTTP [Stevens 1996], IIOP [Object Management Group 2002], SOAP [Simple Object Access Protocol 2002], and others. The Directory Service can map to RMI [Remote Method Invocation 2002], LDAP [Howes 1997], UDDI [UDDI 2002, UDDI 2000], and others.

Since JAS is an extension of to Java platform, agents written in the JAS environment also have access to the rich Java platform API. This implies that agents hosted within JAS based environments will also benefit from the continued development of Java and related technology.

Grid Computing

Grid computing is an evolution of distributed computing and is based on the metaphor of the electrical power grid. The goal of grid computing research is to make computational power as easily accessible and useable as electrical power. In the electrical infrastructure, power is transmitted from producers of electricity to consumers through a national power distribution grid. A computational grid tries to provide computational power in much the same manner. The emphasis is on transparent availability of needed computational and data resources regardless of geographical location.

Computational grids interconnect large supercomputing centers and large data warehouses in order to provide high-performance computing to end-users. Grids can be hierarchical with regional grids interconnecting local ones and national grids interconnecting regional ones. One of the enabling technologies for grid computing is high-speed and high-capacity networks. As broadband connectivity becomes more pervasive, more and more end users will be able to tap into computational grids.

Foster and Kesselman list several application areas for grid computing including distributed supercomputing, high-throughput computing, on-demand computing, data-intensive computing, and collaborative computing. [Foster 1999].

Computational grids are also viewed as an enabling technology for new kinds of applications. Example applications include teleimmersion, telemedicine, global weather forecasting, molecular modeling, material science, and genetics research.

An interesting difference between electrical grids and computational grids lies in the role of the consumers and producers. In electrical grids, producers are often distinct from consumers. In computational grids, consumers may be able to act as providers of resources when their end-systems are idle [Suri 2001, Gimps 2002]

Large-scale grid computing efforts include the NASA Information Power Grid [Johnston 1999], The NSF-sponsored TeraGrid [TeraGrid 2002], and the European Union sponsored DataGrid [DataGrid 2002] among many others. The Global Grid Forum [Global Grid Forum 2002] is an organization that facilitates cooperation amongst grid computing researchers and publishes open standards for adopters.

Recently, computational grids and web services have started to merge. The goal of web services is to provide infrastructure for services to be advertised, found, and accessed over the Internet using Web-style protocols. Web services and computational grids face many similar challenges: description and advertisement of services, description and matchmaking (lookup) of service requests, invocation of services (possibly with attached contracts regarding quality of service or other constraints), and the accounting mechanisms to charge for resource usage. Recognizing the overlap, the Globus group have recently announced the Open Grid Services Architecture (OGSA) [Argonne National Labs 2002] that brings grid services and web services infrastructure closer together.

Traditional computational grids differed from agent-based systems in their scope, features and capabilities as well as their target application domains, with the goals for grid computing being closer to traditional distributed systems. However, recent work in grid and web services are making them more dynamic, late-binding, and long-lived. These are also characteristics of multi-agent systems. Many research areas in grid computing such as network communications infrastructure, service discovery, and resource management overlap with agent systems research. As the economic benefits help drive companies such as IBM and Microsoft to foster web services, agent-based systems can take advantage of grid computing and web services infrastructure.

Peer-to-Peer Architectures

Traditional distributed computing architectures use the client-server pattern where the number of clients far exceeds the number of servers. Moreover, the systems are specialized to act either as clients or servers but not both. Peer-to-peer architectures, on the other hand, blur the distinction between the client and the server, with a majority of the systems having dual personalities: they can behave as clients or as servers. The goal is to move away from centralized servers, which are single-points of failure as well as performance bottlenecks. As systems become larger in scope and size, traditional client-server approaches will not be able to meet the scalability demands.

Peer-to-peer networking has been popularized by media-sharing applications such as Napster [Napster, Inc. 2002] and Gnutella [Gnutella 2002]. Both of these applications provide, to varying degrees, peer-to-peer sharing of data. The Napster approach still uses a centralized directory of shared data but peer-to-peer transfer of data. The Gnutella approach uses self-organizing networks with a distributed directory and search mechanisms. However, even the Gnutella approach requires a bootstrap directory of clients in order for a new client to discover other peers and join the network.

While file sharing has popularized peer-to-peer architectures, it is far from the

only application available. Personal computers have long supported peer-to-peer networking over Local Area Networks with additional capabilities such as printer and modem sharing. Computational grids may also use peer-to-peer approaches to locate and share processing resources. Companies such as Entropia are developing computing frameworks that allow cluster computing to be performed on computers connected via peer-to-peer networks. Also, groups such as the Peer-to-Peer Working Group [Peer-to-Peer Working Group, 2002] are developing standards to help accelerate the research and application of peer-to-peer computing architectures.

Software agent architectures have traditionally been peer-to-peer. The autonomous nature of agents implies that agents may at any time initiate requests to other agents as well as receive requests from other agents, thereby naturally making them peer-to-peer.

Risks

Infrastructure is the lowest layer of software agent systems and provides critical services such as administration and management, secure messaging, resource management, and visualization. Using infrastructure does not add any risks to software agent systems. In fact, well-designed infrastructure and tools can help mitigate risks that other agent technologies might introduce. For example, policy-based management and control of agent systems can help to make sure that agents act within specified bounds and do not misbehave. Visualization tools can help identify inappropriate or undesirable behavior of agents.

Forecast

Forecast Table

Technology element	Near term 2001-2003	Midterm 2004-2006	Long term 2007-2010
Infrastructural Technologies			
Administration and Management Tools	10s of agents; multiple systems; simple security policies	100s of agents; 10s of systems; complex security policies;	1000s of agents; 100s of systems; self-adaptive security policies
Authentication	Authentication of small-scale agent systems	Authentication with revocation for long-lived agents	Seamless authentication with identity protection on a global scale
Registration and Lookup	1000s of agents; local registries	100000s of agents; regional registries	Millions of agents; global registries
Message Transport	Simple policies to govern message sequencing	Sophisticated message sequencing policies	Same
Encryption Services	Efficient, secure encryption algorithms	Same	Same
Resource Management	Resource control for individual agents	Resource allocation and control for groups of agents	Resource management for large groups of agents
Persistence Services	Persistence of agent data state	Persistence of agent execution state	Transparent and automatic persistence of complete agent state
Logging Services	Limited logging of small groups of agents	Logging of large groups of agents and of mobile agents	Fully distributed, secure, and efficient logging of all agent actions
Coordination Services	Research prototypes for coordination in small-scale ad-hoc networks with host mobility.	Adoption of an XML-based coordination language for small devices.	Adoption of standardized coordination protocols for wireless devices.
	Intelligent resource discovery prototype for wireless devices.	Semantics of coordination languages codified.	Integration of MA technology with coordination languages.
Debugging Tools	Debugging of individual agents	Debugging of mobile and multi-agent systems	Debugging of large-scale agent systems
Visualization Tools	Limited visualization of small multi-agent systems and small groups of mobile agents	Visualization of 1000s of agents	On-demand, secure visualization of large-scale agent systems
Deployment Tools	Limited deployment tools for local networks	Deployment tools for large intranets and the Internet	Global deployment of agent platforms and agents

Figure 1. *Agents Infrastructure Forecast Table*

Summary and Recommendations

Infrastructure for software agents consists of services and tools that used by software agents as well as their developers and users. The infrastructure needs to address challenges such as security, scalability, robustness/fault tolerance, efficiency, accessibility, and interoperability. While somewhat similar to distributed systems, software agents have a number of characteristics that are unique such as long-lived identity and state and an autonomous execution model with the ability to adapt and be proactive. Software agent infrastructure needs to address agent specific requirements although it can build upon existing research and implementations of distributed system infrastructure.

Services that should be provided by the infrastructure include identity generation, authentication, registration and lookup, message transport, resource management, persistence, logging, and encryption. In additional tools to support administration and management, debugging, visualization, and deployment are needed.

A number of efforts are currently underway to develop software agent infrastructure. The CoABS Grid is an integration framework for multiple agent systems and infrastructures. The Java Agent Services (JAS) effort specifies a standard API for the Java™ Platform from Sun Microsystems. FIPA is a standards organization that has published a number of specifications on infrastructure for supporting interoperability between agents. Grid computing is a more traditional distributed systems approach that also has an impact on agents.

Our recommendation in this area is that research should be continued in both the distributed systems infrastructure area as well as the software agents infrastructure area. Key challenges such as scalability, distributed security, administration and management tools, and deployment must be addressed.

Many of the issues discussed in this chapter are the subject of ongoing work. This needs to be both encouraged and made use of where possible. Other areas currently have a lower commercial priority, such as high-level security and deployment, and as such will require more extensive investigation and development.

References

Agentcities. 2002. On-line reference. http://www.agentcities.org

America On-Line. 2002. AOL Instant Messenger 2002. On-line reference. http://www.aim.com/.

April. 2002. On-line reference. http://www.nar.fujitsulabs.com/aap/.

Argonne National Labs. 2002. On-line reference: http://www.globus.org/ogsa/deliverables/OGSA-January-2002-v3.pdf

Atkinson, M., and Jordan, M. 2000. A Review of the Rationale and Architectures of Pjama: a Durable, Flexible, Evolvable and Scalable Orthogonally Persistent Programming Platform. Sun Microsystems, Inc.

Babaoglu, O., and Marzullo, K. 1993. Consistent Global States. In Distributed Systems, ed. S.Mullender, 55-93. Reading, Mass: Addison-Wesley Publishing Co.

Bellavista, P., Corradi, A., and Stefanelli, C. 1999. A Secure and Open Mobile Agent Programming Environment Proceedings of the Fourth Int'l Symposium on Autonomous Decentralized Systems (ISADS'99), 238-245. Tokyo, Japan: Conference Proceedings, IEEE Computer Society Press

Bellifemine, F., Poggi, A., and Rimassa, G. 2001. JADE: a FIPA2000 compliant agent development environment. Agents 2001: 216-217.

Barkai, D. 2002. Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net. Intel Press.

Birman, K. 1996. Building Secure and Reliable Network Applications. Greenwich, CT: Manning Publications Co.

Bloomer, J. 1992. Power Programming with RPC. Cambridge, Mass: O'Reilly & Associates Inc.

Bradshaw, J.M., Suri, N., Canas, A.J., Davis, R., Ford, K.M., Hoffman, R., Jeffers, R. and Reichherzer, T. 2001. Terraforming Cyberspace. IEEE Computer. July 2001: 48-56.

Bradshaw, J.M. et al., 1999. Agents for the Masses?. IEEE Intelligent Systems Mar./Apr. 1999: 53-63.

Bradshaw, J.M. et al., 1997. "KAoS: Toward an Industrial-Strength Generic Agent Architecture. Software Agents. Mass: AAAI Press/The MIT Press: 375-418.

Clip2. 2002. The Gnutella Protocol Specification v0.4. On-line reference. http://www.clip2.com/GnutellaProtocol04.pdf

Damianou, N. et al. 2000. Ponder: A Language for Specifying Security and Management Policies for Distributed Systems. Version 2.3. TRDOC 2000/I. London. Dept. of Computing. Imperial College of Science, Technology and Medicine.

DataGrid. 2002. On-line reference. http://eu-datagrid.web.cern.ch/eu-datagrid/

Ferreira, L., and Dahab, R. 2002. Bliended-Key Signatures: Securing Private Keys Embedded in Mobile Agents. Applied Computing 2002: 82-86.

FIPA_OS. 2002. On-line reference. http://fipa-os.sourceforge.net/.

Foster, I., and Kesselman, C., 1999. The Grid. Blueprint for a New Computing Infrastructure. San Francisco, CA: Morgan Kaufmann Publishers, Inc.

Foundation for Intelligent Physical Agents. 2002. On-line reference. http://www.fipa.org/.

Foundation for Intelligent Physical Agents. 2001. FIPA Abstract Architecture Specification XC0001J.

Foundation for Intelligent Physical Agents. 2001b. FIPA Agent Management Specification XC00023H.

Foundation for Intelligent Physical Agents. 2001c. FIPA Agent Message Transport Service Specification XC00067D.

Foundation for Intelligent Physical Agents. 2001d. FIPA ACL Message Structure Specification XC00061E.

Foundation for Intelligent Physical Agents. 2001e. FIPA ACL Message Representation in XML Specification XC00071C.

Foundation for Intelligent Physical Agents. 2001f. FIPA KIF Content Language Specification XC00010B.

Foundation for Intelligent Physical Agents. 2001g. FIPA RDF Content Language Specification XC00011B.

Foundation for Intelligent Physical Agents. 2001h. FIPA Communicative Act Library Specification XC00037H.

Foundation for Intelligent Physical Agents. 2001i. FIPA Interaction Protocol Library Specification XC00025E.

Foundation for Intelligent Physical Agents. 2001j. FIPA Ontology Service Specification XC00086D.

Foundation for Intelligent Physical Agents. 2001k. FIPA Domain and Policies Specification PC00089D.

Gimps. 2002. On-line reference. http://www.mersenne.org/prime.htm

Global Grid Forum. 2002. On-line reference. http://www.gridforum.org/

Global InfoTek, Inc. 2002. Control of Agent Based Systems. On-line reference. http://coabs.globalinfotek.com/.

Gnutella. 2002. On-line reference. http://www.gnutella.com/.

Graves, M., Holmback, H., and Bradshaw, J.M. 2002. Agent Conversation Policies. Handbook of Agent Technology, ed. Bradshaw, J.M. Mass. AAAI Press/The MIT Press, submitted for publication.

Grimstrup, A., Gray, R.S., Kotz, D., Cowin, T., Hill, G., Suri, N., Chacon, D., Hofmann, M. 2001. Write Once, Move Anywhere: Toward Dynamic Interoperability of Mobile Agent Systems. Technical Report TR2001-411, Computer Science, Dartmouth College.

Howes, T., Smith, M. 1997. LDAP. Programming Directory-Enabled. Applications with Lightweight Directory Access Control. Indianapolis, IN: Macmillan Technical Publishing.

ICQ. I Seek You. 2002. On-line reference. http://web.icq.com/.

IBM. Web Sphere MQ Series. 2002. On-line reference. http://www3.ibm.com/software/ts/mqseries/.

JADE. 2002. On-line reference. http://sharon.cselt.it/projects/jade/.

Jansen, W., and Karygiannis, T. 1999. Mobile Agent Security. NIST Technical Report.

Johnston, W., Gannon, D., and Nitzberg, B. 1999. Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. In 8th IEEE International Symposium on High Performance Distributed Computing. Redondo Beach, California.

Kerberos. The Network Authentication Protocol. 2002. On-line reference. http://web.mit.edu/kerberos/www/.

Knoll, G., Suri, N., and Bradshaw, J.M. 2001. Path-Based Security for Mobile Agents. Proc. 1st Int'l Workshop Security of Mobile Multiagent Systems and 5th Int'l Conf. Autonomous Agents (Agents 2001): 54-60. New York: ACM Press.

Knowledge Systems Laboratory. 2002. Knowledge Interchange Format (KIF). On-line reference. http://www-ksl.stanford.edu/knowledge-sharing/kif/.

Magnin, L., Pham, V.T., Dury, A., Besson, N., and Thiefaine, A. 2002. Our Guest Agents are Welcome to Your Agent Platforms. Applied Computing 2002. Proceedings of the 2002 ACM Symposium on Applied Computing: 107-114. Madrid, Spain: Universidad Carlos III De Madrid.

McCabe, F.G. 2002. Java Agent Services (JSR87). On-line reference. http://jcp.org/jsr/detail/87.jsp/.

McCabe, F.G., and Clark, K.L. 1994. April Agent PRocess Interaction Language. In Intelligent Agents: Theories, Architectures, and Languages. Berlin: Springer-Verlag.

Microsoft Corp. 2002. Windows Messenger. On-line reference. http://messenger.msn.com/.

Microsoft Corp. 2002b. .NET Passport. On-line reference. http://www.microsoft.com/myservices/passport/overview.asp

Mullender, S. 1993. Distributed Systems. Reading, Mass: Addison-Wesley Publishing Co.

Murphy, A., and Picco, G.P. 1999. Reliable Communication for Highly Mobile Agents. First Int'l Symposium on Agent Systems and Applications (ASA'99). Third Int'l Symposium on Mobile Agents (MA'99).

Napster, Inc. 2002. Napster Web Page. On-line reference. http://www.napster.com/.

Oaks, S., and Wong, H. 2000. Jini in a Nutshell: A Desktop Quick Reference. Sebastopol, CA: O'Reilly & Associates.

Object Management Group. 2002. On-line reference. http://www.omg.com/.

Object Management Group. 2002. CORBA/IIOP. On-line reference. http://www.omg.org/technology/documents/recent/corba_iiop.htm

Peer-to-Peer Working Group. 2002. On-line reference. http://www.p2pwg.org/.

Perkins, C.E., Woolf, B., and Alpert, S.R. 1998. Mobile IP Design Principles and Practices. Prentice Hall PTR.

Pinsdorf, U., and Roth, V. 2002. Mobile Agent Interoperability Patterns and Practice. In 9th Annual IEEE Interantional Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'02). Lund, Sweden: Lund University.

Remote Method Invocation. (Java RMI). 2002. On-line reference. http://java.sun.com/products/jdk/rmi/.

Sakamoto, T., Sekiguchi, T., and Yonezawa, A. 2000. Bytecode Transformation for Portable Thread Migration in Java. In Second Int'l Symposium on Agents Systems and Applications and Fourth Interantional Symposium on Mobile Agents, ASA/MA 2000, eds. D. Kotz and F.Mattern, 16-28. Zurich, Switzerland.

Sander, T., and Tschudin, 1997. C.F. Protecting Mobile Agents Against Malicious Hosts. Mobile Agents and Security, ed. G. Vigna. Springer-Verlag.

Schneier, B. 1996. Applied Cryptography. New York: John Wiley & Sons, Inc.

Simple Object Access Protocol (SOAP 1.1). 2000. W3C Note 08. On-line reference. http://www.w3.org/TR/SOAP/.

Stallings, W. 1998. Cryptography & Network Security: Principles & Practice. Prentice-Hall.

Stevens, W.R. 1994. TCP/IP Illustrated, Volume 1. The Protocols. Reading, Mass: Addison-Wesley Pub. Co.

Stevens, W.R. 1996. TCP for Transactions, HTTP, NNTP, and the UNIX® Domain Protocols (TCP/IP Illustreated, Volume 3). Reading, Mass: Addison-Wesley Pub. Co.

Stevens, W.R. 1999. UNIX Network Programming. Volume 2. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Sun Microsystems, Inc. 2002. Java Community Process. Community Development of Java Technology Specifications. On-line reference. http://jcp.org/

Sun Microsystems, Inc. Java Message Service API. 2002. On-line reference. http://java.sun.com/products/jms/.

Sun Microsystems, Inc. Java 2 Platform, Micro Edition (J2ME). 2002. On-line reference. http://java.sun.com/j2me/.

Sun Microsystems, Inc. 2002. Jini.org – The Community Resource for Jini Technology. On-line reference. http://www.jini.org.

Sun Microsystems, Inc. 1999. Jini Architectural Overview: Technical White Paper. Palo Alto, California.

Suri, N., Bradshaw, J.M., Breedy, M., Ford, K.M., Groth, P.T., Hill, G.A., and Saavedra, R. 2002. State Capture and Resource Control for Java: The Design and Implementation of the Aroma Virtual Machine. On-line reference. http://nomads.coginst.uwf.edu/.

Suri, N., Groth, P., and Bradshaw, J.M. 2001.While You're Away: A System for Load-Balancing and Resource Sharing Based on Mobile Agents. Proc. First IEEE/ACM Int'l Symposium Cluster Computing and the Grid: 470-473. Los Alamitos, CA: IEEE CS Press.

Suri, N., 2000. et al., Strong Mobility and Fine Grained Resource Control in Nomads. Proc. 2nd Int'l Symp. Agents Systems and Applications and the 4th Int'l Symp. Mobile Agents (ASA/MA 2000), Berlin 2000: 2-15.

Suri, N., Bradshaw, J.M., Breedy, M.R., Groth, P.T., Hill, G.A., Jeffers, R., and Mitrovich, T.S. 2000b. An Overview of the NOMADS Mobile Agents System. Sixth ECOOP Workshop on Mobile Object System.

Tan, H. K., Moreau, L. 2002. Certificates for Mobile Code Security. Applied Computing 2002. Proceedings of the 2002 ACM Symposium on Applied Computing: 76-81. Madrid, Spain: Universidad Carlos III De Madrid.

Tanenbaum, A., van Steen, M. 2002. Distributed Systems Principles and Paradigms. Upper Saddle River, N.J., Prentice Hall Inc.

TeraGrid. 2002. On-line reference. http://www.teragrid.org/

Truyen, E.,Robben, B., Vanhaute, B., Coninx, T., Joosen , W., and Verbaeten, P. 2000. Portable Support for Transparent Thread Migration in Java. In Second Int'l Symposium on Agents Systems and Applications and Fourth Interantional Symposium on Mobile Agents, ASA/MA 2000, eds. D. Kotz and F.Mattern, 29-43. Zurich, Switzerland.

UDDI. 2002. On-line reference. http://www.uddi.org

UDDI Technical White Paper. 2000. On-line reference. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf

Urzok, A. 2001. Scalable Asynchronous Bot Execution Architecture. Personal Communication with Author.

Vigna, G. 1998. Cryptographic Traces for Mobile Agents. In Mobile Agents and Security, ed. G. Vigna. Berlin: Springer-Verlag: 137-153.

Wooldridge, M., and Jennings, N.1998. Pitfalls of Agent-Oriented Development. In the Proceedings of the Second International Conference on Autonomous Agents, [AA98]. 385-391.

World-Wide-Web Consortium. 2002. Resource Description Framework (RDF). On-line reference. http://www.w3.org/RDF/.