Object Management Group

250 First Ave, Suite 201
Needham, MA 02494

Telephone: +1-781-444-0404

Agent Technology

Green Paper

OMG Document agent/00-09-01
Version 1.0

1 September 2000

Agent Platform Special Interest Group

Homepage: http://www.objs.com/agent/index.html

This paper presents a discussion of technology issues considered in the Agent Special Interest Group of the Object Management Group The contents of this paper are presented to create discussion in the computer industry on this topic; the contents of this paper are not to be considered an adopted standard of any kind. This paper does not represent the official position of the Object Management Group.

Table of Contents

1. Introduction

1.1. Purpose of the Green Paper

Typically, when the word "agent" is used, many people conjure up images from Web spiders and search bots. Yet, these same people feel that there is something more to the concept. For those who wish to know more, the purpose of this paper is to provide:

- a basic understanding of agents and agent-based technology.
- a summary of how agent-based technology is currently being employed.
- brief discussions of the major issues being addressed by agent research and development.
- indications where agent technology is relevant to the OMG—and vice versa.
- suggestions for standardizing areas of this new technology using the OMG process.

1.2. Intended Audience(s)

This green paper is intended for the OMG as a whole and, specifically, those task forces and special interest groups that are working in areas that might involve agent technology. The purpose here is to encourage synchronization of those areas employing agents and agent-based technology.

This paper is also intended for the agent technology industries. Their thoughts and suggestions will help address the commercial significance of the concepts and directions established in the paper.

1.3. Agent Special Interest Group Mission

The mission of the Agent Special Interest Group is to provide a forum for identifying and building consensus and convergence in the industry around agent technology development.[1] One of the first efforts of this group is this green paper—which has two purposes:

- to address the key features of agent technology, and
- to determine what and why the next steps might be for the Agent Special Interest Group.

1.4. Writing Team

Editor: James Odell.

Contributors: David Kerr, Broadcom Eireann Research Ltd.
Heimo Laamanen, Sonera
David Levine, IBM
Gregory Mack, Booz-Allen & Hamilton, Inc.

[1] Agent SIG Mission Statement agent/00-08-01

David Mattox, Mitre Corporation

Francis McCabe, Fujitsu Labs

Stephen McConnell, OSM sarl

James Odell

Kimmo Raatikaine, University of Helsinki

Kate Stout, Sun Microsystems, Inc.

Craig Thompson, Object Services and Consulting

1.5. Questions & Comments

Please address content-related questions or feedback to the Agent Special Interest Group (agents@omg.org) and to the paper editor James Odell (jodell@compuserve.com). In the interests of mailing list efficiency, please address errata or omissions directly to the editor only.

2. What is an Agent?

2.1. Definition of Agent

An agent can be a person, a machine, a piece of software, or a variety of other things.
The basic dictionary definition of agent is *one who acts*. However, for developing IT systems, such a definition is too general: IT-related agents need additional properties. Some of the properties that agents may possess in various combinations include:

- **Autonomous -** is capable acting without direct external intervention. It has some degree of control over its internal state and actions based on its own experiences.
- **Interactive -** communicates with the environment and other agents.
- **Adaptive -** capable of responding to other agents and/or its environment to some degree. More advanced forms of adaptation permit an agent to modify its behavior based on its experience.
- **Sociable -** interaction that is marked by friendliness or pleasant social relations, that is, where the agent is affable, companionable, or friendly.
- **Mobile -** able to transport itself from one environment to another.
- **Proxy -** may act on behalf of someone or something, that is, acting in the interest of, as a representative of, or for the benefit of some entity.
- **Proactive -** goal-oriented, purposeful. It does not simply react to the environment.
- **Intelligent -** state is formalized by knowledge (i.e., beliefs, goals, plans, assumptions) and interacts with other agents using symbolic language.
- **Rational -** able to choose an action based on internal goals and the knowledge that a particular action will bring it closer to its goals.
- **Unpredictable -** able to act in ways that are not fully predictable, even if all the initial conditions are known. It is capable of nondeterministic behavior.
- **Temporally continuous -** is a continuously running process.
- **Character -** believable personality and emotional state.
- **Transparent and accountable -** must be transparent when required, yet must provide a log of its activities upon demand.
- **Coordinative -** able to perform some activity in a shared environment with other agents. Activities are often coordinated via a plans, workflows, or some other process management mechanism.
- **Cooperative -** able to coordinate with other agents to achieve a common purpose; nonantagonistic agents that succeed or fail together. (*Collaboration* is another term used synonymously with cooperation.)
- **Competitive -** able to coordinate with other agents except that the success of one agent implies the failure of others (the opposite of cooperative).

- **Rugged -** able to deal with errors and incomplete data robustly.
- **Trustworthy -** adheres to Laws of Robotics and is truthful.

An industry-standard definition of agent has not yet emerged. Most agree that agents bound for IT systems are not useful without at least the first three of the above properties. Others require IT agents to possess all of the properties listed above to varying degrees. At a minimum, an IT agent is generally regarded to be an autonomous entity that can interact with its environment. In other words, it must be able to perceive its environment through sensors and act upon its environment with effectors.

2.2. Important Forms of IT Agents

The agents found in IT systems have special requirements: they must execute as software, hardware, robotics, or a combination of these. Agent developers have identified several forms of agents that are important for IT system development. The list of agent characteristics presented earlier addresses some of these requirements. Additionally, since IT systems have special needs, software- and hardware-related forms must be considered. Those forms considered the most important to agent developers today are discussed below.

2.2.1. Software agents

Software agents are a more specific kind of agent. At a minimum, a software agent is defined as *an autonomous software entity that can interact with its environment*. In other words, they are agents that are implemented using software. This means that they are autonomous and can react with other entities, including humans, machines, and other software agents in various environments and across various platforms.

Basically, software agents are design patterns for software. Tools, languages, and environments can be specifically developed to support the agent-based pattern. However, the agent design pattern can also be implemented using object-oriented tools, languages, and environments—or any other tool, language, and environment that is capable of supporting software entities that are autonomous, interactive, and adaptive. Agent-based tools are preferable primarily because the agent design patterns are inherent in the software—rather than explicitly programmed. In other words, object technology can be used to *enable* agent-based technology, but the autonomous, interactive, and adaptive nature required by agents is not currently supported within OO technology. While these properties can be (and are being) added to the OO approach, the design patterns for agents and agent-based software are not fully and directly supported.

(Note: from this point on, the term *agent* will usually mean *software agent.* While many of the ideas and technology can be generalized to support machines and people, the emphasis in this Executive Report will be on software implementation.)

2.2.2. Autonomous agents

When an agent has a certain independence from external control, it is considered autonomous. Autonomy is best characterized in degrees, rather than simply being present or not. To some extent, agents can operate without direct external invocation or intervention. Without any

autonomy, an agent would no longer be a dynamic entity, but rather a passive object such as a part in a bin or a record in a relational table. Therefore, autonomy is considered by FIPA and the OMG Agent Special Interest Group to be a required property of agents.

Autonomy has two independent aspects: *dynamic autonomy* and *unpredictable autonomy*. Agents are dynamic because they can exercise some degree of activity. An agent can have some degree of activity—ranging from simply passive to entirely proactive. For example, while ants are basically reactive, they do exhibit a small degree of proactivity when they choose to walk, rest, or eat. A supply-chain agent can react to an order being placed, yet be proactive about keeping its list of suppliers up to date.

Agents can react not only to specific method invocations but to observable events within the environment, as well. Proactive agents will actually poll the environment for events and other messages to determine what action they should take. (To compound this, many agents can be engaged in multiple parallel interactions with many other agents—magnifying the dynamic nature of the agent system.) In short, an agent can decide when to say "go."

Agents may also employ some degree of unpredictable (or nondeterministic) behavior. When observed from the environment, an agent can range from being totally predictable to completely unpredictable. For example, an ant that is wandering around looking for food can appear to be taking a random walk. However, once pheromones or food are detected, its behavior becomes quite predictable. In contrast, the behavior of a shopping agent might be highly unpredictable. Sent out to choose, negotiate, and buy a birthday present for your mother-in-law, the agent might return with something odd indeed or with nothing at all. In other words, the agent can also say "no."

2.2.3. Interactive agents

Interactive agents can communicate with both the environment and other entities and can be expressed in degrees. On one end of the scale, object messages (method invocation) can be seen as the most basic form of interaction. A more complex degree of interaction would include those agents that can react to observable events within the environment. For example, food-gathering ants do not invoke methods on each other; their interaction is indirect through physically affecting the environment. In other words, ants do not receive method invocations; they receive events regarding the state of the environment. Even more complex interactions are found in systems where agents can be engaged in multiple, parallel interactions with other agents. Here, agents begin to act as a society. Finally, the ability to interact becomes most complex when systems involving many heterogeneous agents can coordinate through cooperative and/or competitive mechanisms (such as negotiation and planning).

While we can conceive of an agent that cannot interact with anything outside of itself, the usefulness of such an entity for developing agent-based systems is questionable. Therefore, interaction is considered by FIPA and the OMG Agent Special Interest Group to be a required property of agents.

2.2.4. Adaptive agents

An agent is considered *adaptive* if it is capable of responding to other agents and/or its environment to some degree. At a minimum, this means that an agent must be able to *react* to a simple stimulus—to make a direct, predetermined response to a particular event or environmental signal. Thermostats, robotic sensors, and simple search bots fall into this category.

Beyond the simple reactive agent is the agent that can *reason*. Reasoning agents react by making inferences and include patient diagnosis agents and certain kinds of data-mining agents.

More advanced forms of adaptation include the capacity to *learn* and *evolve*. These agents can change their behavior based on experience. Common software techniques for learning are neural networks, Bayesian rules, credit assignments, and classifier rules. Examples of learning agents would be agents that can approve credit applications, analyze speech, and recognize and track targets. A primary technique for agent evolution usually involves genetic algorithms and genetic programming. Here, agents can literally be bred to fit specific purposes. For example, operation plans, circuitry, and software programs can prove to be more optimal that any product that a human can make in a reasonable amount of time.

An agent that can not respond to its environment or to other agents is another kind of agent whose usefulness is questionable for developing agent-based systems. Therefore, adaptation is considered by FIPA and the OMG Agent Special Interest Group to be a required property of agents.

2.2.5. Mobile agents

While stationary agents exist as a single process on one host computer, mobile agents can pick up and move their code to a new host where they can resume executing. From a conceptual standpoint, such mobile agents can also be regarded as itinerant, dynamic, wandering, roaming, or migrant. The rationale for mobility is the improved performance that can sometimes be achieved by moving the agent closer to the services available on the new host.

For example, if an agent wants to obtain information from several sources on different platforms, it could send information requests to each of the platforms using the equivalent of a remote procedure call (RPC). However, if the volume of information exchanged with the remote site is large, issues of traffic and bandwidth must be considered. Also, the agent might be able to process the remote data more effectively than those services offered at the remote site. In either or both of these cases, relocating the agent to each of the various platforms could be a more efficient way of processing. One disadvantage of such mobility is that the remote sites must provide the CPU cycles to support the mobile agent's processing. This not only brings an additional processing burden to the remote site, it also raises three issues: security, billing the agent for its CPU time, and unanticipated scaleability problems.

The ability of an agent to transport itself from one environment to another is not a requirement for agenthood. Nevertheless, mobility is an important property for many agent-based systems—and necessary for a certain class of application.

2.2.6. Coordinative agents

Human organizations exist primarily to coordinate the actions of many individuals for some purpose. That purpose could be to create such structures as profitable business units, charitable organizations, ballet companies, or Little Leagues. Using human organizations as an analogy, systems involving many agents could benefit from the same pattern. Some of the common coordinative agent applications involve supply chains, scheduling, vehicle planning, problem solving, contract negotiation, and product design. Without some degree of coordination, such systems could not be possible—in either human or agent-based systems.

Furthermore, the analogy requires that we consider a heterogeneous population of agents. Human organizations are not constructed with a population of identical individuals doing the same thing; instead, we diversify, delegate, negotiate, manage, cooperate, compete, and so on. The same approach needs to be employed in multiple agent systems. Careful consideration should be given to designing and structuring agent-based systems, however. This will increase the likelihood that agents will be coherent in their behavior. While this limits and controls agent spontaneity, it still preserves agent-level flexibility.

2.2.7. Intelligent agents

After decades, the term *intelligent* has still not been defined (or understood) for artificial system and applying the term now to agents may not be appropriate. Most tend to regard the term *agent* and *intelligent agent* as equivalent. Perhaps this is just an attempt to communicate that agents have more power that conventional approaches. For example, in comparison to relational tables or objects, agents can be thought of somewhat "smarter." Or, it could just be marketing hype. However, it would be fair to say that the notion of intelligence for agents could very well be different than for humans. We are not creating agents to replace humans; instead, we are creating them to assist or supplement humans. A different kind of intelligence, then, would be entirely appropriate.

The current wisdom is that whatever the term *intelligent agent* means, such agents will require a basic set of attributes and facilities. For example, an intelligent agent's state must be formalized by knowledge (i.e., *beliefs*, *goals*, *desires*, *intentions*, *plans*, *assumptions*) and be able to act on this knowledge. It should be able to examine it beliefs and desires, form its intentions, plan what actions it will perform based on certain assumptions, and eventually act on its plans. Furthermore, intelligent agents must be able to interact with other agents using symbolic language. All this sounds like a model of rational human thinking—but we should not be surprised. Once again, agent researchers are using our understanding of how humans think as a model for designing agents.

FIPA and the OMG Agent Special Interest Group do not have a standard definition of intelligent agent. However, the description above provides a general idea of the group's current direction.

2.2.8. Wrapper agents

This agent allows another agent to connect to a non-agent software system/service uniquely identified by a software description. Client agents can relay commands to the wrapper agent and have them invoked on the underlying services. The role provided by the wrapper agent provides

a single generic way for agents to interact with non-agent software systems. (Note: the wrapper agent would probably not be necessary for objects, that is, objects should still be able to coexist in the same environment.)

The wrapper is considered important by FIPA and the OMG's Agents Special Interest Group. It provides a bridge to legacy code and facilitates the reuse of code for an agent's process, or it transforms a static piece of code into an active processing entity.

2.2.9. Other forms of agents

The kinds of agents listed above are the predominate forms considered for every agent-based system. More detailed examination of an application can identify other forms, such as facilitator agents, broker agents, manager agents, and so on. These forms can be more easily thought as roles that an agent can play—rather than the fundamental approach designed into an agent.

2.3. Single versus Multiagent Systems

Many of the early commercial agents were developed for information search. Here, individual agents were launched on a tether to gather predefined kinds of information and return them to the human requester. In other words, these agents were solo operations that had very little—if any—interaction with other agents. Such an approach certainly has its many uses. However, if we look at the human world, such an approach *alone* could not build the societies or support the organizations that humans have come to enjoy. Instead, we set up networks of people that interact for various purposes. Interaction among agents, then, is not sufficient to build agent societies, we need agents that can coordinate—either through cooperation, competition, or a combination of both. These agent "societies" are called *multiagent systems* (MAS).

Multiagent systems, then, are systems composed of agents coordinated through their relationships with one another. For example in a kitchen, the toaster "knows" when the toast is done, and the coffeemaker "knows" when the coffee is ready. However, there is no cooperative environment here—only an environment with single agents. In a multiagent environment, the kitchen becomes more than just a collection of processors. The appliances would be interconnected in such a way that the coffeemaker and the toaster would ensure that the coffee and toast are ready at the same time.

Some of the rationale for multiagent systems are as follows:

- One agent *could* be constructed that does everything, but fat agents represent a bottleneck for speed, reliability, maintainability, and so on (i.e., there are no omnipotent agents). Dividing functionality among many agents provides modularity, flexibility, modifiability, and extensibility.
- Specialized knowledge is not often available from a single agent (i.e., there are no omniscient agents). Knowledge that is spread over various sources (agents) can be integrated for a more complete view when needed.
- Applications requiring distributed computing are better supported by MAS. Here, agents can be designed as fine-grained autonomous components that act in parallel. Concurrent

processing and problem solving can provide solutions to many problems that, up until now, we handled in a more linear manner. Agent technology, then, provides the ultimate in distributed component technology.

To support multiagent systems, an appropriate environment needs to be established. MAS environments:

- provide an infrastructure specifying communication and interaction protocols.
- are typically open and have no centralized designer or top-down control function.
- contain agents that are autonomous, adaptive, and coordinative.

Clearly, single-agent environments are much simpler, because designers do not have to deal with such issues as cooperation, negotiation, and so on. However, the large-scale requirements of industry necessitate approaches that employ coordination and distribution. As such, FIPA and the OMG Agent Special Interest Group are focusing primarily on multiagent systems rather than single agents.

3. Status of Agent Technology

3.1. Introduction

The emergence of agent technology is similar to the stories many other technologies (e.g. relational, object oriented, GUI). As such, some expect to some of history's lessons to repeat themselves:[Cag97]

- Agent technology is not a single, new, emerging technology—but rather the integrated application of multiple technologies.
- Agents are not necessarily a new, isolated form of application. Instead, it can add a new set of capabilities to existing applications.
- Initially, agent functions will emerge within applications, but later—with experience—will become part of the operating system or application environment.
- Agent applications may strengthen the human-computer interaction.
- Ultimately, applications that do not exploit agent support in the operating system will be severely disadvantaged.

While destined for lofty goals, the current state of agent technology is that:

- It is still an active research area.
- Isolated pioneer products are emerging.
- The full set of technologies are not available.
- The technologies are not integrated with one another.
- There is no consensus on how to support agents at the operating system level
- Despite the hype, agent technology is not in widespread use–nor has it been widely accepted as an inevitable trend.
- There are early adopters who can demonstrate the value of agent technology.

3.2. Current Usage of Agents

As suggested above, the agent industry is in an embryonic state. As such, the deployment of agent-based systems and technology are isolated and few, but exist nonetheless—and are in fact on the increase. Currently, several classes of agents have been deployed to some degree.

- **Network and system management agents**
  The telecommunications companies have been the most active in this area, and indeed seem the group most committed to the agent paradigm. Their goal is to use agents to assist in complex system and network management tasks, such as load balancing, failure anticipation, problem analysis and information synthesis.

  One note: There is a confusing overlay of terms in this area. Network management is often

performed by employing *SNMP agents.* These are a particular mechanism for management, but are not the types of agents discussed here. This is just a simple overlap of names.

- **Decision and logistic support agents**
  Mostly deployed in closed environments, utility companies and military organizations use agents for information synthesis and decision support. These systems may alert an operator to a possible problem, provide information in support of a complex decision. They are closely aligned to decision support systems from the traditional AI community.
- **Interest matching agents**
  These are probably the most used agents, and most users don't even know they are using them. The interest matching agents are used by commercial Web sites to offer recommendations, such as *If you liked "Frank Sinatra's Greatest Hits" you might also like "Tony Bennett's Songs for a Summer Day".*
  Based on Patti Maes work at MIT Media Labs, and later at Firefly, these agents observe patterns of interest and usage in order to make recommendations. They have been deployed at amazon.com, and various CD and video sales sites.
- **User assistance agents**
  These agents operate at the UI level, offering information or advice to users. They are sometimes represented visually as a cartoon advisor. Companies such as Microsoft, Lotus, and Apple have shown the most interest in this area. The best known example of an agent in common use is the animated help characters used in Microsoft Office products. These agents use bayesean networks to analyze and predict possible topics that the user may need help with.
- **Organizational structure agents**
  These agents are structure to operate in a similar manner as human organizations. For example, multiagent supply chain systems would have agents playing the roles such as that of buyers, suppliers, brokers, stock, orders, line items, and manufacturing cells. Operations systems would have resource agents, material agents, process agents, and so on.

Each of these represent some aspect of agents' features. However , none of these represent the full range of possible agent features.

3.3. Current Kinds of Agent Applications

The current kinds of applications that employ agents is still limited. Once the concepts become more accepted and more tools become available, the agent-based approach will become more imbedded it IT applications.

(We need more words for 3.3.1 through 3.3.4)

3.3.1. Enterprise applications

- Smart documents (i.e., documents that `know' that that they are supposed to be processed)
- Goal-oriented enterprise (i.e., work-flow on steroids)
- Role and personnel management (i.e., dynamically attaching roles and capabilities to people)

3.3.2. Business-to-business applications

- Market making for goods and services
- Brokering of the above
- Team management

3.3.3. Process control

- Intelligent buildings (e.g., smart heating/cooling, smart security)
- Plant management (e.g., refinery)
- Robots

3.3.4. Personal agents

- Email and news filters
- Personal schedule management
- Personal automatic secretary

3.3.5. Information management tasks

- **Searching for information -** The amount of information available over a corporate intranet stretches the capability of most users to effectively retrieve useful information. Search agents contain domain knowledge about various information sources. This knowledge includes the types of information available at each source, how to access that information and other potentially useful knowledge such as the reliability and accuracy of the information source. Search agents use this knowledge to accomplish specific search tasks..
- **Information filtering -** Another common task for agents. Information filtering agents attempt to deal with the problem of information overload by either limiting or sorting the information coming to a user. The basic idea is to develop an on-line surrogate for a user that has enough knowledge about the user's information needs so that it can select only those documents that would be of interest. These types of agents usually function as gatekeepers by preventing the user from being overwhelmed by a flood of incoming information. Filtering agents also work in conjunction with, or are sometimes incorporated into, search agents in order to keep the results from searches down to reasonable levels. Typically, filtering agents incorporate machine learning mechanisms. This allows them to adapt to the needs of each user and to provide more precision than that typically provided by keyword filtering approaches.
- **Information monitoring -** Many tasks are dependent on the timely notification of changes in different data sources. A logistics planner my develop a plan for moving equipment from one location to another, but the execution of that plan could be disrupted by the onset of bad weather at a refueling stop. The logistics planner would like to know of any events that would be likely to effect his plan as soon as they happen. Agents are useful for monitoring distributed data sources for specific data. Being software constructs, they have the patience necessary to constantly monitor data sources for changes. Alternately, mobile agents can be

dispatched to remote or otherwise inaccessible locations to monitor data that the user might not normally have access to.

- **Data source mediation -** The data management landscape is populated with a multitude of different systems, most of which don't talk to each other. Agents can be used as mediators between these various data sources, providing the mechanisms that allow them to interoperate. The SIMS Project at ISI developed an information mediator that provides access to heterogeneous data and knowledge bases. This mediator can be used to create a network of information gathering agents, each of which has access to one or more information sources. These agents use a higher level language, a communications protocol, and domain-specific ontologies for describing the data contained in their information sources. This allows each agent to communicate with the others at a higher semantic level.
- **Interface agents / personal assistants -** An interface agent is a program that is able to operate within a user interface and actively assist the user in operating the interface and manipulating the underlying system. An interface agent is able to intercept the input from the user, examine it, and take appropriate action. While interface agents are not directly related to data management, they have to potential to play a large role in assisting users of data management systems. This becomes increasingly important as data management systems become more distributed and group together to form large, complex systems of systems. Agents in the interface can function as a bridge between domain knowledge about the data management systems and the user. These agents could assist users in forming queries, finding the location of data, explaining the semantics of the data among other tasks. Examples of this include intelligent tutoring systems and web browsing assistants [Lieb95]. In addition, Microsoft is now including interface agents in its desktop products to watch the actions of users and make appropriate suggestions.

3.3.6. Nomadic Computing Applications

Current and future development in the area of wireless data communications and mobile computers enable to a great extent mobile computing. The environment of mobile computing is very different compared to today's environment of traditional distributed systems in many respects. Bandwidth, latency, delay, error rate, interference, interoperability, computing power, quality of display, among other things may change dramatically as a nomadic end-user moves from one location to another — from a computing environment to another, e.g. from a wireline LAN via a wireless LAN (from a office) to a GPRS / UMTS network (to the field). The variety of mobile workstations, handheld devices, and smart phones, which nomadic users use to access Internet services, increases at growing rate. The CPU power, the quality of display, the amount of memory, software (e.g. operating system, applications), hardware configuration (e.g. printers, CDs) , among other things ranges from a very low performance equipment (e.g. hand held organizer, PDA) up to very high performance laptop PCs. All these cause new demands for adaptability of Internet services. For example, PDAs cannot display properly high quality images designed to be watched on high resolution displays, and as nomadic users will be charged based on the amount of data transmitted over the GPRS network, they will have to pay for bits that are totally useless for them.

- **The nomadic end user**
  The nomadic end-user confronted with these circumstances would benefit from having the following functionalities provided by the infrastructure: information about expected performance provided by software agents, software agents controlling over the transfer operations, a condition-based control policy, capability to work in disconnected mode, advanced error recovery methods, and adaptability.
  The ability to automatically adjust to changes mentioned above in a transparent and integrated fashion is essential for nomadicity [Klei97]—nomadic end-users are usually professionals in other areas than computers, and today's mobile computer systems are already very complex to use as a productive tool; thus, they need all the possible support, which an agent based distributed system could deliver. Adaptability to the changes in the environment of nomadic end-users is the key issue. Software agents could play a significant role in implementing adaptability. One agent alone is not able make the decision how to adapt, and therefore adaptation is a co-operation effort carried out by several agents. Therefore, there should be at least some level of cooperation between adapting agents.
- **Simplified and faster service deployment**
  Software agent technology (e.g. agents, platforms and conversation protocols) could form a powerful tool:
  - to model new nomadic services at a high abstraction level, and
  - to implement the model by (automatically) integrating various software agents together to perform the service.
- **Increasing availability, reliability, and support for disconnection**
  Software agent technology forms a natural solution to increase availability and reliability and to support disconnected operations by acting autonomously on behalf of a nomadic end user both in the fixed network side and in the mobile computer side while the mobile computer is disconnected from a network.
- **Nomadic computing state of the art**
  Today, no commercially available applications exist that use software agent technology to carry out the above tasks. In addition, there are only few research projects targeting to solve the problems in nomadic computing with software agent technology [Amase, Cameleon, Monads]. Therefore, the state of the software agent technology in this field is very preliminary.
- **Requirements for software agent technology**
  The most important requirement stated by nomadic applications is adaptability to various and highly varying wireless data communications and to very different kinds of mobile terminals.
  General requirements for the agent technology can be summarized as follows:
  - Fast application and service deployment.
  - New types of intelligent, personalized services.
  - User friendly, intuitive end-user interface.
  - Short term (1-30 minutes) prediction of available Quality-of-Service.

- Prediction of end-user geo-location in the near future (1/4-4 hours).

- **Related work on nomatic applications**
  FIPA [FIPA] has addressed nomadic applications. Technical Committee TC E - Nomadic Application Support has developed a specification (current status: draft) that specifies 1) monitor agent and control agent based infrastructure and 2) bit-efficient data representation of ACL to be utilized in the environment of nomadic applications.

3.4. Technologies Currently Being Used

Almost all of the systems being built today are being built with

- Programming language: Java or C++
- Communication language: (for communicating agents): KQML or FIPA ACL
- Represented as strings or XML documents
- Content language: KIF or SL1
- Most mobile agents are moved by using Java serialization

The first commercial tool sets for building agents have entered the market within the last year. Some are purchasable and some can be downloaded for free. These agent building systems vary widely in functionality don't adhere to any standards. Agents built in one system will not work in others. There is no uniform support for communications protocols across these tools either. The following table gives an overview of commercial agent systems available at the time of this writing.

Product	Company	Language	Description
AgentBuilder®	Reticular Systems, Inc.	Java	Integrated agent and agency development environment
AgenTalk	NTT/Ishida	LISP	Multi-agent coordination protocols
Agent Building Environment	IBM	C++, Java	Environment
Agent Development Environment	Gensym	Java	Environment
Agentx	International Knowledge Systems	Java	Agent development environment
Aglets	IBM Japan	Java	Mobile agents
Concordia	Mitsubishi Electric	Java	Mobile agents
DirectIA SDK	MASA	C++	Adaptive agents
Gossip	Tryllian	Java	Mobile agents
Grasshopper	IKV++	Java	Mobile agents
Infosleuth	MCC	Java, Perl, Tk/tcl	Cooperative information agent environment
iGEN	CHI Systems	C/C++	Cognitive agent toolkit
Intelligent Agent Factory	Bits & Pixels	Java	Agent development tool

Intelligent Agent Library	Bits & Pixels	Java	Agent library
JACK Intelligent Agents	Agent Oriented Software Pty. Ltd.	JACK Agent Language	Environment
Jumping Beans Engineering	Ad Astra	Java	Mobile components
Kafka	Fujitsu	Java	Agent library
LiveAgent	AgentSoft Ltd.	Java	Internet agent construction
Microsoft Agent	Microsoft Corporation	Active X	Interface creatures
Swarm	Swarm Development Group	Objective C, Java	Multiagent simulation
Versatile Intelligent Agents (VIA)	Kinetoscope	Java	Agent building blocks
Voyager	Object Space	Java	Agent-enhanced ORB

4. Key Issues for Agent Technology

As mentioned earlier, agent technology requires that agents be autonomous, interactive, and adaptive. To support these properties, the technology to support agents has to address many key issues. This section discusses each of these issues. it also indicates which standard services and specifications might support these issues.

4.1. Agent Communication

Currently, one of the most important areas for standardization is agent communication. If every designer developed a different means of communicating between agents, our agent systems would be worse than a tower of Babel. Not only would the content and meaning of a communication likely be different, but the means of communication could occur in a variety of ways.

4.1.1. Agent communication languages

Messages must have a well defined semantics that is computational and visible. Thus, we need standardized agent communication languages (ACL), so that different parties can build their agents to interoperate. Furthermore, they must have a formal semantics so that different implementations preserve the essential features of the ACL. By specifying an ACL, we effectively codify the basic elements of interaction that can take place between agents.

Possible implementations:

- KQML
- Arcol and FIPA
- KIF
- XML-based

4.1.2. Message transportation mechanism

In agent environments, messages should be schedulable, as well as event driven. They can be sent in synchronous or asynchronous modes. Furthermore, the transportation mechanism should support unique addressing as well as role-based addresses (i.e., “white page” versus “yellow page” addressing). Lastly, the transportation mechanism must support unicast, multicast, and broadcast modes and such services as broadcast behavior, non-repudiation of messages, and logging.

Possible implementations:

- CORBA
- OMG Messaging Services
- JAVA messaging service
- RMI
- DCOM
- Enterprise Java Beans Events

4.1.3. Ontology communication

Agent communication implies that concepts will be part of the communication among agents. Furthermore, agents can have different terms for the same concept, identical terms for different concepts, and different class systems. A common ontology, then, is required for representing the knowledge from various domains of discourse. The two primary challenges here are building them and linking them.

Possible implementations:

- UML
- MOF
- OKBC
- XML schema

4.1.4. Agent interaction protocols

Agents can interact in various patterns called *interaction protocols* (which are also known as *conversation* or *communication protocols*). Cooperative agents work toward a common goal. For example, to produce a coherent plan, agents must be able to recognize subgoal interactions and either avoid or resolve them. Partial Global Planning does not assume any distribution of sub-problems but allows the agents to coordinate themselves. Joint intention frameworks require joint "commitment." The Shared Plan model is based on the mental attitude—the intention to act. Joint Responsibility is based on a joint commitment to the team's goal and the recipe for attaining that goal. Self-interested multiagent interactions are usually centered around negotiation and contract nets. Whichever kind of conversation is chosen, the pattern—or *protocol*—of that conversation must be understood by the participating agents. Each protocol is a pattern of interaction that is formally defined and abstracted away from any particular sequence of execution steps. Standards and guidelines here would greatly accelerate our usage of agents, because they could then "converse" without requiring human intervention.

Possible implementations:
Electronic Commerce Domain Specifications
Community and Collaboration Framework

4.2. Agent Internals

As we design and build agents that are more intelligent, we need to address contructs and notions that must be supported by the each agent internally, such as goals, adaptation, and understanding.

4.2.1. Goal representation and manipulation

An agent's *goals* are the desires, or future states, on whose fulfillment the agent would prefer to act. Goals, then, help the agent determine what actions to take under particular circumstances. Many theories of goals exist. In the early work on problem solving, a goal was a state to be achieved or a proposition to be made true. However, agent goals could be seen as divided into various classes: achievement goals (long-term goals), satisfaction goals (recurrent goals such as resource gathering), preservation goals (for perserving life and property), and delta goals (i.e.,

"other" state changes). Also, in the BDI (beliefs, desires, intents) approach, *desires* describe the agent's goals (sometimes including a motivational aspect) and *intentions* characterize the goals (desires) that the agent has selected to work on. Goals can have sub-goals, particularly in problem solving. We need to address both the maintenance and the communication of goals.

Possible implementations:
CosTrader constraint language and the CosNotification language extensions
OCL
XML-based

4.2.2. Adaptation

Adaptive agents can be simply *reactive* or they can be *deliberative*. Deliberative agents can learn and/or evolve; that is, they can change their behavior based on their experience with other agents and the environment. The are many techniques and philosophies for adaptation. However, the mechanisms for realizing them can often be implemented in many different ways, using many different languages and development environments—environments that can be far from being agent-based. Choosing that adaptation approach and incorporating it into the agent can be technically difficult without a rich adent development environment.

4.2.3. Procedural versus declarative process specifications

Procedural approaches specify how a computation should proceed; declarative approaches specify what an operation should do rather than how. Both approaches have their merits and therefore both should be considered.

Possible implementations:
UML Activity Diagrams
UML process specification language
Workflow Process Definition RFP

4.2.4. Comprehension of an environment

In order for an agent to recognize features and characteristics of an environment, interfaces are required through which an agent can obtain service information. The CORBA 2.2 specification provides the get_service interface (PIDL) at the level of the ORB. An equivalent interface is required to support the registration and retrieval of information concerning available facilities and applicable policies. Such interfaces should enable an agent both to evaluate an environment and to register itself with the environment as a potential services provider.

Possible implementations:

- Resource Registration and Discovery (draft) RFP

4.3. Life-Cycle Management

Agents will be software running in software environments. As such, they must have well understood mechanisms for such activities as starting, stopping, being managed, and being traced. Some agents are implemented as mobile code, which introduces additional lifecycle issues, such as permissions to run, permissions to perform certain tasks, and to have communication occur at

different locations other than its original starting point. Finally agents can evolve and can possibly clone themselves, which introduces issues related to the delegation of responsibilities and permissions.

As we examine lifecycle management, we must also examine the software environments in which agents will run. These can be very small, intermittently connected devices like cell phones or Personal Digital Assistants, to very large clusters of servers capable of running huge numbers of agents. The requirements for each environment are quite different, and each must be accommodated.

Possible implementations:
Mobility: extensions to CosLifeCycle::LifeCycleObject move()

4.3.1. Virtual existence or persistence

Agents may logically exist for indeterminate periods of time during which they may display dormant behavior. Interfaces supporting agent lifecycles need to consider requirements for "logical" existence of possibly very large numbers of agents as opposed to in-memory physical existence. This has a number of implications around storage, messaging and communication, and management.

- **Persistence of agents**
  In systems that currently support long-lived agents, agents can be "put to sleep" and saved (usually to a physical disk, although this is just an implementation strategy). When the agent is persistent, the current state and data are preserved, and are restored when the agent is "awakened." There are often other state transitions associated with "putting the agent to sleep" and "awakening" it, having to do with querying whether it is currently in a situation where this can successfully be performed, and on awaking, a notification that time has elapsed and it might want to reassess the environment.
- **Messaging while "asleep"**
  A variety of messaging models are possible for agent systems. Some may need a full robust event-style model with guaranteed message delivery. Others may need a far more casual model where an agent seeing a message of interest may act on the information. There may be messaging models for direct communications between two agents, for a store-and-forward approach, and for a publish-and-subscribe model. System events may also be passed via messages or some other mechanism. Design decisions about what intersection of these strategies to use depend on the type of agent applications being built.
  These design decisions also impact how messages are handled while an agent is "asleep." If an agent has to be awakened with each incoming message to see if the message is relevant, storing the agent to save running many in-memory copies would be counterproductive. Models must be designed to optimize delivery of messages. There will likely be many different models for this, but over time certain design patterns will emerge.
- **Managing agents**
  Agents that are currently "asleep" also need to be managed. In large distributed systems, agents could be moved to a new server in order to provide dynamic load balancing.

Possible implementations:
POA servant management provides explicit support for the management of many logical instances and services enabling object activation (waking up).
PSS (Persistence State Service) provides a set of standard servant APIs for managing the persistent state of objects.

Business Object Domain Specifications: Task and Session 2.0 provides a framework under which large numbers of information and process centric objects can be managed from a user's viewpoint. Under the specification, a user maintains tasks, where each task represents a user-unit-of-work. The task is itself a view of a process or sequence of sequentially executed processes where such processes may be implemented as agents.

4.3.2. History

Mechanisms are required to provide a historical recording of the agent's actions—so that agent behavior can be audited and that agents can evaluate prior actions. This could include a range of situations, from merely obtaining an agent's state to providing a comprehensive log of events and actions for the agent.

4.3.3. Dynamic and multiple classification

During an agent's lifecycle, the interfaces exposed by the agent may be dynamic, reflecting changes in its state or environment. Mechanisms are required to support the existence of multiple interfaces relative to a single agent identity. Furthermore, control mechanisms must be established to enable and disable an agent's features. Such mechanisms are particularly useful when multiple roles are being supported. Depending on the situation, the roles or behavior of individual agents can change. For example in ant societies, workers can become foragers when food availability increases, and nest-maintenance workers can become patrollers when foreign ants intrude. However, once an ant is allocated to a task outside the nest, it never returns. This implies that an agent can both change its role (dynamic classification) and assume more than one role at any moment in time (multiple classification). This is an important issue primarily because most languages and approaches do not directly support multiple and dynamic classification—yet it is a coomon way of describing the role changes that can occur in agent-based (and OO) systems.

Possible implementations:
interface variation dependent on state
CORBA Component equivalence

4.4. Mobility

Static agents exist as a single process on one host computer; mobile agents can pick up and move their code to a new host where they resume executing. From a conceptual standpoint, such mobile agents can also be regarded as itinerant, dynamic, wandering, roaming, or migrant. The rationale for mobility is the improved performance that can be achieved by moving the agent closer to the services available on the new host.

Mobile agents can also be part of an agent system that has static agents. For example, there may be a large "intelligent" agent coordinating a set of actions (such as a workflow, a negotiation, or data synthesis), which in turn sends out smaller mobile agents to perform a specific task, such as acquiring some data.

4.4.1. Additional requirements for mobile agents

Mobile agents create an additional set of requirements:

- They require an agent server where they can run.
- They introduce the complexities of security and validating the authenticity of mobile code.
- They introduce management complexity on such issues as:
  - Where are they?
  - How do you communicate with them when they are moving about?
  - How do you bring them home when network failures occur?
  - How do you ensure that they have a reasonable way of "timing out" and halting a process if the task is time sensitive?
  - If they must keep functioning despite adversity, how do you ensure they stay alive?
- Naming and identification become very important.

4.4.2. Adaptation of mobile agents

Adaptation to the changes in the end-users' environment is an important issue for mobile agents. There are several tasks that such agents may carry out in doing adaptation:

- **Optimization of protocols and their utilization**. Both OMG [OMG99] and IETF [PILC] are developing an optimized version of their standards for wireless data communications. OMG is developing lightweight Corba, and IETF is in the phase of starting to develop a wireless TCP. Mobile agents can select to use a best possible protocol for each environment; for example, IIOP over TCP in a wireline environment and IIOP over WAP in a wireless environment. In addition, agents may act as gateways, which convert from one protocol to another one.
- **Compression of application data, such as still images, video and audio, XML, etc**. Today's application data in the Internet, such as still images and multimedia, are designed mainly to be handled using high performance desktop PC with high quality display. Therefore, the data is not suitable for mobile computing using wireless wide-area networks and anemic mobile terminals, and it requires bit-efficient data representations. Compression and content modifications according to end-user preferences are of great importance. Also in this case, agents may act as gateways, which convert from one data representation to another one.
- **Communication with other adapting agents** to achieve an optimal solution.

4.4.3. Mobile agents and mobile computers

Mobile computers create a challenging environment for software downloads and updates. Main reasons for the challenges are intermittent connectivity, unreliable and low throughput

communication channel, and variety of types of mobile computers. There are several tasks that mobile agents, for example, may carry out in software download and update:

- Handle disconnection by waiting for the mobile computer to connect to an access point before starting the software download or update.
- Carry a downloadable / upgradeable software with it in an optimal representation and install it regarding the mobile computer.
- Instead of downloading transmission of instruction to build up the upgraded software is, in some cases, a very attractive alternative.

Possible implementations:

- Realtime draft Online Updating RFP

4.5. Agent is a Principal

A key attribute of an agent is that it be able to act autonomously. Agents can then take on a wide range of responsibilities on behalf of users or other system entities—including purchasing goods and services and entering into agreements. Furthermore, agents may have several identities which they can use while operating. Any robust system will consider these identities in its transactions with an agent.

Additionally, an agent will often perform some tasks on behalf of another entity. For example, a software agent could perform a task on behalf of a person. It could also perform on behalf of another piece of software (another agent), an organization, or a particular role (manager, system administrator). This leads to an number of issues that need to be considered:

- An agent needs to be able to identify on whose behalf it is running.
- If it requests an action from another entity, the other entity may want to assess whether it will permit the action, based on the proffered identity. This might include assessing things such as:
  - Knowledge of the identity.
  - Some set of permissions that control what this identity is permitted to do.
  - Whether the identity has a good credit (in a financial transaction) or a good reputation.
  - Results of previous interactions with that identity.

Subsets of permissions are another issue. In human interactions, we delegate portions of our decision making routinely as well as in an ad hoc manner. For example, we might ask one of our co-workers the weekly status meetings staff meeting, knowing that the person will pick a convenient time, include the right people, and schedule an appropriate room. To another co-worker, we may delegate signature authority to spend up to $1000 in our absence. In both cases, co-workers have been delegated authority but given very different permissions. A similar set of constructs needs to be created for agents.

Related Specifications:
CORBA::Current
Electronic Commerce Domain Specifications: Community Framework.

4.6. Agent security, identity, and policy

Agents are software entities that often run in a distributed computing environment and interact with many other software entities, including other agents. When software runs in a distributed environment, security issues are numerous. The possibility of encountering security problems increases in open environments, such as the Internet or a virtual private network, or in any environment where all the entities are not known, understood, and administered by a single group.

Various types of security risks are described below. Many of these risks are inherent to distributed computing environments, particularly when software passes messages which can be intercepted, modified, or destroyed. While this is a threat to agent systems, it is also a threat to any software system that depends on messages being passed reliably. Another risk centers on whether or not the software can assume that it is using trustworthy services.

Generally, the word *security* refers to the actions taken to ensure that something is secure. If the item is free from danger, it can not be taken, lost, or damaged. In practice, security is usually applied only to somewhat valuable items, because implementing security has associated costs. This is true both in the everyday world, where we protect our cars or homes from theft (but not a disposable pen) and in the world of computing, where we may protect some, but not all, company resources.

Security policy refers to how access to valuable resources is controlled. For example, a company may have a policy about which groups can access which data, or when certain types of processing jobs can run, or whether outside entities can connect to the company network. Agent systems will also require security policies which may control where agents can run, what they can do, and with what other entities they can communicate.

Security policies are usually based on *identity*—which is simply something that serves to identify or refer to an entity. In this way, an agent could be referred to by its name, a role that it is playing, or the fact that it is a member of some organization, and so on. An agent, then, can have multiple forms of identity. For example, a particular agent could simultaneously be a purchasing agent working on behalf of user Rolf Smith; be playing the role of a bidder in a negotiation with E-Widgets; having its software composed of elements from company Exdeus; and having the serial number 98734501. Each of these identities might be important in different interactions.

Identity is based on a *credential*, which is a set of data that can be validated by a third-party to prove that the entity is what it says it is. For example, when a user logs into a computer system, he often enters both a username and a password, which is the credential that is validated to indicate that he really is that username. Other common mechanisms for identity and credentials are X5.09 certificates and PGP keys.

4.6.1. Types of security risks

Here are some security threats that could happen in multiagent systems.

- **Unauthorized disclosure**- A breach in the confidentiality of an agent's private data or metadata. For example, an entity eavesdrops on the interaction between agents and extracts information on the goals, plans, capabilities, or other information that belongs to the agents. Or, an entity can probe the running agent and extract useful information.
- **Unauthorized alteration**- The unauthorized modification or corruption of an agent, its state, or data. For example, the content of messages is modified during transmission, or the agent's internal value of the last bid is modified.
- **Damage**- Destruction or subversion of a host's files, configuration, or hardware, or of an agent or its mission.
- **Unauthorized copy and replay**- An attempt to copy an agent or a message and clone or retransmit it. For example, a malicious platform creates an illegal copy or a clone of an agent, or a message from an agent is illegally copied and retransmitted.
- **Denial of service**- An attack that attempts to deny resources to the platform or an agent. For example, one agent floods another agent with requests, and the second agent is unable to provide its services to other agents.
- **Repudiation**- An agent or agent platform denies that it has received or sent a message or taken a specific action. For example, a commitment between two agents as the result of a contract negotiation is later ignored by one of the agents. That agent denies the negotiation has ever taken place and refuses to honor its part of the commitment.
- **Spoofing and masquerading**- An unauthorized agent or agent platform claims the identity of another agent or agent platform. For example, an agent registers as a directory service agent and therefore receives information from other registering agents.

4.6.2. Message passing

In systems where agents pass messages, the importance of avoiding message alteration or disclosure is described above. If a message is altered, it might provide incorrect information or transmit a dangerous action. If a message can be read by or disclosed to other entities, the other entity may use the acquired data inappropriately.

Message alteration is usually avoided by providing a mechanism for authenticating the message. Most of the techniques for doing this are based on public/private key pair technologies, such as X.509 certificates. Additional information is sent with the message that allows the receiver to validate that the message has not been changed. Message disclosure is avoided by encrypting the message which again is based mostly on public/private key pair technologies.

For both threats, the authentication or encryption can occur either by encrypting the message itself or by sending it through a transport that provides authentication or encryption services.

Other threats related to message passing include: copy and replay, spoofing and masquerading, and repudiation. Both in copy and replay and in spoofing and masquerading, an agent may assume the identity of another agent. Using this false identity, it can communicate with another

agent in order to request an inappropriate action. Many agent systems use relatively simplistic naming schemes (or *identities)* with no additional credentials. Therefore, a message claiming to be from "Joe" cannot be validated.

This set of problems can be solved in various ways. By tagging messages with credentials, the message can be sent in a way that ensures authentication. Thus, the message can be sent free from tampering by a third party. Tagging messages with credentials can also help avoid repudiation. If a message is signed using a credential, the signing agent can not later deny that it sent the message.

4.6.3. System components dealing with one another

Agents can use agent platforms to provide services. They can also interact with well known services such as a *directory service* that helps them locate other agents or an *ontology service* that helps them look up ontologies. When two system components interact, several risks can occur—the two most likely being damage or spoofing and masquerading.

In the damage scenario, the agent may do malicious or inappropriate things to the host system, such as corrupting or deleting files. Therefore, the agent platform may want to control which agents can take which actions. Typically, the agent would offer a credential that identifies it to the agent platform. After validating the credential, the agent platform would use a security policy, based on the agent's identity to determine what actions the agent could take, and would enforce that policy. This is very much like the access control lists found in most operating systems. However, agent systems probably want to control much more that simply reading, writing and running files. They might want to control message sending, utilization of various resources, when and where an agent can move, and whether a moving agent can run on this platform.

Just as the agent platform may want to validate what entity it is dealing with, an agent may want to validate that it is dealing with an agent platform it knows to be genuine. Agent platforms and services could pretend to be "legitimate" but in fact have some dangerous behavior, such as recording message transmissions before encryption, cloning copies of the agent for its own purposes, or providing false information.

4.6.4. Other risks to which agents can be exposed

In constructing software for an agent, certain types of risks must be addressed—ensuring that the following things can *not* occur:

- Viewing the private security key of the agent.
- Viewing the private data of the agent (i.e., the highest bid an agent is willing to make on a product).
- Invoking private methods in the agent.
- That the public methods are designed in such a way that security risks are minimized.

4.6.5. More security considerations

When designing agent systems, the following aspects of security, security policy, and identity should be considered:

- Agents and agent platforms can have multiple credentials. Multiple credentials reflect the reality that we have multiple roles. Users may have credentials as part of several organizations, as an individual, as the owner of multiple credit cards, and so on.
- Agents can have their own credentials. They may also have credentials for the user that they represent in an e-commerce application.
- Agents should not be created that can act anonymously. For example, a user may want to get data about drug or alcohol treatment without revealing his or her identity. Obviously, sites can choose to reject these agents, if their security policies do not allow interaction with anonymous entities.
- All aspects of security need to be managed.
- Traceability of actions can be useful.
- Using a lease model on any credential can be helpful. In a lease model, credentials expire after a certain period of time, but can often be renewed from a credential authority. This control can be a very effective way to clean up credentials in a system which uses relatively short-lived agents. Requiring long-lived agents to renew their credentials is also useful, because when an entity with bad credentials is forced to renew, it will be rejected and shut down.
- Identity and credentials are also useful for building reputation services. Such services provide a way of determining whether a particular entity has behaved responsibly.

5. Agent architecture

For the most part, agents will be deployed within conventional enterprises and will draw on the enterprise for many services. CORBA provides a rich source of services and a proven architecture. This section provides a framework for considering how a system supporting agents might draw on CORBA services and facilities. The architectural basis for this discussion will be the FIPA architecture.[2] The FIPA Agent Platform provides a good construct from which to discuss the enterprise-related issues in agent deployment. Figure 1 is a UML depiction of the FIPA98 Agent Management Reference Model that has been augmented to include components to facilitate interfaces to non-agent software and services.

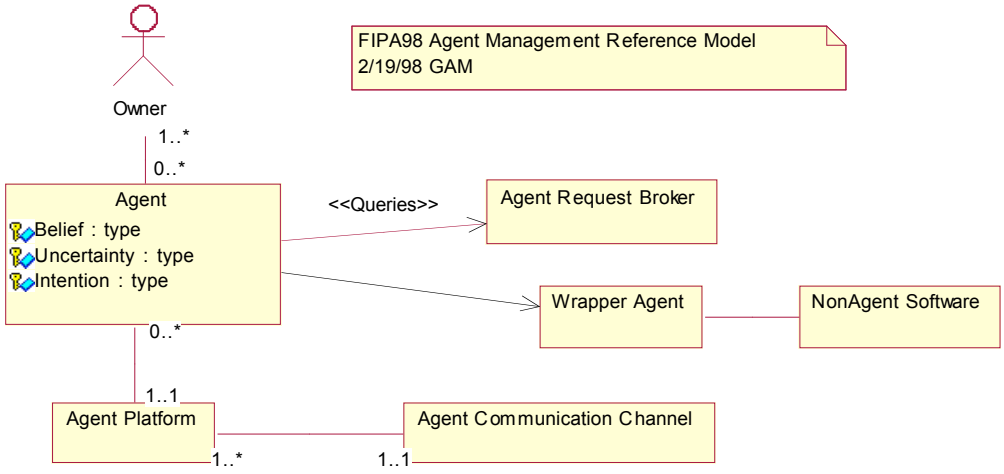

Figure 1. Agent Management Reference Model.

5.1. Agent Platform

The key element to the enterprise architecture is the Agent Platform. An Agent Platform (AP) provides an infrastructure in which agents can be deployed. An agent must be registered on a platform in order to interact with other agents on that or other platforms. Minimally, an AP consists of three capability sets: an Agent Communication Channel, an Agent Management System, and a Directory Facilitator. The Internal Platform Message Transport (IPMT) is the local (possibly proprietary) means of exchanging messages within an AP. Figure 1 is a UML depiction of the FIPA98 Agent Platform Architecture.

[2] *Foundation for Intelligent Physical Agents FIPA98 Agent Management Specification,* Geneva, Switzerland, Oct. 1998.

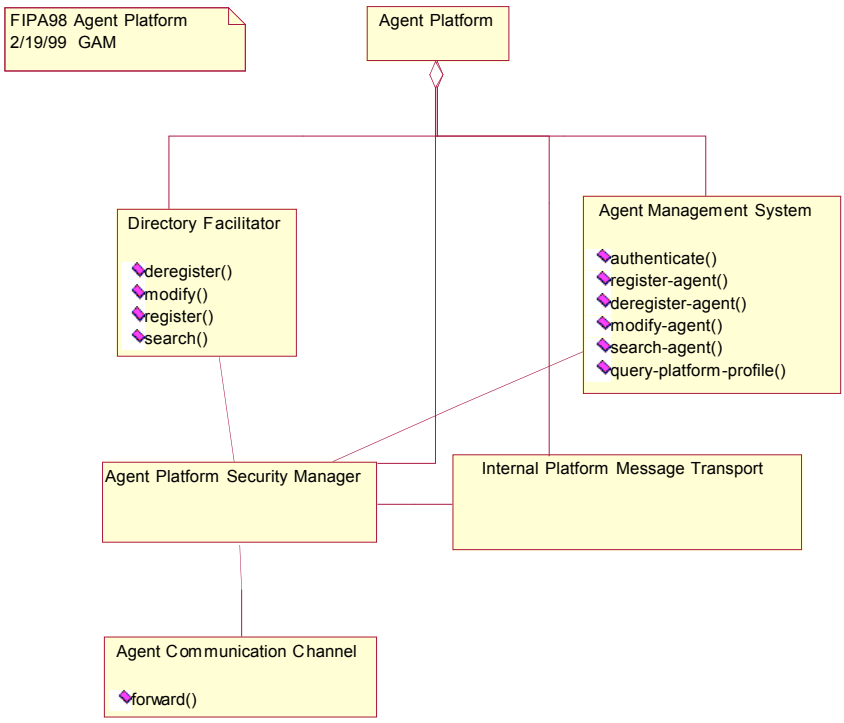

Figure 2. Agent Platform Architecture

FIPA does not specify the physical nature of a platform. However, two cases should be considered, that of a single host and that of multiple processors deployed as a "virtual" platform. If the Platform is virtual, having it fulfill several requirements would be wise. It should have:

- high-speed communications.
- a single system manager.
- a single security enclave.

These last two requirements make the agent system easier to use. The responsibility for humans managing the system is simplified. Also, we avoid the situation where control of the agent lifecycles on a platform is shared by several people. From the system perspective, the lifecycle of all agents in a given platform is controlled by a single entity—the Agent Management System. From the perspective of a human (or agent proxy), the platform itself should also be controlled by a single entity.

5.2. Agent Management System

The Agent Management System (AMS) is an agent that supervises access to and use of the AP. Only one AMS will exist in a single AP. The AMS maintains a directory of logical agent names and their associated transport addresses for an agent platform. The AMS offers "white pages" services to other agents. The AMS is responsible for managing the lifecycle of the agents on the platform [FIPA]. Its actions include:

- Authentication
- Registration
- De-registration
- Modification
- Query platform profile
- Search
- Control of agent lifecycle

5.3. Directory Facilitator

The Directory Facilitator (DF) provides "yellow pages" services to other agents. The DF is a mandatory, normative service. Agents may register their services with the DF or query the DF to find out what services are offered by other agents.

5.4. Agent Platform Security Manager

The Agent Platform Security Manager (APSM) is responsible for maintaining security policies for the platform and infrastructure. The APSM is responsible for run-time activities, such as communications, transport-level security, and audit trails. FIPA 98 security cannot be guaranteed unless, at a minimum, all communication between agents is carried out through the APSM.

The APSM is responsible for negotiating the requested inter- and intra-domain security services with other APSMs in concert with the implemented distributed computing architecture, such as CORBA, COM, and DCE, on behalf of the agents in its domain. The APSM is responsible for enforcing the security policy of its domain and can, at its discretion, upgrade the level of security requested by an agent. The APSM cannot downgrade the level of services requested by an agent but must inform the agent that the service level requested cannot be provided.

5.5. Agent Communication Channel

All agents have access to the Agent Communication Channel. It provides a path for basic interchange between agents, agent services, AMS, and other agent platforms. At least, it must support IIOP. Agents can reach agents on any number of other platforms through the Agent Communication Channel. Access to agents outside of the local namespace could be supported by the CORBA Trader Services.

5.6. Federation of Agent Groups (Grid)

Networking groups of agents is a common requirement for multiagent systems. However, since one of the key properties of multiagents systems is that they are coordinative, something stronger than networking is required. Here, the creation of a coordinating unity out must be

formed using of a number of separate agent groups. The result of uniting these groups of agents in a coalition or alliance is *federation of agent groups*—also referred to as a *grid*.

The term *grid* is increasingly appearing in computer literature, generally referring to some form of system framework into which hardware, software, or information resources can be plugged, and which permits easy configuration and creation of new functionality from existing resources. [MT99]. The "killer applications" for these grid concepts include computational challenge problems (e.g., codebreaking) requiring supercomputing capabilities, universal availability of customized computing services (e.g., access to one's individual desktop and application suite anywhere in the world), global integration of information, computing, and other resources for various purposes, and rapid development and deployment of super-applications configured on-the-fly to solve some problem in a scalable, reliable, survivable way. Several DoD and industry programs use some form of grid concept.

Recently the grid concept has begun to be applied to computer systems involving *agents*, with agents playing both the roles of enablers and customers of grid capabilities. The *agent grid* is a proposed construct that is intended to support the rapid and dynamic configuration and creation of new functionality from existing software components and systems. Requirements the agent grid is meant to address include:

- Humans and agents must be able to connect to the grid anytime from anywhere and get the information and capability they need.
- The grid infrastructure and services must scale to millions of agents such that agents are pervasive and information and computation is not restricted to machine or organization boundaries.
- If one agent goes down, another takes its place.
- It should be possible to add capability modularly.
- Interoperability among agents from agent communities with heterogeneous architectures should be supported. This will enable trans-architecture applications and teams.
- Agents should be able to dynamically discover and establish communication with agents in other agent communities. Agents should be able to find other agents through directory, matchmaking, and broker services.
- Agents should be able to communicate via shared ACLs (communicative acts, vocabularies, ontologies, etc.) or via translation services.
- Interoperability needs to be supported among agents and non-agent software components (objects, legacy applications).
- Local services in agent communities should be leveraged, not replaced, by the services in the Grid.
- The reengineering of components should be minimized (via wrappers, proxy agents, translation and other interoperability services, etc.)
- Existing GOTS and COTS frameworks for component integration should be leveraged: e.g., CORBA, Jini, the Internet, NGII, etc.

We identify three views of an agent grid [MT99]. All are, or can simultaneously be, valid views:

- **A collection of agent-related mechanisms** (brokers, agent communication languages, rules, planners, etc.) that augment object and other technology to increase the ability of developers to model and dynamically compose systems. Many of these mechanisms can be independently studied and could be standardized so that communities that used these standard mechanisms could more easily develop interoperable "agent-based" capabilities.
- **A framework for connecting agents and agent systems.** The framework could be built using the standard mechanisms as well as existing non-agent technology (e.g., middleware). The benefit of this view is to show how the agent mechanisms fit together to make it easier to construct agent systems of systems, using framework-aware (grid-aware) protocols, resources, and services as well as foreign wrapped agent systems and also legacy non-agent systems. The grid as framework is responsible for providing services and allocating resources among its members. One of the issues with this notion of grid is whether there is just one or several grids and if the latter how they interoperate.
- **Model entities that might be used to model an org chart, team or other ensemble.** This view assumes there are many grids but they interoperate according to the standard protocols. In this view, a grid might represent a battalion ensemble which has assigned tasks and resources. Lower level units (agents/grids) use the resources to work on the tasks assigned to the battalion.

The agent grid should be thought of not as a replacement for existing agent architectures and distributed systems, but as a mechanism for integrating them. It is important not to confuse the agent grid as an integration concept with individual grid-like distributed infrastructures (e.g., CORBA, Jini-based distributed Java object networks). The intent is for the agent grid to be a higher-level interoperability infrastructure, largely independent of these more implementation-level techniques, which allows the most appropriate implementation infrastructure (e.g., CORBA, Jini, etc.) to be used in a given situation.

The agent grid is intended to enable the rapid formation and deployment of superapplications. These superapplications will consist of components from several agent systems and from non-agent systems. The function of the grid is to provide infrastructure services to enable formation of these superapplications. The infrastructure needs to provide a way to register grid components. Some of these will themselves be infrastructure services. Also, the grid itself needs to be able to log component interactions and control system wide properties of collections of agents, services, and components. As mentioned, the grid can be implemented through a variety of technologies. Engineering experiments in the DARPA Control of Agent Bases Systems (CoABS) involve implementing the grid on top of Jini, email, search engines, and the web, striving to make the components separable, scalable, secure, and federated, and the architecture open.

The architecture of the agent grid is still undergoing evolution. Open General Issues for the Agent Grid include:

- How much control does a Grid have over individual agents?
- How much domain knowledge does the Grid have?

- How "intelligent" are the entities (components, agents, etc.) that inhabit the Grid?
- What services does the Grid provide?
- What languages, protocols, or other standards is an agent required to adhere to in order to use these services? Are there varying levels of "grid awareness" or "grid compliance"?
- What properties does the Grid need? (e.g., scalability, security, reliability, etc.)
- How does the Grid allow agents from heterogeneous agent architectures to interoperate?
- How do non-agents software components (objects, applications, etc.) make use of the Grid?
- How does the Grid relate to other infrastructures, grids, networks, distributed objects frameworks (like OMG's OMA), the Internet, the Web, etc?

5.7. Maintaining System-Wide Properties in Agent Systems

It is often said that agents are autonomous. They represent and act for their owner and interact with other agents to solve problems. Not much is said about how external conditions affect their individual behaviors. In many problems, the environment of the problem imposes some external constraints on interacting agents. Perhaps resources are scarce (e.g., bandwidth is limited) or communication must happen with real-time constraints or communication must be reliable and secure or the system of agents must assure survivability. How are these system or environmental constraints handled in agent systems?

Generally, such constraints are characterized as system-wide constraints (where a system might be local and a system-of-systems might impose different constraints in different subsystems). Some communities refer to such constraints as *-ilities*, others as *quality of service*. For a good overview of ility and QoS (together called iQuos), see [M99 -"Providing Systemic Properties (Ilities) and Quality of Service in Component-Based Systems"].

Various lists of ilities have been identified. For example: accessibility, accountability, accuracy, adaptability, administrability, affordability, agilityavailability, composability, configurability, degradability, demonstrability, deployability, distributability, durability, evolvability, extensibility, fault tolerance, flexibility, footprint, interoperability, maintainability, manageability, mobility, nomadicity, openness, performance, reliability, responsiveness, safety, scalability, schedulability, seamlessness, security, simplicity, stability, survivability, tailorability, timeliness, trust, understandability, usability.

An open question is how to augment an agent (or component or any other kind of) system so that it has some or all of these properties, or how to extend an existing system to add these properties modularly. It is often said that security cannot be added to a system after the fact. The same seems to be true of many of the properties on this list, that they pervade the design of applications and systems that require them.

Promising work reported at the OMG-DARPA Workshop on Compositional Software Architectures [Tho98] indicates that system properties can often be inserted into the communication paths among components of a system, that this is a reasonable place to log the system's behavior, and that system meters and policy controllers can be added to systems to

monitor and control these kinds of system wide behaviors. There are many open issues about this approach, for instance, how to handle policies that vary from system to system when the systems are composed into a system of systems. It appears that many of the early results from this approach will be applicable to agent systems as the agents are modular. Still, this iquos idea imposes some external control over a collection of formerly autonomous agents. The agent grid is a natural construct with which to associate such system imposed constraints.

6. Agent System Development

6.1. Agent modeling and specification

We need to better understand how to develop applications using agent technology. Therefore, development tools and methods play an important part in agent-based systems. Here, we need to determine what kinds of modeling languages are required as well as the underlying metamodels needed to support agent specification and deployment. Furthermore, we need to ensure interoperability across the lifecycle of agent tools and their designs/work products.

Possible implementations:

- UML
- MOF

6.2. Testing, debugging, validation, and simulation

Concurrency and nondeterminism in execution make it very difficult to comprehend the activity of an agent system. We need visualization tools, debugging facilities, and simulation.

6.3. Agent system development methodologies

We must determine the steps needed to guide the life cycle of agent system development—at both micro and macro levels. Existing methods can be used to some extent, but we need to identify how an agent-based approach will change them.

7. Agent and Object Technologies

7.1. Motivation

Why do we care what the relationship is between agents and objects and about agent-object interoperability? There is widespread interest in both technologies and developers are bound to model problems using one or the other technology. So it is inevitable that the two technologies will need to interoperate if not eventually unify. Also, if the two technologies have similar needs (for instance, to define common middleware services for persistence, registration, trading, time, etc), then having to define these twice, separately for each community, and forcing application developers into decisions on whether to use objects or agents, is counterproductive. So there is a very real need for these two related technologies to co-exist, and even more, to become better integrated, so agents can interact with objects and vice versa. From an organizational point of view, the OMG Agent Special Interest Group, in particular, cares about the relationship of agents to objects because it is focusing on agents within a group that is primarily focused on object technology (which broadly includes middleware, interoperability, and domain services), so we have a vested interest in bridging the gap. On a larger scale, the agent community should care about its relationship to pervasive technologies (like the Web, XML, email, distributed objects, etc.) because it needs migration routes and strategies if it is to become widespread and make a pervasive impact. For instance, many view a more semantic, connected web as an evolutionary manifest destiny, and if agents are to play a significant part, then they must interoperate with these other technologies.

In the argument over the relationship of objects and agents, some hold that agents are objects, others that objects are agents, and some that neither is the case. Taking sides in an emotional debate does not seem to hold much promise. Instead, it appears that we can agree on some of the differences, some of the means to reduce the differences, and identify and work on some unresolved issues.

7.2. The problem of defining terms

As described in *Section 2 What is an Agent?*, there are different ways to define the notion of agent. For instance, *intelligent agents* might be those that use some form of agent communication language (e.g., KQML or FIPA ACL), ontologies, and content languages; *evolutionary agents* might be agents that sense their environment and follow simple rules; and so on. Problems with defining the term agent (or object for that matter) will not be repeated here except to point out that identifying what the relationship is depends on what we mean by terms.

7.3. Comparing objects and agents

Below we offer some points of view on how the relationship of objects and agents can be defined. These come in from different angles and might lead to different conclusions. Over the next few years, we will be interested to see how these points of view can be resolved.

7.3.1. How agents are similar to objects

Objects and agents are alike in several ways. In fact, we can assume that conceptually agents *are* objects or components or entities, in the sense that agents have *identity* (you can tell one agent from another), they have their own *state and behavior* (distinct from that of other agents), and they have *interfaces* by/through which they communicate with each other and with "other things". Like objects, while agents can be any *size*, development guidelines often suggest constructing agents that are small.

7.3.2. How agents differ from objects

Objects and agents differ in several ways. Some of these are conceptual modeling preferences and some imply mechanism differences.

- **Built-in (Intelligent) Functionality** - While there isn't any single grain size that characterizes all agents and multi-agent systems, (intelligent) agents can often be considered to be *more functional (smarter)* than single objects. These agents include complex reasoning components, such as inference engines or neural networks. Often, these resources are implemented as sharable sub-systems, used by many agents, or used as parts of platforms supporting many agents so their presence does not necessarily make agents *larger* (e.g., in lines of code*)* than objects but it does mean that agents perhaps have richer built-in interpreters. On the other hand, we might ask, if we add an inference engine inside an object, will it be an agent as "what's inside" is a question of implementation. In this sense, object interfaces can encapsulate "smart things", e.g., more or less smart agents, and human beings; for example, "thompson@objs.com" is the identifier of an interface to which messages can be sent. Object interfaces can also encapsulate "dumb things", e.g., conventional software objects, or dogs ("on the Internet, no one knows you're a dog"). Still, whether implemented as agents natively or as objects with intelligent interpreters inside, there is still a difference in that agents might then be a smart class of objects that communicate using certain shared agent communication assumptions (see below).
- **Agents are autonomous and reactive -** One of the basic notions about agents is that they are autonomous. They can individually decide whether to respond to messages from other agents. This contrasts with objects that do what they are asked to do (unless forbidden by security or other constraints). Similarly, agents are distinguished as reactive, always listening and participating, in conceptual object terms, containing their own threading capability. Finally, agents are often thought of as proxies for their owners or the things they represent in the real world. In this sense, agents model things that act (e.g., employees) and passive objects might be considered to model things that are acted upon or passive (e.g., equipment).
- **Agent Communication Language** - (Intelligent) agents use a powerful agent communication language to communicate with each other. They can represent complex belief-desire-intention information, use inference, and change their behavior based on what they learn. In effect, they can respond to different messages over time. Objects, by contract, use a fixed set of messages in communication. OMG objects can support multiple interfaces (for instance, a trader could support public interfaces for listing ads and private interfaces for storing and retrieving ads).

On the other hand, objects do have a conceptual need to evolve and take on new roles (from say a child to a parent) but this is not well supported in many object systems.

- **Type-instance distinction** - At least in many agent systems, agents correspond roughly to object instance or individuals but there is no strongly typed class notion. Rather, the agents are all just agents, not employee agents, equipment agents, etc. Of course, they may become specialized through their content and behavior to effectively become employee agents.
- **Inheritance** - Object advocates might point out that objects do some things agents do not commonly do, for instance, inheritance is not commonly viewed as an agent aspect though agents are not precluded from supporting some forms of inheritance.
- **Other differences** - other differences can be highlighted:
  - **Mobile agents -** these appear to be similar if not the same as mobile objects
  - **Information agents -** these might be very like graphs of objects (agents) processing information from data sources and delivering it to customers
  - **User-interface agents** - these may or may not use agent communication language, inference or other agent mechanisms and loose coupling to, for instance, permit new interface agent modalities to be added or others removed from a multimodal agent-based human interface framework.
  - **Agents that evolve** - In some agent subworlds, agents may have additional constraints. For instance, evolutionary agents might be not just be small in lines of code but also small in mass, size, and time.
  - **Small in Mass** - In systems with large numbers of agents, each agent is small in comparison with the whole system. For example, each termite is an almost negligible part of the entire termite hill. As a result, the behavior of the whole is stable under variations in the performance of any single member, and the collective dynamics dominate.
  - **Small in Time (Forgetful)** - Naturally occurring agent systems can forget. Ant pheromones evaporate, and as a result obsolete paths leading to depleted food sources disappear rather than misleading members of the colony. Even the death of unsuccessful organisms in an ecosystem is an important mechanism for freeing up resources so that better adapted organisms can flourish. Furthermore, each transaction can cost an agent, leading to a decrease in energy over time.
  - **Small in Scope (Local Sensing and Action)** - The participants in natural systems usually can sense only their immediate vicinity. In spite of this restriction, they can generate effects that extend far beyond their own limits, such as networks of ant paths or termite mounds. Wolves are built close to the ground and so can't see very far, but manage to coordinate a hunt with other wolves over a wide area.

    In terms of agent support mechansisms (versus objects), it is not clear if these differences should be modeled as constraints on general intelligent agents used to model these subworlds or be used to define a new species of agent called evolutionary agent, as different from intelligent agents as objects are from either.

One more very important way that agents differ from objects is not based on any particular mechanism but rather than on the argument that agents add enough differences to objects to be at a higher (or different) level of abstraction. As such, the problems you use them to solve, the

capabilities they have, and the differences in research involved to understand them is fundamentally different for agents. While this is unassailably true, it does not reduce the need for agent and object technology to both be useful and need to become interoperable if both technologies are used together to solve problems.

7.4. Towards coexistence, interoperability, and/or unification of agents and objects

It appears we have some puzzles to resolve. Agents are objects in some ways and clearly different in others. Further, there are different kinds of agents with different kinds of differences. Finally, we want to have our cake and eat it too, that is, we want to use objects and agents together in problem solving. How can we do this? This section catalogs some points of view on how this can be achieved. We are looking for unification so we can simultaneously explain all these viewpoints in a consistent manner. This unification is important for the reasons given in the Motivation section - if we fail to provide paths for agreeing how agents and objects are related, we set up future interoperability problems, as both technologies have overlapping applicability. But the community has not accepted any particular resolution so the jury is still out. If we can agree, we can perhaps get the best of both worlds sooner. Even if we disagree, we should be better able to describe the source of our disagreements.

7.4.1. Agents and agent systems composed from objects and object infrastructure

While we make the case that agents are not the same as objects, this doesn't mean that agents shouldn't be composed out of objects. In fact, many agent systems have been built using object technology (especially using Java), and, indeed it makes sense to build the infrastructure for agent based systems on top of the kind of support systems used for complex object-oriented software systems

When we begin to decompose agent structure, we rapidly find lots of structures and parts that are reasonable expressed as objects. These might include:

- Agent Names
- Agent Communication handles
- Agent Communication Language Components, including:
  - Encodings
  - Ontologies
  - Vocabulary Elements
- Conversation Policies

Beyond the parts that make up agents, there is an additional layer of software components in multi-agent systems that may be naturally viewed as agent middleware, directly analogous to object middleware, and that can be expressed as objects. Examples of such might include:

- Transport References
- Transport Policies
- Directory elements

- Discovery Services
- Logging Services
- Event Services
- Query Services
- and many more

It does not make sense to re-implement these capabilities using agent technology if we can reuse object components and services that effectively do the same thing.

Explaining how object technology can be used to implement agent technology provides one of ways objects and agents can be related but does not really describe how they might interoperate more as equals. It still leaves us (perhaps naively) wanting to model an employee as an agent because employees are active and a stapler as an object because it is passive as opposed using lower level objects for these in some cases and apparently higher level agents in other modeling cases. So we haven't got the whole story yet.

7.4.2. Agents as "objects with an attitude"

It is tempting to assert "An agent is an object that...", completing the phrase in a variety of ways. (Bradshaw 1997b) refers to agents as "objects with an attitude" in the sense of being objects with some additional behavior added, for instance, mobility, inference, etc. Here, we are using objects as a generic modeling or abstraction mechanism, independently of whether agents are implemented as objects (using object-oriented programming techniques).

The viewpoint might help resolve the problem of multiple kinds of agents. In a sense, we could view the maximal agent as potentially having all behaviors found in the agent attributes list, and that degenerate forms of agents are those containing fewer than all properties. In this view, *objects are agents without these extra agent attributes*. This helps explain how agents might literally be "objects with an attitude." Taking this view, considering agents that use agent communication languages as having the ability to "just say no" to message requests, we can view objects as degenerate agents that always do as they are told.

One can now simultaneously argue that agents are objects and that agents are different from objects. These extra agent capabilities could be added to objects but then they would become agents. The nice thing is, then we have a path for treating agents and objects the same in models and both as first class model elements.

Related to this view, we could distinguish agents that communicate natively as *agent-oriented* and objects that encapsulate and simulate native agent capabilities as *agent-based*, in comparison to the traditional, similar distinction between *object-oriented* and *object-based*.

7.4.3. Resolving the communication differences

If agents use ACL and objects use message passing to communicate, then it is still not quite clear how agents and objects can coexist. In fact, this difference can be viewed as a defining characteristic of how agents and objects are different. Objects are objects if they can communicate

via object messaging and agents are agents if they can communicate via agent communication language.

This observation holds a possible key to how to simultaneously permit both, namely, some entities can do both so they are both objects and agents. This might permit a system to contain agents that use ACL to communicate with other agents, objects that use message passing to communicate with other objects, and some entities that can do both so they are simultaneously objects and agents.

```
            ============
agent --> || ACL | object || <-- object
            ============
```

This point of view might also admit the idea that agents can call objects in an object-oriented way (permitting objects and agents to interoperate as first class objects), that is, behave as objects with respect to objects and so expect objects that they call to return responses in the same way one object expects a response from another that it sends a message to. This might especially be palatable if we view message passing as a subclass of agent messaging.

In a broad way, we might claim that agent messaging and object messaging play similar roles but differ in detail. That is, in some cases the messaging protocols between various kinds of objects will be relatively simple (e.g., conventional object RPC between distributed software objects, or commands sent to hardware), while in other cases they will be more complicated (agent communication language (ACL) sent between agents, or the email flow between people); however, similar abstraction principles can apply to objects at all levels.

But more narrowly, there are differences between object and agent messaging that are not easy to resolve.

7.4.4. Modeling an agent as an object - hard to do

There are several reasons why it is not simple to just model agents as objects directly.

- **Expressive limitations**

  When we model any software entity as an object, we focus on its interfaces and its state. In general, the single key characterization of an agent's behavior is that it accepts messages from other agents, performs some processing and (possibly) responds. This behavior, in terms of interface, may be represented in two forms. One would be to provide a method for every message that the agent is capable of processing. The other would be to provide a single method, AcceptCommunicativeString which would permit the agent to accept arbitrary messages.

  It is tempting to imagine providing an agent with a method for each message it can accept. However, we generally want every agent to be able to be able to receive any message, then decide on its own how to process that message. Certainly, the messages it knows how to and decides to process can change over time.

  [On the other hand, the messages agents send and receive are not arbitrary - to be understood, they are described in a specific agent communication language with agreed upon semantics and

structure. So it might actually make sense for objects that were also agents to be modeled as including the method AcceptACLString. At the same time, it is not satisfying to imagine a large application composed of agents implemented as objects each with one generic method just as it is not satisfying to imagine replacing object methods with a single agent method - too much is lost in this mapping in both directions.]

- **Autonomy**

  Method invocation also has the wrong basic connotations in terms of autonomy. In general, method invocation is done either on the caller's thread of control, or on the thread of a remote proxy for the caller, in the case of remote method invocation. Agents, on the other hand, aren't running as part of the caller, nor are they running with the caller's identity and privileges. In fact, quite the contrary, an agent runs on its own behalf, and with its own identity and privileges While one can imagine doing things with access control such that the additional rights and proper identity were associated with each invocation, it adds steps that are outside the normal notions of method invocation.

  Beyond identity and access control rights, we also run into the notion of autonomy. Unlike a normal object, where access control rights and visibility largely encapsulate notions of whether a method should run, an agent is an autonomous element of the software world. Thus, for every message that the agent receives, it may determine, based on its own goals and state, and the state of the ongoing conversation, whether to process the message and how to respond if it does. This implies that every method, will, in effect, begin with the idiom of making such a determination, something which, in general, is beyond what the majority of objects do at method invocation time. (Parameter checking, and comparison to internal state is fairly normal, but knowing the identity of one's caller, and basing decisions on that, is not)

  Also, agents are active, not passive entities within a software system. Traditionally, an object is passive, with its methods being invoked under the control of another component of the system. An agent, in general, can react not merely to specific method invocations, but to observable events within the environment. Further, as part of the basic structure of multi-agent systems, it is natural for agents to be engaged in multiple, parallel interactions with other agents.

- **Conversations, and long term associations**

  Another problematic issue for thinking of agent messages as method invocation is the question of long term conversations and associations. Agents, unlike simple objects, may engage in multiple transactions concurrently, either through the use of multiple threads, or similar mechanisms. This is expressed naturally in a messaging environment, as each conversation can be assigned a separate identity, and either a unique message destination or a unique identifier can be used to sort out the threads of discourse.

  Similar mechanisms can be built out of objects, of course, but they would bypass, or be in addition to, normal method invocation. One could imagine producing multiple cloned "copies" of the agent object and passing one reference to each counterpart, so as to provide separate communication contexts for each conversation. But the very need to invent this kind of mechanism implies that the agent isn't acting as a simple object.

7.5. Some Issues

The jury is not in. These are some of the challenges that confront us in the area of trying to understand the relationship of objects to agents:

- We know how to wrap legacy code and data with an object wrapper so the legacy code and date can participate in object message passing. Can we do the same thing with code, data (and objects), e.g., wrap them in an agent wrapper so they can communicate as agents? Or do we need to add qualitatively more to turn legacy code into agents? One example of a place where this is important is in agent services. There seem to be useful services like registration, logging, persistence, transactions, matchmaking (trading), administration, replication, security, and many more that agents and agent systems would benefit from. Can we just straightforwardly use object services for these? How important is it to wrap these in agent wrappers to permit them to communicate via ACLs? Is there value in these middleware services having autonomy?
- Can we really add additional features to objects so they become agents? This is consistent with the general view that we progress computer science by adding and packaging useful capability first to climb to abstract data types, then to objects, eventually to agents. It is pretty clear that we added distribution and persistence to objects already. We can similarly add mobility to objects to get mobile objects and most mobile agents are just that. Information agents are generally active objects in a network that implement database-like functions to aggregate data. Traders exist in both the object and agent worlds, playing the same roles. The puzzling cases come when we add BDI, ACL, inference, and ontologies. There does not seem to be an object correlate for these, not yet, at any rate. So maybe our puzzle is, how to deconstruct agents to just add these capabilities.
- If agents are going to be widely useful and common, then how do we go from the current state of practice where agents are in the minority and agent systems are often closed so that agents only communicate with other agents in their own society, to a next generation world where agent technology is comparably dominant to object technology and other possibly related technologies. If agents are to coexist with objects, what implementation changes can be made to add them seamlessly into programming environments like Java of C++? Can we find ways to piggyback agent technology onto other pervasive technologies like email, XML, distributed objects, databases, etc.? Some of the benefits might be scalability and immediate pervasiveness.

In some sense, the burden of getting along with other technologies is a problem for the agent camp - the sooner we provide migration strategies, the sooner agent technology can become more widely available and useful. A way to put the problem in perspective is, pretend you give some application developer an agent toolkit and also conventional tools like scripting languages. At what point will solutions commonly be developed using many of the tools from the agent toolkit? Will they just be used when they are useful, like other tools. For instance, inference might just be used occasionally (perhaps rarely); trading more often; mobility in certain families of situation; and so on.

7.6. Relating agents to OMG

7.6.1. Positioning OMG as the obvious integrating technology

OMG, through CORBA, the OMA framework and its related services, provides almost all of the elements needed to produce a rich framework for supporting agents. Furthermore, through IDL, ORBs IIOP, UML (extended to support agents), and XMI, OMG potentially provides an excellent basis for knitting together abstract agent frameworks instantiated in multiple OMG compatible environments.

Java, with its strong ties to OMG, provides an excellent basis for building Agent Frameworks. Indeed, as mentioned, many agent projects are being built out of Java parts, including Java Beans, RMI, and IIOP. OMG should play a major role in bringing these elements together, and standardizing the usage of various OMG technologies in these areas so as to enhance interoperation and portability.

OMG would need to provide standards in a number of areas, if it desires to act as a major player in the agent arena. Agent Communication Languages, Ontologies, and services for managing these elements would be clear candidates. Infrastructural elements, such as transport factories, agent support platforms and the like would also fall into this purview. See Agent Technology White Paper and Roadmap on OMG Agent WG homepage http://www.objs.com/isig/agents.html.

7.6.2. Integrating Agents in the OMG world view

A more comprehensive proposal would entail integrating agents into the OMG world view as "first class" elements in the overall set of OMG abstractions. This would include supporting the ability of agents and objects to name, invoke and manage each other as first class software elements. Such a proposal would represent a fundamental revisiting of many basic assumptions in the current OMG perspective on software. Such a proposal does not seem warranted today, with the current state of agent technology, and current marketplace acceptance of agent-based systems. However, as the state of the art advances, the Agent Special Interest Group should keep this option available, and, in particular, avoid taking steps that would preclude tighter integration of Agent technologies into the OMG core in the future.

8. Agent RFI Results

8.1. RFI Overview

A Request For Information (RFI) entitled *Agent Technology in OMA* (document ec/99-03-10; ftp://ftp.omg.org/pub/docs/ec/99-03-10.htm) was issued by the Agents WG on March 24, 1999, with a submissions due date of August 2, 1999. This RFI solicited information on available agent technology and how to use it in CORBA-based applications, and addressed several areas: requirements, architectures, designs, projects, products, protocols, and standards. The information provided will be used by the OMG to develop:

- this Agent Technology Green Paper further.
- a Roadmap for technology adoption.
- a series of Requests for Proposal (RFPs) in this area.

The goal of this RFI was to canvas the industry for technologies and required interfaces for exploiting agent technology in applications built using OMG technology. Its scope included, but was not limited to the following areas:

- kinds of agents: mobile, intelligent, information access, user interface, ...
- kinds of agent systems, societies, and meta-agent facilities (e.g., grid)
- kinds of agent communication: agent communication langauges like KQML or FIPA ACL, agent negotiation
- kinds of ontology description mechanisms
- agent and agent system interworking and federation mechanisms
- agent system management and administration, agent life cycle services, system-wide properties in agent systems (-ilities, QoS)
- agent security, risk management, survivability
- planning and scheduling mechanisms
- agent relationship to objects, OMA, CORBA, and CORBAservices
- extensions to existing OMG services and facilities (e.g., traders that support inference, secure agent extensions)
- agent modeling and specification techniques
- tools, testing, debugging, validation, and simulation approaches
- agent-system development methodologies

The nine RFI responses—listed in order of receipt—were as follows:

8.1.1. Object Services and Consulting response

(ec/99-07-01; ftp://ftp.omg.org/pub/docs/ec/99-07-01.htm)

Characterizing the Agent Grid. This working paper covers both a general notion of a grid, as well as a more detailed discussion of the Agent Grid, its properties, and issues related to it. A *grid* is basically a federation of collections of things and their interconnections; an *agent grid*, then, is a federation between groups of agents and their interconnections/interrelationships. (See Federation of Agent Groups, earlier in this paper.)
This paper suggests, first, that there are several kinds of grids: some like the software and the information grid that are applications of infrastructure technology. Second, it submits that the agent grid is, in some way, a technology layer that enables other grids. Third, it proposes that the grid might be related to agent systems either as master registry mechanisms or meta agent systems or similar. And lastly, it suggests that organizational units (ensembles) act like grids in that they control resources. The response concludes that it is less clear that there is likely to be one global construct called *the* agent grid or that such a system makes all optimization tradeoffs and provides all systemic control. (Contact: Dr. Craig Thompson; thompson@objs.com)

8.1.2. Monads response

Joint submission by Nokia, Sonera Finland, and University of Helsinki

(ec/99-07-02; ftp://ftp.omg.org/pub/docs/ec/99-07-02.pdf)

OMG has already addressed wireless access and terminal mobility (telecom/99-05-05). However, this response recommends that the OMG address agent standards that support nomadic applications. The Monads response is based on experiences gained from developing agent services supporting nomadic applications in the research project Monads (http://www.cs.helsinki.fi/research/monads).. The key findings are: first, the OMG should specify an interface for FIPA ACL messaging; second, a location transparent object reference is recommended that is dynamically bound to a location in run-time when the reference is used; third, extensions to CORBA Messaging (orbos/98-05-05) may provide basis for an Agent Messaging Service; last, close cooperation should be maintained between FIPA and OMG is needed.(Contact: Professor Kimmo Raatikainen: kimmo.raatikainen@cs.helsinki.fi)

8.1.3. CLIMATE response

Joint submission by France Telecom, IKV++, NEC Europe CCRL-Berlin, and University of Helsinki

(ec/99-07-03; ftp://ftp.omg.org/pub/docs/ec/99-07-03.pdf)

This submission provided a detail response the RFI questions. The key findings are: 1) a mobile agent Migration Service is needed; 2) an Agent Communication Transport Service is needed; 3) a location transparent object reference that is bound in run-time is needed; 4)UML needs extensions to model mobile agents; 5) Close cooperation between FIPA and OMG is needed; 6) an initial agent system architecture that can be used as a starting point for discussions; and 7) an initial agent management system for agent platforms and agents is needed. (Contact: Professor Kimmo Raatikainen: kimmo.raatikainen@cs.helsinki.fi)

8.1.4. Toshiba Bee-Gent response (first response)

(ec/99-07-06; ftp://ftp.omg.org/pub/docs/ec/99-07-06.zip)

Bee-gent: Bonding and Encapsulation Enhancement Agent Framework for Development of Distributed Systems. This response describes the Bee-gent multiagent development environment. This software includes: support for interaction protocols via its Protocol Engine, support for ACL expressions, and provides a Visual Development Tool for interaction protocols. (Contact: Takahiro Kawamura; kawamura@ssel.toshiba.co.jp)

8.1.5. Toshiba Plangent response (second response)

(ec/99-07-08; ftp://ftp.omg.org/pub/docs/ec/99-07-08.zip)

Plangent is an intelligent agent system that performs tasks for human users. A Plangent agent has movement and planning capabilities; it can move around the network, determine the best course of action in various situations, and act by itself. The most unique point of Plangent system is its agent model which they claim is a combination of intelligence and mobility. The response provides information in the Green Paper catagories of agent definition, mobility, mindspace, reactivity/proactivity, and planning mechanisms. The response also provides information a Plantangent implementation called ITS (Intelligent Transport Systems). (Contact: Masanori Hattori; masanori@ssel.toshiba.co.jp)

8.1.6. Climate response

(ec/99-08-02; **ftp://ftp.omg.org/pub/docs/ec/99-08-02.zip**)

This response consisted of three papers entitled: "Supporting intelligent agents in a distributed environment: a COOL-based approach" ((by Bruno Dillenseger and François Bourdon), "Mobile Agents White Paper" (from General Magic), and "From Interoperability to Cooperation: Building Intelligent Agents on Middleware) (by Bruno Dillenseger) (Contact: Professor Kimmo Raatikainen: kimmo.raatikainen@cs.helsinki.fi)

8.1.7. ATAM response (first response)

Joint submission by Oak Ridge National Laboratory, NIST, and Carnegie Mellon University.

(ec/99-08-03; ftp://ftp.omg.org/pub/docs/ec/99-08-03.html)

Architectural Evaluation of Agent-Based Systems in Manufacturing. This response focuses on system analysis and evaluation issues in developing agent-based systems. Its objective is to inform the OMG agents community of the need for evaluation of emerging agent-based systems from the multiple perspectives of manufacturing enterprises and to describe an approach that starts to address this need. The specific approach described in the response is based on an architecture evaluation methodology called Architecture Tradeoff Analysis Method (ATAM). The ATAM methodology is in development at the Software Engineering Institute and is central to the new NIST-sponsored project called Test Bed for Agent-Based Systems in Manufacturing. The ATAM methodology has a special emphasis on architectural implications and quality attributes such as availability, modifiability, and security. While the specific application area for

this methodology within the response is manufacturing, the methodology itself is applicable across other areas. (Contact:Nenad Ivezic; ivezicn@ornl.gov)

8.1.8. ATAM position paper

(ec/99-08-04; ftp://ftp.omg.org/pub/docs/ec/99-08-04.pdf)

Architectural Evaluation of Collaborative Agent-Based Systems. The Architecture Tradeoff Analysis Method (ATAM) is an architecture evaluation technique currently evolving at the SEI. This paper focuses on a specific architectural view—the *coordination model*—of selected agent systems, and a collection of quality attributes as previously discussed. It is concerned with the identification of typical quality attributes, scenarios, and agent system architectural patterns as a way to expedite evaluations of real agent systems. This position paper supports ec/99-08-03. (Contact:Steve Woods; sgw@sei.cmu.edu)

8.1.8. FIPA response

(ec/99-08-05; ftp://ftp.omg.org/pub/docs/ec/99-08-05.pdf)

This response was a brief letter from the former president of FIPA. It indicated that while FIPA's work is very much work in progress, a number of significant documents have been produced that may be of interest to the OMG's work. These documents are referenced in this RFP—and are also documented in the section 7.1 of this Green Paper. FIPA is also working in the area of agent standards. They provided references to their current specifications, which utilized IIOP, and to their future work. The future work focuses on creating an architecuture that will support multiple mechanisms for implementing Agent systems over distributed compute systems such as CORBA and Java.

9. Agent RFP Candidates

It is anticipated that RFIs and/or RFPs will be issued for agent-based technologies. For example, the overall goal of RFI-1 is to collect information regarding agent technology from various communities. This information will help guide the OMG in the adoption of specifications that will to extend the OMG Object Management Architecture with agent technology functionality. These extension, then, would further populate or align the OMG with emerging agent standards, protocols, tools, and utilities. This list of RFPs, below, is culled from material covered in this Green Paper and discussions at OMG Agent Special Interest Group meetings.

9.1. List of Candidate RFPs

All of these RFP candidates need more work. Here they are only sketched at the level we talked about them during the Boston meeting. Some additional work is needed to insure that

(a) this is the right list of RFPs and we are not missing something important
(b) the list takes into account other relevant OMG work and avoids duplication, and
(c) we are not duplicating work done on agent standards elsewhere (e.g., FIPA)

9.1.1. Agent identity

- Persistent identity over time
  - Identity belongs to authority
  - Identify authority
- Insuring identity
- Authentication
  - Retiring identity (and shunning, time-bound)
  - GC issues
- One unique id—orposibly many under various circumstances (see FIPA work)
- How does this play in existing agent platforms?

9.1.2. Message conveyance

- Durable, reliable; send/receive
- Might need to build a transport gatewaycurrent OMG architecture handles this.
- Concrete realization of FIPA transport
- Use event or messaging style?
- IIOP, events, CORBA, msg, Java msg, com/OLE
- Message source and value types
- FIPA role, OMG role, gateways

9.1.3. Agent discovery

- Registry/directory service - as such how is this related to trader and to EC brokerage registry and directory services
- Feature -> identity or name/address -> agent properties
- Representational issues
- gateways
- Any agent that …
- Query richness – LDAP Vs much more capable
- Ontology issues
  - Can Trader services be extended?
  - Content-based, parameterized search by ACL & ontology
  - Ontology for common attribute or properties
- Content languages
- Wider community than OMG: properties more abstract- OMG prop, JINI, ….
- Rich property service extension to UML

9.1.4. Agent Communication Language (ACL)

- Encoding of ACL
- Important issues: support multiple ACLs and extensible ACLs
- Also support XML/XMI usage, reflection, dynamic IDL
- What is value-added by ACL? Need to make this clear for all OMG
- At least be able ascertain structure and methods, but what else?
- Ontology issues here also
- Frank McCabe will write a piece on the value-added to current OMG technologies.

9.1.5. Ontology

- Could provide a technical basis for defining OMG domains
- There are a lot of ontologies out there; should we support any or all of them?
- While the relationship to XML is to an implementation, XML will be used to define many poor-man ontologies so what does ontology add.
- What is the best bay to express ontology. Can UML help?
- How does this relate to EC Brokerage RFP on resource description using xml-based and XML schema value types

9.1.6. Content language

- Reflective aspect

9.1.7. Agent security ("trust")

- Orthgonalized; similar to OO security but with some twists
- Interop Security RFP
- FIPA- systematically secure…across transports
- In a touchy-feely sort of way, some agent-only issues are:
  - Who agents tell what to
  - Whether they are trustworthy or not
  - Who do you trust and who don't you trust
- Kate Stout will champion for now
- Issues:
  - Sender Identification
  - Tampering
- Encryption

9.1.8. Agent/object mobility

(Note: triaged - consider requirements sooner so as not to preclude, but RFP later)

- Orthoganalized; similar to OO mobility plus msg forwarding

9.1.9. UML profile for agents & ACL & agent platforms

- What UML extensions are needed to support agents?
- How many of these extensions will the OO UML 2.0 decide to support anyway? For those agents requiements outside of UML 2.0, perhaps separate agent, ACL, and agent platform profiles will be useful.
- How does UML relate to ACL and ontologies.

9.1.10. Other possible RFP areas that are out of scope for the near term

Listed below are a number of valid RFP topics but doing the work to refine them into RFPs awaits champions for each. If someone is interested, please come to OMG Agent WG meetings and participate.

- Logging service - including services for visualization of the log
- Negotiation - but see ECDTF Negotiation RFP
- Team Formation
- Agent Information Management Framework
- Multi modal agent-based user interface framework
- Dialog Management
- Content Filtering
- Configurating and Version Control (triaged - consider requirements sooner so as not to preclude but RFP later)

- RT Upgrade management is relevant
- Constraint langauge
- Planning service
- Scheduling service
- list others here ...

9.2. Criteria for Sequencing RFPs

There is more material here than can be considered in one RFP so the RFPs must be sequenced according to some adoption roadmap.

The center of interest in the Agent Special Interest Group is in **multi agent systems** that are used in application development in robust ways in **distributed environments** that operate in LANS, WANs and over the **web**. For the present, we are less focused on: individual agents, desktop agents, collaborative filtering, and agent-user interfaces.

Criteria that can affect the ordering of RFPs are as follows:

- **interoperability** - agent systems will come in various forms and technology that allows agent and agent systems to interoperate will be immediately useful. RFP areas that provide direct help in interoperability are: agent identity, agent registry/discovery, and agent transport
- **ACL communication** - we need standard ways that agents can communicate with each other and standard ways for them to talk about domain entities in different domains. RFP areas that directly provide help here are ACL, Ontology, and Content Language.
- **security** - if agent technology becomes widely popular it will need security models before people trust agent-based applications. So we need to ask what extensions to the OMG security service are needed and must not be precluded in agent system development
- **mobility** - if agents are to realize the claim of mobility, then mobility services should be in place. We view mobility to be largely orthogonal to agents (or objects) and again only need to insure that earlier RFPs do not preclude adding mobility later in a consistent way.
- **distributed, robust, large-scale** - we believe these are general desirable criteria for agent systems as well as object systems. RFPs should not preclude deployment of agents in such industrial strength environments.

9.3. Initial Roadmap

We recommend that the first two bundles of RFPs be:

- **Agent Interoperability RFP** - covering agent identity, agent transport, and agent registration and discovery
- **Agent Communication RFP** - covering agent communication language, ontology, and content language

The first bundle is in several ways orthogonal to the second, at least insofar as different ACLs could be accommodated by the same Agent Interoperability services. Also, within these bundles, the services should be separately specified as they are separable but they must work closely together so they are included in the same bundle. We currently seem to be on the vector to work on the first bundle first and then the second but we can decide about overlap at the Mesa meeting.

Next on the list should be **Agent Security RFP** when we decide what aspects of security are special to agents and go beyond object security.

Other RFPs will be considered at a later date: these will include some from the deferred list.

IDL (if any) and UML extensions needed by agents will be requested as part of every agent RFP.

9.4. Timetable

- Agent Interoperability RFP - issue RFP by Denver, Oslo or San Jose meeting
- Agent Communication RFP - issue RFP by TBD - issue: overlap or sequential
- others - issue later

9.5. Issues with Current Document

- Is there a good reason why this initial list of RFPs is not architecturally derived from the FIPA architecture? should it cover things like agent management, agent wrappers for objects, ...
- agent persistence - should this be broken out
- should the focus be on agent systems and provide gateways that show how at least OMG solutions like IIOP can help but permitting other kinds of implementations -or- are these just going to be OMG agents (a much more rarified market)
- should the interoperability and communication bundles be overlapped or sequenced?
- add other issue here ...

FIPA did approve doing an architecture that could map to various implementations, such as CORBA.

10. Relationship to OMG Technology and Other Work Efforts

10.1. Relevant OMG services

10.1.1. CORBA 2.3.1 (formal/98-12-01)

The base CORBA Distributed Computing Service Platform enabled through the specification of an Object Request Broker. CORBA 2.3 includes the specification of the Interface Definition Language (IDL), the notions of types, inheritance, operations and attributes, exception management, policy and object access through Portable Object Adapters (POA).

10.1.2. Naming sevice (formal/98-12-09)

The Naming Service provides the ability to bind a name to an object relative to a *naming context*. A naming context is an object that contains a set of name bindings in which each name is unique. To resolve a name is to determine the object associated with the name in a given context (see also Interoperable Naming Service).

10.1.3. Event service

The Event Service provides basic capabilities that can be configured together in a very flexible and powerful manner. Asynchronous events (decoupled event suppliers and consumers), event "fan-in," notification "fan-out," and (through appropriate event channel implementations) reliable event delivery are supported. (see also Notification Service).

10.1.4. Life cycle services

The Life Cycle Service defines conventions for creating, deleting, copying and moving objects. Because CORBA-based environments support distributed objects, life cycle services define services and conventions that allow clients to perform life cycle operations on objects in different locations. Lifecycle services provide a basic framework that includes the notion of mobility, however, the notions of mobility are considered insufficient with respect to mobile object application requirements.

10.1.5. Transaction service

The Transaction Service supports multiple transaction models, including the flat and nested models. The ability of an object or agent to support transactional behaviour is exposed by the transaction policy of the implementation.

10.1.6. Concurrency control service

The Concurrency Control Service enables multiple clients to coordinate their access to shared resources. Coordinating access to a resource means that when multiple, concurrent clients access a single resource, any conflicting actions by the clients are reconciled so that the resource remains in a consistent state.

10.1.7. Externalization service

The Externalization Service defines protocols and conventions for externalizing and internalizing objects. Externalizing an object is to record the object state in a stream of data (in memory, on a disk file, across the network, and so forth) and then be internalized into a new object in the same or a different process. The externalized object can exist for arbitrary amounts of time, be transported by means outside of the ORB, and be internalized in a different, disconnected ORB. For portability, clients can request that externalized data be stored in a file whose format is defined with the Externalization Service Specification.

10.1.8. Query service

The purpose of the Query Service is to allow users and objects to invoke queries on collections of other objects. The queries are declarative statements with predicates and include the ability to specify values of attributes; to invoke arbitrary operations; and to invoke other Object Services. Query services may be relevant to identified requirements for directory services.

10.1.9. Licensing service

The Licensing Service provides a mechanism for producers to control the use of their intellectual property. Producers can implement the Licensing Service according to their own needs, and the needs of their customers, because the Licensing Service does not impose it own business policies or practices.

10.1.10. Property service

Provides the ability to dynamically associate named values with objects outside the static IDL-type system.

10.1.11. Time service

Enables the user to obtain current time together with an error estimate associated with it. Time Services may be used to: ascertains the order in which "events" occurred; generate time-based events based on timers and alarms; compute intervals between two events. The Time Service consists of two services: Time Service manages Universal Time Objects (UTOs) and Time Interval Objects
(TIOs), and is represented by the *TimeService* interface; Timer Event Service manages Timer Event Handler objects, and is represented by the *TimerEventService* interface.

10.1.12. Security service

The security functionality defined by this specification comprises:

- **Identification and authentication** of principals to verify they are who they claim to be;
- **Authorization and access control** - deciding whether a principal can access an object, normally using the identity and/or other privilege attributes of the principal (such as role, groups, security clearance) and the control attributes of the target object (stating which principals, or principals with which attributes) can access it;

- **Security auditing** to make users accountable for their security related actions. It is normally the human user who should be accountable. Auditing mechanisms should be able to identify the user correctly, even after a chain of calls through many objects;
- **Security of communication** between objects, which is often over insecure lower layer communications. This requires trust to be established between the client and target, which may require authentication of clients to targets and authentication of targets to clients. It also requires integrity protection and (optionally) confidentiality protection of messages in transit between objects;
- **Non-repudiation** provides irrefutable evidence of actions such as proof of origin of data to the recipient, or proof of receipt of data to the sender to protect against subsequent attempts to falsely deny the receiving or sending of the data; and Administration of security information (for example, security policy) is also needed.

10.1.13. Trader object service

The OMG trading object service facilitates the offering and the discovery of instances of services of particular types. A trader is an object that supports the trading object service in a distributed environment. It can be viewed as an object through which other objects can advertise their capabilities and match their needs against advertised capabilities. Advertising a capability or offering a service is called “export.” Matching against needs or discovering services is called “import.” Export and import facilitate dynamic discovery of, and late binding to, services. To export, an object gives the trader a description of a service and the location of an interface where that service is available. To import, an object asks the trader for a service having certain characteristics. The trader checks against the service descriptions it holds and responds to the importer with the location of the selected service’s interface.

10.1.14. Collection service

Collections are groups of objects which, as a group, support some operations and exhibit specific behaviors that are related to the nature of the collection rather than to the type of object they contain. Examples of collections are sets, queues, stacks, lists, binary, and trees. The purpose of the Collection Object Service is to provide a uniform way to create and manipulate the most common collections generically.

10.1.15. Notification service (telecom/99-07-01)

This document describes a CORBA-based Notification Service, a service which extends the existing OMG Event Service, adding to it the following new capabilities: the ability to transmit events in the form of a well-defined data structure, in addition to Anys and Typed-events as supported by the existing Event Service; the ability for clients to specify exactly which events they are interested in receiving, by attaching filters to each proxy in a channel; the ability for the event types required by all consumers of a channel to be discovered by suppliers of that channel, so that suppliers can produce events on demand, or avoid transmitting events in which no consumers have interest; the ability for the event types offered by suppliers to an event channel to be discovered by consumers of that channel so that consumers may subscribe to new event

types as they become available; the ability to configure various quality of service properties on a per-channel, per-proxy, or per-event basis; an optional event type repository which, if present, facilitates the formation of filter constraints by end-users, by making information about the structure of events which will flow through the channel readily available.

10.1.16. Messaging service (orbos/98-05-05)

Collections are groups of objects which, as a group, support some operations and exhibit specific behaviors that are related to the nature of the collection rather than to the type of object they contain. Examples of collections are sets, queues, stacks, lists, binary, and trees. The purpose of the Collection Object Service is to provide a uniform way to create and manipulate the most common collections generically.

10.1.17. Task and session (dtc/99-08-03)

This specification defines a model that represents the obvious objects and relationships in the end user view of a distributed system. End users are the people that directly interact with the system. These obvious things include:

- the existence of other users, roles, organizations, projects, etc.
- distributed program and information resources
- places where users and resources are found, and processes execute
- tasks that range from ‘print a file’ to ‘build a software product (with other people)’

The Task and Session Model is a first step towards specifying these obvious things in a consistent and common manner. This small first step enables applications to significantly raise the level of collaboration and resource sharing within projects and enterprises, while also raising the level of abstraction for users of distributed systems.

10.1.18. Community framework (dtc/99-07-03)

The Community framework defines a Member that is associated with a abstract Membership. A Membership is a type supporting a managed set of Members. Members within a Membership are characterized by business roles. Membership provides the basis for the definition of Community and Agency interfaces. Community is combination of Workspace and Membership. Agency extends Community to include expose of legal credentials.

10.1.19. Collaboration framework

Collaboration Framework, defines a process through which different models of collaboration rules can be managed. The Collaboration module is based extensively on interfaces from the Community module. A framework and set of collaboration process models are defined supporting *bilateral* negotiation, *multilateral* negotiation and *promissory* commitment. The *bilateral* model defines a policy under which two pricipals can progressively move from a non-agreed to agreed state. The *multilateral* model defines a policy supporting the raising of motions, seconding, amendments, and subsequent voting within a declared membership. Collaborative process models are static data structures used to define rules and constraints. The

models define the semantics of a given collaboration process and provide the building blocks for the creation of custom process descriptions.

10.2. Overlap with OMG Interests

Agent-based technology is a platform technology. As such, it can affect not only other platform technology, but impact domain technology as well. Listed below are the current OMG technical committees with an indication of how agent-based technology might be useful.

(Each of these technical committees needs a brief description of where we think Agent WG interest might overlap.)

Autonomous Decentralized Service System DSIG (adss@omg.org)

Business Object Domain Task Force (bomsig@omg.org)

Command, Control, Computing, Communications and Intelligence SIG (c4i@omg.org)

CORBAgis DSIG (corbagis@omg.org)

Document Management Platform SIG (docman@omg.org)

Electronic Commerce Domain Task Force (ec@omg.org)

Enterprise Customer Interaction Systems (ecis@omg.org)

Finance Domain Task Force (finance@omg.org)

Healthcare (CORBAmed) Domain Task Force (healthcare@omg.org)

Internet Platform Special Interest Group (internet@omg.org)

Life Sciences Research DTF (lifesciences@omg.org)

Manufacturing Domain Task Force (mfg@omg.org)

Realtime PSIG (realtime@omg.org)

Security SIG (secsig@omg.org)

Analysis and Design Task Force (sigad@omg.org)

Analytical Data Management DSIG (statistics@omg.org)

Telecommunications Domain Task Force (telecom@omg.org)

Transportation DTF (transport@omg.org)

Utilities DTF List (utilities@omg.org)

11. Other Similar Standards and Efforts

11.1. FIPA

11.1.1. Introduction

FIPA, the Foundation for Intelligent Physical Agents, welcomes OMG's interest in agent technology. FIPA has been working to develop and promote standardization in the area of agent interoperability since 1996. FIPA has an on-going work program, meeting around the globe on a quarterly basis, with excess of 50 member organisations. With 3.5 years of existence and the extensive specifications produced (see http://www.fipa.org/), as such FIPA is a reference organization in intelligent agents.

A fomal liason has been established between FIPA and the OMG's Agents WG: Frank McCabe is the liason from FIPA to the Agents WG and David Levine is the liason from the Agents WG to FIPA.

FIPA recommends that the OMG consider the specifications that have been developed by FIPA as a background to OMG's possible future standardization efforts. While FIPA's work is very much work in progress, a number of significant documents have been produced that may be of interest to the OMG's work.

11.1.1. Agent communication language

(http://www.fipa.org/spec/f8a22.zip)

This details the syntax and semantics of a high-level agent communication language that is based on speech acts. A primary benefit of employing this language is that the semantics of communication can be preserved in an open manner.

11.1.2. Agent/software integration

(ftp://ftp.fipa.org/Specs/FIPA97/f7a13pdf.zip)

This specification details a standard way in which non-agent based software can be integrated into a FIPA agent platform.

11.1.3. Agent management

(http://fipa.umbc.edu/mirror/spec/fipa8a23.doc)

This specification, which also draws on http://www.fipa.og/spec/f8a21.doc, outlines the necessary specifications needed for managing agents on an agent platform. A point of particular interest to the OMG is the mandatory use of IIOP as the baseline transport protocol. It is envisaged that other transport protocols may be standardised to meet specific needs e.g. for wireless applications.

11.1.4. Human/agent interaction

(http://www.fipa.org/spec/fipa8a24.zip)

This details models for integrating agents with human participation.

11.1.5. Agent mobility

(http://www.fipa.org/spec/fipa8a27.doc)

This details various models and protocols to support agent mobility between agent platforms.

11.1.6. Ontology services

(http://www.fipa.org/spec/fipa8a28.zip)

This details specifications for managing ontology services and ontology models in a consistent and open framework.

11.1.7. Agent naming

(http://www/fipa.org/spec/fipa9716.PDF)

This specification, which is not yet officially adopted by FIPA, outlines models for consistent naming of agents.

11.1.8. Message transport

(http://www.fipa.org/spec/fipa9716.PDF)

This specification, which is not yet officially adopted by FIPA,outlines the requirements and specifications of messaging services between agents and between agent platforms. In addition, we recommend that OMG seriously consider: http://www.fipa.org/spec/fipa9710.pdf.

11.1.9. Conclusion

Many of the questions in the OMG RFI are being addressed in this FIPA Architecture Overview document; however, this is work in progress and may change significantly in the coming months. More generally, FIPA welcomes visitors to it's web pages at

http://www.fipa.org, and the mirror sites in Japan

http://fipa.comtec.co.jp/index-e.html and the USA http://fipa.umbc.edu.

FIPA implementations are currently pre-commercial, but there are in excess of 10 frameworks under development.

OMG members wishing to participate in a FIPA meeting will be welcome and should contact the FIPA Secretariat Teresa Marsico (secretariat@fipa.org).

11.2. CLIMATE

The Cluster for Intelligent Mobile Agents for Telecommunication Environments (CLIMATE) represents a pool of agent technology related projects within the European Union collaborative research and development program on Advanced Communications Technologies and Services (ACTS). CLIMATE was formed in Spring 1998 and currently comprises 14 core projects, which investigate the usage of Agent Technologies in various application areas, such as service

control in fixed and mobile networks, telecommunications management, electronic commerce, multimedia applications, etc. The incorporation of emerging agent standards, particularly the OMG Mobile Agent System Interoperability Facility (MASIF) and the Foundation of Physical Intelligent Agents (FIPA) specifications, is key in this context.

Dr. T. Magedanz is interested in establishing some kind of liason between the Agent WG and CLIMATE. Currently, Kimmo Raatikainen (kimmo.raatikainen@cs.helsinki.fi) is our representative from CLIMATE to the Agents WG. However, no liason efforts are yet underway. For more information, see www.fokus.gmd.de/research/cc/ecco/climate/climate.html.

11.3. KQML/KIF

KQML or the Knowledge Query and Manipulation Language is a language and protocol for exchanging information and knowledge. It is part of a larger effort, the ARPA Knowledge Sharing Effort which is aimed at developing techniques and methodology for building large-scale knowledge bases which are sharable and reusable. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. KQML can be used as a language for an application program to interact with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving. For more information, see www.cs.umbc.edu/kqml.

11.4. US DARPA

The Defense Advanced Research Projects Agency (DARPA) has several research programs that address aspects of agent technology. These include:

- **Control of Agent-based Systems** (http://coabs.globalinfotek.com, contact: Jim Hendler jhendler@darpa.mil). This program has demonstrated how a heterogeneous collection of agent systems and services can be made to operate via a grid like that described in section 4.5 in order to demonstrate agents in a DoD noncombatant evacuation order (NEO) scenario.
- **Advanced Logistics Project** (contact: Todd Carrico tcarrico@darpa.mil). This project, while focused on logistics, has explored an interesting architectural notion called a cluster which contains semantic plug-ins. Clusters are like agents in some regards.
- **DARPA Agent Markup Language** (contact: Jim Hendler jhendler@darpa.mil). This is a new DARPA program, just forming, that will focus on extending the XML-like markup languages with ontology capabilities. See http://www.darpa.mil/iso/ABC/BAA0007PIP.htm.

11.5. EU AgentLink community

AgentLink is Europe's ESPRIT-funded Network of Excellence for agent-based computing. It is a coordinating organisation for research and development activities in the area of agent-based computer systems funded by the European Commission. As such, AgentLink supports a range of activities aimed at raising the profile, quality, and industrial relevance of agent systems in Europe.

As the fifth framework approaches, AgentLink will work, together with other similar groups (such as CLIMATE) to ensure that the potential for agent technology is recognized and fully exploited throughout European industry and academia.

AgentLink divides its activities into four main areas:

- Industrial action
  - facilitating technology transfer through a program of industrial meetings,
  - workshops, standardization updates, and working groups
- Research coordination
  - promoting excellence in European agent research through support for
  - workshops, special interest groups, and dissemination of research results
- Teaching and training
  - establishing agent related skills throughout Europe by support for summer
  - schools and courses
- Infrastructure and management
  - providing an infrastructure through which AgentLink can do its work, including a
  - WWW site, regular newsletter, mailshots, and an awareness program.

For more information, see www.agentlink.org.

11.6. Others

(none know at this time.)

12. Appendix

12.1. Glossary

The Glossary that will be insterted here is currenly located on the Agents WG websit. See http://www.objs.com/agility/tech-reports/9909-agent-glossary.html

12.2. References

[Amase] http://b5www.berkom.de/amase/

Bradshaw, J. (ed.). 1997a. Software Agents. American Association for Artificial Intelligence/MIT Press.

Bradshaw, J. 1997b. An Introduction to Software Agents. In (Bradshaw 1997a).

[Cag97] Caglayan, Alper, and Colin Harrison, *Agent Sourcebook*, John Wiley & Sons, New York, 1997.

[Cameleon] http://www.comnets.rwth-aachen.de/~cameleon/

[FIPA] http://www.fipa.org

[Klei97] Kleinrock, L.: "Nomadicity: Anytime, anywhere in a disconnected world", Mobile Networks and Applications, Vol. 1, No. 4, (Jan. 1997), pp. 351-357.

[Lieb95] Lieberman, H. (1995) "Letizia: An Agent That Assists Web Browsing", *Proceedings of the International Joint Conference on Artificial Intelligence*, Montreal.

[M99] Frank Manola, " Providing Systemic Properties (Ilities) and Quality of Service in Component-Based Systems." 1999. URL: http://www.objs.com/aits/9901-iquos.html.

[Monads] http://www.cs.Helsinki.fi/research/monads/

[MT99] Frank Manola and Craig Thompson, "Characterizing the Agent Grid." 1999. URL: http://www.objs.com/agility/tech-reports/990623-characterizing-the-agent-grid.html.

[OMG99] OMG Telecom DTF RFP on Wireless Access and Terminal Mobility in CORBA, OMG Document No telecom/99-05-05.

[PILC] IETF Working Group on Performance Implications of Link Characteristics (pilc). http://pilc.grc.nasa.gov/pilc.

[Tho98] Craig Thompson (ed), Final Report: OMG-DARPA Workshop on Compositional Software Architectures. 1998. http://www.objs.com/workshops/ws9801/index.html.

12.3. Agent Reference Architecture

It is valuable to try to get a big picture of the scope and landscape of the agent technology area. Figure 1 below is one fairly detailed attempt to list the kinds of technology and component capabilities that an agent reference architecture must eventually account for.

In one interpretation, this figure is just a listing of agent-relevant technology. As such it provides a way to scope the kinds of technology that OMG is likely to be interested in. This scoping is

useful to people to understand the kinds of technology that agent systems might provide or depend on.

Another interpretation (one we will not push too far) is that many of the elements in the picture describe component capabilities that one comes to expect in agent systems (especially multi-agent systems). That is, up to a point, this diagram can be taken as an initial agent technology reference architecture. The right half of the diagram indicates the kinds of elements that one must account for in an agent architecture - for instance, kinds of agents (mobile, intelligent, …) and agent systems that provide agent-friendly environments and resources. In addition, it lists communication capabilities like ACL, coordination capabilities for how agents might interact in ensembles and societies, and agent capabilities like planning. The left half of the diagram focuses more on infrastructure, interoperability, and system-wide capabilities that we expect agent systems to account for. Many of the services that correspond to current and proposed CORBAservices fit the list of services that might be useful in agent systems.

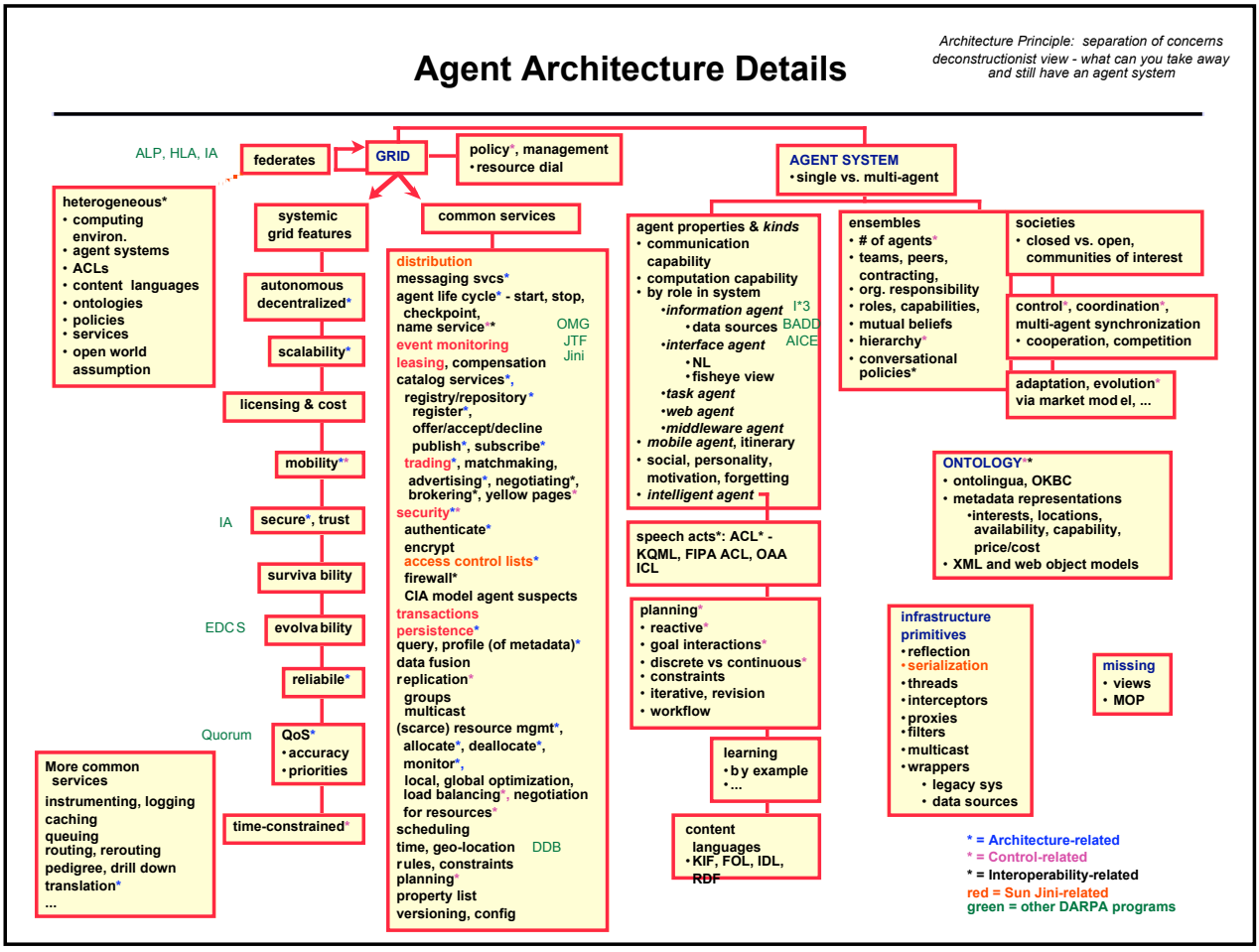

Figure A.1: Agent Reference Architecture

One final interpretation is to view this picture is as an agent technology ontology which provides a listing of the terms used in agent systems along with a little intuition of their meaning. Of course, some terms are worth whole diagrams of their own (e.g., information agents, planning).

When the diagram is coupled with the glossary of Appendix 8.1, then the ontology interpretation is bolstered somewhat.