NOTE TO USERS

Page(s) not included in the original manuscript and are unavailable from the author or university. The manuscript was scanned as received.

pages #176

This reproduction is the best copy available.

UMI®

A Critical Study of Multi-Agent Systems: Models, Architectures and Applications

Zichao Wang

A Major Report in the Department of Computer Science

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

May 2003

© Zichao Wang, 2003

National Library of Canada
Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services
Acquisisitons et services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-83921-4
Our file Notre référence
ISBN: 0-612-83921-4

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

Canada

ABSTRACT

A Critical Study of Multi-Agent Systems: Models, Architectures and Applications

Zichao Wang

This is a review paper on multi-agent systems with focus on the model, architecture and applications. Multi-agent is a software abstraction that represents users in various tasks performed in a heterogeneous environment. This is a contentious topic that attracts many researchers and repels others. It represents a research frontier in a number of fields including computer science, artificial intelligence, robotics etc. Multi-agent system has been an extreme active research area since 1992, and now has raised considerable interest in the research community and in industry. But the long promised deployment has not materialized yet. This review paper looks back at the history, present status and potential future of multi-agent systems on critical view base. It consists of four chapters: Chapter 1 focuses on multi-agent system modeling; Chapter 2 on multi-agent systems architecture, Chapter 3 on multi-agent system applications, platforms and development tools, Chapter 4 gives a brief summary over multi-agent system history during the last 10 years, the present development status and future direction. This report has demonstrated that multi-agent system is still far away from complete. It expects a convincing story from either the artificial intelligence society or software engineering community. However, this does not prevent people from believing that agents would become next generation software programming abstraction. Multi-agent systems would play significant roles in many application domains including software engineering, computer network, robotics, automated manufacturing, etc.

Acknowledgements

It was 2002 Spring quarter when I took COMP7231: Distributed Computation. Prof. H. F. Li gave this course, and this was the first time I had the chance to listen to his lecture. I was very impressed with his style of teaching. Distributed Computation is well known for its difficulty, not only the theory, but the way that a teacher transmits the knowledge to students. The entire class felt lucky since Prof. Li knows how to handle all at the best. The final result was great. All students were able to finish the assignments and the major project successfully. Our group (three students) chose a topic relevant to memory sharing problem, which nobody can say "Oh, piece of cake", but we made it working. We can't attribute this entirely to our hard working, since Prof. Li guided up through from the very beginning. After that two of the three students came to Prof. Li for supervision, either for Master Thesis or Major Report, just to have another chance to benefit more from Prof. Li's teaching.

When we first sat together for a brain storming over potential projects, Prof. Li brought this topic to me. His rich knowledge in distributed/parallel computing makes him confident that the Multi-Agent System could be another frontier in next decade. Anyone, who devotes himself to this area, could learn the most. I became convinced and started a brief survey. Within a few days, I made up my mind, and started this project. It has turned out to be a great journey, and once again I have learned from Prof. Li the ways of facing the challenge and new technology.

This project started in November 2002, and the first draft was submitted at the end of April 2003. During the last six months, I always brought chunk drafts to Prof. Li, discussed with him during the appointment (it was always weekend due to my schedule, sorry for that), and then turned them into readable papers with his suggestions. This is the way I gradually made the final version. It is impossible for me to finish this project without Prof. Li's supervision. There is one phrase, frequently mentioned by Prof. Li, "BE CRITICAL", which I finally realize as a "driving force" for a new thing. To understand a new thing, one must be critical when reviewing it. Then it is possible for one to "dig the root" out, and create his own world. This is a very efficient way for people to understand the Multi Agent Systems. Thanks for Prof. Li, I am able to follow this way and has touched the essentials of this great research area.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ACL:	Agent Communication Language
ABA:	Agent-Based Application
ADT:	An Agent Development Tools
AIP:	Agent Interaction Protocol
ALL:	Agent Language Library
AO:	Agent Object
AOP:	Agent Object-Oriented Programming
ARA:	Autonomous Robot Agent
AsOP:	Aspect-Oriented Programming
BSL:	Behavior Specification Language
CORBA :	Common Object Request Broker Architecture
CPN:	Colored Petri Nets
DAI	Distributed Artificial Intelligence
DCOP:	Distributed Constraint Optimization
DPS	Distributed Problem Solving
FIPA	Federated Intelligent Physical Agents
US DARPA	US Defense Advanced Research Projects Agency
HCI	Human-Computer Interface
IMAJ:	Intelligent Mobile Agents in JAVA
JADE	Java Agent Development Framework.
JAE:	Java Agent Environment
JCAFE:	Jini Compositional Agent Framework for the Enterprise
JECF	Java Electronic Commerce Framework
Klaim:	Kernel Language for agent interaction and mobility
LEAP	Light Extensible Agent Platform
MAAPL:	Mobile Agents Abstract Plan Language
MAsS	Mobile Agents System

MAS:	Multi-Agent System
OMG	Object Management Group
OMGAW	Object Management Group Agent Work Group
OO:	Object-Oriented
OOP:	Object-Oriented Programming
PAI	Parallel Artificial IntelligenceI
PN:	Petri Nets
ROC:	Remote Object Communication
SW:	Software
UML:	Unified Modeling Language

Chapter 1 Agent, Agent Systems, Agent Modeling

What Are In This Chapter: *This chapter focuses agent, agent system and agent modeling. We first look at what can be defined as an agent and an agent system, then what are the agent abstractions in Section 1.1. In Section 1.2, we introduce agent modeling. The current proposed major agent models have been presented and discussed. To understand why agent/agent system becomes popular, a comparison study is made on agent paradigm and process paradigm in Section 1.3. Detailed studies have been made on object/Agent, AO-program/OO-program, agent-modeling/processing modeling and the differences between agent and object. We further investigate tools applicable for agent development in Section 1.4. UML, AUML, AsOP, PN and Patterns and Toolkits have been discussed in this section. Finally, a brief discussion has been made over limitations, applicability and missing aspects of agent models.*

1.1 Agent and Agent Systems

1.1.1 Agent, Agent Program and Agent System

1.1.1.1. Agent and Agent Program

No one knows when and how the term "Agent" was first introduced into computer sciences. But it has been assumed that it came originally from the Artificial Intelligence Community (AI), where an Agent has been defined as an integrated system performing some tasks on behalf of a user. The recent trend in Distributed Artificial Intelligence (DAI) extends the concept of "Agent" to "Software (SW) Agent" by considering it as Software units that may be customized and composed to form a complex system (Sycara et al., 1996). This has been accepted widely in software engineering. On the other hand, the AI community insists that the notion of agents should be taken in a broader sense, encompassing a wide spectrum of computational entities that can sense their local conditions and accordingly make decisions on how to react to the sensed conditions by performing certain behaviors in the task environment (Liu, 2001).

A *SW agent* is often referred to as an autonomous software entity that can interact with its environment. A SW agent is implemented using software that perceives its environment through its sensors, and acts upon its environment via effectors.

To simplify, we would limit ourselves within SW engineering application domain, and unify all names (*Agent, SW Agents, Agent Program*) into a single term: *Agent*. We can think of an agent as an extension of traditional objects based on the fact that an agent abstraction should correspond more to its functional aspects than to blocks of executable code. This can be viewed from the following aspects:

- An agent, like an object, has an internal state, which reflects its knowledge. This internal state is specified and refined during an agent's lifetime.
- An agent has reasoning capabilities that determine its internal behavior. The agent behavior may change dynamically at run time, and may be specified in a declarative way.
- An agent shows an external behavior consisting of communicative acts to other agents or control actions on SW or hardware (HW) devices. The communication with each other goes through standard agent-independent communication languages (Genesereth and Ketchpel, 1993).
- An agent has an identity, which distinguishes one agent from the others in the same system, and it is expressed in term of a name, address, and a description of the service (role) it provides. An agent's identity can be communicated to other agents and agents can freely appear and disappear in the system or change their location.
- An agent distinguishes itself from a traditional object by the way it communicates with others; the level of information exchange is often higher.

1.1.1.2 Strong Agent and Weak Agent

Woodldridge and Jennings (1995a) first proposed the notion of "strong agent " and "weak agent" concept. An agent is considered to adhere to a weak notion of agency if it is comprised of the following properties:

- *Autonomy*. Once launched with the information describing the bounds and limitations of their tasks, an agent should be able to operate independently from their user, that is, autonomously in the background (Castelfranchi, 1995). To this end, an agent needs to have control over its actions so that it can determine what to do when an action succeeds or fails. Moreover, an agent must be

able to augment its internal state so that it can make rational decisions based upon the information that it has gathered.

- *Social ability*. To effect changes or interrogate their environment, an agent must possess the ability to communicate with the outside world (Genesereth et al., 1997). This interaction can exist at a number of levels depending upon the remit of the agent. But typically an agent would need to communicate with other agents and the local environment (to maintain/discover information) and users (to appraise them of their progress).
- *Reactivity*. Agents need to be able to perceive their environment and respond to changes to it in a timely fashion, depending upon their remit. For example, an agent's task could be to monitor a local file system, informing the user when changes occur to a particular file set. This implies that the agent has an awareness of the appropriate filing system and how to interrogate it; agents need not only be aware of their environment, but also be aware of what the changes mean and how to react to them.
- *Pro-activeness*. To help differentiate an agent from another piece of software, agents need to be able to exhibit pro-activeness, that is, the ability to effect actions to achieve their goals by taking the initiative. This means that an agent needs to appreciate the state of their environment and to decide how best to fulfill their mission target. In other words, an agent has the ability of reasoning and learning based on their knowledge.

Further notions of attributes for agents include descriptions that possess more specific meaning than weak agency, that is, they are attributed characteristics and tasks that would normally be ascribed to human. People consider this type of agent ***Strong Agent***. Shoham (1993) describes the mentalistic notions of knowledge, belief, intention and obligation that might be attributed to strong agents above and beyond those defined for a weak agent. One of the major reasons of using "mentalistic" notion rather than knowledge is that it is not always easy to reach an exact match between the formal properties of these formal constructs and common sense, but rather aim to reach a balance between computational utility and common sense (Shoham, 1993). Dennett (1987) describes such an agent as an *intentional system*. An intentional system is a system that can be best described by the *intentional stance*; the ascription of abstraction notions to systems for the purpose of describing how they work. For example, although the

technical description of a computer system may be available, it is too complex to use when describing, say, why a menu appears when a mouse button is clicked over a certain area of the display. The intentional notions as described by Shoham (1993) are useful for providing convenient and familiar ways of describing, explaining and predicting the behavior of complex systems. Dennett (1987) suggests that strong agents are best described by the intentional stance.

Bates uses this concept of strong agents and takes it into anthropomorphic areas by considering the implications of believable agents, that is, agents that try to model a human approach to their interaction with the user by displaying emotions (Bates, 1994). Additionally, Maes (1994a) talks about representing agents, which visually attaches an icon or a face to associate them with cartoon or computer characters. These types of agents are being used in both Human Computer Interaction (HCI) scenarios to help the social interaction between a user and their agents, and also in the computer gaming community to produce virtual characters that react in believable and human ways to given situations.

1.1.1.3. An Agent-System, or a Multi-Agent System

An agent system or a multi-agent system refers to an agent-based system, which usually consists of more than a single agent, and each of them is of different type. For example, there could be a *BookBuyer Agent*, *BookSeller Agent*, and *Broker Agent* in an E-Commerce book trade agent system. In this agent system, these three agents, *BookBuyer Agent*, *BookSeller Agent* and *Broker Agent* perform their roles on behalf of the buyer, the seller and the broker (Silva et al., 2000). Such a system has been developed to solve specific tasks in a distributed fashion, involving multiple agents of different capabilities. The agents in the system should coordinate, cooperate, and sometimes compete among themselves in order to accomplish a given task.

1.1.2 Mobile Agent and Mobile Agent System

If an agent is viewed from its physical locality in the runtime system, we can see two types of agents: stationary agent and mobile agent.

Stationary Agent: This type of agent does not move. It is physically stationary and communicates with its environment through conventional means, such as remote procedure calling and messaging.

Mobile Agent: it is not bound to the system on which it begins execution. It is free to travel among the hosts in its environment. Created in one physical host (execution environment), it can transport its state (typically means the attribute values of the agent that help it determine what to do when it resumes execution at its destination) and code (class code necessary for an agent to execute) with it to another execution environment in the network, where it resumes execution. The content to be transported along with mobile agent is a challenging issue. Most researchers believe that the mobile agent should bring only the agent context, but not the data and the source code.

Scope: a mobile agent has the unique ability to transport itself from one system in a network to another in the same network. This ability allows it to move to a system containing an object, or agent, with which it wants to interact and then to take advantage of being in the same host or network as the object. This defines *the scope of a mobile agent.*

Mobile agent should be viewed as a technology that can solve a lot of problems in a uniform way rather than a technology that enables completely new things that weren't possible otherwise. Kotz and Gray (1999) suggest that Mobile agents are programs that can migrate from host to host in a network, at times and to places of their own choosing. The state of the running program is saved, transported to the new host, and restored, and allowing the program to continue where it left off (Kotz and Gray, 1999)

Mobile agents differ from "applets", which are the programs downloaded as the results of a user action, then executed from beginning to end on one host. Mobile agent system differs from process-migration system in that the agent moves when they choose, typically through a "jump" or "go" statement, whereas in a process-migration system the system decides when and where to move the running process (typically to balance CPU load).

1.1.3 Agent Abstraction

Although Agent-Oriented Programming was proposed as early as in 1993 (Shoham, 1993), agent abstraction is not clear even today. Part of the reason is attributed to the fact that agent is far more complicated than an object, which forms the center of object-oriented programming (OOP). OO concepts are introduced along with the evolution of programming languages. In OOP, emphasis is given to identifying the entities to be manipulated in solving a problem and extracting their essential features to build a model, or selecting an existing model to the user. The OO model is basically data model (input and

output data) and procedural components for problem domain with well-defined structure (a descriptive model). Thus, the class concept, a natural abstraction for object, provides the most successful mechanism for a user defined data type built on an object.

As abstraction, agents and objects serve complementary roles. Agents act autonomously, driven by their goals and plans, reacting to their environment and cooperating with other agents. Objects encapsulate data structures and operations and provide services to other objects. In this sense, Wooldridge et al (2000) states "There is a fundamental mismatch between the concepts used by object-oriented developer... and the agent-oriented view". This comes from the fact that objects as mere service provides has its origins in the paradigms of sequential OO programming, and is no longer adequate when considering concurrency for concurrent objects (agents may be considered as concurrent objects, but they are more than that). To bridge this gap, Object Management Group (OMG) proposed the concept of active objects (Odell and Bock, 1999), which addresses the agent paradigm. What is still missing in the active objects is the idea of goal-driven behavior or pro-activity of agents and the related concept of autonomy.

We have not seen a well-established abstraction for an agent, which can be accepted as a universal model or simplification for a physically realistic agent. We even do not know if there exists such a universal agent due to our limited understanding from the following three aspects:

1. A general (technical) definition of the term agent and their typical aspects/behavior are still forthcoming. Although we say that agent can be modeled as an extension of object, some important characteristics of agents can be identified which distinguish them from object or programs (Franklin and Kraesser, 1997). These include, but not limit, reactivity, autonomy, pro-activity, and cooperation. These are intrinsic properties of an agent, and are related to the change of a dynamic environment. For example, reactivity is the capability of an agent to perceive its environment and react to changes. It has been proposed that a base agent abstraction should take at least these four properties into account.

2. How are these typical behaviors captured in an agent abstraction? There could be different approaches. For example, reactivity can be considered as a prerequisite for purposeful autonomy of agents, and it is already captured within the concept of active objects. In Depke et al's (2001) approach, an agent's behavior is specified by a set of transformation rules modeling its operations.

In agent abstraction, these typical behaviors of an agent can be considered as properties for modeling. We need an approach to capture these basic properties in any agent abstraction.

3. Separation of mobile agent society from intelligent agent society. This separation arose since the agent concept was introduced into the SW engineering society. Mobile agent builder tends to concentrate on the subsystems for shipping any piece of code/context around, and stress mobile computation with resource/data sharing in a distributed heterogeneous environment. The intelligent agent community tends to focus on application specific problems. Some people emphasize intelligent agents, while others focus on agent mobility, and their research and development effort do not integrate. This leads to differences in ontologies, languages, goals, and ways of looking at an agent. For example, intelligence is neither the focus, nor the particular interest in a mobile agent system (Silva et al., 2000). The point is that not all agents in agent applications should be intelligent and/or mobile. This depends on their requirement.

1.1.4 Discussions

The term "agent" frequently appears in three research communities: AI, various applications, such as manufacturing automation (AMA), and mobile agent. They focus on different aspects. But there is something in common; an agent performs tasks on behalf of a user. The difference lies largely on the behavior/properties of an agent, such as strong versus weak agent, mobile versus stationary agent. Most AI agents are strong agents, and manufacturing automation also requires strong agent. Agents of operating system (OS) could be less intelligent (weak agent), but most of them must be mobile.

Thus, an agent is not well defined without reference to a particular notion of agenthood. We need an agent model. Some models are very austere, defining an agent in automata-theoretic terms, and others use a lavish set of vocabulary. With certain extensions, a universal agent definition may be possible.

To summarize, an agent can be considered as a SW/HW entity with a well-known identity, state and behavior, with some autonomy degree of representing its user. These components are defined in a somewhat precise manner, and sometimes correspond to physical entities. In this view, agenthood is in the mind of the programmer: what makes any HW or SW component an agent is precisely the fact that people choose to analyze and control it in these properties. This conceptual agent abstraction is application dependent and should be acceptable to all three research and user communities. From a more technical

point of view, an agent should be implemented based on an active object of medium granularity. This means that an agent is an instance of some defined abstract class – with its own group of threads, state and code – identified by a unique global identity. From a higher-level, conceptual perspective, agent is a basic but powerful abstraction to design complex, distributed and dynamic applications. From yet another perspective – the human-computer-interface (HCI) – agents may be viewed as a new interface paradigm to help end-users access future Internet applications.

Some of the properties that agents may possess in various combinations include:

- **Autonomous** - is capable of 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** - is 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** – conducts interaction that is marked by friendliness or pleasant social relations, that is, where the agent is affable, companionable, or friendly.
- **Mobile** – is 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** – is 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** – is able to choose an action based on internal goals and the knowledge that a particular action will bring it closer to its goals.
- **Unpredictable** – is able to act in ways that are not fully predictable, even if all the initial conditions are known. It is capable of non-deterministic behavior.
- **Temporally continuous** - is a continuously running process.
- **Character** – has believable personality and emotional state.
- **Transparent and accountable** - must be transparent when required, yet must provide a log of its activities upon demand.

- **Coordinative** – is able to perform some activity in a shared environment with other agents. Activities are often coordinated via plans, workflows, or some other process management mechanism.
- **Cooperative** – is able to coordinate with other agents to achieve a common purpose. (*Collaboration* is another term used synonymously with cooperation.)
- **Competitive** – is able to coordinate with other agents except that the success of one agent implies the failure of others (the opposite of cooperative).
- **Rugged** – is able to deal with errors and incomplete data.
- **Trustworthy** – is truthful.

An agent abstraction must represent many of these properties. Thus, one can construct some basic agent, which has the minimum property set. The other type of agents can be built on these base agents. I believe that the Autonomous, Adaptive, Pro-active and Intelligence are the four basic properties, which an agent must have. Here we can see the Intelligence is of particular importance. Mobility is another important property for a mobile agent. Since the intelligence and mobility are two common properties for most agents in the application level, they should exist in a higher-level abstraction. The next level of abstraction will be the level, on which the properties of an agent should reflect specific application requirements. For example, an interface agent and an information-retrieval agent have specific application characteristics. Thus we may view an abstraction as a hierarchy structure shown in Figure 1.1, in which the agent abstraction can be constructed in three levels: base agent, mobile agent and application agent.

One can observe that the agent abstraction at different level can capture different properties of an agent, and there is an inheritance relationship from one level to the next. In general, the lower level agent abstraction should inherit all properties of its higher-level agent abstraction.

1.1.4.1 Basic Agent Abstraction

This is the base agent, which should have all the common behavior/properties of a weak/strong agent. All behavior/properties of an agent at this level should be application/application domain independent. It basically includes intrinsic properties of a weak/strong agent object. The following are some typical base agent abstractions proposed by OMG (2001):

*(a) **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 also to observable events within the environment. Proactive agents will actually poll the environment for events and other messages to determine what action they should take. In short, an agent can decide when to say

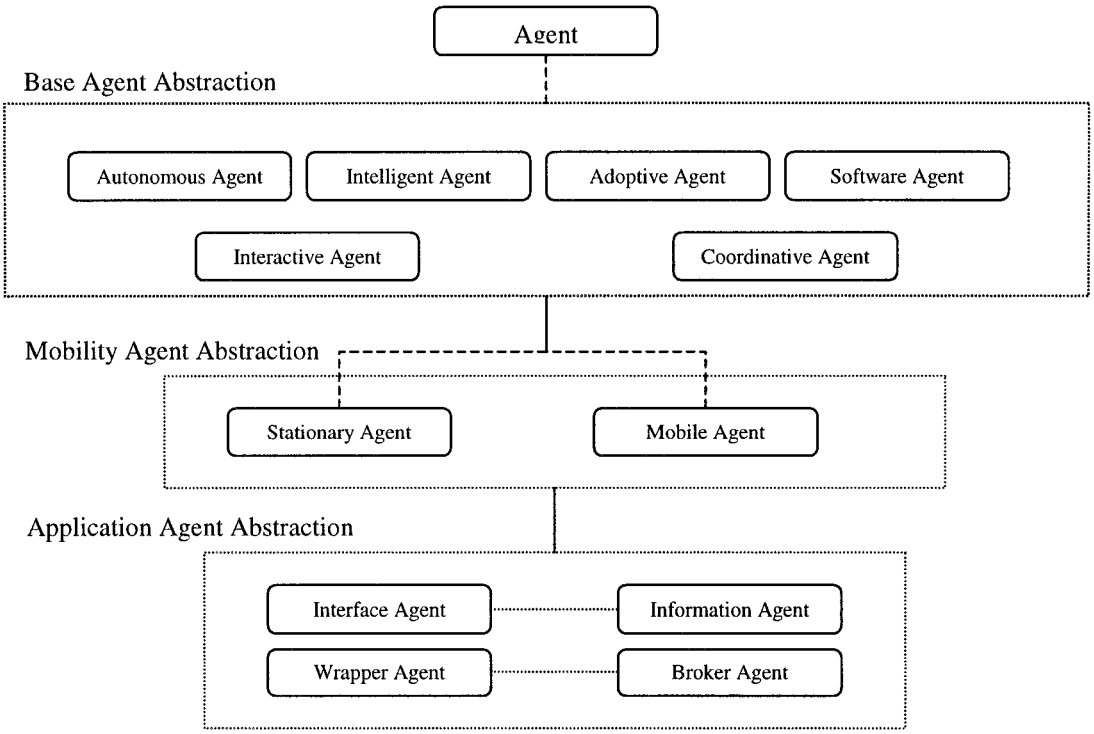

Figure 1.1. Agent abstraction hierarchy structure

"go". Agents may also employ some degree of unpredictable (or non deterministic) 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 is 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."

(b) 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; they interact indirectly by 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.

(c) 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 boys fall into this category. Beyond the simple reactive agent is the agent that can *reason*. Reasoning agents react by making inferences. These include patient diagnosis agents and 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 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 than any product that a human can make in a reasonable amount of time. An agent that cannot respond to its environment or to other agents is another kind of agent whose usefulness is questionable for developing agent-based systems.

(d) 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 coordination-based 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.

(e) Intelligent agents: After decades, the term *intelligent* has still not been defined (or understood) in artificial intelligence 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 agent-based SW have more "power" than conventional software. For example, in comparison to relational tables or objects, agents can be thought of somewhat "smarter." Or, it could just be a marketing hype. However, it would be fair to say that the notion of intelligence for agents could very well be different from human intelligence. 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, the state of an intelligent agent 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.

(f) Software agents: They are more specific kind of agents. 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 human, 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 are 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.

1.4.1.2 Mobile Agent Abstraction

Mobility is added to the agent abstraction. OMG (2001) defines a typical mobile agent. While a stationary agent exists as a single process on one host computer, a mobile agent can pick up and move its context to a new host where it 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. 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.

1.1.4.3 Application Domain Agent Abstraction

Agent can be constructed at this level when it is defined for a particular application domain. This is the level, in which all actual agents should be constructed. Most of the SW agents can be in this category. Each agent created in this level is task/role oriented. Properties/behavior related to the task/role should be easily extracted. Multiple Agent System (MAs), Distributed Artificial Intelligence (DAI) and Mobile Agent System (MAS) are all typical examples constructed at this level. Most agent frameworks, agent models and agent patterns deal with agent system at this level too. It can be noticed that decomposition, based on the task/role of the agents, can be the most powerful means to build agent abstraction for a given agent system.

I would like to stress that a weak agent is autonomous, social, reactive, and pro-active; it is the combination of these behaviors that distinguishes an agent from objects, actors, and robots. Here social behavior means that agents utilize structured messaging and protocol for coordination and negotiation. A strong agent, in addition to the capabilities defined for weak agents, is knowledge based, and it applies reasoning to determine what to do next. To simplify the agent abstraction, the base agent can be considered as abstraction of a weak agent. Mobile Agent and Stationary Agent are inherited from the Base Agent. A Stationary Agent can be modeled as either a strong agent or a weak agent depending on the application. A mobile agent shows a more complex appearance, but it should have the mobility/migration property as its basic property. I may also assume that the base agent has all behavior of a weak agent as minimum property set, which should be inherited to its children agent based on OO concept.

This agent abstraction hierarchy aims at helping to extracting intrinsic behavior/properties of an agent group such that agent model/framework/pattern can be built on agent behavior/properties.

1.2 Agent Models

1.2.1 Logic Model

A logic agent model is based on Independent Choice Logic (ICL, Poole, 1997). The logic model allows what is arguably a natural specification of multi-agent decision problems. It models an agent as a

logic program that specifies how the outputs are controlled by certain inputs. This logic program can use the internal values and sense values but cannot use those values that the agent cannot access. Both natural and artificial agents can be modeled as simple logic circuit or in some traditional programming language.

1.2.1.1 Model Structure

The logic model distinguishes "controller" agent from "plant" agent. The controller is the part to be optimized. It receives digital signals ("observations") and sends output digital "controls" or " actuator commands". The "plant" or "body" is the physical embodiment of the agent that includes input devices such as sensors. The plant receives "percepts" from the environment and sends observations to the controller. The observations are usually correlated with the percepts received. Agent communicates with each other only through the environment, and multiple agents interact in the environment.

1.2.1.2 Model View

The logic model can be viewed from the perspective of agent specification. From the controller's perspective, the inputs will be observations and the outputs will be controls. By generalizing the ICL to the "dynamic ICL" through the use of agent specification modules, it is possible to reach a more concise representation of the decision problems.

Logic model may be well suitable for an idealized single agent in a well-defined environment with applications in areas such as "information seeking actions", "decision-theoretic planning", and a "game: tic-tac-toe".

1.2.2 Behavior Model

A behavior model is based on the theory of behavior (Kinny et al., 1996). The refinement of the behavior model largely depends on the behavior extracted from the agent designed. There are at least three types of agent behavior models (sub-models): Social Model, Behavior Model and Believable Model, extracting/stressing different aspects of agent behavior/properties.

1.2.2.1 Social Model

The social behavior of an agent focuses on communication, interaction and coordination with other agents in order to bring about a coherent solution. This model does not provide a good total solution for an agent due to its lack in extracting other important behavior/properties of agent. It is used only as supplement of other agent model for refinement.

1.2.2.2 Behavior Model

This model considers the behavior of a single agent. One good example is the *Multi-Plane State Machine Agent Model*, in which the behavior of the agent is represented as a multi-plane state machine. It has the following features:

- Behavior embedded into a data structure;
- Explicit concurrency;
- Dynamic agent modification, agent surgery;
- Integration of heterogeneous behavior models;
- Possibility of automated programming.

To obtain an agent abstraction, the agent is decomposed into a group of active objects linked together by a data structure. The four major components of an agent are: the model, the agenda, the state machine, and strategies. Boloni and Marinescu (2000) stressed that the multi-plane state machine structure in this behavior model provides an elegant way to express the multifaceted behavior of an agent, and every plane expresses a facet of the behavior of the agent. These facets include reactive behavior, active behavior, reasoning, planning, and interaction.

1.2.2.3 Believable Model

This model is built on believability, which was originally raised for building an *Intelligent Virtual Environments* (IVE), in which one of the major concerns is believability of the agent. There are two main investigation directions when considering the *Agent Believable Model*, one is to achieve believability and rational behavior of the agent. Damasio (1990) and the others take a more empirical and artistic approach in the agent modeling process with this model approach (Loyall, 1995).

One of the challenges of the believable model is that it does not consider agent interaction as a first order property, thus the agent built is not able to effectively interact with the other agent. This limits the model application domain. For example, this model is not applicable for a Multi-Agents Systems (MAS).

1.2.3 Component Model

This agent model was originally proposed to resolve the innate difficulty of constructing a MAS in a distributed environment, where an integrated collaborative agent-building environment is essential (Nwana, 1998, 1999).

A typical Agent Component Model, such as ZEUS, views an agent system as composed of three layers at the highest level of abstraction: a definition layer, organization layer and a coordination layer. At the definition layer, the agent is viewed as an autonomous reasoning entity, i.e. in terms of its competencies, rationality, resources, beliefs, preferences, etc. At the organization layer it is viewed as in terms of its relationships with other agents, e.g. what other agents it is aware of, and what abilities it knows they possess. At the coordination layer the agent is viewed as a social entity, i.e. in terms of its coordination and negotiation techniques. This agent model is supplemented with protocols that implement inter-agent communication and an interface to application programmers that enable the agent to be linked to the external programs that provide it with resources and/or implement its competencies.

Agent components are implemented based on various aspects (properties/behavior) of agent functionalities, such as communication, ontology, and social interaction. With these basic components (named as agent building blocks), a generic agent (base agent) can be built. Any other application-specific agents should be constructed by specializing the generic agent.

There are a number of agent creation toolkits using the Component Model, which focuses on different aspects of the agent. ZEUS and DIVA are two of them (Erol and Lang, 2000).

1.2.4 Layered Model

A layered agent model is introduced based on agent behavior/property. This model assumes some basic characteristics, which an intelligent agent must have:

- Internal state which reflects their knowledge, which is specified and refined during an agent lifetime;

- Sensory ability to detect the change of the external world;
- Believing/Reasoning capabilities, which determine their internal behavior. Agent behavior may change dynamically at run time, and specified in a declarative way;
- An external behavior consisting of communicative acts to other agents or control actions on SW or HW devices. The communication with each other goes through standard agent-independent communication languages;
- Internal intention to perform certain jobs assigned to this agent;
- An identity, which distinguishes one agent from the others in the same system. The identity is expressed in terms of a name, an address, and a description of the service (role) it provides. An agent's identity can be communicated to other agents and agents can freely appear and disappear in the system or change their location.
- Ability to collaborate with other agents outside.

These behaviors/properties are not sufficient for a complete description of the agent model, but they are representative for most of the agents in the real world. Layered agent abstraction/model extracts these agent behavior/properties, decompose and put them into different layers in a fashion that an agent performs its job with its knowledge. Figure 1.2 shows a sample from Kendall (2000).

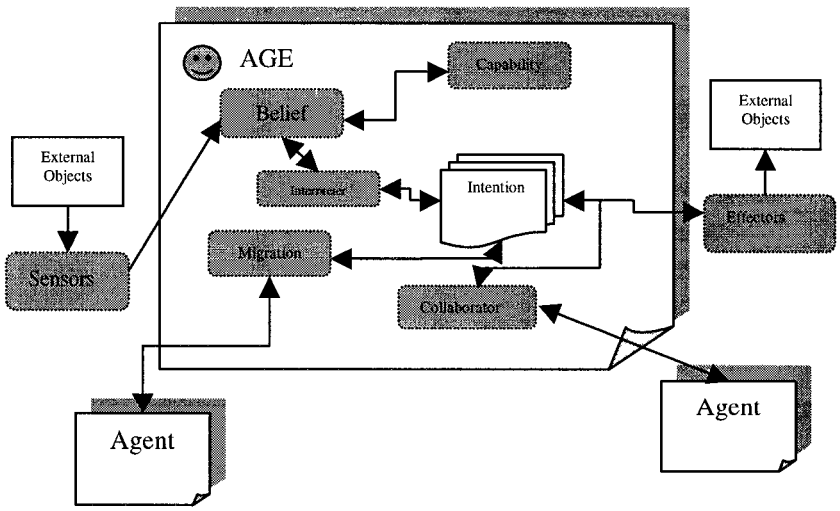

Figure 1.2. An agent layered model for an AI agent (From Kendall, 2000)

Here the agent model extracts sensory and effectors as external behavior/properties, belief, capability, interpreter, migration and collaboration as internal behavior/properties. This is a simplified agent abstraction of a strong agent. It is knowledge-based. As long as these behavior/properties of an agent in a real world have been extracted, Buschmann (1996) proposed that it should be decomposed into layers. In the layered agent model, Kendall and Malkoun (1996), Kendall (1997, 1998, and 2000) suggested that: i) higher layer behavior depends on lower layer capabilities; ii) layers only depend on their neighbors; and iii) there is two-way information flow between neighboring layers.

Figure 1.3 shows a simplified agent layered architecture structure for JAFIMA agent model. The seven layers can be derived from agent representation in Figure 1.2. Each layer is identified from the agents' real world counterpart. Once these layers have been established, they can be utilized to structure the framework.

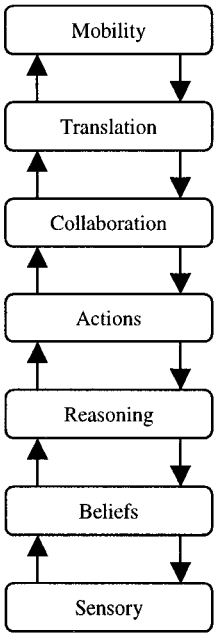

Figure 1.3. The layered agent architecture based on agent behavior from Figure1.2 (After Kendall, 1998)

Here is a two-way information flow from top/down to down/top:

Layer 1 (sensory), Layer 2 (beliefs), and Layer 3 (Reasoning): The sensory layer is responsible for proactively sensing the environment and updating the beliefs. The Sensor needs to cooperate with domain specific interface classes. The design of Sensor layer uses the Adapter pattern. The beliefs layer stores

Beliefs, and its design uses the Composite pattern. The reasoning layer determines what the agent should do next. It interprets beliefs and requests an appropriate reaction.

The Action Layer: It is responsible for carrying out the plan selected by the Reasoning layer. The Action layer incorporates Collaboration Intentions that utilize an interface to the Collaboration layer known as Collaborators, and Reaction Intentions that use an environment specifically known as the Effectors. There are also Scheduler and a Prioritizer when need.

The Collaboration layer: It is responsible for carrying out the exchange of services and negotiation with other agents, determining how to collaborate and address different coordination protocols. It implements send, receive and reject requests, and other replies to messages. There are two types of collaboration: centralized and decentralized. Centralized collaboration is the responsibility of a Mediator or Manager. The Mediator pattern allows an agent to freely collaborate with other agents without direct knowledge of their existence. Decentralized collaboration agents deal directly with one another, the agent maintains the state of each conversation to avoid endless loops. A clear difference is the role: In a decentralized agent system, the role of an individual agent changes (e.g. a role of agent in a system can be client in one case, but server in another case), but this is not the case for a centralized agent system.

The Collaboration Layer consists of sub-layers with the following patterns:

- Synchronized Singleton – it manages the collaboration threads;
- Decorator – it changes the behavior of the thread dynamically;
- Active Object – it schedules the requests forwarded by the action layer;
- Future – it is asynchronous method invocation;
- Strategy – it converts a message into the corresponding language of the destination agent.

Transportation layer: This layer is responsible for translating incoming and outgoing messages according to ontologies used.

Mobility layer: It supports virtual migration by providing location transparency between collaborating agents. It supports actual migration by providing a framework for cloning an agent in another environment.

The layered model gives a general framework as for how an agent should be decomposed into layers in accordance with the behavior/properties extracted from a possible real agent world. Details of each layer can be different depending on the behavior of the agent system, and can be further decomposed into sub-

frame. Figure 1.4 shows the architecture of a centralized Mobility layer. Figure 1.5 shows another example for a sub-layer architecture for the Action layer. This addresses the problem of how an agent commit to performing reactive and proactive behavior.

A layered agent model demonstrates that agents can be designed and implemented using objects following the properties/behavior extracted from real-world agent. We can see that the Agent Abstraction Hierarchy proposed in Figure 1.1 can be naturally constructed with a layered model as behaviors of an agent at each level is well established. As Kendall et al pointed out (2000) that layered structure is conceptual and the actual layers to be built are dependent on the agent to be modeled. The number of layers can grow or shrink as long as the agent behavior can be properly allocated to the layers, and there is well-defined internal data flow between layers. There are a few such examples: JAFIMA is more comprehensive and addresses more aspects of agency (Kendall, 2000). Concordia emphasizes mobility but does not address reasoning (Walsh et al., 1998). Zeus (Nwana, 1998) concentrates on reasoning and collaboration; it does not cover mobility.

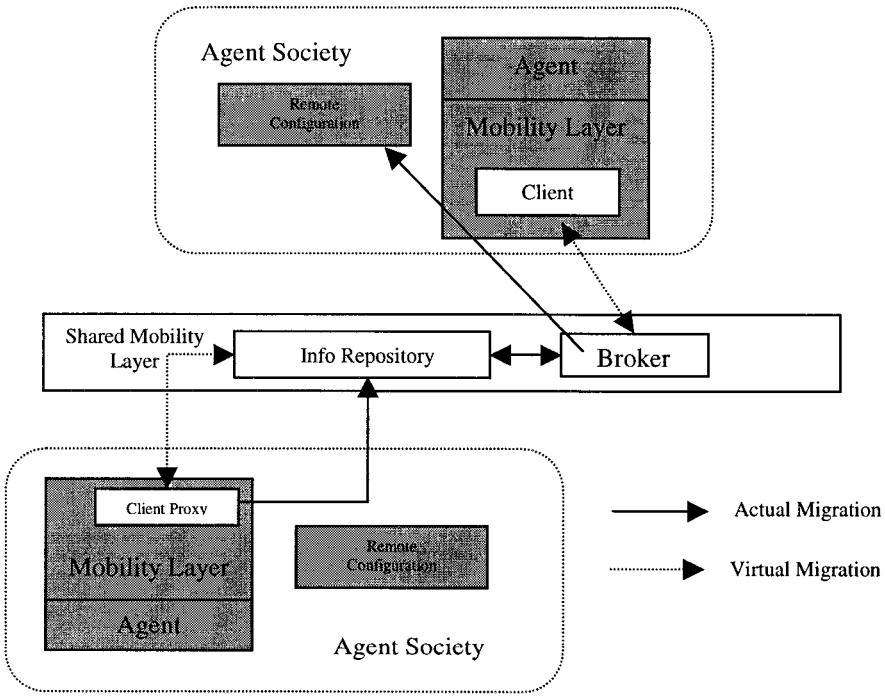

Figure 1.4. Architecture of mobility layer (From Kendall, 1998)

The design for creating individual layers and integrating them uses special patterns, which are usually called Agent Builder and Layer Linker. The Agent Builder addresses the question of separating the agent construction from its representation. The same construction process can create different representations. The Layer Linker addresses the question of integrating the various layers of an agent together. These are resolved using OO Design patterns (Gamma et al., 1994).

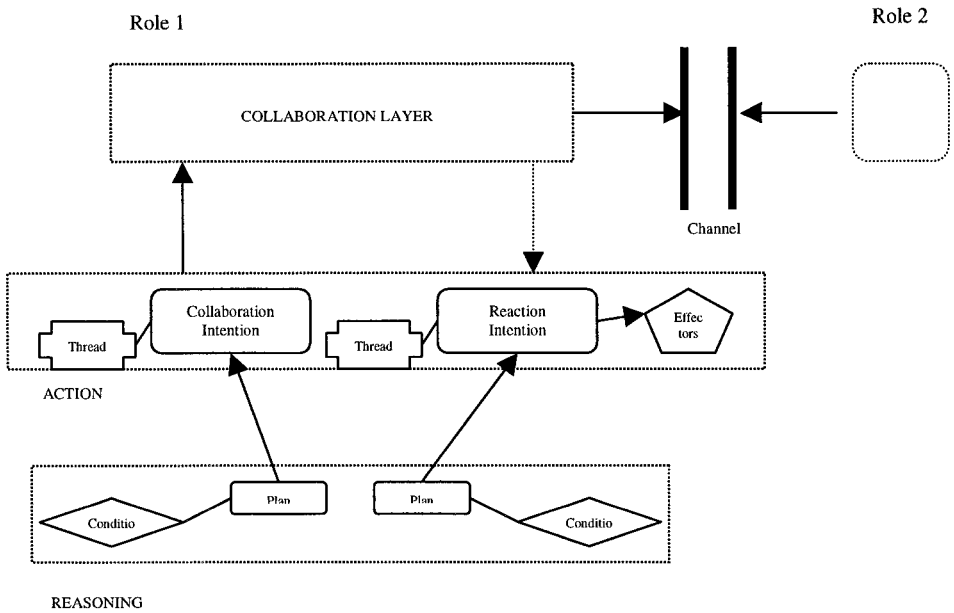

Figure 1.5. Intention in the action layer, plan and conditions in the reasoning layer (From Kendall, 1998).

1.2.5 Role-Based Model

Role model is used to specify, analyze and design MAS on the basis of the roles that an agent plays. Role-based agent model stresses interactions and collaboration among agents. Research on role-based agent system can be traced back to the 90s. The early pioneering works can be found in Barbuceanu et al.(1998), Iglesias et al.(1996, 1997), Kinny et al. (1996), Veloso et al.(1998). These works focus on collaboration and co-ordination. However there has not been a methodology for realizing these representations in *an automated or semi-automated system.*

1.2.5.1 Role Concepts / Role Theory

In Sociology the concept of roles and role theory goes back to the Fifties. Nowadays sociologists and computer scientists use this term as a bridge in common interdisciplinary work. A role becomes a kind of

"interface" to the environment of the agent. Between individual agents and actors, interaction takes place in the course of the execution of the according roles.

Role theory also allows an agent to fulfill multiple roles, each determined by the agent's position and the current personality structure. A position defines how an agent should fit in the social system (e.g. a hybrid MAS), and personality refers to a set of typical motivations and strategies of an agent, that is pursued independently of the position given in the social system. Thus the role concept is good for an agent model in a cooperative environment, where several agents have common goals and social action must be distinguished from joint action. Agents in this cooperative environment may ask for generic sets of expectations of the behavior and competencies from one agent to another. This is where role concept comes into play, where roles are defined by position, personality and interaction situation. Roles may vary with respect to 'personality, position and the tasks that are given by the interaction situation'. Thus the role-based agent model is highly abstract.

Role concept also sees that interaction history may influence the current interaction not only by changing the interaction situation, but also by adaptations. This implies that role-based agent may undergo a learning process and also indirectly influence other agents.

1.2.5.2 Agent Role Model

Agent role model can be established based on roles, which can be easily considered as the specification and evaluation of an agent system (Kendall, 1998). Since roles tend to be seen in observation terms, as tools for the specification and evaluation of agent system, the true importance of roles lies in providing a means to distribute "tasks" among agents. In a more complex situation, role corresponds to a generic set of expectations for behavior and competence.

When a role is captured in an agent, it has a number of properties. These properties form the basis of role-based agent model design and implementation:

- *Abstractivity*: roles can be organized in hierarchies;
- *Aggregation/Composition*: a role can be composed of other roles, with varying visibility;
- *Dependency*: a role cannot exist without the agent;
- *Dynamicity*: a role can be added or removed during the lifetime of an agent;

- *Identity*: the role and the agent have the same identity. The agent and its role are seen and can be manipulated as one entity;
- *Locality*: a role only has meaning in a role model;
- *Multiplicity*: several instances of a role may exist for a given agent at one time. An agent may play several roles at once;
- *Visibility*: access to the agent is restricted by a role.

These are properties of a role, which should be considered in defining the roles for an agent.

Role model may be of particular significance in a hybrid agent system that involves agents and human actors. In this case the modeling is highly asymmetrical between agents and human actors. In most cases, roles are determined by the communication constraint and the complexity of the delegated tasks. This suggests that role-based model is applicable for modeling social action in multi-agent system, and offers the opportunity to model organizational structures with heterogeneous goals. The roles are defined by organizational structures, which determine the position and personality of a social entity.

In a role model, the first thing is to characterize structural properties of entities to be modeled. A typical example is given to an organizational type application. Here one abstracts an organizational structure into five view substructures:

- Formal
- Communication
- Knowledge
- Authority
- Competition

Any real organization will have specific forms in each of these sub-structures. Depending on pre-defined structural properties of organization, a variety of special positions are offered, and personality is given. Thus the roles can be well defined. As an example, in universities there are professors, scientific collaborators, students and others. Each of these roles is described by a number of rules constraining what a role is allowed to do or not – authority-, what they have to know – *knowledge* -, with whom they have to *communicate*, stand in *competition* and what *formal* rights they have. The owner of a position normally gets the formal role.

Transforming organizational structure into sets of roles can do modeling of roles in a computation manner. The formal part of a role can serve as a basic set of the expectations of the behavior and competence of what agents have about other agents in this role. In agent-oriented (AO) modeling, agent roles are used for capturing goals, tasks, or functions assigned to an agent. Thus agent analysis should begin with goals and goal centric use case. Goals should be partitioned, which means that they should be assigned to individual roles. Any agent application will in fact encompass many role models. Relevant role models should be identified during agent analysis.

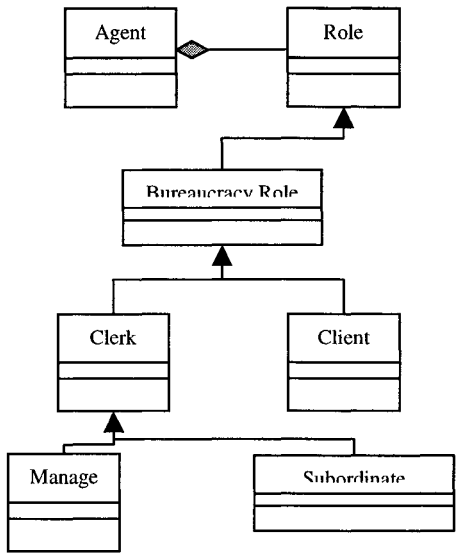

Figure 1.6 Role objects pattern design for bureaucracy agent (From Kendall, 2000)

This suggests that Role-Based Agent modeling starts from capturing roles in the design of agent system. Composing or assembling the responsibilities, interfaces, expertise, and protocols from individual roles form the design of an agent class. Since an agent is treated as an extension of objects, a role-based agent model is similar to OO analysis. The refined model can be structured into sub-models: a structural model, a dynamic model and a functional model. As structural model, an agent class diagram specifies the types of role objects and agents, their attributes, associations, and messages. Figure 1.6 shows the role object pattern design for a Bureaucracy Agent.

In general, role-based agent model can be designed with OO approach that involves structural model, dynamic model and functional model. During the design, the roles in a role-based agent model are assigned

to the agents; an agent plays or carries out the roles that are assigned. Although the model applies OO-design principle, there are differences:

1. The operations of an agent are autonomous, that is, they are never called by another object or agent but only executed under control of the agent itself. As a consequence, we should distinguish between messaging and operations of an agent;
2. Classes do not effectively capture collaboration. The functional and dynamic model for role-based agents should be extended (Figure 1.7).

A role model design pattern can be established based on the application. At least five role-based models are studied in different application domains: Supply Chain, negotiate for services, Contract Net, Iterated Contract Net and Auction.

Role model

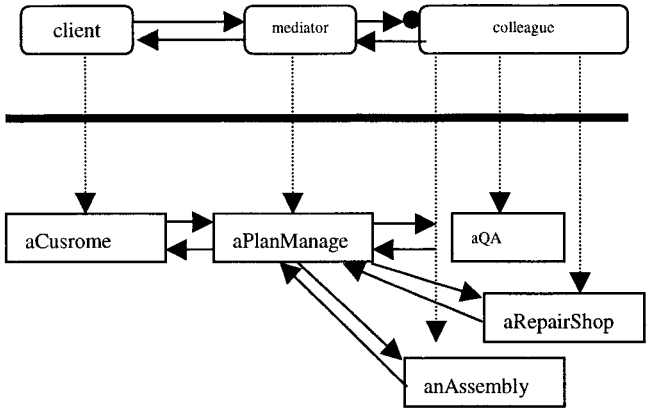

Figure 1.7. Role model: Collaboration diagram (From Kendall, 2000)

1.2.6 Brief Discussions

1. We can see that both *Logic Model* and *Believable Model* are rule-based. They allow for a form of parameterized rules by the use of logical variables. These rules construct large computation trees from smaller components. All the actions taken are primarily rule-based, and depend highly on the knowledge that the agent has gathered. They do not place too much effort on agent collaboration. Hence they may be inadequate in modeling complex coordination among agents.

2. Although the field of agents is witnessing the convergence of research from multiple disciplines, serious problem remains. Some see agents as a natural extension of the object-oriented programming paradigm. Contemporary work on the theory of behavior provides the foundations for a theoretical model of agents. Interface agents are considered the next generation evolutionary step in the development of visual interfaces. The field of robotics is using agents to model the behavior of their artifacts. This diversity of views as well as the wide range of applications of agents leads to numerous programming paradigm, languages, communication methods, and design concepts. All of these pose difficulty in unifying an appropriate agent model. The goal of an agent model should be to maximize the flexibility and provide a rich palette of options in the choice of programming paradigm, language, communication and operation mechanism.

3. The innate difficulty of constructing a sophisticated agent has motivated agent developers to move away from developing point solutions to point problems in favor of developing models for building more complex agents in a multiple agents system. This philosophy led to the development of the Component Model. How do components in an *Agent Component Model* map into the layer in a *Layered Agent Model*? This is a question that deserves further analysis.

4. Role-Based Agent models capture the structure of elements. After decomposing roles of practical systems, this model focuses on how they interact with each other in collaborations. Since the role of an agent in the real world exhibits patterns, role pattern is introduced into this model. This makes the agent modeling easier. A typical example is the structure of bureaucracy, in which the model can be well decomposed into five major roles: Director, Manager, Subordinate, Clerk and Client. Comparing to other agent models, role model is more flexible in terms and captures agent interaction well. Role model allows more dynamic behavior change of an agent.

5. Current proposed models all focus on only some aspects of agent. A universal agent model that captures all desirable abstraction of an agent has yet to emerge. This can be viewed from some basic properties of agents, which I would attribute as *Agent Property*, *Agent Interaction/Collaboration*, and *Agent Personality*. A good agent abstraction should contain all key elements of these.

Agent Property Abstraction (APA): Each agent has certain intrinsic properties, which must be described or "advertised" with the agent model by instantiating the APA. Other agents may use the

instantiated APA provided by an agent to determine what service that agent can provide, as well as the SW interoperability criteria required to access such service.

Agent Collaboration Abstraction (ACA): Each agent has certain properties that are relevant to collaboration. For example, agents will communicate with each other. ACA is an abstract framework that specifies the conditions under which one agent may invoke another one, together with the formal mechanism used in such an invocation;

Personalized Agent Abstraction (PAA): It is entirely possible that there are certain tasks that a user often performs. The PAA specifies an abstract structure, which, when instantiated, creates an agent that is personalized to the needs of the user.

Unfortunately, all of the established models fulfill only part of these three abstractions, and we do not have agent abstraction that can establish all the *APA*, *ACA* and *PAA*.

1.3. Agent Paradigm: Distinction from Process Paradigm

The agent paradigm can be seen as an extension of the notion of (active) objects using concepts such as autonomy, cooperation, and goal-oriented behavior. Process paradigm does not account for these particular aspects. In many domains, agent paradigm competes with process paradigm, but the differences are obvious. Table 1.1 lists some major features of a process paradigm and agent paradigm.

Table 1.1. Agent paradigm versus process paradigm

	Process Paradigm	Agent Paradigm
Basic Unit	Object	Role/agent
Parameter defining state of basic units	Unconstrained	Beliefs, commitments, capabilities, choice, …
Process of computation	Data as attribute passed by member functions	Message passing, and response methods
Types of message	Unconstrained	Inform, request, offer, promise, decline,
Constrains on methods	None	Honesty, consistency,..
Model Approach	Procedure driven, event driven, user driven	Task-driven, goal driven, behavior driven

I would suggest that the most fundamental differences are the basic unit, process of computation and the way of modeling object and agent.

1.3.1 Objects and Agents

The atomic programming unit is an object in OO-process paradigm, but it is an agent in agent oriented (AO) paradigm. The object in an OOP has a well-defined class abstraction. In AO paradigm, agent class has been proposed and used. In a role model, in which role is the atomic unit, the notion of a role focus on the position and responsibilities, while the object class stipulates the capabilities of the individual object it models.

Process paradigm focuses on object. An object has a formal, mathematically sound, yet practically realizable definition for any program based on OO concept. A class is a highly abstracted representation of the object, and object's entities can be well mapped into class attributes and methods in a program.

It has been suggested that an agent can be treated as an extension of objects (Kendall et al., 1997, Kendall, 1998). Agent encompasses all the features that objects have, with the additional characteristics of autonomous, proactive, social, reactive, and intelligent behavior. Therefore, additional facets are required. Table 2 compares the facets of an object and an agent in terms of their roles.

Table 1.2 Comparison of facets for object and agent in terms of role.

	Object	*Agent*
Role model	Context	Context
Responsibilities	Services, tasks	Services, tasks, goals, obligation, interdictions
Collaborators	Roles with which it interacts	Roles with which it interacts
External interfaces	Access to services	Access to services
Relationships to Other roles	Aggregation, generalization, refinement, role sequences	Aggregation, specialization, generalization, role sequences
Expertise		Ontology, inference, problem-solving knowledge
Co-ordination and negotiation		Protocol, conflict resolution, knowledge of why other roles are related, permissions
Others		Resources, learning/adaptability

1.3.2 AO-program and OO-program

In today's world, not all SW programs can be referred to as an agent, but they can be extended to an agent. Given any program P which is not an agent according to the provided definition, there must be a way to extend P in some minimal fashion to a new program P' such that P' is an agent according to the definition. This suggests that OO programs can be agentized. For a program P to be considered an agent, P must have the ability (1) to know what properties it has, (2) how and with whom it can collaborate, and (3) how it can personalize its services to satisfy a human or another agent. These three criteria lead to three abstract mathematical definitions, with corresponding data structure in an OO program. These three criteria distinguish an agent from an OO program, and they also indicate how an OO-program can be extended to an AO program.

1.3.3 Agent-Modeling and Process-Modeling

Agent-based modeling competes with process-based modeling that identifies system variables and evaluates or integrates sets of procedures relating these variables. Both approaches simulate the system by constructing a model and executing it on a computer. The differences are in the form of the model and how it is executed. In agent-based modeling, the model consists of a set of agents that encapsulate the behaviors of the various individuals that make up the system, and execution consists of emulating these behaviors. In process-based modeling, the model is a set of procedures/equations, and execution consists of evaluating them. Thus 'simulation' is the general term that applies to both models, which are distinguished as agent-based emulation, and process-based evaluation respectively.

If agent-based modeling and process-based modeling are viewed at a high level, they could differ in two ways: the fundamental relationships among the entities that they model, and the level at which they focus attentions. Process paradigm begins with a set of procedures that express relationships among observable. The evaluation of these procedures produces the evolution of the observable over time. These procedures may be algebraic equations, or they may capture variability over time, or over time and space. The modeler may recognize that these relationships result from the interlocking behaviors of the individuals, but those behaviors have no explicit representation in process paradigm. Agent-based modeling begins with behaviors through which individuals interact with each other. These behaviors may involve

multiple individuals directly or indirectly through a shared environment. The modeler pays close attention to the observable as the model runs, and may value a parsimonious account of the relations among these observables. The modeler begins by representing the behaviors of each individual, then turns them loose to interact. Direct relationships among the observable are an output of the process, not its input.

1.3.4 What Are the Differences between Agents and Objects

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

1. 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* (in terms of code size) than objects but it does mean that agents perhaps have richer built-in interpreters. On the other hand, does an inference engine inside an object make it an agent? In this sense, object interfaces can encapsulate "smart things", e.g., more or less smart agents, and human beings. Object interfaces can also encapsulate "dumb things", e.g., conventional software objects, or dogs. 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 agent communication protocols.
2. 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.

3. Agent Communication Language - Agents use a rich 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 contrast, use a fixed set of messages in communication.
4. Type-instance distinction – In some agent systems, agents correspond roughly to object instances or individuals but there is no strongly typed class notion. Rather, the agents are all just agents.
5. Inheritance - Object advocates might point out that objects do some things which 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.
6. 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.

An important aspect is that agents are higher-level abstraction of behavioral entities than objects. 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.

1.3.5 Brief Discussion

I have turned to consider agents as an extension of objects, but it is not simply as a computational entity in an OO process paradigm. The agent approach is best applied in dealing with tasks that are less structured or ill defined. In such tasks, complete mathematical or computational solution may be either unavailable or too expensive to use. Like the concept of Objects developed in 70's, agent has become a new abstraction in software systems. Implication of agent technology is still to be further explored.

1.4 Agent Modeling Techniques

The research community of AO SW engineering has not yet identified an accepted process/ tool/ technique to support the design/modeling of an agent system (Bergenti and Poggi, 2001). AO SW engineering requires a modeling technique, which works at agent level abstraction, and differs from that in an OO system design, where focus is given to an object level abstraction. The level of agent level abstraction considers agents as atomic entities that communicate to implement the functionalities of the system, and this communication is supported by an agent communication language, such as FIPA ACL (FIPA, 1999) or KQML (Finin, 1997), and by ontology used to associate a meaning with content message. An accepted diagrammatic notation does not yet support the agent level abstraction even if a number of proposals are available. The challenge is to find a complete model of an agent class that describes the feature set of an agent of that class. These features include:

- Support interaction protocols;
- Accepted content messages, taking into account the ontology;
- Semantics of each message.

1.4.1 UML

The SW engineering community tackles the problems listed above by investigating the possibility of extending UML to support the basic AO concepts such as agent, ontology, and interaction protocol. This has led to a number of UML extensions (FIPA, 1999; Odel and Bock, 1999, Odell, 2000).

The Unified Modeling Language (UML) provides a complete set of standard notations for OO SW. Two of them, which specify system dynamic behavior, are state machines and collaboration diagrams. State machine may be associated with an agent class to specify its reactions to external environments. Meanwhile, UML interaction diagrams can be utilized to specify how agents may interact by exchanging messages. UML has been utilized to model the dynamic behavior of an agent. The behavior of an agent is specified by the UML sequence diagram that an agent goes through during its lifetime in response to the environment, together with its response to the environment. The UML interaction diagram, such as collaboration diagram, shows an interaction, consisting of a set of agents and their relationships, including

the messages that may be dispatched among them. The collaboration diagram, in particular, stresses the structural organizations of agents that send and receive messages.

Lind (2001) suggests that standard UML can be applied to describe agent interaction protocols by capturing one of the core concepts of multi-agents system – interaction. Interaction is the foundation for cooperative or competitive behavior among autonomous agents and thus encapsulates the most fundamental design within the development of multi agent systems. Since UML strongly focus on object-oriented software design, it is not right away suitable for agent-based software. In order to make it fit some special requirement of agent-oriented software, Lind (2001) has proposed two possible ways:

- To extend the UML by providing new structural elements and diagrams that enhances the expressive power of the base language. The developer of AUML model favors this.
- Another approach is to use the UML for describing agent-specific aspects of a software system, with the major goal of remaining within the boundaries of the original language and to use only those extension mechanisms that were explicitly admitted by the language designer. This may be considered as a Standard UML approach for modeling the interaction of agent systems.

Bergenti and Poggi (2001) argue that capturing interaction of the agent system is necessary, but not sufficient to model a complete AO SW life cycle. Instead of using UML interaction and state chart diagrams, they propose constructing *ontology diagram*, *architecture diagram*, *role diagram* and *protocol diagram* as tools for modeling different aspects of an AO SW.

1.4.1.1 Ontology Diagram

The problem of describing the rest of the world to an agent is traditionally solved by using an ontology that outlines a model of the world in terms of entities and relations between such entities. An ontology diagram describes the entities in terms of classes. Such entity classes are used to construct an ontology diagram. These are mapped into UML exploiting public relations between entities classes.

The agent abstraction does not deal with implementation details because these are normally tackled at the object level. This implies that entity classes are not allowed to contain private methods and attributes. Similarly, public methods associated with a class of object, which are dedicated to specify the message objects, is not allowed at the agent abstraction because the actors are agents and not the entity belonging to

the ontology. Therefore, entity classes defined in an ontology diagram are allowed to be characterized only in terms of public attributes. It is also noticed that the entity classes are defined with extension of stereotype, which is introduced to allow an architect to keep the entities belonging to the ontology apart from the objects used at the object level. Figure 1.8 shows an ontology diagram describing a subset of the ontology defined in the FIPA AVEB specification. It defines a class of entities called FIPAAVDescription characterized by a string called title (the only public attribute). It also allows the modeling of (public) relations between the entities in the ontology. The Ontology diagram extends the UML class diagram by applying the stereotype entity. Moreover, this stereotype allows an agent-enabled CASE tool employing a different diagrammatic conversion for object classes and entity classes, just like what is normally done for actors and objects.

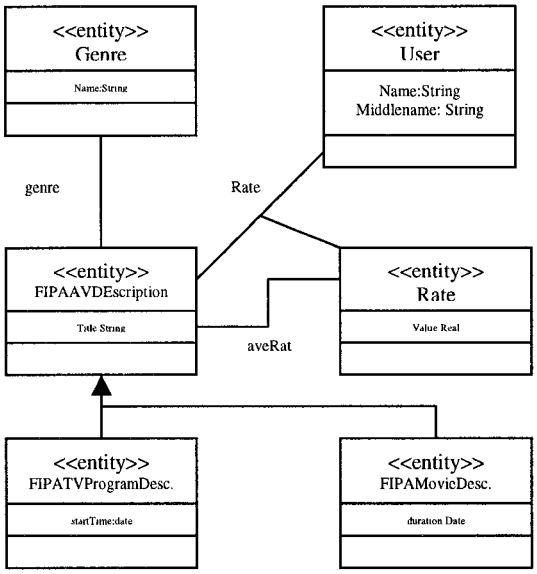

Figure 1.8 Part of the ontology diagram for the FIPA AVEB ontology

1.4.1.2 Architecture Diagram

Architecture is composed of a set of agent classes and each class represents a class of agents with a precise set of responsibilities. Hence an architecture diagram is actually defined as a UML class diagram, and contains agent classes and relations between such classes (Figure 1.8). An agent class can contain only public methods corresponding to the actions that an agent in that class can be asked to perform. These methods must be declared with no return value. Moreover, the parameters passed to these methods must

belong to entity classes defined in an ontology diagram. Architecture diagram also contains relations between agents to allow the multi-agent architecture modeling the relations between agents as shown in Figure 1.8 by the links among agent classes.

The modeling of agent architecture is not that easy. In fact, the architecture diagram specifies the MAS architecture by means of a textual description. This description contains a list of agent classes and a set of relations among them. Moreover, each class is associated with a set of responsibilities and a set of messages that an agent belonging to that class is requested to understand. This protocol has not been given, and the architecture diagram does not support the interaction protocol.

1.4.1.3 Role Diagram and Protocol Diagram

Traditional UML models the dynamic behavior of a system by using interaction diagram and state chart diagram. They are not directly suitable to model the interaction protocol in agent. Interaction protocols are valuable aspects of an agent system. We need to model the interaction between agents without taking into account the formal semantics of the chosen ACL. Moreover, it should represent off-the shelf solutions to a large number of problems. Object protocols, the OO counterpart of interaction protocols, are not considered as a fundamental element in the design of an OO system at the object level. This is the main reason why UML does not provide a diagram to model object protocols. To model interaction protocols at the agent level of abstraction, UML can be extended. There are a number of ways to resolve this. Protocol diagram and Role diagram (Bergenti and Poggi, 2001) are two of them.

Protocol diagram and role diagram are both UML-based notation to model interaction protocol at the agent level of abstraction. They are based on collaboration diagrams (or sequence diagram) and class diagrams respectively. A protocol diagram is a collaboration diagram that comprises only classes declared in a role diagram. These classes are labeled with *stereotype* role to keep them from other kinds of classes, such as agent or entity classes. A role class models a role-played by an agent in an interaction protocol.

1.4.1.4 Focus of Applying UML in Agent Level of Abstraction

This approach focuses on agent interaction, which defines that the agents must know which messages they can expect in a particular situation and what they are supposed to do when a certain message arrives. This part of the interaction process is controlled by interaction protocols.

1.4.1.5 Goal of Applying UML in Agent Level of Abstraction

With the UML approach, the goal is to identify the major elements of interaction protocols. It has to decompose the participants (agents) into different groups, and each group has a set of associated incoming and outgoing messages and behaviors that are associated with a group of agents as a role that can be played by an agent. There are two things to be considered by the interaction protocol:

- Participating roles
- Temporal ordering, or the flow of the control.

A protocol established within this model is a collection of several distinct automata where each automaton can have an arbitrary number of interaction points with other automata. These interaction points are called channels and they control the message exchange between different automata.

1.4.1.6 Tools

Protoz protocol specification (Philipps and Lind, 1999). The main tool is a compiler that generates Oz code from a given protocol specification. The protocol is defined by a collection of roles where each of these roles is specified as an extended finite state machine;

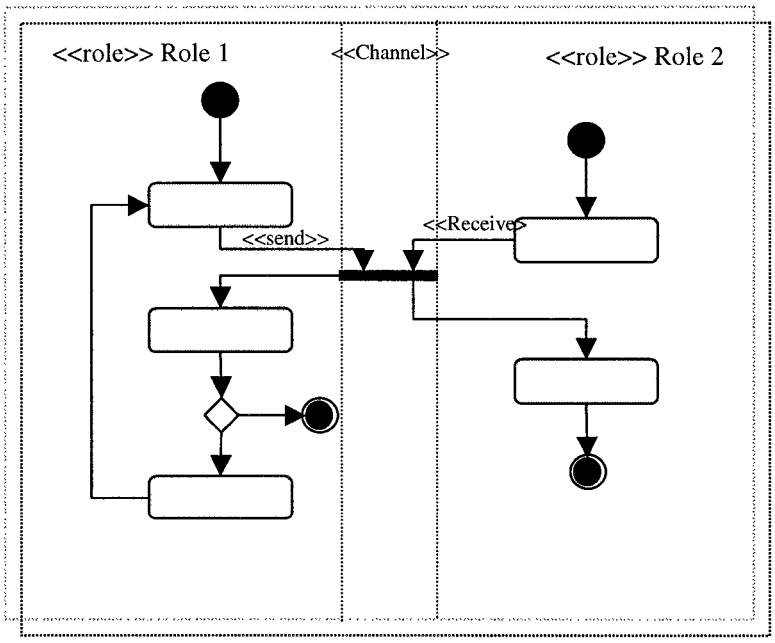

Figure 1.10 Activity diagram for agent interaction protocol

Lind's AO-based notations/domain-specific diagrams. They can be used to model the basic elements characterizing the agent level of abstraction: the architecture of the multi-agent system, the interaction protocols supported by agents and the ontology followed by agents.

Tailoring UML: define a couple of stereotypes that are used to model agent interaction protocols.

- Role → stereotype <<role>> that occurs within the application
- Channels → <<channel>> that manages the message exchange between two roles
- <<synchronization point>> that denotes the sending and the reception of messages, respectively (<<send>> and <<receive>>).
- <<timeout>> whenever timeout is reached and no message has been delivered, the control flow of the respective receiver resumes at the state pointed to by the timeout transition.
- The use of an activity diagram for specifying agent interaction protocols is shown in Figure 1.10. Activity diagram represents agent interaction protocols. Here are the key points:
- The swim lanes indicate the control flow spaces that are associated with each role with the agent interaction protocol;
- The control flow of each role is modeled using the structural elements provided by UML activity diagram;
- The self-contained control flow spaces are linked via a communication channel that holds one synchronization point that links the activity diagrams of the interacting roles.

1.4.2 AUML

AUML stands for Agent UML, which is an extension of standard UML. AUML (Odell, 2000) defines a diagrammatic notation to model interaction protocols. It is not compatible to UML sequence diagram. It has been developed for specifying the interaction/coordination protocols of multi agent systems by introducing a completely new diagram type called protocol diagrams. These diagrams combine elements of UML interaction diagrams and state diagrams to model the roles that can be played by an agent in the course of interacting with other agents. The new type of diagrams allows for specification of multiple threads within an interaction protocol and supports protocol nesting and protocol templates based on generic protocol descriptions.

1.4.2.1 Focus

The approach stresses two aspects: 1) Relating agent system to the nearest antecedent technology (OO SW development); 2) Using artifacts to support the development environment throughout the full lifecycle with AUML. The AUML basically provides a set of UML idioms and extensions in a three layer AUML representation for agent interaction protocols: template and packages to represent the protocol as a whole; sequence and collaboration diagrams to capture inter-agent dynamics; and activity diagrams and state charts to capture both intra-agent and inter-agent dynamics.

1.4.2.2 Concept, Scheme and Goal

Representing Agent Interaction Protocol in UML is the concept of this approach.

Scheme for AUML:

- Present agent as an extension of active objects.
- Exhibit both dynamic autonomy (the ability to initiate action without external invocation) and deterministic autonomy (the ability to refuse or modify an external request).
- UML bases: 1) support static models→ class and package diagram describing the static semantics of data and message; 2) dynamic models → capturing interaction of objects; 3) use cases → the specification of actions that a system or class can perform by interacting with outside actors.
- Suggest agent-based extensions to the UML representations: package, interaction diagrams, state charts, and activity diagrams.

The goal of this approach is to express agent interaction protocol (AIP) with UML, and present a specification technique for AIPs with both formal and intuitive semantics and a user-friendly graphical notation. Here the AIP describes a communication pattern as an allowed sequence of messages between agents and the constraints on the content of those messages.

A layered approach to protocol is the basic concept in the model, in which each analogous problem domain can be considered in one layer, in which a further interaction protocol can be defined. Figure 1.11 shows an interaction process in leveling. Here it first shows how a message is exchanged between two agent/roles; It also shows two more concepts: 1) the protocol is treated as an entity in its own right (the

protocol is package), a conceptual aggregation of interaction sequences; 2) package protocol can be also treated as a pattern that can be customized for analogous problem domains.

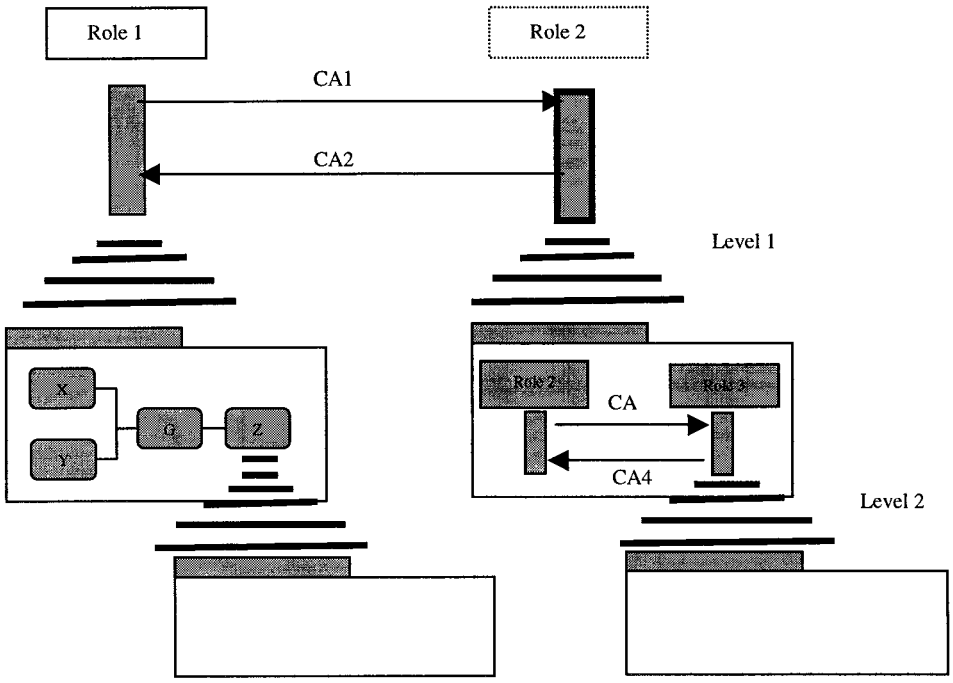

Figure 1.11 Interaction protocol using a combination of diagrams

Figure 1.11 illustrates the leveling concept. Level 1 represents the overall protocol. Level 2 represents interaction among agents. Level 3 represents internal agent processing

Leveling can continue "down" until the problem has been specified adequately to develop or generate code. The interaction protocol at the top has a level of details below, which in turn has another level of details. Each level can express intra-agent or inter-agent activity.

1.4.2.3 Tools: UML Dynamic Diagrams

This AUML applies UML dynamic diagrams as tools. It has several features: 1) It uses UML sequence diagram as a protocol for the contract net. 2) The protocol as a whole is treated as an entity in its own right, and it is a package used for expressing nested protocols (Figure 1.12, showing interaction in a supply chain). The sequence diagram itself describes the inter-agent transactions needed to implement the protocol. 3) Intra-agent communication can be supported by UML's activity diagrams and state charts.

Level 1: It represents the overall protocol. AUML proposes patterns in a practical context. Agent interaction protocols provide us with reusable solutions that can be applied to various kinds of message sequencing between agents. AUML uses package and templates from UML for protocol solutions in this level.

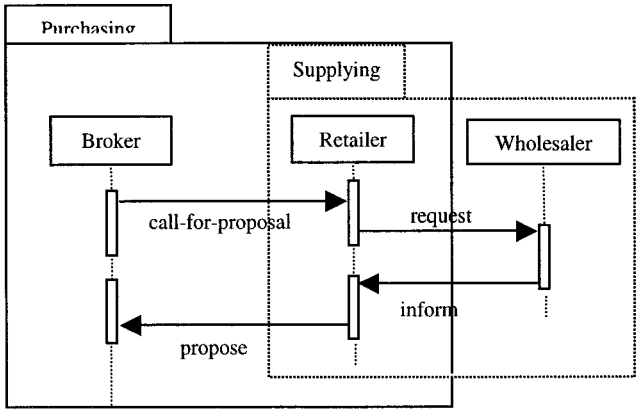

Figure 1.12 Using package to express nested protocols

Level 2: It represents interactions among agents. AUML applies UML dynamic models for expressing interactions among agents. It uses interaction diagrams, sequence diagram and collaboration diagrams, to capture the structural patterns of interactions among object. It also uses UML activity diagram for process flow at this level.

- Chronological sequence of communications: it can be simulated with a sequence diagram;
- Association among agents: it can be simulated with a collaboration diagram;
- Flow processing in the agent community: it can be simulated with an activity diagrams and state charts.

Level 2 is suitable for a single agent concurrently processing the multiple cases

Level 3: It is the lowest level, and represents internal agent processing. A UML sequence diagram expresses a sequence of messages exchanged by a set of objects, while an AUML diagram models all sequences of messages that a set of agents may exchange in the scope of an interaction protocol. These diagrams are not generally adopted because off-the-shelf CASE tools do not yet support them.

1.4.3 Aspect-Oriented Programming (AsOP)

AsOP has been used as a tool in modeling MAS recently (Kendall, 1999). The base reference for this is the Role Object pattern. As a design tool, role object pattern is used mainly at the implementation level, and

has introduced some problems, namely object schizophrenia, significant interface maintenance, and no support for role composition. AsOP is aimed to alleviate some of these concerns.

AsOp is specially proposed for agent Role Model. Fowler (1997) evaluates various approaches, and concluded that the role object pattern is the most common one. AsOP provides an individual class for every role. The roles are organized in a hierarchy, with subclasses for more specialized role behavior. A core object contains the roles that it plays as a set of role instances. Dynamic role assignment is supported because the instances that represent the current roles can be changed at runtime.

AsOP has not been widely applied in agent design. Kendall (1999) tried to extend the AsOP concept and establish a set of model notation, which can be set as a standard model tool. They have proposed five different options, but concluded finally that no one is complete. They further proposed a hybrid approach to compensate the model. In this modified approach, role specific (extrinsic) behaviors are localized in the aspect source code. There are many issues remaining open for the AsOP approach as agent modeling tool. It is hard to understand, lacks well-defined notation, and does not support adequately some basic features of the role model. It is not easy to apply to other agent models.

1.4.4 Petri Net (PN)

An agent system could be a heterogeneous/complex entity made up from agents with different characteristics. Typically, such an agent system composes of parts, which are mainly behavior control dominated, and parts, which model the flow of messages/data/materials, as well as parts, which combines these aspects. These parts of the agent system could be highly concurrent, and the interaction/collaboration among these agents is very complex and very different in nature. UML alone is not sufficient to model this heterogeneous system. For example, how agents accomplish a goal task is usually specified by the MA plans built from basic actions of agents. A critical problem with such an approach is how the designer can make sure the plans are reliable. To tackle this problem, a high-level formalism of PN is introduced, which adds additional levels of abstraction to compensate the lack from UML in modeling agent systems.

PN (Murata, 1989) is a formal and graphical appealing language, which is appropriate for modeling systems with concurrency. The PN could be one of the basic features of an agent system. PN defines a minimum set of expressions needed to describe the functional aspects and dynamic behavior aspects of the agent system.

Anil and Shatz (2001) has developed a model called Object Petri Net Models (OPMs) from UML state chart diagrams and connecting them using UML collaboration diagrams. In this way, the single agent system level Petri net could be analyzed by formal OPN analysis techniques. OPN provides a formal semantic framework for the UML notation and can be considered as a complementary tool for UML when applied to the agent modeling. The approach of Agent-Oriented Colored PN (Moldt and Wienberg, 1997) redesigns Shoham's paradigm of agent-oriented programming by means of object-oriented colored PN. Xu and Shatz (2001) extend G-net to model inheritance of agent classes in MAS, which provides a clean interface between agents with asynchronous communication ability and supports formal reasoning. Friha (1991) specify a multi-agent system for resource allocation problems using concurrent object-oriented PN.

An alternative is the PrT Nets, which has been proposed by Xu (2002). The PrT net is a high-level formalism of the PN, and is used to answer a critical problem: how is an agent plan reliable and feasible. It considers the totality of agent capabilities specified by preconditions and post-conditions as a behavior model of what the agents as a group can do, and take the multi-agent plans as a description of the process for moving from some start state, through this model, to a goal state. This is well suited for modeling MAS by representing agent capabilities as transitions.

The common issues with the PN model are: 1) they suffer from high complexity in reaching ability analysis through occurrence graphs, and 2) except the PrT nets, they are not concerned with agent plans.

1.4.5 Patterns and Toolkits

Pattern is a useful solution for a reoccurring problem in a particular context. Kendall et al developed interesting and preliminary work in agent patterns (Kendall, 1997; Kendall and Malkoun, 1996). They discussed and proposed a set of well-known patterns (such as Active Object, Mediator, Proxy, Adapter, Negotiator, and so on) used to support basically the "strong agent" vision (Woodldridge and Jennings, 1998). Aridor and Lange (1998) presented some agent design patterns in 1998. They presented these design patterns from the agent application design point of view: traveling (Itinerary, Forwarding, and Ticket), task (Master-Slaver, and Plan) and interaction patterns (Meeting, Locker, Massager, Facilitator, and Organizer Group). Silva and Delgado (1999) proposed a generic agent pattern for mobile agent system, which is said to be useful to develop dynamic and distributed applications in an open and large-scale distributed

environments. It encapsulates a business specific class (a specialization of the Agent class) with some user identification and a specific security policy.

The basic idea of identifying the strong agent from weak agent is largely from the agent design point of view. Figure 1.1 shows the relationship between the different classes of agent design approaches involved with the support and development of dynamic and distributed agent applications. It shows three interrelated classes of framework, each one uses features provided by the previous one.

1.4.6 Discussions

The tools discussed above are more or less based on UML. Agents are autonomous and interactive, and their activities include goals and conditions that guide the execution of defined tasks. While objects need outside control to execute their methods, agents know the conditions and intended effects of their actions and hence take responsibilities for their needs. Agent acts both along and with other agents. Thus there is no direct adaptation of UML to agent modeling. Both FIPA and the OMG Agent Work Group (OMGAW) are exploring uses of and recommending extensions to UML (Bauer, 1999, 2001; Odell, 1999). Depke et al (2001) discuss graph transformation and roles on an agent-based UML. Odell (2000) works on a comprehensive scheme for AUML and focus on the AIP. In general, these approaches are successful in using UML as a specification technique for AIP with both formal and intuitive semantics and a user-friendly graphical notation. The semantics proposed allows a precise definition that is also usable in the SW engineering process. The graphical notation provides a common language for AIP communication, particularly for people not very familiar with the agent approach. I would say that several tools could be applied directly to agent-based system by adopting simple idioms and conventions. In other cases, we could suggest several straightforward UML extensions that support the additional functionalities that agents offer over the current UML version. UML-based approach is dominated as a tool in agent modeling. The key issue is that the tool must help to identify the potentially problematic or insufficient parts of UML.

However problems do remain:

- No formalism yet exists to sufficiently specify agent-based system development. UML/AUML/ or any extension of UML techniques lacks the well-defined semantics and standard notation. The UML off-the-shelf CASE tool does not fully support proposed UML/AUML extension in modeling the agent system. For example, AUML is to extend the UML by providing new

structural elements and diagram that enhance the expressive power of the UML base language. However, it has the major drawback that it violates the ideal of the UML as a general design language.

- Non-determinism: The original UML state machine for OO SW is based on a dynamic computation of a pre-determined, static calculation of possible state transitions in responding to input events. This is not the case for an agent-based SW, which is described as a non-deterministic system in most applications. Agent systems can be multiple-states (or composite state), and its state chart behavior is more than a pre-determined, static calculation of limited states. We may expect a few ambiguities in the semantics of UML state machines when applied to AO SW. This implies that the UML approach is not suitable directly to AO SW, as it is limited to a single state machine. It seems that some of the more advanced constructs of state machines, such as synch and choice states and deferred events need to be implemented. Another major criticism is about predefined agent plans, which makes the model less flexible, but may be required for deterministic MAS.
- Collaboration in OO SW and AO SW: Collaboration for OO SW in UML is well defined with a set of standard notations since the communication among object is parameterized, there can only be one instance of any given class, and most of them occurs within a process. This is not, to some extent, the case for AO SW, where message event cannot be completely parameterized, and agent could cooperate or interact in unforeseen ways.
- One alternative approach is to introduce agents as an extension of active objects, and promote the use of standard representations for methods and tools to support the analysis, specification, and design of agent software. The former aspect of agent UML leads one to focus on fairly fine-grained agents. More sophisticated capabilities should be added where needed, such as mobility, and knowledge. These are not well defined in recent UML, but have been proposed for the future framework of agent UML (Bauer, 2001).
- Dependencies of intra-role and inter-role introduced by the other roles during the design process.
- Software language support for the design of interaction protocols. A number of protocol specification languages have been proposed ranging from low-level communication protocol up to

high-level specification language for multi agents applications. However none of these (perhaps except for Estelle (5)) has gained widespread acceptance.

My conclusion is that we shall continue to identify agent-specific requirement for the UML, try to solve the upcoming problems within the standard and suggest extensions only in those cases where solutions within the standard are not possible. Recently, cooperation has been established between the FIPA and OMG. As a first result of this cooperation, they analyzed the requirement for such an endeavor and proposed the framework of AGENT UML (Bauer, 1999, 2000a, 2000b, 2001). The core part within AGENT UML is the mechanisms to model protocols for MA interaction and an extension of class diagram for agents. This is achieved by introducing new diagrams into UML: protocol diagrams and agent class diagrams. Protocol diagrams extend UML state and sequence diagrams in various ways; other extensions include agent roles, multithreaded lifelines, extended message semantics, parameterized nested protocols, and protocol templates. An agent class diagram extends usual class diagrams with agent specification information. This may provide powerful tools for MAS modeling, but there is no practical application yet.

1.5 Limitations, Applicability and Missing Aspects of the Agent Model

1.5.1 Lack of Universal Semantic Base for Standard of Model Notations

Almost all agent development to date has been "home grown" and done from scratch, independently, by each development team. This has led to a few problems:

- Lack of an agreed definition, notation, and standard. An agent model developed by one team sees little chance to be ported by another team with some modifications. Agent models show neither significant standardization, nor customization;
- Duplication of efforts. Since the lack of universal notation in agent model, there has been little reuse of the agent architectures, designs, and components;
- Inability to satisfy industry strength requirement. To meet the industrial requirements, agent models must support the existing software development tolls and computer infrastructures, which require an integration of agent models with the current well-accepted modeling tools such as UML. The current adaptation has very limited success.

To overcome this problem, agent patterns, as an approach for agent modeling, has been proposed. A number of MASS (JAFIMA, JADE, and et al) has been built by using patterns as the basic building blocks. Although as many as 80 patterns have been built, they still are far from sufficient due to the fact that an agent always shows a complex behavior, and the behavior which each agent model to capture is diverse. However the pattern does provide a strong software engineering foundation for agent based systems.

Pattern does make modeling life easier, but it is still not sufficient as a model since it could be part of a modeling tool, but not all. A well-defined semantics base for a standard model notation is of more importance. This is a missing part, or at least, an incomplete part of the agent model.

1.5.2 Handling Complex Behavior of Real World Agent

Certain behaviors of a real world agent are difficulty to translate into a consistent formalism, which could be due to the complexity of an agent world. One suspects that the only realistic way to incorporate the complex agent in a model will be by implementing a black box, thus relaxing the decision algorithm in a agent model. It is hard to model general-purpose MAS. One good example is the planning algorithm to generate a plan that is optimal in certain sense. But the criteria vary from application to application. A plan with minimum steps does not necessarily have minimum execution time because the different actions usually have different durations of execution time.

1.5.3 Embed Dynamic Planning; Limitation from Supported Platforms

Many of the agent models are designed statically, meaning that the model captures the essence of an agent with predefined properties, roles, and interactions. This may be due to the agent platform: application can't be recompiled at runtime if they are implemented with standard programming language such as C++. Java might be an exception; this is why many of the MAS are built on Java.

Role model may be the only one, which can handle the dynamic aspect of the role with the help from AsOP. But the modeling process is too complicated, and the notation is not well established. We see also that the AspectJ compiler does not support some key features of the role model (Kendall, 1999). Thus the dynamic aspect of agent paradigm remains as a problem to be resolved since MAS should be designed in a way that they can run on different platforms. Van Hilst and Notkin (1996) implement and compose roles with templates in C++, which composes roles only at compile time, and it does not, obviously, support

dynamic changes at runtime. It is expected that the dynamic aspect of agent could substantially limit the potentials of the agent paradigm, in particular, when the MAS is designed for running in a heterogeneous environment or distributed system.

1.5.4 Model Validation and Verification

This Waterfall model of software engineering emphasizes the importance of software developing life cycle, in which system validation and verification is of extreme importance. This should include two things: fulfillment and performance. The first stresses if the model is sufficient to reach all the design goals, and the second is to tell how efficiently the goal could be reached. I have seen little effort in this direction except that in one of the PN agent model.

An agent model is mostly goal-driven. In MAS, how agents accomplish a goal is usually specified by the agent plans. A critical problem with such an approach is how can the model make sure the plans are reliable. A model must be able to propose several ways for modeling and analyzing agent plan. This is still missing.

1.5.5 Dependence on Environment of Infrastructures

There are numerous works in OS process migration over the decades. But the problems associated with agent paradigm are different. Interpreted languages (or Java at the most) are commonly used to support agent execution in heterogeneous architectures, operating systems, or even heterogeneous administrative domains, such as the Internet. This creates a problem: how is this supported in the agent model? Agent exists in higher-level abstraction, but they are too heavy in the packet world.

1.5.6 Security

Security is one of the major technical hurdles in MAS. Numerous challenges remain in agent model, which are related to security:

- Protecting the machine without artificially limiting agent access rights;
- Protecting an agent from malicious machine;
- Protecting groups of machines that are not under single administrative control.

There is no adequate solution for any of them.

1.5.7 Exploit Concurrency

One of the original advantages of using agent is in parallel computation, which involves exploiting agent system concurrency. Depending on the agent, concurrency has been taken into account in decomposition of the system along functional, organizational, physical or role consideration. As Wooldridge and Jennings pointed out (1998), no single agent model is universally best in terms of developing agent system concurrency. One of the most obvious features, which do not handle properly, is the amount of concurrency. Solution for this issue is comparatively small or even in extreme cases non-existent. For instance, in a layered agent model, task must be sequentially performed, one agent does some processing, produce some results, and then enters into idle state, this is internally a sequential model. This is an unsatisfactory design because there is only ever a single thread of control: concurrency, one of the most important potential advantages of MAS solution is not exploited.

1.5.8 Error and Exception Handling

This is another aspect, which seems to have been ignored. Compared with OOP paradigm, there are more challenging issues arising from the higher level of abstraction. In an OOP, errors and exceptions are well-defined in-proc, and can be picked up and propagated through the entire system easily. There are many mechanisms to handle error and exceptions; the best solution is always at the hands of programmers. Since the scope of handling error and exception is limited, the system can be always reset to a predefined proper state.

MAS is far more complicated, and we have not seen a clear framework for handling this issue. A possible explanation is the varied ways in which an agent works and communicates with others. This complicates definition of error and exception. The openness of an agent makes such definitions difficult. Intuitively, it also may be somewhat contradictory.

Chapter 2 Agent Architectures, Frameworks and Platforms

What Are In This Chapter: *The agent architecture is reviewed for single agent systems, multi-agent systems and mobile agent systems respectively since they focus on different aspects of agent-based systems, and apply different approaches. When agent/agent systems are well defined, the agent/agent system architecture design becomes the key point. This chapter focuses on agent architectures, agent frameworks and platforms. We discuss first the design architectures for single agent systems (Section 2.1), then multi-agent systems (Section 2.2), finally the mobile agent systems (Section 2.3). In each section, we start with proposed agent design architecture(s), give examples to show how the proposed architecture(s) is(are) applied to an agent system design. We will further look into the multiple agent system and mobile agent system frameworks and stress a few important research issues in agent architectures, frameworks and platforms in Section 2.4. We discuss particularly a few commercial agent platforms such as IBMAglet, MiLog ZEUS, Concordia ObjectSpace and AgentOS, which are representatives of the current multi-agent systems.*

2.1 Single Agent System Design Architecture

This is an area most interested by AI society. The purpose is to build human-like agents for use in distributed, interactive virtual environments (or worlds). These agents are expected to behave appropriately and timely in a complex, dynamic environment, and ideally in the same way as real humans behave in a real world. Such agent systems are, in essence, autonomous with human appearance and intelligence, i.e. having their own perception, behavioral and motor systems. Typically, system designers would concentrate on behavior models for a specific kind of agent in a given environment, and implement a limited set of behavior based on the requirements of the agent system.

They usually focus on the basic properties of the agent system, and apply three basic underlying architectures: BDI, Subsumption and Reactive. Majority of the proposed architectures emphasize behavioral control of an agent.

2.1.1 Approaches

2.1.1.1 Logical Control Approach (LCA)

A logical approach is usually adapted to high-level agent control architecture. It uses a logical formalism to model virtual worlds from the point of view of an animated agent. A cognitive model is used to design a reactive architecture that governs what an agent knows, how that knowledge is acquired, and how it can be used to plan actions. A high-level behavior specification language (BSL) has been developed to provide an intuitive way to specify the knowledge of an agent, the preconditions and the effects of its action (Chen, 2001). It can also be used to program control and agent interaction.

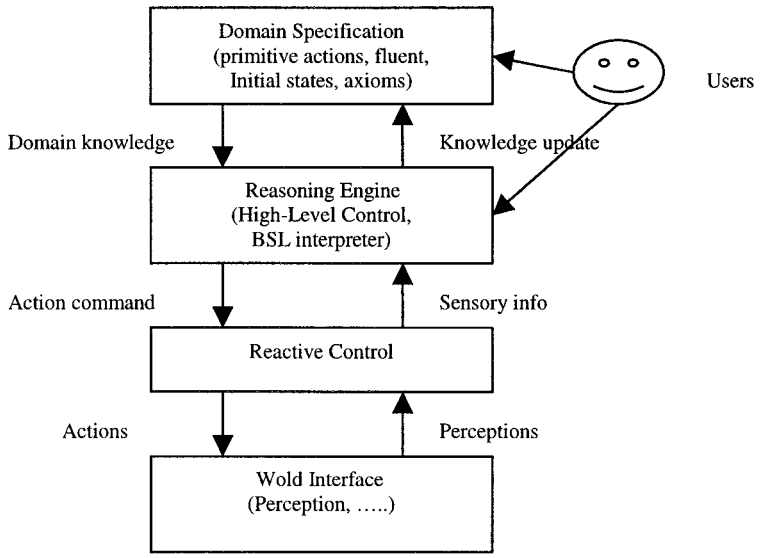

Figure 2.1 A simplified Logical Control Agent Architecture

Figure 2.1 illustrates simplified agent logical control architecture. It consists of four layered control modules: Domain Specification, Reasoning Engine, Reactive Control and World Interface. This layered agent control allows an agent to be controlled at different levels of abstractions. This logic control approach uses logic as a medium of representation and theorem as the means of computation. This approach may form a general architecture to design a single agent system. However, the proposed high-level logic control approach is restricted to task-level (or goal-driven). It does not know how to control agent's behavior through mental states such as motivation and emotion. That leads to the behavior control approach presented later.

2.1.1.2 Body/Head/Interface Component Approach (BHICA)

This generic model to design agents follows a Body/Head/Interface metaphor. The design architecture consists of three basic components called the Body, the Head and the Interface. The Body is in charge of task execution, which carries out the specific tasks of the agent in the application domain. A task can consist of pre-existent SW, public domain programs, and SW or HW components for specific purpose. The Head is devoted to coordinate the different functionalities and to manage the representation of the world. It consists of four sub-components: the controller, the reasoner, the knowledge base and the working memory. The Interface is in charge of communication with users, other agents and the environment. It consists of: KQML/ACL interface that deals with the interactions between agents; Sensors and Actuators that allow the agent to exchange data with the rest of the SW environment; a User Interface Manager that communicates with the user following his preferred modalities.

2.1.1.3 Behavior Control Approach (BCA)

This approach tries to unify the principles and characteristics associated with agent intelligence and uses behavior as its basic control components. Behaviors are selected dynamically and their actions are combined according to the intentions of the agent. Introspection of its reactions is the major ability given to an agent in this approach. The central part of the agent architecture is the control unit design of the agent system. It is commonly conjectured that AI agents need behavior control, which is based on mental concepts, e.g. believe, desire, intend, mood and emotion etc. This approach is involved in producing various kinds of internal and external behaviors of the system, including external physical actions and more subtle processes such as motive generation, attention switching, problem solving, information storage, skill acquisition, or even modification of the architecture.

Figure 2.2 shows a generalized behavior control architecture, which consists of six modules. The Behavior module is the central unit, which is a behavior-based system. To have a more general mechanism for behavior-based control, more modules have been added to the system: the External Situation module evaluates special external conditions in the environment, which can affect behavior selection. The module selects behaviors according to the need and goals of the agent. The Cognition module considers internal

parameters, and proposes recommendations. These three modules suggest the use of different behaviors to the Final Selection module, which establishes the activation of the behavior.

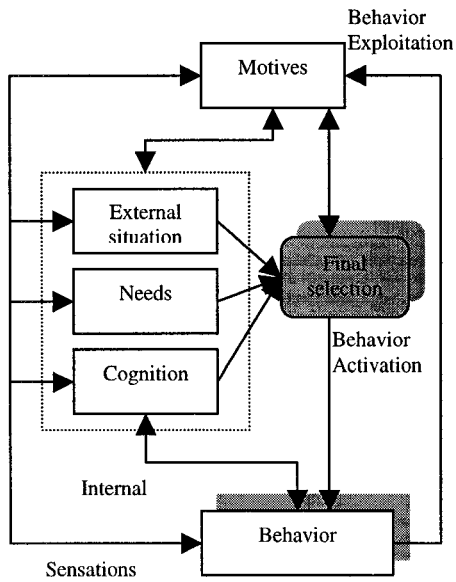

Figure 2.2 Generalized behavior control architecture

2.1.1.4 Brief Discussion

Autonomy, learning, adaptation, perception, reasoning, modeling, judging, planning, deliberation, goals, needs, behaviors, reactivity, emergence, holism, introspection, motivations, emotions: these are all concepts associated with intelligence. The unification of all of these concepts is not an essential condition to characterize a natural or an artificial system as intelligent, but it can help improve the level of intelligence manifested or required. Logic approach uses a logical formalism to model virtual worlds, with a goal of building a reactive system. It can be based on concepts such as learning, behavior, and autonomy of an agent depending on application. Behavior-control approach stresses agent behavior. The agent system set the behavior as the basic control component. The behavior is dynamically selected and their actions are combined according to the intentions of the agent to be designed. In this approach, the implementation of control can be also based on logic methodology.

In building intelligent agent system, a clear boundary between behavior-control approach and logic-control approach doesn't seem to exist. For example, a logic control approach may be applied to the behavior module in Figure 2.2 where a fuzzy behavior is dominant, and it requires rules and linguistic

variables (such as BSL) to establish the relation between sensations and actions. These are two fundamental approaches. Behavior Control stresses the intelligence of agent system, and considered as an Intelligence Approach. Logic Control stresses logical formalism, and considered as a Methodological Approach.

2.1.2 Examples

Cathexis: The Cathexis models emotion, moods, and temperaments as a network composed of special emotional system (Velasquesz and Maes, 1997). It follows a behavior-control approach to develop a distributed system for the generation of emotions and their influence in the behavior of autonomous agents. This model has been inspired in different fields including among others, Psychology, Etho-logy and Neurobiology. Figure 2.3 shows a high level view of the architecture, which can be derived from the generic architecture shown in Figure 2.2. Here the central components are: Emotional Generation, Emotional Behavior, Action and Stimuli. Decomposing it into different sub-components can customize the Emotional Behavior.

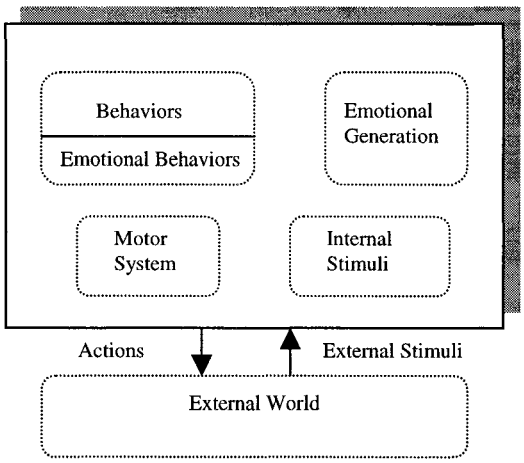

Figure 2.3. The Cathexis architecture (after Velasquesz and Maes, 1997)

Cathexis has been implemented in its totality as part of an OO framework that allows an agent developer to create emotional agents. This framework has been implemented in C++, and can be also implemented in Java. It is platform independent. Clearly, Cathexis allows one to model different kinds of emotional phenomena and gives enough flexibility to design various affective styles for an intelligent agent.

2.1.2.1 An Autonomous Robot Agent (ARA)

Maria and Oliveira (1998) proposed a design architecture for ARA as shown in Figure 2.4. It follows a *Body/Head/Interface Component* approach for the entire system design, but applies Logic Control Approach for some core part of the Control System. The system is made up of a Mobile Platform (MP), Sensors/Actuators Interface (SI), specific Control System (CS) and a User Interface (UI) via a component called Mediator (M), that communicates with users and knowledge database (KDB). The SI is the module that establishes the interface between the CS and the MP. The CS is the core component of the ARA and a Behavior-Control approach is applied to the architecture design of this component. The ARA architecture proposed focuses on the ability of the robot to perform an assigned mission. The model designer gives a Mediator module to interact with both the user and ARA. The Mediator gets a mission from a user. Then it installs the cognitive and behavior agents. Whenever ARA has not enough competences to deal with a particular situation it can ask for help from the KDB via the Mediator, or interact with the user to redefine the mission.

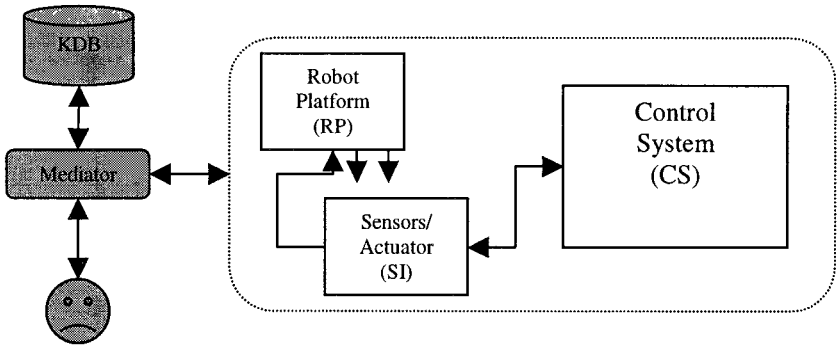

Figure 2.4 The ARA architecture

Neves and Oliveira (1997) believes that this proposed ARA architecture helps to build an agent with features of situation and goal oriented, robustness and extendibility.

2.1.2.2 Cooperative Intelligent Real-Time Control Architecture (CIRCA)

This agent control model focuses on hard real-time plan execution. Figure 2.5 shows the CIRCA architecture originally by Musliner et al (1993). The CIRCA domain knowledge base specifies how system states may change via a set of actions and temporal state transitions, and contains a set of sub goals which,

when achieved in order, enable the system to reach its final goal. During planning, the external world model is created incrementally based on initial state(s) and all available transitions. The CIRCA follows a general Logic Control Approach, but modifies it with the context of CIRCA. As we can see in Figure 2.5, the core component is the Planner. There are a few versions of the CIRCA with the Planner module allowing single task or multiple tasks to be scheduled. Thus it can run either on a uni-processor or multiple processor platforms.

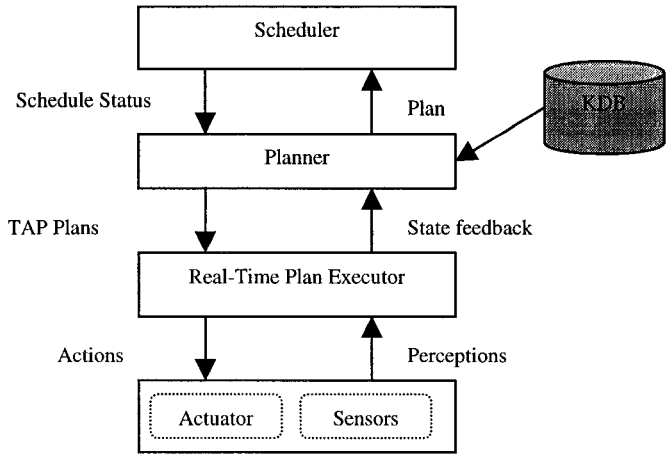

Figure 2.5 CIRCA architecture

2.1.3 Discussions

2.1.3.1 Logic Control Architecture (LCA)

LCA provides a basic framework for agent architecture design in application domains such as intelligent robot, intelligent real-time reaction system, and an agent system with fuzzy-behavior control. This approach focuses on maximizing the behavioral competence of autonomous agent, towards the goal of increasing the autonomous of robots, which will improve their competence at difficult environments. There are a few strengths:

1. The LCA allows a system designer to build a virtual agent system via high-level control constructs;
2. The high-level behavior applies rules to provide an external control mechanism;
3. The LCA can be easily turned into layered architecture, which incorporates cognitive modeling methodology and support extensibility, reusability and multiple levels of instructions;

The level-control is one of the most powerful features of this agent architecture. The power of the core layer (i.e., the Planner in the CIRCA architecture) can be tuned to achieve optimal performance in terms of the application requirements. But it could be also a drawback since it does not allow multiple level controls in the model. This limits the application of this architecture. At this point, the agent-layered architecture can be considered as an extension of the LCA architecture, in which the agent system is controlled at different levels.

4. The LCA provides a framework for generic agent architectures such as CIRCA. More complex systems can be built based on it. For example, Musliner et al. (1993) extend the CIRCA by considering multiple resources during plan scheduling and exhibit limited tolerance to resource failure (i.e. "internal faults") during plan execution.
5. Another major feature is the bi-direction internal communication between neighboring components. It governs the state transitions of an agent. If viewed by this feature, the LCA is with an underlying layered agent structure. But the LCA emphasizes control mechanism. The system can be decomposed into different layers; each layer takes corresponding responsibilities. LCA extends the layered concept of an agent generic architecture that addresses simple agent systems. LCA itself can be applied to a sophisticated agent system. It decomposes the core layer into sub-control layers. Thus LCA is flexible.
6. The LCA is originally proposed for single agent systems, built largely on the intelligence of an agent system. The LCA provides a generic framework, with a basic control cycle consisting of the traditional sense-think-act loop. It does stress a logic control methodology, but it is the application that specifies intelligent aspects emphasized in the model.
7. It does not consider collaboration and interaction of agent system.
8. Another potential drawback is that LCA requires high-level behavior specification language to provide an intuitive way to specify the agents' knowledge about their domain in terms of actions, preconditions and effects. This is particularly true when a set of rule-based criteria must be pre-established to guide the control mechanism in the core control layer. This could be very much application-domain dependent, and requires additional work to handle dynamic behavior of system.

9. LCA requires bi-directional communication between neighboring layers. This means that behavior of each layer depends on neighboring layer. This leads to strong coupling among system component.

The LCA can be related to layered agent model by correlating agent behavior and their reaction to the environment. It is goal-driven, and provides explicit and integrated support for single intelligent agent system in the area of robotics, aircraft simulation control, traffic control and so on.

2.1.3.2 Body-Head-Interface Control Architecture (BHICA)

BHICA provides a framework for an agent system, which automates and enhances the task assigned by the users. The design goal is on both intelligence and reaction of the system to external world. It has been applied widely in information searching, retrieval and processing, and E-Commerce, where an intelligent agent system is designed as a SW component that works in an integrated environment. Example instances include WebACE agent, CiteSeer agent (Bollacker, 1998), IVE agent, PBA and MASMA (Cesta, 1997).

1. BHICA is a generic agent architecture, which is flexible and adaptable enough to guarantee an incremental and modular development of an agent system in the application domain;
2. BHICA does not address the intelligence of the agent system, but encapsulates them into the head module, which is devoted to coordinate different functionalities. Since the head module has the ability of managing working memory and a built-in KDB, it does not only decide the action to be performed, but it also controls its resources through well-defined ways.
3. BHICA focuses on autonomy and reasoning specially designed to support for applications with standardized assistance to some applications such as E-Commerce and Web-based agent system.
4. The BHICA supports modularization, meaning that each module can be designed with architectural building blocks that support the specification of behaviors of the agent system in a way that allows periodic actions, interleaving of planning and execution, and the concurrent activation of multiple behaviors with asynchronous components.
5. The BHICA supports reusable agent behaviors. A behavior is a partially ordered set of basic actions. By "reusable", it means that the behaviors are specified in terms of domain-independent abstractions and can be reused in building an agent for a new domain or task. For example,

MASAM has an underlying BHICA architecture. There are two kinds of agents: personal agent and service agent, and each agent is a specialization of BHICA.

6. In general, the BHICA is facilitated by the abstraction of the local schema (KDB), a set of generic SW component for knowledge representations, and agent reasoning. It is weak in modeling interaction with other agent(s).

2.1.3.3 Behavior Control Agent Architecture (BCAA)

1. Generalization of the architecture: BCAA provides another generic agent architecture, which focus on the behavior control of agent system. Since agent intelligence is common to all agent worlds, we may say that agent behavior is reusable. Agent's reusable behavior is facilitated by its reusable architecture, i.e. the domain-independent abstraction of the local KDB schema, and a set of generic SW components for knowledge representation, agent control, and interaction with other agents. For example, the generic SW components, based on DECAF (Decker and Lesser, 1993; Oates, 1995), are common on all classes of information agents as well as to the task and interface agents built. The architectures presented in many applications are consistent with the agent behavior/component model. Decker (1997) proposed a behavior-based information agent architecture, in which the larger part of the architecture is shared by all of SW agent classes from BDI agent theory. This reusable component in BCAA makes the architecture design easier, which may follow design patterns by allowing the use of building blocks representing common agent intelligence properties, and the control mechanism. For example, Agent Behavior Editor is under construction in some research group (Decker, 1997), which will allow more rapid construction of new classes of agents through the reuse and combination of existing behaviors, specification of new behaviors, and new task reductions.
2. On the other hand, reusable behavior of agents does not block the model from customization. This means that the BCAA provides only a generic framework. Particular agent architecture can be designed based on new requirement from applications. For example, both SIMS and Decker's Information Agent are based on the DECAF architecture, a type of BCAA. But their differences are obvious. The Decker's Information Agent architecture design focuses on the ability to

interleave computational actions from many concurrent behaviors, to interleave planning and execution, to schedule periodic activities and activities that have dead lines, and to handle behaviors that are strung out in time and where the next step may be externally and asynchronously enabled. While SIMS agents do not have most of these capabilities. This is from the application requirement, and the BCAA gives the flexibilities for modification to reach a better behavior control of the agent.

3. BCAA stresses that behavior specification is the proper level for allowing people to construct new classes of SW agents in a structured, well-defined way. "Information Level" architecture is close to the requirements specified by SW engineers. This uses a design level of descriptions that always refer to the whole agent. The fashionable alternative view of architectures composed only of large and complex collections of very low-level components, without any modularity at intermediate levels. This is a distinct feature from LCA, which emphasizes the control at different level of agent.
4. We also notice that the BCAA focuses on functional aspects of an agent behavior, and combines control flow and data flow in each SW component. Thus it is best for some agent application such as Information Agents.
5. BCAA does not work well in agent collaboration and interaction, and may not be well suited for MAS.
6. BCAA fits the best to information processing control agent system: which is producing various kinds of internal and external behavior of the system, including external physical actions and more subtle processes such as motive generation, attention switching, problem solving, information storage, skill acquisition, or even modification of the architecture.

2.2 MAS Design Architectures

MAS architecture has added at least one more important focus: collaboration/interaction among agents. A framework of MAS architecture should represent and develop cooperation knowledge and protocols, and enable agents to work together and coherently achieve their common goals and those of the whole multi-

agent community. The following three important issues involved in the architecture design should be answered:

- Communication: how to enable agents to communicate? What communication protocols to use?
- Interaction: what language the agents should use to interact with each other and combine their efforts?
- Coherence, coordination and coupling: how to ensure that agents coordinate with each other to bring about a coherent solution to the problem they are trying to solve.

2.2.1 Three-Tier Architecture (TTA)

2.2.1.1 Approach

TTA designs MAS in a three-tier pattern: Interaction Tier, Information-Content Tier, and Glue-Tier (Erol and Lang, 2000). TTA considers interactions among agents as the most important aspect of MAS, which determines the coupling and coherence of the system through the communication protocol. TTA assumes application domain independence in such pattern. In the design, each interaction has one or more participating agents, and each participant agent plays a designed interaction role. TTA separates the syntax of an interaction (i.e. the protocol) from its domain-specific aspects. This decouples the internals of an agent from the roles it can play. In this way, an interaction can be used in multiple applications across many domains. The Information-Content tier is about local information and expertise of an agent. The intelligence, which an agent may posses, is implemented here and is passive. However, the TTA provides an API that allows easy access to its encapsulated capabilities. The third is the Glue-Tier, which is designed to integrate the interaction layer and the local-information content layer. It consists of a set of adapter objects, and each of them implements the interfaces required by its corresponding roles. The adaptive objects do that by making the appropriate scenario where role objects are readily available for the application at hand.

The TTA architecture separates the design of interactions from the design of internals of an agent, and breaks interactions into smaller, reusable components. Separation of information content and interactions divides a complex design task into manageable subtasks, promotes reusability and maintainability.

2.2.2 Component Based Agent Architecture (CBAC)

2.2.2.1 Approach

Component based agent architecture follows an OO approach, and it is introduced for Agent Oriented (AO) Application Frameworks. It has been mostly applied to multi-agent systems with the application domains in electronic market (E-Commerce), telecommunication and manufacture. CBAC emphasizes reusability, maintainability, and responsibility decomposition. Components are considered as building blocks for the entire agent system. As a reuse technique, AO architecture closely resembles both component OFF-The-Shelf (COTS) development and OO Application Framework. Thus the development consists in building new applications by assembling previous-existing black-box SW components (from COTS) and customizing the variable aspects of each single agent (from OO framework). Thus: COTS give the ability of reusing existing components as building blocks for new (agent) application systems, and the OO creates relationships between the agents and determines the information and control flow between them.

The components can be derived through the decomposition of application domain/tasks into sub-domain/sub-task, which is entirely based on the agent system requirements. It could be considered as a generic architecture, on which specific architectures can be constructed. For example, Belief-Desire-Intension (BDI) agent architecture can be considered as an alternative of the CBAC.

2.2.2.2 Example: RETINA Framework

RETINA framework provides a good example of applying the CBAC to agent application. The components can be classified as elemental, basic design and domain dependent (Brugali, 1993) that are the elemental components, basic design components and domain dependent component in the RETINA framework.

1). The elemental components

It is the building block for the implementation of individual agents. As an example, the RETINA framework provides the following elemental components:

- *The Communication and Coordination Component*, which accepts and interprets messages from other agents, and send replies;

- *The Planning Component*, which produces a plan that satisfies the agent's goal and tasks;
- *The Scheduling Component*, which schedules the plan actions;
- *The Execution Monitoring Components*, which initiates and monitors action execution;

2). The Basic Design Components: *The Basic Design Component* integrates the elemental components of the framework. Each has its distinct sets of internal structures (layered, blackboard, subsumption) and distribution control protocols they adopt (client/server, peer-to-peer, pipeline). As an example, the RETINA framework provides a Belief-Desire-Intension (BDI) agent architecture, which integrates the *elemental components* using the following data structures:

- *Task Schema Library*: contains both domain independent and domain dependent plan fragments indexed by goals. These plan fragments are retrieved and incrementally instantiated according to the current input parameters;
- *Belief Database*: contain facts, constrains and other knowledge reflecting the agent's current model of the environment and the state of execution;
- *Schedule*: depicts the sequence of actions that have been scheduled for execution.

2). The Domain Dependent Components

The domain dependent components consist of the customizable implementation of agents with domain-specific functionality (e.g. the Investment Agent is a reusable component for the financial domain). They can be classified in three broad categories: Information Agents, Task Agents and Interface Agents.

- Information Agent: accesses information sources, such as databases or a web-server;
- Task Agent: has specific reasoning capabilities or wrap legacy systems, which can be treated as procedural code invoked by the execution monitor;
- Interface Agent: manages the GUI, which transforms user commands into agent's objectives and display results.

3) Agent Generations and Customization

For a multi-agent system, a set of collaboration agents belonging to all these categories is required. The component agent architecture supports concrete agent generation by customizing the variables of the agent framework components. There are three kinds of agent customization:

- White-box customization: the framework provides a generic implementation of the base agent architecture, which the user specializes for encapsulating specific capabilities and resources (e.g. GUI);
- Black-box customization: the framework provides specific implementations of each elemental components (e.g. the Planner) that are to be plugged in into the base agent architecture:
- Gray-box customization: for each specific agent implemented when the framework is used (e.g. a Task Agent), the developer has to specify the agent Knowledgebase and Task Schema library.

In general, this approach is used for agent system integration. The entire behavior of the application is determined by the behaviors of agents and their interactions. Interoperability between agents is guaranteed by the common infrastructure and architecture provided by the framework used to implement them. In particular, for each agent we specify (1) the knowledge and capabilities (data, rules, constraints) that the agent needs in order to fulfill the assigned responsibilities; (2) the inter-agent communication protocol (that is, how the agent may be used by other agents), and (3) the external dependencies between an agent's functionalities and resources (task Schema).

This approach is aimed at raising the level of reuse of single agent components to the entire architecture (multi-agent system) and the development of application frameworks.

2.2.3 Agent-Group-Role Architecture (AGR)

2.2.3.1 Approach

AGR architecture is proposed specifically for agent role model to resolve the design issues: how the roles and agents are correlated to each other. It is applicable for most applications involved in MAS. Abrami (2002) have proposed an Agent-Group-Role model for supporting a co-decision procedure. This is based on the notion of role, group and agent (Figure 2.6). In their theory, groups describe collective structures through behavior types given by roles. Agents executing roles modulate these collective behavior types through their individual features. Agents carrying out several roles undertake a superposition of behaviors induced by collective dynamics. In this way, the agent-group-role model is applicable for modeling both individual and collective system. In an organizational approach, the model could be built on three levels:

- A global level, where behavior types are defined by the set of roles and their relationships in the organization level;
- An individual level: where agents execute and interpret the roles they are playing;
- A co-evolution level: where simultaneous dynamics of organizational levels interact through roles and agents.

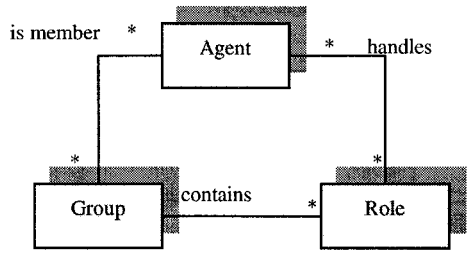

Figure 2.6 AGR pattern (after Abrami, 2002)

The AGR approach appears to be able to tackle both individual and collective aspects of a system. From this paper, the AGR has been given clear definitions based on organizational model:

- An agent is defined as an active communicating entity, no constraints other than those trigged by the ability to play a role;
- A group is defined as a set of agents;
- A role is defined as "a representation of an agent function, service or identification within the group". The role encapsulates the way an agent should act within a group. Roles are local to the group.

For agent-group based model, the global activities are given the first consideration to reach temporal consistency in the modeling process at an organizational level. The global activities are described by the position of the agents within these activities. The position can be attributed as part of the roles, since a role defines the position and function of the entity (agent) within an organization "from outside".

Thus AGR Model has the flexibility for an "ideal" multi-agent system, isolated from its environment and individual constraints. This "ideal" system is modulated by the way the agents play their roles.

2.2.3.2 Example

An AGR-based MadKit platform has been developed, and used in various research teams in projects covering a wide range of applications, from simulation of hybrid architectures for control of submarine robots to evaluation of social networks or study of multi-agent control in a production line. One typical sample is the Biology Assistant Agent (BAA). The BAA deploys a set of agents providing services and supports for collaborative work among biologists. Several groups have been specified and implemented: Viewer Group, which contains agents, specialized in genetic sequence rendering. Roles within this group correspond to specific rendering methods. An agent in this group typically receives messages containing genetic information, can display it, and send message to other agents, containing snippets of DNA, selections or actions made by the users, etc. The Service Group gathers agents specialized in processing of genetic sequence. The User Assistance Group is the agent involved in the interaction with the user, memorizing actions and habits, serving as "note-pads" for actions. Combined with the direct manipulation interface provided by the G-Box, the GAR architecture allows building chains of processing easily customized. Furthermore, it allows to cooperative work solutions seamlessly. As agents can be in distributed groups, user agents can communicate without modification. A processing chain can be distributed on many machines, with sequences viewed and annotated by multiple users with the help of their agents (Ferber and Gutknecht, 1998).

2.2.4 Discussions

2.2.4.1 The Focus of MAS Architecture

TTA focuses on agent interactions. It separates the agent interaction from the design of agent internals. This decouples interaction roles from local information of agents, and improves communication among MAS designers and implementers. Thus it reduces SW design, development, debugging, and maintenance time by automating the process of coding agent roles. It is relatively easy to apply different agent models into TTA since the agent internals are independent from system architecture.

CBAC follows an OO approach, and emphases the reusability, maintainability and decomposition of agent's responsibility. MAS is constructed by their functional components, which are considered as

building blocks for the entire agent system. There is no constrain on functional components, but they are mostly based on the agent behaviors.

The AGR simply provides a framework for role-based agent system. It places no constrains on the internal architecture of agents so that it is well suitable for generic MAS.

2.2.5.2 Strengths and Weakness of Agent Architectures

AGR and LAA have full flexibility for internal agent architecture and provide a strong structural model for an agent system. They allow seamless integration within the system. Both of them can reveal appropriateness between the model and the actual agent task, as they can be well mapped into some agent models such as agent role model and agent-layered model. The AGR and LAA could be built in a very heterogeneous environment since they are characterized and built from various individual agent architectures, and support various agent interactions through language heterogeneity (with KQML, ACL or other communication schema, e.g. ad-hoc). At the opposite end, BCA and CBAC do not manage these kinds of heterogeneity. MAS built on these two architectures can be well defined with application requirements.

2.2.5.3 How Each Architecture Is Open to Different Platform/Framework

The underlying framework for AGR follows an organizational metaphor, in which activity and interaction are defined for groups, roles and their relationships. This is an open architecture and can be built/run on different platforms. MADKIT and SEIT are two such examples (Ferber and Gutknecht, 1998).

LAA and CBAC are also open architectures. They provide general frameworks for MAS in various domains. For example, JAFIMA framework is built based on the LAA, and RETINA framework is on CBAC.

A good example is the PTM (the Personal Travel Market). The PTM shows how AGR architecture can be built on an existing agent platform. The PTM is a MAS modeled after the existing travel agency. It incorporates electronic equivalents of travel agents and service providers and an electronic assistant acting on behalf of a user. An overview of such a system is shown in Figure 2.7. As can be seen from this figure, there are two independently developed agent platforms and three different types of agents/roles involved in the architecture. The PTA (Personal Travel Assistant) resides in the Agent Service Layer (ASL platform)

whereas the Travel Broker Agent (TBA) and Travel Service Agents (TSAs) reside in the JADE agent platform. Access to online web resource occurs outside of the FIPA domain.

This proposed PTM deploys three types of agents: the PTA, TBA and a number of TSAs. The FIPA ACL realizes the communication among these three types of agent. Each agent also has its own communication language, which must be specified inside the agent architecture. The PTM works across two agent platforms: the ASL and JADE, and it utilizes all the functionalities provided by them. These two platforms support the development of agents in a number of languages including C++, CLIPs, Java, JESS and Sicstus Prolog. This support is provided in the form of language bindings and an agent shell for each language that can be tailored to its roles.

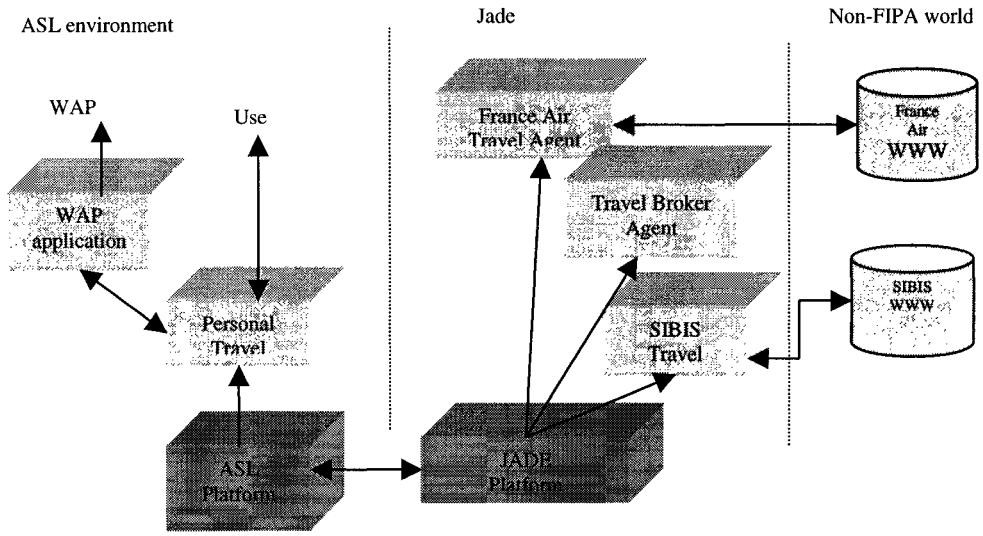

Figure 2.7. The PTM architecture (Modified from Ferber and Gutknecht, 1998)

2.2.5.4 How This Architecture Reach the Collaborations/Communication among Agents.

For the LAA architecture, there is a special layer called *Collaboration* layer, that handles all interaction and collaboration among agents inside the system. Inside the collaboration layer, agents determine their approach to cooperate and work with other agents. This layer can be further refined with conceptual, architectural, and design patterns for messaging (conversation), centralization (facilitator), decentralization (agent proxy), and social policies (protocol). However the LAA has some limitations on how communications are regulated among agents: it forces a structured message-passing pattern in

sequences rather than in isolated acts. This raises at least two problems: 1) successive messages between agents may be related, and 2) system may be blocked if an endless loop of messages occurs. The second problem may require a termination mechanism to manage it properly.

The AGR has less restriction in collaboration. The entire system is viewed as a framework for activity and interaction through the definition of groups, roles and their relationships. From the (organizational) structural point of view, it is regarded as a structural relationship between collections of agents. Thus it can be described solely on the basis of its structure, i.e. by the way groups and roles are arranged to form a whole, without concern with how the agents actually behave. MAS will be analyzed from the "outside", as a set of interaction models.

For the CBAC, special components called *Communication* and *Coordination Component* are generally required. They are two of the elemental components in the system and play the role for the *Communication and Coordination*. They accept and interpret messages from other agents, and send replies.

2.2.5.5 Coherence and Coupling in MAS

Current design approaches, such as BCA and CBAC, have focused on agent's intelligence and overlooked interaction. An agent in a complex application is often involved in multiple tasks concurrently, and most tasks require interaction with other agents, with little or no knowledge of other agents' internals (autonomy). Interactions are the most critical aspect of MAS and they constitute the first tier in agent architecture. The requirement of system coupling depends very much on how interaction is defined in the system. When the interactions are not properly designed, agents participating in these interactions can be highly dependent on each other, and this may increase the coupling of the system. It may be against the advantages of localization of the MAS. The resulting MAS may be brittle, and susceptible to failures when any part of the system is modified.

On the other side, MAS is an integrated system. Agents should work together towards some common goals. This requires coherence. Coherence is attainable with carefully designed collaboration protocol among agents. LAA and AGR have the ability to handle special requests for agent collaboration.

From architecture design perspective, BCA and CBAC requires stronger coupling. The other agent architectures consider less system coupling, and do not provide implementation details for communication

protocol among the agents. Thus we cannot tell if they can be applied for an agent system with high coherence.

2.2.5.6 Availability of the Architecture to the Design of the Agent System

The AGR provides the most flexibility to the design of the agent system since its designer is responsible for choosing the most appropriate agent model as internal agent architecture.

2.2.5.7 Scalability of MAS in a Heterogeneous Environment

In the context of MAS, it will need to scale in a number of different dimensions: (1) the total number of agents; (2) the total number of systems or platforms; (3) the size of the data (rules) maintained by an agent; and (4) the diversity of agents. There should be different scenarios for these cases. Scalability becomes more important when performance is of concern.

2.3 Mobile Agents System (MASS) Architecture

Mobile Agent is a classification of agents whose predominant feature is the ability to transport between nodes on a network or between nodes across networks. Thus agent mobility/migration is one of the fundamental aspects, which the system architecture must take into account beyond the intelligence and collaboration for single agent system architecture and MAS architecture. In terms of mobile agent system design, there could be two concepts: one is the mobile agent architecture and the other the mobile agent system architecture.

2.3.1 Mobile Agent Layered Architecture

2.3.1.1 Approach

To the majority of the mobile agent community, a mobile agent is regarded as a SW component/program, which can be sent as an independent application, and run on a remote location. This agent should run on different OS platforms, and does not have to be recompiled. Thus it is designed for working in a heterogeneous, distributed system. Since a mobile agent needs to work at different platforms, and this is common to all mobile agents designed for different application domains, there is suitable generic agent architecture. Figure 2.8 shows proposed mobile agent architecture.

The bottom layer is designed for communication, which is specified with proper protocol, and generally written in platform independent language such as Java. For example, in Concordia System (Concordia White Paper, 2002), this layer contains code written in Java. In general, a mobile agent has four aspects in design view:

- Implementation Language (most of them are implemented on Java);
- Security (Agent ID, Agent Authentication, Security and Integrity);
- Mobility;
- Communication protocol.

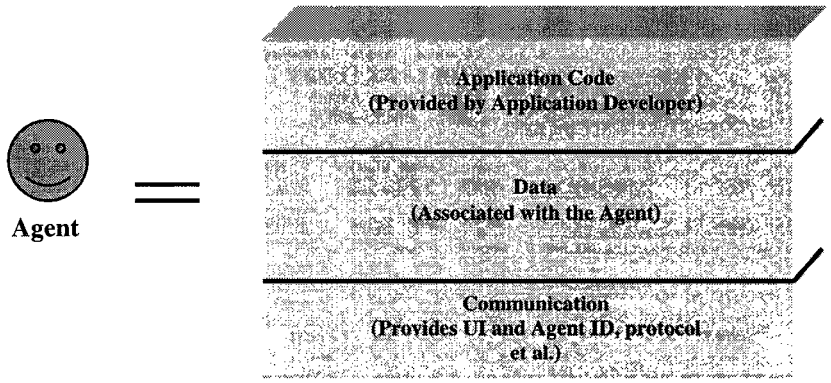

Figure 2.8. Mobile agent architecture (layered for incorporating into destination system)

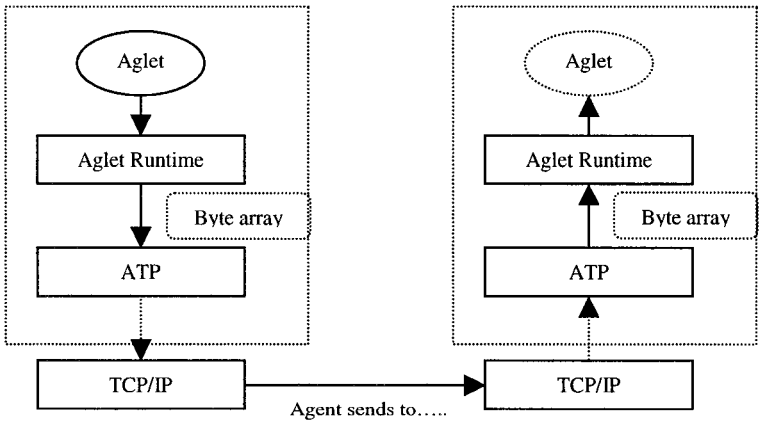

Figure 2-9. Aglet agent architecture and transportation

Figure 2-9 shows how this mobile agent architecture should work (http://www.trl.ibm.co.jp/aglets/api/com.ibm.aglet.AgletInfo.html). The difference among mobile agent design lies on the Agent

Transport Protocol (ATP). The bottom communication layer protocol depends on the environment in which the agents communicate.

2.3.2 Client/Server Architecture

After the mobile agent architecture is individually defined, a mobile agent system (MASS) architecture will be considered. Due to the requirements from management and application of the MASS, most of them are based on server/client architecture.

2.3.2.1 Approach

Most of the current MASS show underlying client/server architecture to some extent, and is treated as transportable straight extension of the client/server technology. In the client/server paradigm, communicating entities have fixed and well-defined roles; a *server* offers a set of services and a *client* makes use of those services. This model also implies a strict sense of dependency; clients are dependent upon servers to provide the services that they require. The communication mechanism that takes place between a client and a server is through a message passing protocol since network communication is assumed. However, message passing has been criticized as being too low level, requiring programmers to determine network addresses and synchronization points. Figure 2-12 shows the system architecture for Concordia.

The Client/Server architecture is a generic architecture, which provides a framework for MASS. It is the application's responsibility to refine the design with suitable platform for each system component.

2.3.2.2 Examples

1). Remote Procedure Call (RPC) System: The Remote Procedure Call (RPC) system developed by Sun Microsystems Incorporated has underlying client/server architecture. This is achieved by allowing the client to request a service to be executed on a server in the same way that it would make a local function call. These services are represented by *stubs*, which are template function calls that pass through to the RPC subsystem. The location of the server, the initiation of the service and the transportation of the results are handled transparently to the client.

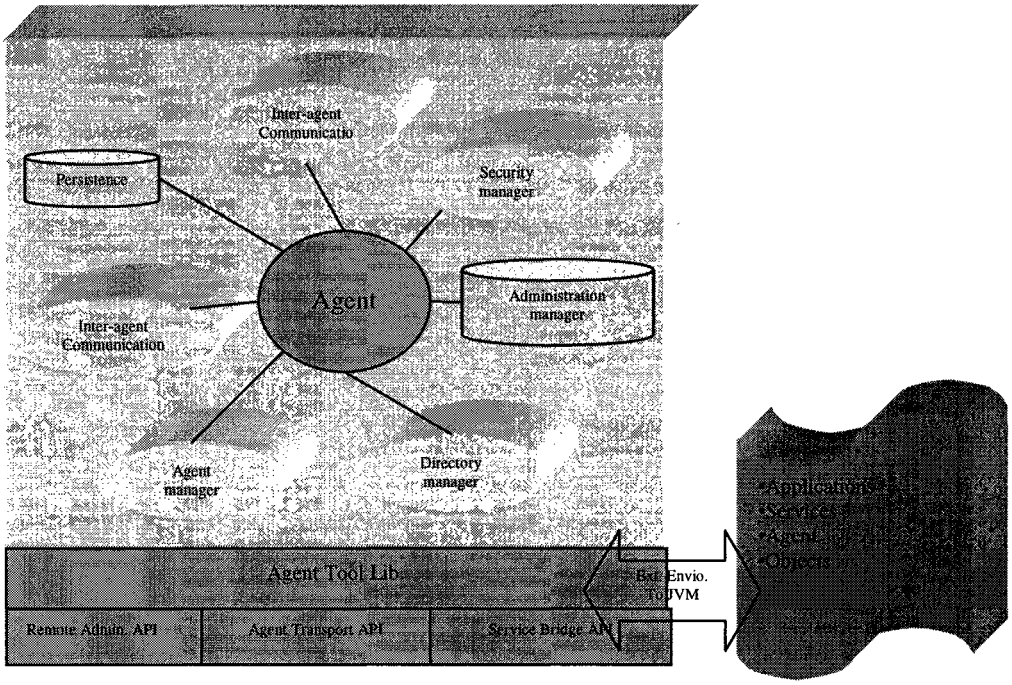

Figure 2-12 Concordia MASS architecture

However, a fundamental problem exists with client/server architectures when considering distributed information management. If the server does not provide the exact service that the client requires, for example the server only provides low-level services, then the client must make a series of remote calls to obtain the end service that it requires. This may result in an overall latency increase and in intermediate information being transmitted across the network, which is wasteful and inefficient, especially for large amounts of data. Moreover, if servers attempt to address this problem by introducing more specialized services, then, as the number of clients grows, the amount of services required per server becomes unfeasible to support.

2). Common Object Request Broker Architecture (CORBA) is another example, which can be applied in the MASS (Object Management Group, 1993). CORBA attempts to make the client/server paradigm more accessible by adopting the object-oriented principles of object reuse, inheritance and encapsulation. Distributed Computing Environment RPC (Open Software Foundation, 1992) offers security and authentication facilities and an interface of user-level threads instead of sockets to achieve a higher level of abstraction.

3). An HTTP-Based Infrastructure for Mobile Agents: The HTTP-Based Infrastructure for Mobile Agents (Lingnau, 1995a) is a project being developed at the Goethe University in Germany. It is designed to provide a low-level infrastructure to support agent mobility and communication through the use of HTTP.

The architecture itself consists primarily of an *agent server*, which is a process that executes on every host that can be accessed by agents. Its tasks include accepting agents, creating an appropriate runtime environment for agents to execute within, supervising the execution of agents and terminating agents if required. In addition to this, the agent server must also organize the transfer of mobile agents to other hosts, manage communication between agents and their users, and perform authentication and access validation.

4). Agent TCL System: The Agent TCL system (Gray et al, 2001;, 2002) is a model for supporting transportable (mobile) agents that is being developed within the Department of Computer Science at Dartmouth College. The architecture of Agent TCL is based upon the server model advocated by Telescript (White, 1994) and the initial language implementation is centered around an augmented form of the Tool Command Language (TCL) (Ousterhout, 1994). The alpha release of Agent TCL only supports synchronous, network-oriented communication through the commands agent_meet to initiate a communication with another agent, and agent_accept, which completes and synchronizes the two agents. Also, movement between sites is predetermined with a set of machines being specified in the agent's code. However, future work will look at making agents aware of their environment so that they can plan a migration strategy.

The Agent TCL architecture has been used in three information retrieval applications. The first is in the domain of technical reports, the second in text-based medical records and the third in three-dimensional drawings of mechanical parts.

5). TACOMA: Tromosø And COrnell Moving Agents (TACOMA) (Johansen et al., 1995a; Johansen et al., 1995b) is a joint project that is being developed by the University of Tromosø and Cornell University and is primarily concerned with providing operating system support for agents. TACOMA has been through many stages of revision, but the latest prototype uses TCL/HORUS, which is a version of the TCL scripting language that uses HORUS (van Renesse et al., 1995) to provide group communication and fault tolerance. TACOMA also has mechanisms for firewall agents (tag_firewall) and RPC-style communicating

agents (tag_rpc). The tag_firewall agent provides an entry point to a site and a convenient point to perform such functions as authentication, access control and accounting. tag_rpc allows client agents to block while they are awaiting results from a server agent. The future of the TACOMA project lies in investigating mechanisms for fault tolerance, for example, by using rear guards. A rear guard is a special agent that is left behind when an agent migrates; it is responsible for launching a new agent should a failure cause an agent to vanish and for terminating itself when it is no longer necessary.

6). Telescript: Telescript (White, 1994) is a commercial product developed by General Magic Incorporated to support mobile agents for an electronic marketplace. Telescript is an object-oriented programming language in which state-oriented migration is seen as the basic operation which is provided by the go instruction and a *ticket* argument that determines the destination site in "varying levels of specification" (White, 1995). A Telescript *engine* exists at each site to accept and authenticate migrating agents and to restart the execution of agents at the statement immediately after the go command.

Telescript is currently being used in the AT&T PersonalLink(TM) network for mobile communications and is seen as a potential mobile computing mechanism for Personal Digital Assistants (PDAs).

2.3.3 Peer–Peer Architecture

2.3.3.1 Approach

Compared to the Server/Client architecture, Peer-Peer architecture follows decentralized approach. Figure 2-13 shows a sample from AgentSpace.

This type MASS architecture is basically a middleware for distributed environment with OS platform independent language (such as Java). For example, AgentSpace has been designed to run on various OS platforms including Windows, MacOS, Solaris and Linux.

However, as an agent is highly autonomous, security is the first concern in case the host does not have sufficient firewall to identify if an incoming agent is safe.

2.3.4 Discussions

2.3.4.1 Focus

Client-Server Architecture: Sub-programming alleviates the problem of client/server architectures somewhat by allowing clients to launch subprograms at the node where the service is located. In this way, any number of requests can be initiated locally and the subprogram can process the intermediate results before transmitting the actual results to the client once it has finished. Example sub-programming systems are SUPRA-RPC (Stoyenko, 1991), Remote Evaluation (REV) (Stamos and Gifford, 1990) and Network Command Language (NCL) (Falcone, 1987).

Although subprograms can migrate on their initial launch, they cannot subsequently move to other systems or resources. This restricts their ability to communicate with each other since they may not execute for large periods of time and also maintain the rigidity of the client/server relationship. Additionally, subprograms are generally written explicitly for a specific client, so their reusability is limited.

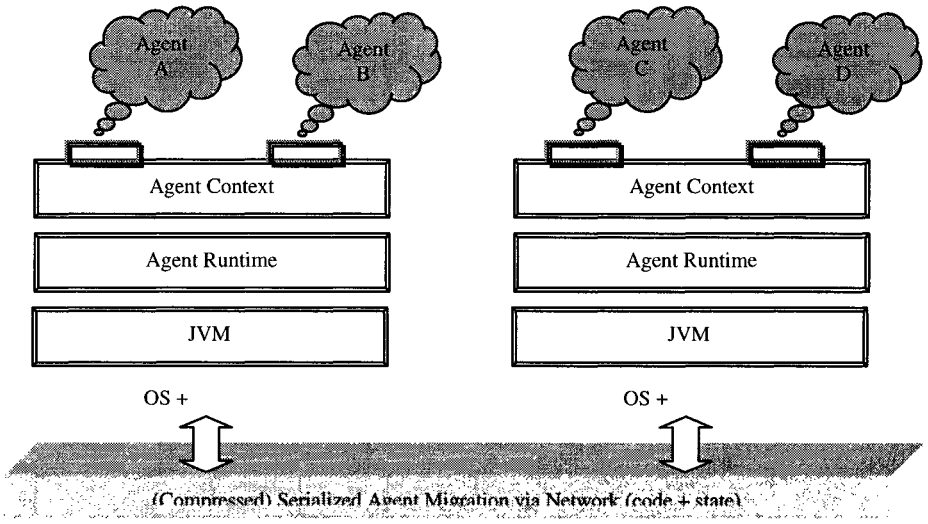

Figure 2-13 AgentSapce MASS architecture

The mobile agent paradigm attempts to address the issues that are raised by the client/server and sub-programming paradigms. Typical characteristics of mobile agents are their ability to migrate at will, autonomy in their actions, a peer-to-peer personality and a processing and network independence from their

original location. The question is which agent architectures is capable of justifying descriptions in terms of success. This is an issue to be addressed later.

2.3.4.2 Open Agent Architecture (OAA)

There are many MASS architecture proposals that have roughly the same purpose and present a common set of functionalities. Nevertheless, they also present some important technical and even conceptual differences. For instance, Telescript, which is based on Client/Server architecture, is used to be the representative reference for MASS with a high technological base. However it is (or it was) a proprietary system, with a difficult-to-learn programming language, and is not suitable for a dynamic and open environment such as the Internet. On the other hand, some MASS, such as ffMAIN is language based and system independent (Lingnua, 1995a, 1995b), but it shows severe limitations on the overall performance and difficulties in developing complex application. Aglets – today the reference Java-based MASS/MAS – lacks an elaborate object model, e.g. without the notion of execution places hierarchically organized, and without management operation on agent families and clusters. It also lacks some technical capabilities, e.g. just two ACL's (access control levels). It does not have the notion of agent class manager, "open channel" capability, or even the notion of users transparency.

The architecture of all three types listed here, is more or less application domain dependent. Thus it is not a perfect architecture. We expect that, in the future, they will be improved in order to support the development and execution of more flexible, reliable, secure and efficient agent-based application (ABA). In order to achieve that objective, a MASS architecture should incorporate a good combination of features from all of them. For example, they should support the independence protocol of TCP/IP and/or HTTP-based MASS, the technology for migrating agents (threads) as applied in the Telescript, and the tight integration with Java as supported by Aglets.

This comes from the idea of the OAA, which is a conceptual MASS/MAS architecture, composed of three complementary components: AES, for Agent Execution System; ACS, for Agent Class System; and AEE, for Agent Execution Environment. These three components should be the "Building Blocks" of an MASS/MAS. There could be other additional components needed for particular application domains.

Here are five important aspects for OAA architecture:

- Agent System Description: involves the agent architecture/object model, and the notion of execution hierarchy;
- Languages: involves system implementation language and agents design language
- Mobility: involves transport protocol at inter/intra system level;
- Communication: involves the communication inside an agent, between agents in agent level (ACL) and system level (communication between different agent system through mobile agent);
- Security: which involves the agent security and also host security;

2.3.4.3 Applications

To be able to run on multiple platforms, the OAA needs to provide sufficient libraries (Agent Language Library: ALL). For instance, OAA agent libraries exist for the following languages and platform:

- Quintus Prolog: SunOs, Solaris, Windows
- ANSI C (Unix, Microsoft, Borland): SunOs, Solaris, SGI IRIX, Windows
- Common Lisp (Allegro and Lucid): SunOs, Solaris
- Java: Any Java Platform
- Borland Delphi, Windows
- Visual Basic: Windows

The OAA should provide also a Agent Development Tools (ADT), which is a set of utilities for guiding a programmer through the steps required for defining and adding new agents to the OAA. The current OAA contains three separate utilities: a) PROACT allows a programmer to define the capabilities of a new agent; b) LEAP is a tool for adding linguistic information to an agent; and c) PROJECT is used to define an agent-based application as a particular collection of domain agents.

2.3.4.4 Technical Hurdles for MASS

There are several technical hurdles that must be cleared before MASS can be widely applied. They are also important research issues.

- Performance and Scalability: Current MASS saves network latency and BW at the expense of higher loads on the services machines. Agents are often written in a (relatively) slow interpreted

language for portability and security reasons, and must be injected into an appropriate execution environment upon arrival. Thus, in the absence of network disconnections, mobile agents often take longer to accomplish a task than other traditional implementations. This is due to time saved from message communication, which is less than the time penalties from slower execution and the migration overhead. One solution for this is to have just-in-time compilation (most notably for Java), SW fault isolation, and other techniques that allow mobile code to execute nearly as fast as natively compiled code. In addition, the migration overhead must be reduced. Together, these efforts would lead to a system in which accepting and executing a mobile agent involves only slightly more load than if the service machine had provided the agent's functionality as a built-in, natively compiled procedure.

- Portability and Standardization: Nearly all MASSs allow a program to move freely among heterogeneous machines. The code is compiled into some platform-independent representation such as Java bytecodes, and then either compiled into native code upon its arrival at the target machine or executed inside an interpreter. For mobile agents to be widely used, however, the code must be portable across mobile-code systems, since it is unreasonable to expect that the computing community will settle on a single mobile-code system. Making code portable across systems will require a significant standardization effort. The OMG MASIF standard is an initial step. But it addresses only cross-system communication and administration, leading to a situation in which an agent cannot migrate to the desired machine, but instead only to a nearby machine that is running the "right" agent system. This is what we called inter/intra MAsS communication standardization. We may stress three aspects here: 1) network oriented, 2) node oriented, and 3) synchronous and asynchronous communication.
- Security: It is possible now to deploy a mobile agent system that adequately protects a machine against malicious agents. Numerous challenges remain, however: (a) protecting the machine without artificially limiting agent access rights; (b) protecting an agent from malicious machine; and (c) protecting groups of machines that are not under single administrative control. An inadequate solution to any of these three problems will severely limit the use of mobile agents in a

truly open environment such as the Internet. A good MASS architecture will consider the agent security (agent) and host security (system) as a whole in the design phase.

2.3.4.5 Limitations for MASS

Although each of the proposed mobile agent architectures support differing levels of functionality, they attempt to address the same problem, namely, enabling portions of code to execute on different machines within a wide-area network.

The weakest system is the HTTP-based infrastructure for mobile agents due to its primitive agent migration mechanism and simplistic communication model. For example, it is difficult for the mobile agents to pass real-time data between each other. However, asynchronous communication is desirable when an agent needs to issue multiple information requests and then process the results at a suitable time. The concept of a shared information space where agents can advertise their services is useful in arranging for mobile agents to meet and share data.

TACOMA also suffers from a stateless migration policy that hampers the ease with which migration can be achieved from the point of view of the programmer writing a mobile agent. Additionally, TACOMA lacks dynamic determination. This means a programmer must be aware of the locations that an agent can visit before it can be launched. This is advantageous for predicting the path and position of an agent, but limits its functionality. However, TACOMA's data storage model is very flexible and the actual way in which information is transferred between agents and servers is beautifully simple.

The main problem with the Telescript architecture is that it only supports the Telescript language (which *must* be entirely supported by Telescript engines) and is not open to researchers to develop. On the other hand, the Telescript language was designed for writing mobile agents so that migration is well supported. Security is intrinsic to the system and communication between Telescript agents is handled through meeting places and also directly between agents.

Agent TCL appears to be the most flexible architecture: it supports state-oriented migration, multiple languages and networking protocols and a comprehensive communication subsystem. Once the predetermined routing of agents is resolved and some security measures are implemented, it becomes a flexible and powerful mobile agent system.

One feature that all of these mobile agent architectures have failed to address explicitly is in defining a domain of applicability; they all concentrate on the mobility of agents rather than the integration of agents with information resources. The mobile agent community concentrates the Telescript language too closely on the migration aspect. It does not consider integration with the desktop and third party applications. To remedy this situation, General Magic looked into integration. They combined Telescript mobile agents with a desktop in a heterogeneous fashion by using the abstract windows toolkit (AWT) in the Java programming language. Further, they recently produced a set of integration tools, called Telescript Active Web Tools, which provide class hierarchies to enable Telescript agents to integrate with Web servers and resources.

2.4 MAS/MASS Frameworks

2.4.1 IBMAglet Agent Framework

The IBMAglet is a 100% Java mobile agent technology (workbench). The Java Aglet API (J-AAPI) contains methods for initializing an aglet, message handling, and dispatching, retracting, deactivating/activating, cloning, and disposing of the aglet. An aglet is an atomic working unit in the IBMAglet. It is implemented in the same principle as an agent in terms of functionality. Application developers can write platform-independent aglets and expect them to run on any host that supports J-AAPI.

2.4.2 MiLog Mobile Agent Framework

This is a logic-based mobile agent framework for constructing intelligent information agents. In the framework, inference and planning process of an agent is written in a compact logic program. The framework contains a new generation logic program execution engine, which provides a strong migration capability. It is used to help developers to construct more responsible and flexible information gathering systems.

A good example is the BIDDINGBOT. Fukuta et al. (2000a, 2000b) propose it based on the MiLog Model. It is a system, which can support bidding to several auction sites simultaneously by the cooperative-bidding agents (Ito, 2000). BiddingBot consists of one leader agent and several bidder agents. Each bidder agents is assigned to an auction site, and implemented by using the MiLog framework. The bidder agents

cooperatively gather information, monitor, and bid in the multiple auction sites simultaneously. Using shared clause DB and querying to other agents implement communications among agents. The leader agent facilitates cooperation among the bidder agents as a matchmaker; send a user's request to the bidder agent, and presenting bidding information to user. The leader agent interacts with a user as a WWW server on the MiLog framework. The cooperation protocol is implemented by finite state machine in which the state is changed by receiving a certain query from other agents.

2.4.3 ZEUS Agent Framework

The ZEUS is an agent-build tool-kit, developed at the Intelligent Systems Research Unit at British Telecom. It is being used to create a prototype distributed MAS with the main application in the field of electronic marketplace within a LAN. ZEUS framework provides complete agent class library. The need to provide a generic, customizable, and scaleable industrial-strength collaborative agent building tool-kit motivated this framework. The tool-kit itself is a package of agent classes implemented in Java, allowing it to run on a variety of hardware platforms. ZEUS has the ability to delineate the domain-level problem-solving power from the agent-level functionality. In other words, it provides classes that implement communication, co-operation, co-ordination, task execution and monitoring, and exception handling, leaving developers to provide the code that implements their agents' domain-specific problem-solving abilities.

2.4.4 CONCORDIA Framework

Concordia is a 100% Java-based mobile agent framework developed at Mitsubishi E.I.T.America. It contains:

- Concordia server
- Java virtual machine and
- Agents(s)

The agents are programmed in Java and execute on server with JVM. It provides a number of services including: security, mobility and communication. For communication, it uses TCP/IP protocol for the transportation with lightweight Agent Transport API (application embedded). The inter-agent

communication (registration, posting and notification of event) is also provided. This is an agent framework widely applied in information processing and E-Commerce.

2.4.5 ObjectSpace/AgentSpace Framework

It has been built on the top of Voyager. It is a 100% Java agent development platform that allows Java programmers to create network applications using both traditional and agent-enhanced distributed programming techniques. Voyager was the first version, and designed to use regular Java message syntax to construct remote objects, send messages, and move them between applications. The latest version is renamed as AgentSpace.

2.4.6 AgentOS Framework

It is a distributed computing environment or virtual operating system supporting the development of distributed client-server and object-oriented application using agent. Agents are active, autonomous objects containing both computational logic and state information, with the capability of navigating an agent system, accessing the services offered by the nodes within the system to complete an assigned task, on behavior of a human user or another agent. AgentOS is a Java-based agent-host that provides support for the execution of agents, a network-transparent event-based communication mechanism, and a model for accessing available services. AgentOS is augmented and extended by a system of services, which are implemented as agents themselves.

2.5 Important Research Issues

We may conclude that agent system architectures are very diverse, but there are general rules to follow. Single agent systems, in general, focus on the intelligence of agents. MASs add on collaboration and interaction among agents inside the system, and the collaboration and interaction among the agents plays more important role in building the MAS. For MASS, system design should consider the mobility or agent migration from place to place. We can argue that large and complex agent systems should be able to cope with heterogeneity of models, communications and individual agent architecture, which could be generic architecture. If this is true, there are some important research issues:

- How the generic agent architecture should be defined and built? Should it be built at individual agent level or agent system level, and what are the basic requirements: agent intelligence, collaboration/interaction, and/or agent mobility?
- If there is generic agent architecture, how to evaluate the success of an agent architecture design?
- Can OAA provide generic architecture that is capable of all kinds applications?
- How is security handled at agent level and at system level?
- Cognitive architecture: architecture can be defined simply as the portion of a system that provides and manages the primitive resources of an agent. For cognitive architectures, these resources define the substrate upon which a system is realized. Addressing many issues surrounding the choice, definition, extent, and limits of these resources and their management is one of the research interests. This analysis attempts to assist in determining the necessary, sufficient and optimal distribution of resources for the development of agents exhibiting general intelligence.

Architectures, in general, have divergent features that lead to different properties. For example, some utilize a uniform knowledge representation, some a heterogeneous representation, and others, no explicit representation at all. These decisions then lead to the support of specific capabilities. They follow some explicit methodological assumptions, often driven by the domains and environments in which the architecture will be used. The varieties of these choices are what are responsible for the variety of architectures. One way to further constrain the number of choices is to use examples of psychological or neuroscientific validity in architecture design. An additional advantage of this approach is that there is synergistic interchange between the studies of artificial and biological intelligence; in particular, Newell (1959) has proposed that computer-modeling tools as represented by cognitive architectures now allow the formulation of unified theories of cognition.

However, many researchers purposely ignore the constraints posed by human cognition. Often this is because they are interested in developing agents, which populate and behave effectively in some environment. Studying the interactions between the architecture and the environment (which could be a static, problem-solving situation or a highly dynamic, reactive environment) is of primary concern. In this sense, the term *cognitive architecture* is a little misleading. Although it is used throughout this report, a better term might be *agent architecture* which would include both those systems that made an explicit

attempt to model human psychology (i.e. cognitive architectures) and those which simply explore some aspects of general intelligent behavior.

Chapter 3 Applications

What Are In This Chapter: *This chapter focuses on multi-agent system applications. Both MAS and MASS communities expect broadband applications of agent systems. We will discuss some fields, which agent systems may apply for. We first look at agent types and agent systems in Section 3.1. Both AI and MAS/MASS communities have defined a number of agent types and agent systems. Some of them have been proposed for specific applications and are currently commercially available. In Section 3.2, we introduce a few of MAS/MASS applications by giving the case studies for each of them. These case studies include the areas of Organization Management, E-Commerce, Information Procession, Distributed Computation and Supply Chain. All of them have been recognized as the most active application areas for MAS/MASS. In Section 3.3, we introduce agent development tools and agent system platforms; these include ADK, MadKit, LEAP, AgentBuilder, JADE and JAT. We discussed also briefly the other Agent Platforms recommended by FIPA in this section. Finally, we discuss a few issues concerning the success and failure of these agent systems (Section 3.4).*

The notion of agents takes a broad sense, encompassing a wide spectrum of computational entities that can sense their local task conditions and accordingly make decisions on how to react to the sensed conditions by performing certain behaviors in the task environments. With the characteristics of being autonomous, adaptive, robust, and easy to implement, agent-based approaches have found many potential applications in dealing with tasks that are less structural or ill defined. In such tasks, complete mathematical or computational solutions may be either unavailable, or too expensive to use.

Agent-based system has found its roles in many applications domains. Examples of agent systems include personal software assistants that search for, filter, and size information in data and knowledge intensive tasks. Agents can be also physically embodied, such as robotic systems that cooperatively manipulate objects in a Cartesian task environment and move them from one designed location to another. Agent systems have been widely applied in software engineering, where an agent is computationally coded. A good example is optimization agents that coordinately test certain numerical values and efficiently narrow down a large search space to a smaller set of possibilities. Another example for this application

domain is the ant-colony paradigm, which has been used in telecommunication networks to perform load balancing. There is no systematic classification for agent systems, but two scientific communities have developed most of the existing agent-based systems: intelligent agent/multiagent systems (MAS) and the mobile agent system (MASS). The MASS builders tend to concentrate on the subsystems for shipping any piece of code around. For instance, Tacoma system is built to support more or less any piece of code moving about, whether or not it has some aspect that is called intelligence. On the other hand, the intelligent agent community tends to focus on application specific problems.

Regardless of their domains of application, agents often have one thing in common; namely, they locally interact with their task environments, in the course of problem solving. Responding to different local constraints received from their task environments, the agent can select and exhibit different behavioral patterns. It is not our purpose to cover all possible application domains in this research report. We rather will focus on agent-based system applications including the following features:

- Enough mobility;
- Must be autonomous;
- Having loose coupling among system components;
- Communicate with ACL, but not through ordinary message passing;
- Effective in both resources (remotely and/or locally) and security;
- Developed in a distributed fashion;
- Multiple agents which will coordinate and sometimes compete among themselves in order to most effectively accomplish a given task;

There could be different approaches to group agent systems applications, but we would like to classify them by application domains, which may requires all features listed above. To name a few:

- Supply-Chain;
- E-Commerce Automation (Negotiation System);
- Information Processing;
- Distributed Computation;
- Robotic System.

3.1 Agents and Agent System

3.1.1 Agent Types

There are a number of agent types defined by OMG. They have special requirements for different application domains (OMG Green Paper, 2001). To identify the forms of an agent is of extremely importance in MAS/MASS development. The list of agent characteristics presented earlier addresses some of these requirements. Additionally, since individual MAS/MASS has special needs, software- and hardware-related forms must be considered. Agent forms, which are considered as the most important to agent developers, are abstracted as *Software Agents*, *Autonomous Agents, Interactive Agents, Adaptive Agents, Mobile Agents, Coordinative Agents, and Intelligent Agents*. They are discussed in 1.1.4.1 (Basic Agent Abstraction). For MAS/MASS application, very special agents may also be required. Here we give two examples:

- 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 of the wrapper agent provides a single generic way for agents to interact with non-agent software systems,
- Other types of Agent: The kinds of agents listed above are the predominated 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.

3.1.2 Single versus Multiagent Systems

Many of the early commercial agents were developed for information search. Here, individual agent was launched on a tether to gather predefined kinds of information and return them to the human requester. Such an approach certainly has its many use cases. However, this approach *alone* could not build the societies or support the organizations. 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 *Multiagent Systems*. When part of the system is mobile, it becomes a *Mobile Agents System.*

MAS/MASS, then, are systems composed of agents coordinated through their relationships with one another. Some of the rationales for MAS/MASS are as follows:

- One agent could be constructed that does everything, but such 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.
- MAS better support applications requiring distributed computing. 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.

Clearly, single-agent environments are much simpler, because designers do not have to deal with issues such as cooperation, negotiation, and so on. However, the large-scale requirements of industry necessitate approaches that employ coordination and distribution. As such, the majority of IT industry are focusing primarily on MAS/MASS rather than single agent system. We will discuss only MAS/MASS applications, and will not touch the applications accomplished by single agent.

3.1.3 A Survey on Current Agent Systems

There are over a hundred agent systems running worldwide. Table 3.1 lists 76 of them, which have complete information and known running/development states.

Table 3.1 Multiple agent systems running world wide

No	Agent System Name	Description	Developer	Lang-uage	Application	Ver-sion	Date

1	AgentSpace (UK)	Agent building platform	Univ of Hull / Univ of Sunderland	Java	General purpose		2001
2	AgentSpace (a)	Agent building platform	Univ Técnica de Lisboa (IST/INESC)	Java	General purpose	1.2	1999
3	AgentSpace	Mobile agent framework	Alberto Sylva	Java	Support for dynamic and distributed applications		2002
4	Agent Tcl	Transportable agent system	Univ. Dart	Tcl Tk	Information management		2002
5	Aglets	Java class libraries	IBM Tokyo	Java	Internet	beta 1	2000
7	aIsland	Exposes objects (graphic module, audio module, neural network, fuzzy logic ...) to an agent developer, who can glue these modules together	JXTA aIsland Project	Java scriptin g	Distributed computation, asynchronous	1.0	2002
8	Ajanta	Network mobile object	Univ of Minnesota	Java	General purpose	Alpha 1.0	1999
9	AMASE	- optimzed for wireless mobile communication environment - resource negotiation before migration, resource control - agent signing; access control based on ACLs - trust concept for defining which platforms an agent user trusts - service trading with Meta-Keywords	AMASE Consortium	Java + JDK	Information service, asynchronous messages synchronous RPC	1.1	1999
10	AMETAS	* Usage of encryption / signatures / roles / policies for agents * Strict autonomy * Event management * Integration of users and external entities by adapters and services * Agent type system * KQML * Hierarchical place naming system * Temporary and permanent places	Goethe-Univ, VSB	Java + JDK	E-Commerce (Asynchronou s message passing)	1.0	1999
11	Anchor Toolkit		Lawrence Berkeley National Laboratory	Java + Unix	General purpose	1.0bet a	1999
12	Ara - Agents for Remote Action	Agents for remote action	Univ of Kaiserslautern	C/C++, TCL, Java	Partially connected c. D.D.B	1.0a	2002
13	ARCA - Autonomous Remote Cooperating Agents	Autonomous Remote Cooperating Agents. text-based and GUI-based agent server, console to control local or remote agent servers	Ins of Telecommunicatio ns and Computer Science, Univ of Catania - ITALY	Java + JDK	synchronous two-way, asynchronous two-way and one-way invocation	2.3	1999
14	Bee-gent - Bonding and Encapsulation Enhancement aGENT	- Mediation Agents - The methods of information exchange and communication are highly suited to Internet and Web-Top computing. - GUI-based Interaction Protocol development tool. - Conformance to the FIPA Spec. of the agent technology.	TOSHIBA Corp.	Java + JDK	Asynchronous Communicati on / HTTP; XML based-messaging; Distributed computation	Beta1. 2	1999
15	Bond	A Multi-Plane State Machine Agent Framework	Computer Sciences	Java Python,	KQML and XML based	release	1999

			Dept.,Purdue Univ.	Jess embed ded, + JDK	messaging		
16	CBorg	This system featurs STRONG migration: at any time can a proces migrate to another place. Second, agent names are LOCATION TRANSPARENT. If an agent migrates you dont need to know its new location. A specially build router will take care of this.	Vrije Univ. Brussels	Windo ws Linux Solaris Macint osh PalmO S	message sending & serialisation of data	3.0	2000
17	Concordia	Framework for agent development	Mitsubishi Electric ITA	Java	Mobile computing, Data base	V1.1.3	1999
18	D'Agents	MASS system	Dartmouth College	Tcl, Java, Schem e + Unix	Message passing; Distributed Information Retrieval	2.0	1999
1`9	Dejay	Using Dejay. It allows remote creation, remote references and migration of objects.	Hamburg University	Java, Dejay + JDK	Distributed computation	0.9.4	2000
20	dynamicTAO	MAS/MASS system	Univ. of Illinois at Urbana-Champaign	DCP & Java + Solaris, Linux, Windo ws	Emphasis on dynamic configuration of CORBA component-based applications	1.0	2000
21	Evolutionar Agent Societies (EAS)	MAS/MASS. Self-organization of agents based on ecological model. Agents support unique, evolvable personas.	ANSER	Java + JDK	Com Transport layers: (asynchronou s sent) messages	1.0 Alpha	1999
22	FarGo	An integrated MAS/MASS collection of development and deployment tools	Technion - Israel Institute of Technology	Java + Windo ws, Solaris, Linux	regular method invocation (remote-call semantics). Dynamic Layout of Distributed Applications	0.2.2	1999
23	ffMAIN	Cross-language agent communication; fine-grained security mechanism available; WWW integration; agent server is feature-complete, efficient HTTP server	Fachbereich Informatik, Johann-Wolfgang-Goethe-Univ Frankfurt	Tcl, Java; Perl + Unix, Win32	HTTP, General purpose	2.0	1999
24	Grasshopper	MAS/MASS for supporting management of agency	IKV++ GmbH	Java + Windo ws, Solaris	asynchronous/ synchronous messages General purpose	1.2.2	1999
25	Gypsy	Component-Oriented Design, Mobile Agent as Programming Paradigm, LDAP Agent/Place Registry, Supervisor-Worker Concept	Technical Univ of Vienna	Java, Python + JDK	one-way-, asynchronous messages, General purpose	0.5	1999
26	Hive	Decentralized, peer to peer architecture	MIT Media Lab	Java, + JDK	Java RMI, Networking	0.6	1999
27	Imago Prolog	Implement a Lightweight Virtual Machine for Prolog programing	Imago_Lab UofGuelph	Prolog-LVM +	Active Messages	0	2002

		language		Linux			
28	IMAJ	Supporting IMAJ agents	UNINOVA/FCT + ISEL	Java,JINNI JESS proprietary MAAPL + LDK	LINDA tuple space, Cooperative Information Agents; Monitoring in Adaptive Mobile Agents	alpha release	2000
29	J-SEAL2		CoCo Software Engineering GmbH	Java + JDK		1.0	2001
30	JAE - Java Agent Environment	MAS/MASS	Aachen Univ of Technolog.	Java + JDK	Message Passing, Blackboard (Tupple Space), Service Center, RPC. E-Commerce; Distributed computation	1.2	1999
31	JAM	BDI-theoretic agent framework underlying the reasoning capabilities of the agent. Mobility is just one of many capabilities that JAM agents have at their disposal.	Intelligent Reasoning Systems	Java 2 + Java	Socket, + General purpose	0.61+0.79i	1999
32	JAMES	Support for Fault-Tolerance. GUI for system and agent monitoring. Caching and Prefetching mechanisms. Support for Resource Management. Dynamic software update Support for Network-Management	Univ of Coimbra, Portugal, Siemens S.A, Portugal	Java + JDK	JavaSpaces, General purpose	v2.0.1	1999
33	JA Title	Java agent framework dev/KAML	Standford Univ.	Java	Information retrievial, interface agent	N/A	2001
34	JavaNetAgents	Support MAS/MASS	LIP6/ONERA	Java + JDK	asynchronous message passing, General purpose	0.9	1999
35	JavaSeal	Security of agents on a site. Each agent executes in a protection domain. Protecting the host and other agents from potentially malicious agents.	University of Geneva	Java + JDK	channels transfer, + AgentOS	1	1999
36	JCAFE - Jini Compositional Agent Framework for the Enterprise	Jini Compositional Agent Framework for the Enterprise. MAS	MITRE Corporation	Unix, Windows, + Java	KQML over RMI; General purpose	1.0	1999
37	Jinni	Knowledge processing with embedded Prolog engines	BinNet Corp.	Java, Prolog, + any platform running Java	Mobile code through sockets + encryption; AI agency	2002	2002

38	Jumping Beans	Mobile CORBA, MASS	Ad Astra Engineering, Inc.	Java + Java	Networking, distributed computation	1	1999
39	Kaariboga	MAS/MASS	none	Java; / JDK, Windows, Linux	asynchronous messages; General purpose	2000-08-06	2000
40	Kafka	Multiagent libraries for Java	Fujitsu Lab. Japan	Java / Unic based	General purpose	N/A	2001
41	Kali Scheme	Distributed impl. Of scheme	NEC Research I	Java	Distributed data mining, load balancing		
42	Klaim (Kernel Language for agent interaction and mobility)	Node type MAS/MASS	Univ of Florence and Univ of Pisa	Java / JDK	tuple spaces / Networking	1.1	1999
43	Klaim (Kernel Language for agent interaction and mobility)		Univ of Florence and University of Pisa	C/C++		1.1	2000
44	Knowbot Operating Environment	Research infrastructure of MASS	CNRI	Python	Distributed system / internet	1.0 alpha 2	1999
45	Messengers	Autonomous messages	UCI	C (Messenger-C)	General purpose	N/A	2002
46	M0 Messengers	Autonomous messages	Centre Univ d'Informatique. Univ of Geneva, Switzerland Uppsala Univ Sweden	C (Messenger-C)	General purpose	0.27.11	1999
47	MAgNET	Focuses on agent economies where buyers and suppliers use mobile agents to search the web and find out about each other	University of California, Santa Barbara	Java/ Aglets	TCP/IP socket, TCP/IP message passing / Networking, E-Commerce	1.0	1999
48	MAP	- GUI for monitoring agent and agent system - The platform integrates the paradigm of remote execution and code on demand	Univ of Catania - IIT	Java / Java	Asynch and synch msg. passing / Distribute Computation	2.0	1999
49	MATS - Mobile agent teams	Heterogeneous architecture, very specialized functions per agent	British Telecom	Java / Java	Asynch and synch msg. passing / distributed parallel processing	Beta	1999
50	MESSENGERS	Application-specific logical network used for navigation; it supports fault-tolerance by automatic snapshots and restarts	University of California, Irvine	C / SparcStation4/ Unix	randezvous event wait/signal shared node variables / general-purpose high-performance computing	2.0	1999
51	MILLENNIUM	Provide special security features	University of Oviedo.	Java / Java	N/A / General Purpose	Version 1.0	1999

52	MiLog		Nagoya Institute of Technology	Prolog / JD, Windows, MacOS 8 MacOS X IRIX.	HTTP based inter-agent communication mechanism / Information Retrivals, Robats	1.2	2001
53	MIPLACE	* Code generator for agent-application * Useful GUI for management * Support for permanent object	NEC Corporation	Java / Solaris, Windows	JavaRMI, JavaIDL, CORBA / General purpose	2.1.2	1999
54	MOA	Mobile Object and Agents	OpenGroup, UK	Java	General purpose		
55	Mobidget	An agent can be distributed over several nodes. Each piece of an agent, residing on a node, can independently migrate.	NEC Corporation	Mobidget / Windows Solaris	Remote method invocation sending object references / Networking, distributed computation	1.1	1999
56	Mogent1.0		NanJing Univ,P.R.China			1.0	1999
57	Mole	First Java-based MASS system	Univ of Stuttgart, IPVR	Java, Unix	General purpose	3.0	2000
58	muCode	Structured itinerary: Full separation of itinerary from function body Three flexible migration modes:Select,Sequence and Dynamical itinerary modification	Politecnico di Milano	Java / SUN OS 5.x	MPI-based communicator aided, send / general purpose	1.0	2000
59	Nomadic Pict	Nomadic Pict describes both a strongly typed programming language and the runtime system which implements the code execution and mobility	Univ of Cambridge, Computer Laboratory	Nomadic Pict / platforms supported by OCaml	Communication on channels in the Nomadic Pi Calculus style / MASS, general purpose	alpha	2002
60	NOMADS	Provides transparent mobility for Java agents. This transparent mobility has been achieved by developing a new clean-room Virtual Machine.	University of West Florida	Java / Windows NT Solaris Linux	Raw message transfer / MASS with Strong Mobility and Fine-Grained Resource Control	0.11 Alpha	2000
61	OAA	Open Agent Architecture	SRI Int. AI	C, C-Lisp, Java, VB	General purpose		
62	Odyssey	Set of Java Agent class libraries	General Magic, Inc.	Telescript	E-Commerce	None.	1999
63	owchii		sourceforge	Java / Java	automatic proxy and routing of messages	2	2002
64	Pathfinder	Stronger Management and Mobility Support.	National Chung-Hsing Univ TAIWAN	Java /Java	ATP and DMI / Active Networks	0.9	1999
65	Planet	Layered architecture: Neutrality for both programming-language and bytecode designs.	Univ of Tsukuba, Japan	C & C++ / Solaris,	Remote memory-mapped files /	alpha	1999

				Linux	Distributed computing		
66	Plangent	Intelligent agent system	TSHIBA Corp.	Java	Intelligent tasks	1.0beta7	1999
67	rmi64	Provides a demo sample	Technische Univ Wien	Java / java+rmi	N/ A / A simple showcase	1.0.0	1999
68	SeMoA (Secure Mobile Agents)	Extensive security mechanisms. Extendible architecture. Pluggable security filters for incoming and outgoing agents	Fraunhofer IGD	Java / JDK platform	SMTP, POP3, Raw socket / Distributed Computing	0.3	1999
69	SOMA	CORBA facility SECURITY facility	DEIS -Univf Bologna	Java / Windows, Solaris	Message exchange / MOBILE COMPUTING support	2.0	1999
70	Tacomas	Tromso and Comel Moving Agent	Noeway & Cornell	C, Unix-based	Vlient/Server model issues / OS support		2002
71	TAgents		The TAgents project	Java / Java JVM	Messages of Java serialized objects / General purpose	0.8.3	2001
72	Telescript	Places abstract the locales of agents and thus the sources and destinations of the trips that agents make. Permits control agent consumption of resources, esp. time and space. Place and agent state safe-stored to disk	General Magic, Inc.	Telescript / Solaris	Agent migration. / Networking, distributed computing	None.	1999
73	The Rube	Mobile code system	UK	Scheme	Remote execution of scheme		
74	TuX (Tacoma UniX)	Wrappers (coming soon) to encapsulate mobile agents.	University of Tromsø	Tcl, C, Phyton, Perl, and Scheme	Sync or async send / MASS for general purpose	2.0	1999
775	Voyager ORB Professional	Platform for distributed application	ObjectSpace, Inc.	Java	Support for agent systems	3.1.1	1999
76	WASP	Strong migration for Java by using code instrumentation - agent payment mechanism based on JECF - graphical agent construction tool - Java Card usage for agent authorizaton - Java Card integration as trusted computing base - System configuration managable by management agents, 'hot swap' of some system components possible	Darmstadt Univ of Technology	Java / JDK	RMI, ROC / E-Commerce	2.0	1999

As we can see from Table 3.1, the listed agent platforms have been developed for different application domains. Some of them are for general purpose, and the others for specific purpose such as Internet, or E-

Commerce. To see how these MASS/MAS systems support different applications, some basic features of them are relevant:

- Supported platforms;
- Supported Languages;
- Implementation standard;
- Type of migration;
- Format of data;
- Local communication type;
- Global communication type;
- Code migration;
- Communication style;
- Agent tracking;
- Other features.

When an application is developed based on one of the available agent platform/framework, application specific issues should be addressed. These will be discussed in more details in the following.

3.2. MAS/MASS Applications

Although many MAS/MASS systems have been proposed and implemented, only a few have been put into practical use. However application specific agent systems are well defined, and corresponding frameworks have been built for some of them.

3.2.1 Current Usage of MAS/MASS Agents

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

3.2.1.1 Network and System Management Agents

Telecommunications industry has been the most active in this area, and indeed seems to be more committed to the agent paradigm. Notable applications include assisting in complex system and network management tasks, such as load balancing, failure anticipation, problem analysis, and information synthesis.

3.2.1.2 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.

3.2.1.3 Interest Matching Agents

These are probably the most used agents. The interest matching agents are used by commercial Web sites to offer recommendations, based on Maes' work (1995a) 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.

3.2.1.4 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.

3.2.1.5 Organizational Structure Agents

These agents are structured 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 represents some aspect of agents' features. However, none of these represent the full range of possible agent features.

3.2.2 Current MAS/MASS Application Types

The current kinds of applications that employ agents are still limited. This is partly due to incomplete agents concepts and agent tools available. As progresses are made in these two aspects, the agent-based approach will become more imbedded in MAS/MASS applications.

3.2.2.1 Enterprise Applications

- Smart documents (i.e., documents that `know' 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).

3.2.2.2 Business-to-Business (B2B) Applications

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

3.2.2.3 Process Control

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

3.2.2.4 Personal Agents

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

3.2.2.5 Information Management tasks

- Searching for information;

- Information filtering;
- Information monitoring;
- Data source mediation.

3.2.2.6 Interface Agents / Personal Assistants

An interface agent is a program that is able to operate within a UI and actively assist the user in operating the interface and manipulating the underlying system. OMG defines the functionalities of this agent as:

- 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 play a large role in assisting users in data management;
- It can function as a bridge between domain knowledge about the data management systems and the user;
- To 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 (Lieberman et al., 1999). In addition, Microsoft is now including interface agents in its desktop products to watch the actions of users and make appropriate suggestions.

3.2.2.7 Nomadic Computing Applications

Current and future development in the area of wireless data communications and mobile computers enable to a great extent mobile computing. This is the domain of mobile agent coming into play.

3.2.3 Organizational Management: Case Study

In this application domain, MAS/MASS are described on the basis of social structures built from group roles, independently from the actual nature of the agents. The system is analyzed from the outside, and modeled as a framework, which is aimed at bringing particular insights into the interrelations between temporal, spatial and social scales of management process at different levels. The approach is based on AGR models discussed in section 1.2.5.2. Groups describe collective structures through behavior types

divided into various roles. Agents executing roles modulate these collective behaviors induced by collective dynamics.

One success case is the INRM (Integrated Natural Resources Management) by Abrami and et al (2002). In this application, three levels of interests of the organization are exemplified:

- At the global level, behavior types are defined by a set of roles and their relationship in the organizational levels;
- At the individual level, agents execute and interpret the roles they are playing;
- At the co-evolution level, the dynamics of organizational levels interact dynamically through the roles played by various agents.

In this application, they distinguish different types of groups, roles and relations, and instantiated during simulations. Farm Group, Irrigation System group and Individual Irrigation Organization Group describe water management with regarding to the farm, the irrigation system, and the individual irrigation coordination respectively.

This is a pure MAS case based on the AGR model. It works on issues concerning co-evolution of individual and collective dynamics within a system. However it lacks spatial aspects of the system, well-defined relationship between social and spatial structures is missing. For example, do social structures imply some boundaries in space, and how?

3.2.4 E-Commerce Automation (Negotiation): Case Study

E-Commerce application operates in a dynamic and distributed environment, dealing with a large number of heterogeneous information sources with evolving contents and dynamic availability. They typically rely on distributed and autonomous tasks for information search, fusion, extraction and processing, without centralized control. An E-Commerce scenario typically involves the following activities: identifying requirements, brokering products, brokering vendors, negotiating deals, or making purchase and payment transaction.

There are many MAS/MASS systems, which have been implemented for E-Commerce Automation (Odyssey and Voyager). But IBM Aglet provides a story as it meets major requirements for such an application.

An IBM Aglet is a Java-based autonomous software agent. As used here, a ***software agent*** is a program that can halt itself, ship itself to another computer on the network, and continue execution at the new computer. The key feature of this kind of software agent is that both its code and state are mobile.

3.2.4.1 Aglets are autonomous

Once you start, they decide where they will go and what they will do. They can receive requests from external sources, but each individual aglet decides whether or not to comply with external requests. Also, aglets can decide to perform actions, such as travel across a network to a new computer, independent of any external request.

Table 3.2 IBM Aglet platform features

Features	Descriptions
Supported platforms:	Windows, Solaris, AIX 4.x, MacOS
Supported languages:	Implementation on Java 100%, Agent programmed in Java and executed on server with JVM
Implemented standards	Began investigation into FIPA ACL issues
Mobility/Migration	Via Aglet object (or a proxy: AgletProxy). Migration is done with the object serialization of java. Object are marshaled then un-marshaled, and Aglet execution at destination
How many threads or processes are possible per agent	An active aglet is associated with one thread
Communication	TCP/IP protocol for global communication Message passing (synch, or asynch) Proxy objects are used for the communication with an aglet. All messages are sent to the proxies. The proxy transfers messages into the aglet. When the aglet is placed on the remote host, the proxies forward messages to the aglet via network. The aglet server who hosts the aglet retrieves messages and convert system messages into system events. The remains are sent to the aglet. At that time, security mechanism control messages pass or not.
How the communication partner can be addressed	A proxy object (AgletProxy) is used as a partner of a communication. And every aglet has its own identifier (AgletID). An AgletID object can be converted into an AgletProxy object in an aglet server with AgletContext#getAglteProxy(AgletID) method.
Security	Aglet Instance has identity, owner id, creation date and trust. It implements a security manager with network or file access control
Agent tracking	Logging facility is provided on the aglet server
Directory of services	A Finder as an experimental feature is provided
Type of code migration	Necessary classes are archived and transferred to the receiver. A Jar file is supported, all classes in a jar file are transferred as a bundle. Other classes are transferred on demand from code base server.
Other special features	Security control

3.2.4.2 Aglets Are Mobile

People use the term "software agent" to talk about more than just mobile agents. Two other meanings of the term are *intelligent agents* and *representatives*. Intelligent agents are endowed to some degree with artificial intelligence. They may or may not be mobile. A representative is a piece of software that represents you, like an attorney or an assistant. Representatives stand in for you in your absence. Depending on your instructions to them, representatives can make decisions or even consummate deals on your behalf. Representative agents can be mobile.

Aglets can potentially be endowed with artificial intelligence or serve as representatives, but they need not be either. Fundamentally, they are mobile agents: Java programs that can halt execution, travel across the network (with both code and state in tact), and continue execution at another host.

Besides the mobility, IBM Aglet agent has other capabilities, which are necessary when working in a particular E-Commerce environment. Here is a list of some of the more common applications:

Data Collection from Many Places

One of the main differences between mobile code, such as applets, and mobile agents is its itinerary. Whereas mobile code usually travels just from point A to point B, mobile agents have an itinerary and can travel sequentially to many sites. One natural application of mobile agents, therefore, is collecting information spread across many computers hooked to a network. An example of this kind of application in E-Commerce is a network backup tool that periodically looks at every disk attached to every computer in a network. Here, a mobile agent could roam the network, collect information about the backup status of each disk. It could then return and report to its point of origin.

Searching and Filtering

Given the ever-increasing amount of information available on the Internet and other networks, the activity of collecting information from a network often amounts to searching through vast amounts of data for a few relevant pieces of information. Filtering out the irrelevant information can be a very time-consuming and frustrating process. On behalf of a user, a mobile agent could visit many sites, search through the information available at each site, and build an index of links to pieces of information that match a search criterion. Searching and filtering exhibits an attribute common to many potential

applications of mobile agents: knowledge of user preferences. Although mobile agents do not have to be "representative" or "intelligent," they often are. Here, an agent is given knowledge of user preferences in terms of a search criterion and an itinerary, and sent out into the network on the user's behalf. It sifts through huge amounts of data for those pieces of information of particular interest to the user. At some point, it returns to the user to report its findings.

Monitoring

Sometimes information is not spread out across space (on the disks of many different computers hooked to the same network), but across time. New information constantly is being produced and published on the network. Agents can be sent out to wait for certain kinds of information to become available. For example, an agent could go to a stock market host, wait for a certain stock to hit a certain price, then buy some of it on behalf of its user. Another example is personalized newsgathering. An agent could monitor various sources of news for particular kinds of information of interest to its user, then report back when relevant information becomes available.

This kind of application highlights the asynchronous nature of mobile agents in an E-Commerce environment. If you send out an agent, you need not sit and wait for the results of its information gathering. You can program an agent to wait as long as it takes for certain information to become available. Also, you needn't stay connected to the network until an agent returns. An agent can wait until you reconnect to the network before making its report to you.

Targeted Information Dissemination

Another potential use of mobile agents is to distribute interactive news or advertising to interested parties. Unfortunately, this means mobile agents, like e-mail, can be used for Spam -- indiscriminate distribution of information, usually advertising.

Negotiating

Besides searching databases and files, agents can gain information by interacting with other agents. If, for example, you want to schedule a meeting with several other people, you could send a mobile agent to interact with the representative agents of each of the people you want to invite to your meeting. The agents

could negotiate and establish a meeting time. In this case, each agent contains information about its user's schedule. To agree upon a meeting time, the agents exchange information.

Bartering

Electronic commerce is another good fit for mobile agent technology. A mobile agent could do your shopping for you, including making orders and potentially even paying. For example, if you wanted to fly from Silicon Valley to an island in the South Pacific, an agent could visit databases of flight schedules and prices for various airlines, find the best price and time, make reservations for you, and pay with your credit card number. Electronic commerce also can take place *between* agents. For example, there could be an agent host dedicated to the buying and selling of automobiles. If you wanted to buy a car, you could give agent knowledge of your preferences, including a price range and potentially a negotiation strategy. You would send your agent to the dedicated host, where it would mingle and haggle with agents seeking to sell a car. If a potential match were found, your agent could report back to you, and you could contact each other in person to make the final arrangements. Alternatively, your agent potentially could consummate the deal on your behalf. If the opportunity is a good one, your agent may have only a few microseconds to act before someone else's agent buys the car.

Another critical requirement for E-Commerce is security. IBM Aglet is designed with strict control over the access to the mobile agent. Each Aglet instance is assigned unique agent ID, owner ID and trust degree. A security manager from the system can trace them and monitor the file access and state.

Two successful applications of IBM-Aglet in E-Commerce are the Tabican and Vertual Community. The Tabican is used to find a package tour or flight ticket through Internet (http://www.trl.ibm.co.jp/aglets/, 2002). The Vertual Community is used for shopping on-line.

There are limitations for IBM Aglet when apply to E-Commerce:

- Its' global communication follows asynchronous message passing, but synchronous messaging is often required in E-Commerce;
- A clear collaboration among agents is missing. The system consists of only two APIs and two implementation layers. When more agents are involved, they have to migrate through network;
- The strict security control reduces its performance and functionalities required by the application.

- Aglet agent is usually heavyweight agent since it carries all code required to perform the computing when the destination is reached.
- It has a lower compatibility since it is supported on platforms, which require JVM.

3.2.5 Information Processing: Case Study

3.2.5.1 Basic Requirements for Information Processing

Information searching

The amount of information available is huge over a corporate intranet. This 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

This is 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 may 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 affect 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 do not 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.

3.2.5.3 Concordia: Data Base Query and Information Processing

Concordia provides a general agent framework for mobile computing and data base systems. It can be used to build MASS for applications such as Remote Database Query, Smart Messaging, Groupware Manager and Information Retrieval. It follows client/server architecture and is composed of three basic components including Concordia server, JVM and Agent(s). Table 3.3 lists the major features.

There are three general applications in this information-processing domain:

Remote Database Query

This is implemented for querying data from a remote DB. Figure 3.2 sketches the structural diagram. A user agent actively collaborates with a query agent to access remote DB. The user agent sends requests, and query agent gets requests and communicates with the data base server. When information is retrieved, the query agent performs filtering, sorting and all other tasks based on the request and its own knowledge.

Smart Messaging

This is a Web Server service provider. The central part is a mail agent, which does all the management of email sending/receiving according to the user agent instruction, but performs its own task based on its knowledge.

Information Retrieval

It retrieves user information from web sites. User Web/Mail Agent gets orders from a UI (Web Server), and sorts the requests and performs the requests on behalf of the user.

Table 3.3 Concordia Agent Platform Features

Features	Descriptions
Supported platforms:	Windows, Solaris, Linux, HP/UX, AIX.
Supported languages:	Implementation on Java 100%, Agent programmed in Java and executed on server with JVM
Implemented standards	Began investigation into FIPA ACL issues
Mobility/Migration	Weak migration (but with multiple method entry points via Itinerary). Queue Manager (inbound and outbound) for reliable transport of agents Java object serialization scheme
How many threads or processes are possible per agent	In general use, Concordia will create 2 threads, one main thread for Itinerary execution and a second thread for asynchronous event delivery. Agents can create more threads if granted the proper security privilege.
Communication	TCP/IP protocol Light weight Agent Transport API (application embedded) Local communication mechanisms: Distributed Events and Agent Collaboration. Global communication mechanisms: Distributed Events and Agent Collaboration Data format: arbitrary object subclasses from Concordia base class.
How the communication partner can be addressed	Publish-subscribe type model. Receivers register interest in certain classes of events with central Event Manager. When event is posted, Event Manager forwards event to all registered listeners. Event may be sent directly to agent via its unique Agent ID. Concordia also supports group-oriented events in which messages can be transparently sent to all members of a group of cooperating agents.
Security	User identification Agent authentication, security and integrity Resource protection (Resource access control mechanism). Server configured access control list. Privileges are granted based on the identity of the user who launched the agent
Agent tracking	Home register via mobile agent debugger
Directory of services	Global and local using a string identifier. Global directory maintained by optional Directory Manager service.
Type of code migration	All supported: on demand from sending host, on demand from code server, all classes as a whole from sending host, all classes as a whole from code server.
Other special features	Encryption of agent during transit. Transactional message queuing for reliable agent delivery. Agent persistence. Graphical server remote administration tool. Graphical Agent Debugger. Agent collaboration framework (distributed information sharing and synchronization for a group of related agents. Agent compression during transit.

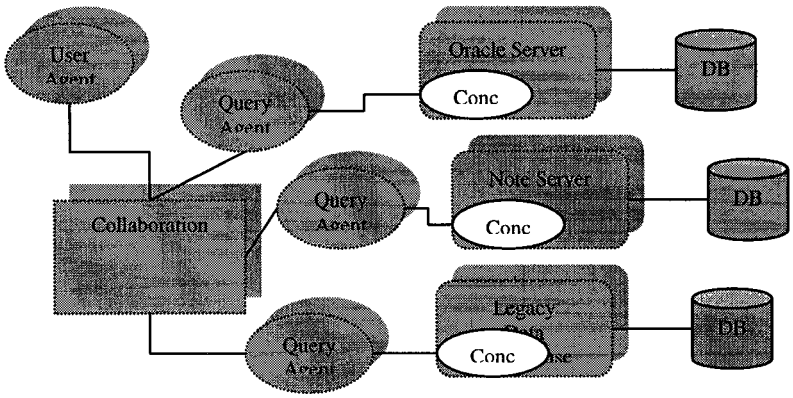

Figure 3.1 Concordia application: remote database query (After Mitsubishi, 1997)

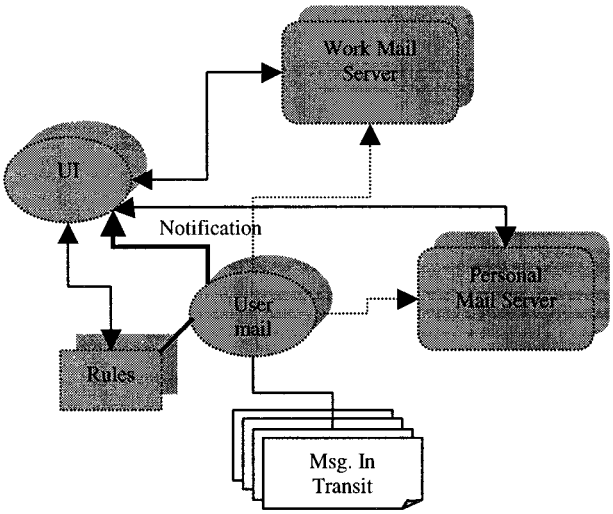

Figure 3.2 Concordia application: smart messaging (After Mitsubishi, 1997)

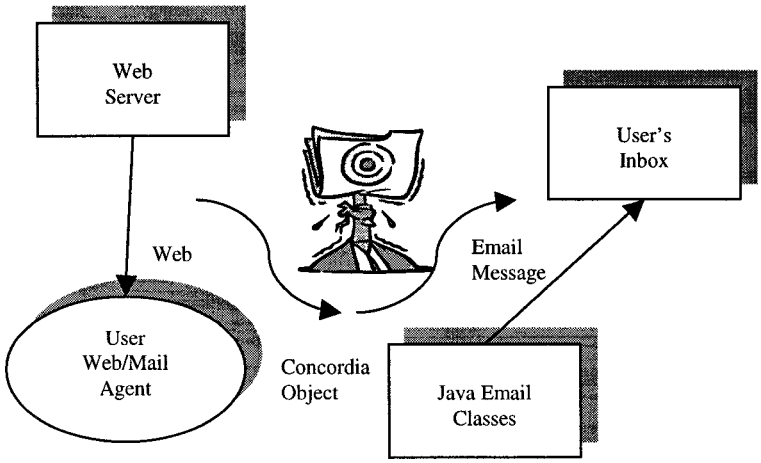

Figure 3.3 Concordia applications: information retrieval (After Mitsubishi, 1997)

3.2.6 Distributed Computation: Case Study

3.2.6.1 Basic Requirements

The environment of mobile computing is very different compared to traditional distributed systems. Performance measures such as bandwidth, latency, delay, error rate, interference, computing power, and quality of display may change dramatically as nomadic end-users move across the network such as from a wire line 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 rapidly. 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 range 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 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 —nomadic end-users are usually professionals in areas other than the computer industry. Current mobile computer systems are already very complex to use as a production tool. Hence, these users 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 to make the decision, and therefore adaptation is a cooperative 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:

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. 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 highly varying wireless data communications and to different kinds of mobile terminals. General requirements for distributed computing agent 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 nomadic applications:

FIPA has addressed nomadic applications. Technical Committee TCE - Nomadic Application Support has developed a specification (current status: draft, FIPA, 2003). It 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

Parallel processing:

Given that mobile agents can move from node to node and can spawn subagents, one potential use of mobile agent technology is to administer a parallel processing job. If a computation requires so much CPU time as to require its breaking up across multiple processors, an infrastructure of mobile agent hosts could be an easy way to get the processes out there.

3.2.6.2 AgentOS: for Mobile Agent System for Ubiquitous Computing

There is so far no MAS/MASS system built exclusively for this application. AgentOS could be mostly closed adaptable to this application domain. The AgentOS platform investigates applications in an environment designed for supporting agents in wide-area, heterogeneous networks. AgentOS has been developed to fulfill the basic requirements listed above. It focuses on the following:

- Network, distributed, and location transparency;
- Distributed, network-centric data and application hosting
- Replication and consistency management
- Flexibility, mobility, openness, and dynamism

The AgentOS framework provides features as:

- Agent execution model
- Agent management
- Mobile computing and data management service
- End-user capabilities.

With these design features, AgentOS is able to provide different service to an application:

- Agent context service
- AgentManager service
- AgentInfoCreator service
- Class factory service
- UUIDGen service
- Registry service
- Directory service
- Transport service
- Active kernel service
- Routing Directory service
- Agent-based mobile service

AgentOS has a well-defined research agenda. It has now completed it's fifth phase, which focuses on mobile computing applications. Distributed computing is one of them, and involves a collaborative multi-user software development environment.

AgentOs meets most but not all of the requirements from OMG for distributed computation. Missing features include availability, reliability, and support for disconnection.

3.2.7 Supply Chain: Case Study with ZEUS

This case study describes the implementation of an application that simulates the manufacture of Personal Computers (PCs). Building a PC involves the acquisition and integration or many components, usually from different manufacturers, thus providing an excellent example of a supply chain. The model consists of a keyboard, a monitor, a CPU (motherboard), and either a laser or inject printer only for

simplicity. Each participant in this model will be capable of performing specific tasks. Here are the participants: MakeComputer (Dell), MakeMonitor (TXCAN), MakeMotherBoard (Intel), MakePrinter (HP), MakeInkCartridge and MakeTonerCartridge (CartridgeCorp) (Figure 3.4). The whole design is realized using the ZEUS Agent Generator tool.

3.2.7.1 Application Analysis: Select Role Model

With a particular role model in mind this case associates roles to participants, starting with the standard *Zeus Application* role model from which it inherits.

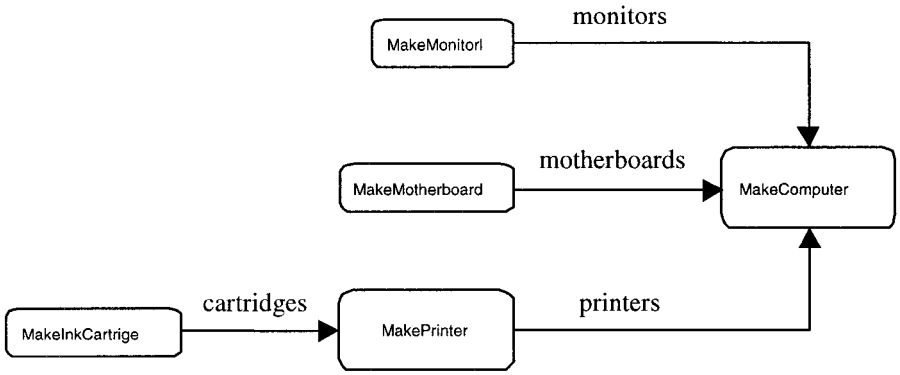

Figure 3.4. The participants of the PC supply chain (After Collis and Lee, 2001)

After deciding on the domain and considering its constituent role models several agents can be created to fulfill the roles found within the role model (Table 3.4):

Table 3.4 Role models for the ZEUS

Agent Name	Roles Played
Dell	SC Head (Negotiation Initiator, Consumer)
HP	SC Participant (Negotiation Initiator, Negotiation Partner, Supplier, Producer, Consumer)
Intel	SC Tail (Negotiation Partner, Supplier, Producer)
Taxan	SC Tail (Negotiation Partner, Supplier, Producer)
CartridgeCorp	SC Tail (Negotiation Partner, Supplier, Producer)

3.2.7.2 Agent Responsibilities

From Simple Supply Chain role model agent responsibilities can be obtained for the 2 constituent roles of an agent in the Supply Chain Head role. The responsibilities involved can be categorized as social or domain responsibilities, the former involving interaction with other agents, and the latter involving some local application-specific activity.

The next role to consider is the Supply Chain Participant, the role played by those agents that produce resources and consume resources from others.

The third role is the Supply Chain Tail, which is intended for agents that will supply resources without consuming resources that have been produced externally.

3.2.7.3 Application Design

The application design process consists of two phases. The first is an agent design phase where the roles assigned to each agent in the previous stage are refined by mapping each of the responsibilities to a generalized problem, and then choosing the most appropriate solution. The second stage of the design process is to model the declarative knowledge that will be used by the agent roles. This stage should result in the concepts inherent to the application (termed Facts within ZEUS), their attributes and possible values (also known as constraints).

ZEUS does not use OO design tools for the first stage design, but uses internal tool called Ontology Editor for the second design phase.

3.2.7.4 Application Implementation

ZEUS provides a number of tools for agent implementation as soon as the design phase is finished, i.e. the ZEUS Agent Generator. The implementation process combines the steps necessary to create a generic ZEUS agent with the steps necessary to implement the role-specific solutions identified during the previous phase.

The agent implementation consists of the following activities:

- Ontology Creation
- Agent Creation, for each task agent this consists of:

- Agent Definition;
- Task Description;
- Agent Organization;
- Agent Co-ordination;
- Utility Agent Configuration;
- Task Agent Configuration;
- Agent Implementation.

The purpose of these stages is to translate the design we have derived from the role models into agent descriptions that can be automatically created by the ZEUS Agent Generator tool.

During the agent creation, ZEUS tools specify Agent Definition Process, Task Description Process, Agent Organization Process, and Agent Co-ordination Process. These are the pattern processes, and it is applied to each agent to be created.

3.2.7.4 Agent Utility Implementation:

ZEUS provides Agent Utility that can be implemented to facilitate the property of agents created. ZEUS also provides Visualiser and the Society Viewer window to observe how the agents interact with each other. For the output results,_The Statistics tool may also be useful to plot price fluctuations over time, and the Report Tool is useful for monitoring the status of each of the tasks in the supply chain.

ZEUS provides a success SC case for applying to a agent. It works well for simple case, which involves only negotiation. The agent implemented is autonomous, but insufficient since Case Reasoning Engine is not provided.

3.3 Agent System Development Tools

3.3.1 AgentBean Development Kit (ADK)

ADK provides all the facilities to make an application become an agent. All you have to do is to select the AgentBean component from the Component Palette and drag it onto your application during the design phase. You must set ALL its properties in order for the AgentBean to be able to connect a facilitator. To connect the AgentBean to the Facilitator, you have to call its 'connect()' method, and to disconnect it, its

'disconnect()' method.

The AgentBean contains an AgentLib object, and will receive from it the 'doEvent(String KS, String func, String args)' notification. This notification can be passed to an application. The later has to register as a 'DoListener' for this AgentBean object. This interface comes with a unique method, 'doEvent(DoEvent e)', and the 'DoEvent' argument contains field filled with the values of the original notification from the Facilitator. The application gets a reference to this AgentLib using both 'getAgentLib()' or the 'lib' field.

Application Areas:

1) Network and Systems Management

The project current focus is on network and systems management. It has implemented agents that collect various data using SNMP (i.e. network load, uptime) and an agent that discovers the network topology. In the future it plans to implement components necessary for an agent to compute the placement of replica servers and try to find out the network topology without having access to a router's Management Information Base.

2) Computer Supported Cooperative Network

This work will be implemented as a distributed time planner using the ADK, which is an on-going project.

3.3.2 MadKit

MadKit is an agent-oriented middleware with which one can build distributed applications. One of the most interesting features of MadKit is that distribution is done transparently. One does not have to care whether an agent is remote or not. Madkit, is handling all the necessary communications and connections to make agents situated on different platforms work together transparently. A developer has to take care of: 1) the message passing between agents, 2) Run MadKit on the two remote kernels (let us call the computers toto.org and riri.net), and open a Communicator agent on both of them.

MadKit works on a 'peer to peer' basis, i.e. there is no "server" and "client". Every kernel is both a server and a client. MadKit takes care of all the low level communications. MadKit handles sockets, since RMI is good for client/server applications, but is much too slow to be used in peer-to-peer applications.

MadKit is running on Java SDK supported platform, and the source code is written in Java. It is possible that MadKit is embedded into a java application.

3.3.3 LEAP Project

The LEAP (Light Extensible Agent Platform) project is addressing the need for open infrastructures and services, which support dynamic, mobile enterprises. It developed agent-based services supporting three requirements of a mobile enterprise workforce: *Knowledge management* (anticipating individual knowledge requirements), *decentralized work co-ordination* (empowering individuals, coordinating and trading jobs) and *travel management* (planning and coordinating individual travel needs). Central to these agent-based services is the need for a standardized Agent Platform. Project LEAP developed an agent platform that is: lightweight, executable on small devices such as PDAs and phones; extensible, in size and functionality; operating system agnostic; mobile team management application enabling, supporting wired and wireless communications and FIPA compliant.

The LEAP project starts by defining application requirements as well as reviewing current FIPA standard. The design of LEAP is based on the development of an innovative, scaleable and "operating system agnostic" architecture for devices ranging from PDAs and phones to desktop systems. This architecture and an initial version of the LEAP application are integrated and deployed in Lab trials. The Lab trials assess run-time parameters, application usability, allow performance evaluations, and are used to steer operational use in field trials.

Actually, LEAP can run on any device that supports one of the following standardized versions of Java: J2SE, J2ME or Java (JDK 1.1.x). JADE-LEAP version 2.1 has already successfully been run on:

- J2SE devices (desktops, laptops);
- J2ME devices (POSE, Palm IIIc, Palm Vx, Motorola A008, Motorola i95cl);
- Java devices (Psion 5mx, iPAQ, Siemens SX45, Motorola HDT600).

The LEAP provides building blocks to easily and quicker develop agent-based applications supporting mobile workforces. There are three families of services: knowledge management, teamwork co-ordination and travel management. They run on top of JADE-LEAP.

Success stories: FIPA Agents now running on lightweight PDAs under J2ME. The LEAP project reached its first milestone (LEAP Version 1.0) by running FIPA compliant Agents on a PDA. LEAP has been tested and is running now on both: a PALM IIIc and a PSION 5MX through a wireless connection to the Internet. LEAP is the first FIPA Platform and Agents running on lightweight PDAs under J2ME.

The LEAP platform has been successfully run on a Motorola Accompli008 along with some agents. This is the first FIPA compliant agent platform ever run on a MIDP device.

HP announced that they ported LEAP to their Jordana PDA and Chai VM.

3.3.4 AgentBuilder

AgentBuilder is an integrated tool suite for constructing intelligent software agents. AgentBuilder consists of two major components - the Toolkit and the Run-Time System. The AgentBuilder Toolkit includes tools for managing the agent-based software development process, analyzing the domain of agent operations, designing and developing networks of communicating agents, defining behaviors of individual agents, and debugging and testing agent software. The Run-Time System includes an agent engine that provides an environment for execution of agent software.

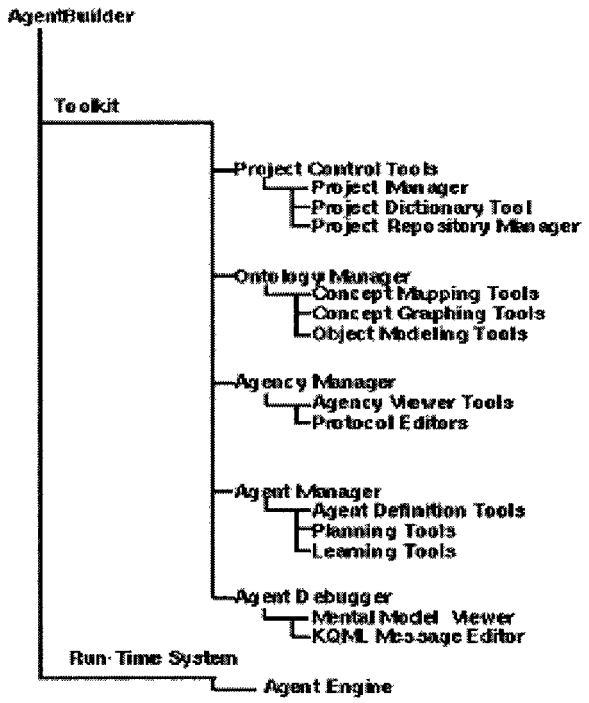

Figure 3.5 Structure of AgentBuilder

Agents constructed using AgentBuilder communicate using the Knowledge Query and Manipulation Language (KQML) and support the performatives defined for KQML. In addition, AgentBuilder allows the developer to define new inter agent communication commands that suit his particular needs. All components of both the AgentBuilder Toolkit and the Run-Time System are implemented in Java. This means that agent development can be accomplished on any machine or operating system that supports Java and has a Java development environment. Likewise, the agents created with the AgentBuilder toolkit are Java programs so they can be executed on any Java virtual machine. Figure 3.5 shows structure of AgentBuilder

AgentBuilder allows software developers with no background in intelligent systems or intelligent agent technologies to quickly and easily build intelligent agent-based applications. AgentBuilder reduces development time and development cost and simplifies the development of high-performance, robust agent-based systems. It supports Windows XP compatible, JAVA 1.3, CORBA and HP E-Speak/\.

3.3.5 JADE

JADE (Java Agent Development Framework) is a software framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middleware that claims to comply with the FIPA specifications and through a set of tools that support the debugging and deployment phase. The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another, as and when required. JADE is completely implemented in Java language and the minimal system requirement is version 1.2 of JAVA (the run time environment or the JDK). Recently, JADE has been integrated with the results of the LEAP project. The synergy between the JADE platform and the LEAP libraries yields a FIPA-compliant agent platform with reduced footprint and compatibility with mobile Java environments all the way down to J2ME-CLDC.

The goal of JADE is to simplify the development of multi-agent systems while ensuring standard compliance through a comprehensive set of system services and agents in compliance with the FIPA specifications: naming service and yellow-page service, message transport and parsing service, and a

library of FIPA interaction protocols ready to be used. The JADE Agent Platform complies with FIPA specifications and includes all those mandatory components that manage the platform, that is the ACC, the AMS, and the DF. All agent communication is performed through message passing, where FIPA ACL is the language to represent messages. The agent platform can be distributed on several hosts. Only one Java application, and therefore only one Java Virtual Machine (JVM), is executed on each host. Each JVM is basically a container of agents that provides a complete run time environment for agent execution and allows several agents to concurrently execute on the same host.

The communication architecture offers flexible and efficient messaging, where JADE creates and manages a queue of incoming ACL messages, private to each agent; agents can access their queue via a combination of several modes: blocking, polling, timeout and pattern matching based. The full FIPA communication model has been implemented and its components have been clearly distinct and fully integrated: interaction protocols, envelopes, ACL, content languages, encoding schemes, ontologies and, finally, transport protocols. The transport mechanism, in particular, is like a chameleon because it adapts to each situation, by transparently choosing the best available protocol. Java RMI, event-notification, HTTP, and IIOP are currently used, but more protocols can be easily added via the MTP and IMTP JADE interfaces. Most of the interaction protocols defined by FIPA are already available and can be instantiated after defining the application-dependent behavior of each state of the protocol. SL and agent management ontology have been implemented already, as well as the support for user-defined content languages and ontologies that can be implemented, registered with agents, and automatically used by the framework.

Basically, agents are implemented as one thread per agent, but agents often need to execute parallel tasks. Further to the multi-thread solution, offered directly by the JAVA language, JADE supports also scheduling of cooperative behaviors, where JADE schedules these tasks in a light and effective way. The run-time also includes some ready to use behaviors for the most common tasks in agent programming, such as FIPA interaction protocols, waking under a certain condition, and structuring complex tasks as aggregations of simpler ones. Among others, JADE also offers a so-called JessBehavior that allows full integration with JESS (Java Expert System Shell), where JADE provides the shell of the agent and guarantees (where possible) the FIPA compliance, while JESS is the engine of the agent that performs all the necessary reasoning.

The agent platform provides a Graphical User Interface (GUI) for the remote management, monitoring and controlling of agents, allowing, for example, to stop and restart agents. The GUI also allows creating and starting the execution of an agent on a remote host, provided that an agent container is already running. The GUI also allows controlling other remote FIPA-compliant agent platforms.

3.3.6 Java Agent Template (JAT)

The JAT provides a fully functional template, written entirely in Java, for constructing software agents, which communicate peer-to-peer with a community of agents, distributed over the Internet. Although portions of the code that define each agent are portable, JAT agents are not migratory but rather have a static existence on a single host. (However, using the Java RMI, JAT agents could dynamically migrate to a foreign host via an agent resident on that host). Currently, all agent messages use KQML as a top-level protocol or message wrapper. The JAT includes functionality for dynamically exchanging "Resources", which can include Java classes (e.g. new languages and interpreters, remote services, etc.), data files and information included in the KQML messages. JAT agents can be executed as either standalone applications or as applets via the appletviewer (browsers such as Netscape Navigator will not work without special extensions due to restrictions on networking and file IO). Both configurations support graphical and non-graphical agents. An Agent Name Server provides coordination. The architecture of the JAT was specially designed to allow for the replacement and specialization of major functional components including the GUI, low-level messaging, message interpretation and resource handling. Consequently, the JAT should be used as a platform for building a wide range of agents for different domain applications.

The JAT_0.3 includes the core JAT packages (JavaAgent.context, JavaAgent.agent and JavaAgent.resource). These packages support a remote service paradigm (RemoteService.context, RemoteService.agent, RemoteService.resource and ImageSelector). The JAT_03 also provides packages, which implement a number of demo examples. Specifically, the following directories should be found under the JAT_0.3/ directory:

- Classes: Contains all java and class files as well as some Makefiles.
- Scripts: C shell scripts for simplifying execution.
- Working: Contains directories in which agents can store files during execution, including logfiles.

- Files: Contains directories for files and classes, which will be shared between agents.
- API: Javadoc generated documentation.
- Applet: html files for applet execution.
- Documentation: html documentations.

3.3.8 Others FIPA Agent Platforms

The above tools are partially successful (at least to some extent), because they provide most of the services required by building a complex MAS/MASS.

Following FIPA standardizations, the following agent platforms have been developed as tools for building agent systems:

- Mecca (Siemens);
- FIPA-OS (Nortel);
- FIPA-Smart;
- CoABS.

These are actually not success cases due to limitations of integrating agents into standard computing environments, and they have faded away from the market place.

3.4 Discussions

There are many MAS/MASS developed, and more are coming to ride the current wave of popularity of agent-based system. Each claims that MAS/MASS technologies are going to change the way we live and work, and each wants to be the ones that provide the breakthrough system we all end up using.

3.4.1. Why They Are Successful: a Few Common Things

From samples given above, successful MAS/MASS have some things in common:

- Application/goal-driven in design. Particular agent system is always designed/built for specified application, which has the capabilities to accomplish the task assigned to the agents. For instance, AgentOS for distributed computing; Concordia for information processing. This may formalize an agent system, but on the other hand, make them less flexible;

- They have all been implemented in Java, and apply Java Remote Method Invocation (RMI) as the communication framework. RMI provides an RPC-like mechanism for Java processes, permitting Java objects to communicate through method calls across address spaces. The primary motivations for using RMI today are that it
    - Support multiple transport protocol (TCP/IP, HTTP, with others on the works);
    - Provides a basic object registry/bootstrap mechanism;
    - Flexibility of moving code, data or both, between address space, but not just data;
    - Still be able to communicate with standard ACL among the agents.
- Tools. Each successful MAS/MASS platform provides well-established tools for agent creation, system setting and customization for application. The building process of agent-based application is fairly simple:
    - The package is easy to install
    - The example agent applications are "flashy" (they have GUIs);
- Multiple platform support. This makes agent application running in a distributed, heterogeneous environment possible.
- Comprehensive documentation and example applications, which make systems easy to learn.

3.4.2 Why They Are a Few Miles Away from Perfect

1) Majority of MAS/MASS applies the communication paradigm evolving from artifacts of conventional client/server architecture, such as RPCs. These mechanisms rely on fixed, well-defined client and servers, and do not have enough flexibility to deal with fluidity of wide-area mobile networks.

2) They are incomplete in functionalities. Due to the limitations in supporting platform, implementation language and communication protocol standardization, no single MAS/MASS provide full functionalities required by application or has the ability to modify/add new functionalities. For instance, IBM Aglets defines a general-purpose mobile agent framework, but lacks essential functionalities such as merging and splitting, dynamic agent generation and spawning, and inter-agent communication. Similar incompleteness has also been seen in other MAS/MASS systems.

3). None of the current MAS/MASS meets completely the requirements made by OMG for specified application domain. OMG has made clear requirement set for MAS/MASS systems applicable for each application domain. For instance, for a MAS/MASS system built for E-Commerce, OMG has set specific requirements focused on these aspects:

- Data collection from many places;
- Information Searching and filtering;
- Information Monitoring;
- Targeted information dissemination;
- Agent Negotiating;
- Agent Bartering;
- Mobility.

Current MAS/MASS systems may emphasize a few of them, but not all. This is partly due to the fact that these requirements have been established only recently, and the OMG has seen the importance of standardization of agent systems. However this does not impose any constraints on each individual research group, and that needs coordination.

3.4.3 Performance and Optimization

Though agents are autonomous and mobile in most of MAS/MASS system, the agent itself might not be lightweight. This could be particularly true when it carries both code and data to travel across distributed, heterogeneous environments. It will significantly reduce system performance if an agent is required to perform a task that is code/CPU intensive, and there is no way for the code to be written in high-level language such as C/C++. One solution for this issue is to follow a coordination mechanism, called CP&CR mechanism, where problem constraints are partitioned by constraint type and constraint connectivity, and are assigned to different agents. Thus the agent could become light-weighted.

To make agent application working in a distributed/heterogeneous environment more efficiently (performance issue), two solutions are practically possible: make lightweight agent and simplify distributed constraint optimization. The first one has been limited by agents' property itself if they have to carry not only the code, but also the data for a complex task. The second one may have more advantage since it is

practically possible for almost any kind of agent system. This is often referred to as Distributed Constraint Optimization (DCOP) in an agent system.

DCOP imposes considerable complexity in the coordinated search for an optimal solution. MAS/MASS, in general, consists of a number of discrete subsystems and components that interact with each other through various interfaces to provide the required functionality. Designing such a system requires finding values for interface variables that are compatible among the various components, and is analogous to the constraint optimization problem, with subsystem and components playing the role of constraints among the variables. Although DCOP often introduces additional complexity in agent coordination, many agent applications often exhibit special structures that can be exploited to facilitate problem solving. There are a few research works concerning agent-based DCOP (Parunak, 2000, Liu and Sycara, 2001; Hannebauer and Muller, 1998) with focus on the scheduling of agent system. For instance, Liu and Sycara (2001) present a coordination mechanism for DCOP that takes advantage of disparity among subproblems to efficiently guide distributed local search for global optimality.

Although DCOP has been stressed in resolving the MAS/MASS system distributed computing performance, no clear progress has been seen yet. This could be a potential direction for future research.

3.4.4 Coordinative Agents

Since most of MAS/MASS involve multiple agents actively participating in a single task, collaboration among agents has become a challenging issue. Here is a simple example: A mobile agent A is deployed to place M, and mobile agent B has been deployed to a place N. At a time t, agent A sends a message from place M to agent B at place N. While waiting for response from agent B, agent A decides to move to place K at its own willing. When agent B finally sends response to place M, it cannot reach agent A, but the message wanders around, and the message might not be needed any more by agent A, since it could be either out of date or agent A is assigned a new task.

There are many practical use cases like this, but the current algorithm simply can't handle all of them. Some systems implement an agent tracing solution, which works together with agent collaboration. But we have not seen a substantial improvement yet. This will limit flexibility/applicability of agent system from many applications involving wide-area distributed system.

3.4.5 Is There Space for Further Improvement?

The answer is "Yes". The next generation global computer network is poised to become a ubiquitous medium for communication, collaboration, and personal information management, allowing access to personalized and collaborative computing services anywhere, through a variety of desktop and mobile computing devices. Such a vision can only be realized through further evolution of Internet and mobile computing technologies---in order to support component-oriented dynamic applications, scalable resource sharing, and large-scale group (multicast) communication, all for a large number of mobile and nomadic users. Deployment of a large-scale mobile and nomadic computing system requires a uniform treatment of these distributed systems issues, as opposed to the ad-hoc treatment that these issues receive today. That is, an operating system for wide-area mobile networks is necessary.

It is necessary to build a distributed operating system for wide-area mobile networks using the autonomous object, or mobile agent, programming paradigm. The mobile agent paradigm is an extension of distributed objects, exhibiting features such as active threading, run-time code mobility with autonomous navigation, and knowledge-based inter-agent communication. These characteristics favor a uniform implementation of essential mobile computing services such as multicast communication, intelligent fault-tolerant routing, proxy server/client handling, pessimistic and optimistic data replication management, and multi-level security models. Such services will enable the rapid construction and secure, scalable deployment of wide-are mobile and nomadic computing applications.

The first stage of potential research involves building a mobile agent system suitable for deployment in mobile networks. Fundamental issues in mobile agent systems include efficient agent migration, inter-agent communication (including intra-node, inter-node, and distributed group), agent behavior modeling and verification, and debugging support for mobile agent systems.

Chapter 4 Agent Past, Present and Future: A Brief Summary

What Are In This Section: *This chapter summarizes briefly knowledge we have gathered about agents, agent systems. We will have a brief look at SW agent history, present status, and future development directions. This brief summary aims at giving a broad view of the major progresses made during past a few decades. We divide this chapter into three sections: Section 4.1 discusses the agent history up to the year of 2002. In this section we focus on the major breakthrough made before 2002. Section 4.2 discusses the agent present status. In this section, we will look at the open issues for the agent platforms and critical questions the agent technology is facing. In section 4.3, we discuss the agent future with focus on major concerns and a few challenging issues. We will suggest a few key issues for agent future based on what we have learned now.*

The introduction of "Agent" concept addresses broad areas involving in both AI and SW Engineering societies. Agents can be implemented in many ways: Software, Network, People, Machine, Robotics, Traffic Control, Nuclear Reactor Control, Automated manufacturing, and so on. I cannot cover all of them, but just focus on the Software Agent, which are the primary focus from the AI and Software Societies. SW Agents can be used with many other technologies, and it has been an extremely active research field since the early 90's. It will remain to be so in the foreseeable future. The following figure shows how the "SW Agent" is applied in various application domains, and it applies to the past, current and future development (Figure 4.1). We have specified a few MAS/MASS in Chapter 3, and showed how these agent systems have been built and applied for particular applications domains. Just as OO-programming has replaced structured programming in the later 70s, autonomous SW Agent provides additional layers of abstraction that conventional OO does not address. "Agent" paradigm is growing, and we are interested to analyze if its impact will evolve further in software design and programming.

Agents exist in many different ways, and refer to software, network, people, machine robotics, traffic controller, nuclear reactor controller and etc. (Figure 4.2). Hence an agent can have different appearances and acquire different standardizations.

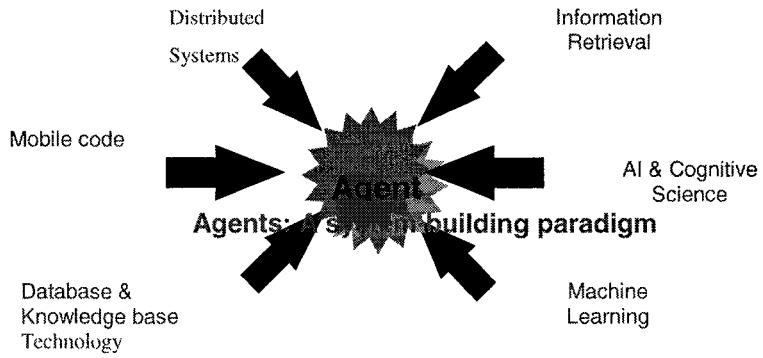

Figure 4.1 Agent paradigm and its application domains (modified from Labrou 2000)

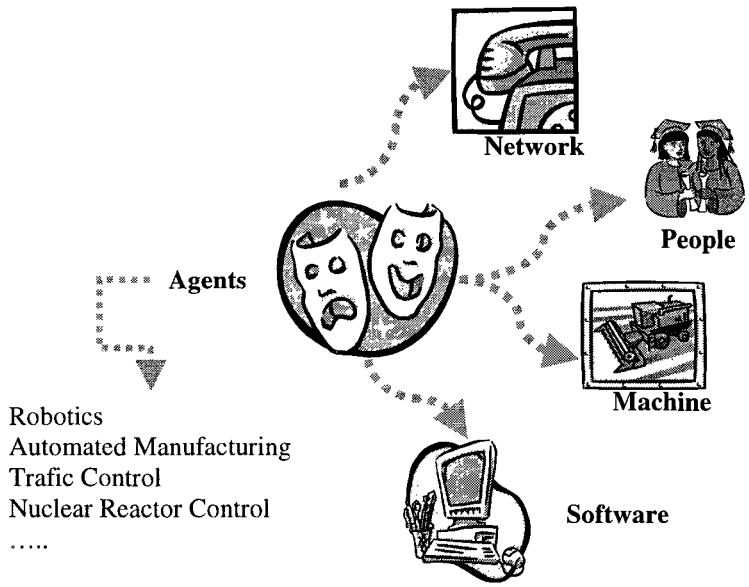

Figure 4.2 Agents can be implemented in many ways

4.1 Agent Past: ? - 2002

In SW engineering, we have experienced the following evolution:

1978 – 1990:	Structured Programming;
1982 – Now:	Object Oriented Programming;
Later 90s - ?:	Agent Paradigm Programming..

Agent paradigm was actually proposed a long time ago (back to 50's), but there was no common knowledge base for what is an agent between AI and SW Engineering societies until later 90s. The AI research emphasis is on the use of AI techniques to create software that performs information filtering and other autonomous tasks for users. "Intelligent" agents of this sort have been developed at Xerox PARC, at MIT's Media Lab, and by researchers at other companies and universities. These agents may or may not display any explicitly anthropomorphic features.

The SW Engineering research emphasis is on the agent as an interface metaphor that aids the user. These agents may or may not incorporate new AI techniques--their essential function is to act as effective bridges between a person's goals and expectations and the computer's capabilities. The agent metaphor is used to make the interface more intuitive and to encourage interactions that might be difficult to evoke with a GUI. Agents of this sort need not be explicitly anthropomorphic, although this is the arena in which the expressive qualities of 'characters' are being explored.

4.1.1 AI View of the Agents

The 1990's have seen the dawn of a new paradigm in computing - software agents in AI society. Many researchers were very active in this vibrant area, drawing from more traditional research within the artificial intelligence (AI) and human computer interaction (HCI) communities. Kay (1990) and others argue that the software agent systems have the potential to revolutionize computing as we know it, allowing us to advance from direct manipulation of systems to indirect interaction with agents. Removing the requirement for people to manage the small details of a task liberates individuals, empowering them to accomplish goals otherwise requiring experts. The next section describes some of the important landmarks that happened along the way to where we are today.

Alan Turing, famous for his work on computability (1937), posed the question "Can machines think?" (1950). His test, where a person communicates via a Teletype with either a person or a computer, became known as the Turing test. The Turing test requires a conversational computer to be capable of fooling a human at the other end. It is the Turing test that inspired the birth of the artificial intelligence community. The discipline of artificial intelligence (AI) was born in the 1950's. Minsky (1986), after some work with neural networks (deemed a failure at the time due to the difficulty of learning weights), teamed up with John McCarthy at MIT to work on symbolic search-based systems. At the same time at Carnegie-Mellon,

Allen Newell and Herbert Simon were successfully exploring heuristic search to prove logic theorems. Initial successes thus led to heuristic search of symbolic representations becoming the dominant approach to AI. The 1960's saw much progress. Then at Stanford, McCarthy (1960) had just invented LISP and set about representing the world with symbols, using logic to solve problems (1969, 1983). At the same time Newell (1959) created the General Problem Solver which, given a suitable representation, could solve any problem. Problems solved were in simple, noise and error free symbolic worlds, with the assumption that such solutions would generalize to allow larger, real world problems to be tackled. Researchers did not worry about keeping computation on a human time-scale, using the increases in hardware performance to constantly increase the possible search space size, thus solving increasingly impressive problems. During the 1970's, search became well understood. Symbolic systems still dominated, with continuing hardware improvements allowing steady, successful progress. Robots were created, for example Shakey Robot (Nilsson, 1984), that lived in special block worlds, and could navigate around and stack blocks sensibly. Such simplified worlds avoided the complexity of real world problems. The assumption, underpinning all the symbolic research, which simple symbolic worlds would generalize to the real world, was about to be found. Expert systems were created to try to solve real problems. McCarthy (1983) had realized that "common sense" was required in addition to specialized domain knowledge to solve anything but simple micro-world problems. A sub-field of AI, knowledge representation, came into being to examine approaches to representing the everyday world. Unfortunately the idea of "common sense" proved impossible to represent, and knowledge-based systems were widely viewed to have failed to solve real-world problems. At the same time, the back propagation algorithm caused a resurgence of interest in connectionist approaches, previously deemed a failure, and Minsky (1986) examined an agent-based approach for intelligence. The late 1980's and early 1990's saw the decline of search-based symbolic approaches. Brooks (1991) convincingly challenged the basic assumptions of the symbolic approaches, and instead created embodied, grounded systems for robots using the "world as its own best model". This bottom up approach was termed *nouvelle AI*, and had some initial successes. However, it too failed to scale up to real world problems of any significant complexity. Connectionist approaches were aided by new parallel hardware in the early 1990's, but the complexity of a parallel architecture led such systems to fail in the marketplace. Knowledge engineering, now widely seen as costly and hard to re-use, was superseded

by machine learning techniques borrowed from AI. Towards the end of the 1990's, pattern-learning algorithms (1997) could classify suitable domains of knowledge, such as news stories and examination papers, with as much accuracy as manual classification. Hybrids of traditional and nouvelle AI started to appear as new approaches were sought. The mid 1990's saw Negroponte (1970) and Kay's (1990) dream of indirect HCI coupled with Minsky's (1986) ideas on intelligence lead to the new field of agent-based computing. Experiments with interface agents that learnt about their user (1994), and multi-agent systems where simple agents interacted to achieve their goals (1995) dominated the research. Such agent systems were all grounded in the real world, using proven AI techniques to achieve concrete results (applying the maxim "a little AI goes a long way").

User modeling changed in the 1990's too, moving from the static hand crafted representations of the 1980's to dynamic behavior based models (1993), such as LCA and BCA (see Chapter 2). Machine learning techniques proved particularly adept at identifying patterns in user behavior. General application domains have been recognized during agent system development inside AI, which include:

- Auction/market domain. Typical samples are *Kasbah* (1996) and *Sardine* (2000);
- Believable/entertainment domain. *ACT* (1996) and *ALIVE* (1996) are typical samples. *Cathexis* (1997) is a believable agent with modeled emotions, as is the *Oz* project (1994).
- Email filtering domain. Typical samples are *MailCat* (1999), *MAGI* (1996), *Maxims (1994)*, *Re:Agent* (1998).
- Expert assistance domain. Typical samples are *Coach* (1994), *Eager* (1991), *GALOIS* (1997), *GESIA* (1998), *Open Sesame!* (1997).
- Matchmaking domain. Typical samples are: *ExpertFinder* (1998), *ReferralWeb (1997)* and *Yenta* (1997).
- Meeting schedulers. Typical samples are: *CAP* (1994) and *Meeting scheduling agent* (1993).
- News filtering domain. Typical samples are *ANATAGONOMY* (1997), *Butterfly* (1999), *IAN* (1996), *Krakatoa Chronicle* (1995), D*ude(1998)*, *NewsWeeder* (1995), *NewT* (1994) and Pannu's (1996).
- Recommender systems. Typical samples are *GroupLens* (1997), *PHOAKS* (1997), *Ringo* (1994), *Citeseer* (1997).

- Web domain. Typical samples are *AARON* (1996), *Amalthaea* (1997), *ARACHNID* (1998), *CiteSeer (1998)*, *Fab* (1997), *Jasper* (1996), *Letizia* (1996), *Personal WebWatcher* (1999), *WebMate* (1998), *WBI* (1997).
- Other domains:
    - *CIMA* (1996) is a text prediction agent, which suggests completions of sentences in a text editor. Heuristics learn from observed examples, hints and partial specifications.
    - *CILA* (1995) is an agent tested in an artificial, abstract domain. It tests constructive induction-based learning against AQ15c and selective induction.
    - *COLLAGEN* (1995) is an agent whose interaction style is modeled on human collaboration.
    - *Grammex* (1999) learns grammar (e.g. email structure) from user examples and performs actions when it detects new occurrences of this grammar.
    - *Mondrian* (1996) learns graphical operations, which are explicitly programmed by users.
    - *Softbot* (1994) plans internet-based actions from incomplete user goal specifications.
    - *Remembrance agent* (1996) suggests documents related to the user's current context.
    - *UCI GrantLearner* (1997) is a system to identify interesting grant funding opportunities.

All of these applications have been developed inside AI around middle to later 90s, and most of them have been developed based on agent intelligence. Figure 4.3 shows an agent system and applications from the AI society (AI agents).

	Character-based agent	Social agent	Recommender system	Monitor user behaviour	Explicit feedback	Initial training set	Programmed by user	Behavioural model	Knowledge-based model	Stereotypes
ACT	o			o				o		
ALIVE	o									
Cathexis	o									
CAP		o		o				o		
COLLAGEN		o								
Do-I-Care		o			o	o				
ExpertFinder		o		o				o		
Kasbah		o					o			
Maxims		o		o			o	o		
Meeting scheduling agent (Maes)		o		o	o			o		
Sardine		o					o			
Yenta		o				o				
GroupLens			o		o					
PHOAKS			o	o						
Ringo			o		o					
Siteseer			o			o				
Referral Web			o			o				
Fab			o	o	o					
AARON				o	o			o		
Adaptive website agent (Pazzani)				o		o		o		
Amalthea				o	o			o		
ANATAGONOMY				o	o			o		
CEA				o	o	o		o		
CIMA				o				o		
Coach				o					o	o
Eager				o				o		
GALOIS				o				o	o	o
GESIA				o				o		
Jasper				o	o			o		
Kitakata Chronicle				o	o			o		
Letizia				o				o		
LAW				o	o			o		
Let's Browse				o		o		o		
MAGI				o	o			o		
Margin Notes				o		o				
NewsDude				o		o		o		
Open Sesame!				o				o		
Personal WebWatcher				o				o		
Remembrance agent				o				o		

	Character-based agent	Social agent	Recommender system	Monitor user behaviour	Explicit feedback	Initial training set	Programmed by user	Behavioural model	Knowledge-based model	Stereotypes
Sentence completer (Schlimmer)				o				o		
Travel assistant (Wuszkiewicz)				o	o			o		
WebACE				o				o		
WebMate				o	o			o		
WebWatcher				o		o		o		
WBI				o				o		
ARACHNID					o	o				
IAN					o					
Learning personal agent (Sycara)					o	o				
LIRA					o			o		
NewsWeeder					o	o		o		
NewT					o	o	o	o		
Syskill & Webert					o	o		o		
MailCat					o	o				
ReAgent					o	o				
UCI GrantLearner					o	o		o		
Butterfly						o				
CiteSeer						o				
Grammex							o			
Meeting scheduler (Haynes)							o			
Mondrian							o			
Softbot							o			
SARD									o	o

Figure 4.3 Agent system / application classifications

4.1.2 SW Engineering View of the Agents

Agent software is a rapidly developing area of research in SW engineering. However, the overuse of the word 'agent' has tended to mask the fact that, in reality, there is a truly heterogeneous body of research

being carried out under this banner. SW engineering gives a typology of agents, places agents in context, defines them and then goes on, inter alias, to overview critically the rationales, hypotheses, goals, challenges and state-of-the-art demonstrators of the various agent types in the typology in SW engineering. Hence, it attempts to make explicit much of what is usually implicit in the agent's literature. SW engineering also proceeds to overview some other general issues which pertains to all types of agents in the SW engineering. SW engineering view of agents contains some strong opinions that are not necessarily widely accepted by the AI agent community. The SW engineering society adapted the "agent" concept to MAS, which is "Autonomous". It was, later on, introduced into a concept of Mobile Agent" for moving code around in a network environment. The MASS has been mainly a research area in the early 90s, but became popular in the late 90s.

Software agents have evolved from multi-agent systems (MAS), which in turn form one of three broad areas, named DAI (Distributed Artificial Intelligence), the other two being Distributed Problem Solving (DPS) and Parallel AI (PAI). Hence, as with multi-agent systems, they inherit many of DAI's motivations, goals and potential benefits. For example, thanks to distributed computing, software agents inherit DAI's potential benefits including modularity, speed (due to parallelism) and reliability (due to redundancy). It also inherits those due to AI such as operation at the knowledge level, easily maintenance, reusability and platform independence (Huhns and Singh, 1994). The concept of an agent in SW engineering can be traced back to the early days of research into DAI in the 1970s - indeed, to Carl Hewitt's concurrent Actor model (Hewitt, 1977). In this model, Hewitt proposed the concept of a self-contained, interactive and concurrently executing object, which he termed 'actor'. This object had some encapsulated internal state and could respond to messages from other similar objects: an actor "is a computational agent which has a mail address and a behavior. Actors communicate by message-passing and carry out their actions concurrently" (Hewitt, 1977). Broadly, SW engineering society tends to split the research on agents into two main strands: the first (MAS) spanning the period 1977 to the current day, and the second (MASS) from 1990 to the current day too.

MAS works on smart agents, which begun in the late 1970s and all through the 1980s to the current day, concentrated mainly on deliberative-type of agents with symbolic internal models. We type these as collaborative agents as in Chapter 3 from the FIPA standards. A deliberative agent is "one that possesses an

explicitly represented, symbolic model of the world, and in which decisions (for example about what actions to perform) are made via symbolic reasoning" (Wooldridge, 1995).

Initially, MAS work concentrated on *macro* issues such as the interaction and communication between agents, the decomposition and distribution of tasks, coordination and cooperation, conflict resolution via negotiation, etc. Their goal was to specify, analyze, design and integrate systems comprising of multiple collaborative agents. These include classical work such as the actor model (Hewitt, 1977), MACE (Gasser *et al.*, 1987), DVMT (Lesser and Corkill, 1981), MICE (Durfee and Montgomery, 1989), MCS (Doran *et al.*, 1991), the contract network coordination approach (Smith, 1980; Davis and Smith, 1983), MAS/DAI planning and game theories (Rosenschein, 1985; Zlotkin and Rosenschein, 1989; Rosenschein and Zlotkin, 1994). These 'macro' aspects of agents as Gasser (1991) terms them, emphasize the *society* of agents over *individual* agents, while *micro* issues relate specifically to the latter. In any case, such issues are well summarized in Chaib-draa *et al.* (1992), Bond and Gasser (1988) and Gasser and Huhns (1989). More recent work under this strand includes TÆMS (Decker and Lesser, 1993; Decker, 1995) DRESUN (Carver *et al.*, 1991; Carver and Lesser, 1995), VDT (Levitt *et al.*, 1994), ARCHON (Wittig, 1992; Jennings *et al.*, 1995). Note that game theoretic work should arguably *not* be classified as a macro approach. It may likely move to the micro end of the spectrum. In addition to the macro issues, more work has also been characterized by research and development into theoretical, architectural and language issues. In fact, such works evolve naturally, though not exclusively, from the investigation of the macro issues. This is well summarized in Wooldridge and Jennings (1995a), and in the edited collections of papers: Wooldridge and Jennings (1995b) and Wooldridge *et al.* (1996). However, since 1990 or thereabouts, there has evidently been another distinct strand to the research and development work on software agents - the range of agent types being investigated is now much broader. This is the MASS: "Mobile Agent System", which we have discussed in detail in Chapters 2 and 3.

Some cynics may argue that MAS arises because everybody is now calling everything an agent, thereby resulting, inevitably, in broadness. We sympathize with this viewpoint; indeed, it is a key motivation for us - to overview the extensive work that goes under the 'agent' banner. In addition to investigating macro issues and others such as theories, architectures, languages, there has also been an unmistakable trend towards the investigation of a broader range of agent types or classes.

Part of the reason for answering the broadness of agents in SW engineering is that the diversity of interests from industries and universities in pursuing agent technology. It includes small non-household names (e.g. Icon, Edify and Verity), medium-size organizations (e.g. Carnegie Mellon University (CMU), General Magic, Massachusetts Institute of Technology (MIT), the University of London) and the real big multinationals (e.g. Alcatel, Apple, AT&T, BT, Daimler-Benz, DEC, HP, IBM, Lotus, Microsoft, Oracle, Sharp). Clearly, these companies are by no means completely homogeneous, particularly if others such as Reuters and Dow Jones are appended to this list. The scope of the applications is even more impressive: it really does range from the mundane (strictly speaking, not agent applications) to the moderately 'smart'. Lotus, for example, will be providing a scripting language in their forthcoming version of Notes, which would allow users to write their own individual scripts in order to manage their e-mails, calendars, and meetings. This is based on the view that most people do not really need 'smart' agents. Towards the smart end of the spectrum are the likes of Sycara's (1995) *visitor hosting system* at CMU. In this system, "task-specific" and "information-specific" agents cooperate in order to create and manage a visitor's schedule to CMU.

To achieve this, first of all, the agents access other on-line information resources in order to determine the visitor's areas of interest, name and organization and resolve the inevitable inconsistencies and ambiguities. More information is later garnered including the visitor's status in her organization and projects she is working on. Second, using the information gathered on the visitor, they retrieve information (e.g. rank, telephone number and e-mail address) from personnel databases in order to determine appropriate attendees (i.e. faculty). Third, the visitor-hosting agent selects an initial list of faculty to be contacted, composes messages, which it dispatches to the calendar agents of these faculties, asking whether they are willing to meet this visitor and at what time. If the faculty does not have a calendar agent, an email is composed and dispatched. Fourth, the responses are collated. Fifth, the visitor-hosting agent creates the schedule for the visitor, which involves booking rooms for the various appointments with faculty members. Naturally, the system interacts with the human organizer and seeks her confirmation, refutations, suggestions and advice. Most would agree that this demonstrator is pretty smart, but its 'smartness' is derived from the fact that the 'value' gained from individual stand-alone agents *coordinating* their actions by working in *cooperation*, is greater than that gained from any individual agent. This is *where* agents

really come into their existence. More examples of applications are described later but application domains in which agent solutions are being applied to or investigated include workflow management, network management, air-traffic control, business process re-engineering, data mining, information retrieval/management, electronic commerce, education, personal digital assistants (PDAs), email, digital libraries, command and control, smart databases, scheduling/diary management, etc. Indeed, as Guilfoyle (1995) note "in 10 years time most new IT development will be affected, and many consumer products will contain embedded agent-based systems". The potential of agent technology has been much hailed, e.g. a 1994 report of Ovum's, a UK based market research company, is titled "Intelligent agents: the new revolution in software" (Ovum, 1994). The same firm has apparently predicted that the market sector totals for agent software and products for USA and Europe will be worth a substantially higher number by the year 2000 in contrast to an estimated 1995 figure (Guilfoyle, 1995). Moreover, as King (1995) notes telecommunications companies like BT and AT&T are working towards incorporating smart agents into their vast networks. Entertainment, e.g. television, and retail firms would like to exploit agents to capture our program viewing and buying patterns respectively. Computer firms are building the software and hardware tools and interfaces which would harbor numerous agents. Reinhardt (1994) reports that IBM plans (or may have already done) to launch a system, the IBM Communications Systems (ICS), which would use agents to deliver messages to mobile users in the form they want it, be it fax, speech or text, depending on the equipment the user is carrying at the time, e.g. a PDA, a portable PC or a mobile phone. At BT Laboratories, they have also carried out some agent-related research on a similar idea where the message could be routed to the nearest local device, which may or may not belong to the intended recipient of the message. In this case, the recipient's agent negotiates with other agents for permission to use their facilities, and takes into consideration issues such as costs and bandwidth in such negotiations (Titmuss *et al.*, 1996). At MIT, Pattie Maes' group is investigating agents that can match buyers to sellers or which can build coalitions of people with similar interests. They are also drawing from biological evolution theory to implement demonstrators in which some user only possesses the 'fittest' agents: agents would 'reproduce' and only the fittest of them will survive to serve their masters; the weaker ones would be purged.

To reach a common base for placing *existing* agents into different agent classes, SW engineering society investigates a agent typology (Nwana, 1996). A typology refers to the study of types of entities. There are several dimensions to classify existing software agents.

Firstly, agents may be classified by their mobility, i.e. by their ability to move around some network. This yields the classes of *static* or *mobile* agents.

Secondly, they may be classified as either *deliberative* or *reactive*. Deliberative agents derive from the deliberative thinking paradigm: the agents possess an internal symbolic reasoning model and they engage in planning and negotiation in order to achieve coordination with other agents. These agents on the contrary do not have any internal symbolic models of their environment, and they act using a stimulus/response type of behavior by responding to the present state of the environment in which they are embedded (Ferber, 1994).

Thirdly, agents may be classified along several ideal and primary attributes, which agents should exhibit. BT Labs have identified a minimal list of three: autonomy, learning and cooperation. These three minimal characteristics derive four types of agents, which should be derivable from the basic agents listed in the Figure 1.1:

- *Collaborative agents;*
- *Collaborative learning agent;*
- *Interface agents;*
- *Truly smart agents.*

However, these distinctions are *not* definitive. For example, with collaborative agents, there is more emphasis on cooperation and autonomy than on learning. Hence, we do not imply that collaborative agents never learn. Likewise, for interface agents, there is more emphasis on autonomy and learning than on cooperation. Some SW engineering people do *not* consider anything else which lie outside the 'intersecting areas' to be agents. For example, most expert systems are largely 'autonomous'. But they usually do not cooperate or learn. Ideally, in our view, agents should do all three equally well, but this is an *aspiration* rather than the reality. Truly smart agents do not yet exist. Maes (1995a) noted that "current commercially available agents barely justify the name", but alone the word 'intelligent'. Foner (1997) is even more incandescent. Though he wrote this in 1997, it is still applicable today: "... I find little justification for most

of the commercial offerings that call themselves agents. Most of them tend to excessively anthropomorphize the software, and then conclude that it must be an agent because of that very anthropomorphization, while simultaneously failing to provide any sort of discourse or "social contract" between the user and the agent. Most are barely autonomous, unless a regularly scheduled batch job counts.

In principle, by combining the two constructs so far (i.e. static/mobile and reactive/deliberative) in conjunction with the agent types (i.e. collaborative agents, interface agents, etc.), one could have static deliberative collaborative agents, mobile reactive collaborative agents, static deliberative interface agents, mobile reactive interface agents, etc. But these categories, though quite a mouthful, may also be necessary to further classify existing agents.

To summarize, SW engineering groups identify seven types of SW agents to date:

- Collaborative agents
- Interface agents
- Mobile agents
- Information/Internet agents
- Reactive agents
- Smart Agents

These agents are more application specific, and should fit into the application agent abstraction level proposed in Figure 1.1.

There are some applications, which combine agents from two or more of these categories, and they are referred as heterogeneous agent systems. Such applications already exist even though they are relatively few. Another issue (for completeness sake) is that agents need not be benevolent to one another. It is quite possible that agents may be in competition with one another, or perhaps quite antagonistic towards each other. However, we view competitive agents as potential subclasses of all these types. That is, it is possible to have competitive collaborative-type agents, competitive interface agents, competitive information agents, etc.

4.1.3 Historical Milestone

1996:

- KQML was used in building agent system, which was a language and protocol for exchanging information and knowledge.
- Buschmann et al, proposed a layered agent architecture, an agent layered model was proposed based on the agent behavior;
- FIPA worked to develop and promote standardization in the area of agent interoperability since this year.
- ACL: Service language was a new class of language designed to facilitate service description, service discovery, and application data exchange. Examples include:
    - toolTalk (DEC circa '96)
    - E-speak (HP)
    - BizTalk (Microsoft et. al.)

1997:

- JAFIMA agent platform was built on the agent layered architecture by Kendall et al.
- Approaches to sharing objects in a distributed system evolved over the past 15 years.
    - CORBA
    - Distributed Computing Environment (DCE) developed by the Open Group in the early 90's
    - Java
    - Jini
    - RMI
    - Enterprise Java Beans (EJB).
    - OLE/COM/DCOM/ActiveX (Microsoft).
    - SOAP: The Simple Object Access Protocol.
- First International Autonomous Agent Conference

- OMG recommended standards for agent technology where appropriate – particularly the OMG's Object Management Architecture (OMA)

1998:

- Second International Conference On Autonomous Agent.
- Agent Standardization: US DARPA had several research programs that addressed various aspects of agent technology. These included: Control of Agent-based System; Advanced Logistics Projects; DARPA Agent Makeup Language.
- Agent Standardization: CLIMATE (Cluster for International Mobile Agents for Telecommunication Environments. It represented a pool of agent-related projects within the European Union. CLIMATE was formed in Spring 1998, and currently comprises of 14 core projects, and investigates various application areas, such as service control in fixed and mobile networks, telecommunications management, E-Commerce, and multimedia applications.

1999:

- Mobile Agent and Mobile Agent System formalized.
- FIPA published standard requirements for FIPA Agent Platform, which contained four core components: Agent Software (A), Agent Management System (AMS), Directory Facilitator (DF) and Agent Communication Channel (ACC). IIOP was used as transport protocol.
- Over 60 MAS/MASS systems finished beta version, and put into commercial running. These included: AgentSpace, Ajanta, Concordia, and others (see Table 3.1)
- More Agent development platform were built and put into practical use. These included JAE, JavaNetAgent and others. (see Table 3.1).

2000:

- Agent Standardization: AgentLink. It was Europe's ESPRIT-fund network of excellence for agent-based computing. It was a coordinating organization for research and development activities in the area of agent-based computer systems aimed at raising the profile, quality, and industrial relevance of agent system in Europe.

- Over 20 MAS/MASS systems finished beta version, and put into commercial running. These included: Aglet, CBorg, and others. (see Table 3.1)
- More Agent development platforms were built and put into practical use. These included DynamicTAO, IMJA and others. (see Table 3.1).

2001:

- Aalaadin AGR model was proposed for agents system;
- Over 10 MAS/MASS systems finished beta version, and put into commercial running. These included: JA-Title, TAgents, and others. (see Table 3.1)
- MiLog was finished and put into the practical use. This was the first MASS utilizing HTTP based inter-agent communication mechanism for information processing (see Table 3.1).
- The second version of AgentSpace was released.

2002

- First Intelligent Agent and SW Engineering Joint International Conference;
- An AGR (Agent-Group-Role) model was proposed, and a first AGR based modeling framework was built for a participative water management support. This was the first agent application in an organization-modeling framework.
- The mobile version of AgentSpace was released.
- Another about 10 MAS/MASS systems were released including Agent-Tcl, AgentOS, and others.

4.2 Agent Present

Agent paradigm seems to have potential in almost everywhere in SW engineering, but it is still far away from any application that shows reasonable industry strength. If agents are going to matter, they must first be easily integrated into a standard computing environment. This includes:

- Integrated into standard programming languages
- Able to interact with standard services (e.g. LDAP servers)
- Easily integrated into applications that matter (e.g., SAP, MS Office apps, etc.)

- Able to work well on a heterogeneous network environment. The current agent platform are still facing many issues in web browsers and web servers.

4.2.1 Open Issues for Agent Paradigm

4.2.1.1 Protocol Design for Complex Interactions in Multi-Agent Systems

We will require a generic approach or protocol engineering to the specification, analysis, and verification when several agents are involved. This approach should be three folds:1) Starting from semi-formal specification by means of Protocol Diagrams (e.g. AUML), both formal specification of interaction protocols and their verification are allowed; 2) Debugging and qualitative analysis: Interactions are based on distributed observations associated with true concurrency semantics (i.e.CPN unfolding) and ; 3) CPN formalism is extended to Recursive CPN (RCPN) with abstraction in order to deal with open protocols. The main interest of abstraction is the design of flexible protocols giving agents more autonomy during interaction. In addition, abstraction allows concise modeling and easier verification.

4.2.1.2. The Design of Agent Communication Languages

At the agent level, the collaboration among the agents is realized through ACL. In implementing an application, one should convince oneself that neither (1) a database approach nor (2) a distributed object approach is appropriate. The open issue is to allow agents talk freely (agent conversation). It defines allowed/useful/desirable sequences of messages for particular tasks and indicates what messages may fit best. The following should be considered:

- Allow more intuitive and convenient method for handling messages in context.
- Through conversation composition, scale to different levels of conversation granularity.
- Provide conversation management independent of agent implementation.
- Facilitate communication through conversation sharing.
- Both KQML and FIPA ACL include specifications for conversations (or conversation protocols).
- Conversations are not part of the semantic definition of the ACL.
- Conversations focus on an agent's observable behavior.
- Programmers might find conversations more useful than formal semantics.

The meaning of primitives is often context/situation dependant and conversations can accommodate context. Modern ACLs are powerful enough to:

- Encompass several different ways to achieve the same goal
- Achieve several different goals with the same message.

4.2.1.3. Scalability, Stability, and Robustness: Performance Issues

These issues have not been addressed, but are important for the success of agent-based systems. Empirical investigations need to be carried out to establish suitable minimum levels of performance and, clearly, these systems have to be elaborated. Alternatively, their stability would need to be proven formally. Though these issues are *non-functional*, they are crucial nonetheless.

Agents in deployed multi-agent systems monitor other agents to coordinate and collaborate. However, as the number of agents monitored is scaled up, two key challenges arise: (i) the number of monitoring hypotheses to be considered can grow exponentially in the number of agents; and (ii) agents become physically and logically unconnected (unobservable) to their peers. We need to examine these challenges in teams of cooperating agents, focusing on a monitoring task that is of particular importance to robust teamwork: detecting disagreements among team-members. This is not a well-addressed issue, although a few highly scalable disagreement-detection algorithms have been discussed to guarantee sound detection in time linear in the number of agents. Clearly this is one area that needs to be explored analytically and empirically before large scale agent systems can succeed.

4.2.1.4. Separation Between SW Engineering (Multiagent Systems) and the AI Agent Community

The distance between SW engineering and Al agent community in their agent research has been narrowed a bit. But there is not enough direct coupling between these two communities until more recently. They speak different languages, set different goals, and have different ways of looking at these "Agents". It is perhps the time for both the communities to merge their ideas, concepts and development appraoches. Al agent system builders could be using some of the many MAS/MASS systems as an agent deployment mechanism. MAS/MASS system builder could then learn what is needed to support in a mobile agent

system. For example, MAS/MASS might have a lot of applications and be useful without them necessarily being intelligent. On the other hand, some of AI agents do not think much about mobility, so they do not even think about the particular kinds of problems that MAS/MASS might have in being autonomous.

4.2.1.5. Engineering the Construction of Collaboration in an Agent System

The Pleiades system is a good step in this direction but there is much more research to do. We must move away from point solutions to point problems, and design methodologies/meta-tool, which allow for quicker implementation of collaborative agent-based systems.

4.2.1.6. Inter-Agent Coordination

This is a major issue in the design of these agent systems. Coordination is essential to enabling groups of agents to solve problems effectively. Without a clear theory of coordination, anarchy or deadlock can set in easily in collaborative agent system. Furthermore, should agents be totally truthful when negotiating with others or should they be allowed to 'lie' when it suits them? Coordination is also required due to the constraints of resource boundedness and time. Much experimental and/or formal work is still required to address these issues of coordination and negotiation.

4.2.1.7 Legacy Systems:

The thorny issue of what to do with legacy systems is still with us and will always be a problem. Established techniques and methodologies for integrating agents and legacy systems are still required.

4.2.1.8. How Do These Systems Learn?

Would learning not lead to elaboration? What are the appropriate architectures for different types of problems? How do you ensure an agent does not spend much of its time learning, instead of participating in its setup?

4.2.2 Evaluation of Collaborative Agent Systems

This problem is still outstanding. How are they verified and validated to ensure they meet their functional specifications? Are unanticipated events handled properly? How else would you trust such systems to run power stations, nuclear installations and chemical plants?

4.2.3 Mobile Agents and Mobility

Mobile agents are software abstraction that can migrate across the network (hence mobile) representing users in various tasks (hence agents). This is a contentious topic that attracts many researchers and repels the same number of others. The ideas of mobile abstractions are probably as old as distributed systems, first appearing in distributed operation system, such as MOSIX, V Kernel, Sprite, and Mach some 10 years ago: each of them had a process-migration implementation. Unfortunately, these researches did not lead to industrial adoption in these features.

The term "Mobile Agent" was first introduced by Telescript, which supported mobility at the programming language level. Many mobile agent systems followed, most implemented in Java, but also in scripting languages, to supporting mobile code. This is the original idea of designing a mobile agent.

Mobile Agents seem now following the same path. However it is still too early to conclude if it would fail. In a brief summary, mobile agents have raised considerable interest in research (Agent Tcl, Tacoma, and Mole...) and in industry (Aglets, Concordia, Jumping Beans...). These prominent application domains have emerged:

- Data-intensive applications where the data is remotely located and owned by a remote service provider. This includes E-Commerce, Information Retrieval, and Distributed Computing.
- Disconnected computing, such as laptops and PDAs, or extensible servers.
- Dynamic deployment of software in heterogeneous environment. One good example could be the System Administration and Management (in a network environment), extending a mobile agent out to a remote device to do the work for a user.

However, we are facing a few critical issues:

- Application domains have not recorded big successes. There is no MASS deployed in an industrial setting.
- How do MASS relate to other environment/infrastructures that MASS could either contribute to or benefit from? These include OS migration, middleware environments (Jini, DCOM, CORBA), networking (active networks, for example), mobile computing.
- Justification of mobile agents with respect to performance, scalability, compatibility, manageability, fault isolation, and so on.

- Better understanding of what should be mobile and when. What should be migrated: data, code or agent context (for example thread state transfer, which decides the weak/strong mobility of a mobile agent)? The current trend favors process migration, including the original code. This could produce a serious problem for both the parent and the child nodes. A better idea may be to move only the agent context: or in a simple word, move whatever it must be moved, but not all the data and code.

Among these issues, the agent mobility could be the most essential. Is mobility intrinsically a hard problem to solve, or is it just at its early development stage? If it is an intrinsic property of an agent, what should be considered? How does the mobility of an agent affect other related issue in a system such as security, system performance, collaboration, consistency, locality of reference, flexibility, customization and other properties? We are still not very certain that problems addressed by MASS can be equally well, and perhaps more securely resolved by static clients that exchange messages.

Mobile agents are adept to changing platform and environment, but stationary agents are less capable in this regard. In other words, mobile agents must be able to work in a heterogeneous environment. If an agent is not mobile, it must use the network to exchange information. This will

- Reduce complexity required by mobility;
- Encourage specialization within platforms;
- Employ well-established protocols;
- Supports closed-environment philosophy;
- Results in performance problem in situations requiring high volume or frequency;
- Results in processing inefficiencies when the sum of the specialized agents makes more work than having a single mobile agent;
- Reducing effectiveness when a connection is lost.

4.2.4 A Few Critical Comments and Conclusions

4.2.4.1 Basic Features of Agents

An AI agent is usually designed with asynchronous message passing. The system shows better stability. A SW agent is transaction driven. Asynchronous message passing might be less frequently used.

- A MAS/MASS agent system must be autonomous. This implies a loose coupling of the system.
- Al agent systems and MAS/MASS require different levels of collaboration, they should be considered at different granularity.
- Fault tolerance might not be an important requirement for AI agents.
- MASS and MAS: mobility is the property of an agent for MAS/MAS, but not a basic property of any kind of agent systems. It is the agent context which should be migrated. There are two issues in making an agent mobile: (a) security, and (b) group communication / group coverage
- Killer application: Mobile agents may require a "Killer" application. The mobile agent paradigm is in many respects a new and powerful programming paradigm, and its use leads to better performance in some cases. Nonetheless, most particular applications can be implemented just as cleanly and efficiently with a more traditional technique. Thus, the advantages of mobile agents are modest when any particular application is considered in isolation. Instead, researchers must present a set of applications and argue that the entire set can be implemented with much less effort. At a minimum, making such an argument demands that the mobile-agent community actively supports anyone who is writing a high-quality survey of mobile-agent applications, since no one group will be able to implement a sufficient number of applications. Once a clear quantitative argument is made, a few major Internet services can be convinced to open their sites to mobile agents, since they will recognize that agents will lead to more applications based around their services and hence more users. From there, more Internet services will follow.
- If the MAS/MASS is highly mobile, security will be an intrinsic property of agents. The importance of security may depend on the system the agent is designed to work. If agents are designed to work within a closed system, security is less critical. However if the agents are designed to work in an open system such as VPN, WAN, or a hybrid system, the security is a critical issue for agents and the system as well.

4.2.4.2 Three Hurdles traced by MAS/MASS

1. Lack of agreed definitions: Agents built by different teams have different capabilities and definition. Similar agent can be built on different models.

2. Duplication of efforts: There has been little reuse of agent architecture, design, or components. People have discussed a lot over this issue, but failed to come to a common base due to the interests on agents
3. Inability to satisfy industrial strength requirements: Agents must integrate with existing software and computer infrastructure. They must also address security and scaling concerns.
4. In conclusion, despite the criticisms on agent systems for both AI and MAS/MASS by those within and without other agent camps, there are many industrial applications that would benefit significantly from them, in just the same way, as there are applications that would benefit from reactive agents.

4.2.4.3. What Have Made the MAS/MASS Failure?

To be fair, I personally think that the biggest failure comes from the lack of industry strength, or a practical link to the real application of the agent paradigm. It is true that we do see some commercial systems. Companies are selling agent-based products or software, but the market share is far beyond the target expected by the agent community. Both Al and SW engineering community have failed to deploy commercially useful agent systems although there are over a hundred available online. None of them is driven by real applications, but simply show cases. The current agent infrastructures and languages tend to be driven more by notions of computational elegance than practical need. The agent area still remains as complicated hard-to-understand topic.

4.2.4.4 What Have Convinced Us at Present

Agent as a paradigm for SW engineering: SW engineers have derived a progressively better understanding of the characteristics of complexity in software. It is now widely recognized that interaction is probably the most important single characteristic of complex software. They have believed that agents can be used as a tool for understanding human societies since mobile MAS/MASS provide a novel new tool for simulating societies, which may help shed some light on various kinds of social processes.

An agent can be considered as a computer system that is capable of independent action on behalf of its user or owner. The MAS/MASS is one that consists of a number of agents, which interact with one-another. In order to successfully interact, agents need the ability to cooperate, coordinate, and negotiate.

To be cost-effective, agents system should be developed to work in a distributed and heterogeneous environment, and has the ability to share resources. This requires the agents being capable of independent, autonomous action in order to successfully carry out the tasks that we delegate to them.

4.3 Agent Future

Software agents offer a new paradigm for very large scale distributed heterogeneous applications. It focuses on the interactions of autonomous, cooperating processes, which can adapt to humans and other agents. The mobility should be an orthogonal characteristic. Intelligence is always a desirable characteristic, but is not required by the paradigm. After this project, my primary conclusion is that "*The paradigm is still under construction*". This is based on the following considerations.

4.3.1 Major Concerns and Challenges

4.3.1.1 Using XML to Describe ACL Messages

Both KQML and FIPA ACL are using a LISP-like syntax to describe properly formed ACL messages. ACL messages have "deeper" semantics (KR-like) than account for the Communicative Act, the Sender and the Receiver. The deep semantics, in the case of FIPA ACL are described in SL. An ACL message as a syntactic object has parameters that are not accounted for in the semantics (language, ontology, in-reply-to, etc.). Syntactically, ACL messages introduce pragmatic elements and a particular syntax useful for parsing and routing. The syntactic form (e.g., LISP-like) need not be unique. ACL messages can be thought as having an "abstract syntax". The abstract syntax "allows" for multiple syntactic representations or encoding. Examples of encoding are: Lisp-like balanced parenthesis list, XML or even a Java structure.

Beyond XML: RDF (Resource Description Framework) is a W3C metadata standard built on top of XML RDF data consists of nodes and attached attribute/value pairs, providing the expressive power of *semantic networks.* Nodes can be any web resources (pages, servers, basically anything for which one can

give a URL) including other RDF expressions. Attributes are named properties of the nodes, and their values are either atomic (e.g., strings, numbers) or other resources or metadata instances.

Other standards are built on top of RDF, including:

- P3P: *Platform for Privacy Preference* for the exchange of privacy practices and preferences among Web sites, agents and users.
- PICS: an infrastructure for associating metadata labels with Internet content.
- SHOE is an HTML-based knowledge representation language developed by Jim Hendler et al at the University of Maryland (http://www.cs.umd.edu/projects/plus/SHOE/). It provides an XML compliant way of encoding horn clauses and embedding them in web pages a namespace for shared ontologies defined on other web pages. Similar examples include Interlingua (Grosof et. al.) and XML encoding of KIF.
- DAML (Darpa Agent Markup Language) will be the next iteration of these ideas.
- Can ACL be a high level language to be understood? Does it provide additional degree for presentation using efficient solution?

4.3.1.2 Modeling, and Architecture Design

We have discussed a lot of the MAS/MASS modeling, architecture and design, and many examples have been listed. However, the majority of the current available models, architectures and designs are given to specific project(s). In other words, they may be application specific. An open architecture design will be of high priority for both MAS and MASS. There are a number of papers discussing the requirements for an open architecture of agents, which should provide a relative universal agent platform and fits the major application domains.

4.3.1.3 Mobility and Mobile Agents

It is still not very clear how, when, and where the mobility should be introduced to the agent, and which agents should be mobile. If an agent is mobile, what should it carry along? Current requirement for a mobile agent is to carry mobile code, but not data. A mobile agent is sent to a remote destination to perform particular task with the use of local data. When the task is finished, the agent should bring the result/output back. Thus, the mobile agent, in any sense, is not a lightweight object, which prevents its mobility at the

current network environment due to the security and network bandwidth. A potential solution in future is that a mobile agent carries only its context, which contains whatever required for performing the task without any outside interference. An agent works on an assigned task with local resources, and gets the result out, but not any modification to local host resources. This could make it a lightweight object, and enhance the security of the system. The final goal of the mobile agent should be working in an open system.

4.3.1.4 Agent Communication Architecture

There is no well-defined transport protocol for agent systems; it depends highly on the systems itself. The following agent architectures have been proposed by the FIPA:

Communication protocols:

- Unicast – sending a packet when there is only one sender process and one specific recipient process.
- Broadcast – sending only one packet and all the hosts in the network recognize and read it.
- Multicast – sending only one packet and all the host that have registered interest recognize and read it.

Application protocols:

- Publish/subscribe – decoupled, asynchronous, many-to-many, event-driven communication.
- Request/reply – decoupled, synchrounous, one-to-many, demand-driven communication.
- Solicit/response – asynchronous request/reply.

Message routing:

- Subject-based.
- Content-based.

Message properties:

- Format repository service.
- Self-describing format.
- Transforming/transition service.
- Message priority.

- Message expiration.

These are just propsoed agent communication architectures, we may need a standard for it. There is an urgent need for communication agents to agree on the agent-level protocols they will use. This protocol is often conveyed via an extra parameter on a message (for instance, ask: from Alice: to Bob...: protocol auction 42). The current common protocols are : Contract Net, Various Auction Protocols, Name Registration. We also require the protocol defined in terms of constraints on possible conversation and can be expressed as: Grammars (e.g. DFAs, ATNs) Petri Net Rules or axioms ete. We have to decide which of these requirements are neccessary, and which are optional (application dependent).

4.3.1.5 Agent Platform Development Tools

Most available agent platform development tool kits are Java based, since this is the programming language easily run across platforms. However, most of the Java program code is relatively hervyweight, and does not work well through network if the agent (seen as a SW program) is too complicated. We need a more general agent platform development tool kit which can build lightweight agents for a complicated application. This means that the agent platform supports a high-level programming language.

4.3.1.6 Industry Strength

We have seen rapid progress of MAS/MASS system development since 90s. However this trend has not shown any industry strength yet. There could be many reasons for this, but I would say that this technology is not matured to meet the basic requirements as an industry product. There is no final release date. The future seems to be there, but we do not know how far away it is yet.

4.3.1.7 What Does It Take to Remain an Important Distributed Paradigm in the Next Decade? Can Agent Paradigm be the One?

I would suggest the following merits associated with the agent paradigm:

- Naturally resource sharing.
- Autonomous, decentralized and loose coupled.
- Security for a remote application running on hybrid systems.
- Data consistency.

- Mobility.

It seems that the agent paradigm posses all of them, thus could be a good candidate for the next generation distributed system.

4.3.2 A Few Key Ideas for Agent Future

Based on the current agent paradigm status, and its history in the past 10 years, it is still hard to predict if the paradigm will be dominant over the others in a foreseeable future. I do see a period, during which substantial progresses have been made in the agent paradigm. This is the period between 1996- 2001. However the agent paradigm lacks the industry strength from the very beginning, and it has not shown any sign of greater improvement. I would ask myself a few questions before going to the key ideas:

- Is the agent paradigm suitable for all applications?
- What does it take to remain an important distributed paradigm in the next decade?

Frankly speaking, I do not have answers for them. However this does not prevent me from suggesting the following key ideas for the agent paradigm in future based on my own analysis:

- Software agents offer a new paradigm for very large scale distributed heterogeneous applications.
- The paradigm focuses on the interactions of autonomous, cooperating processes, which can adapt to humans and other agents.
- Agent communication language is a key enabling technology;
- Mobility is an orthogonal characteristic, which someone considers as a peripheral issue.
- Intelligence is always a desirable characteristic but is not strictly required by the paradigm. The paradigm and associated technology are still evolving.
- Agent communication: Agent-agent communication is key in realizing the potential of the agent paradigm. Since interoperability is a defining characteristic of agents, standards are important! Candidates for standardization include
    - Agent architecture.
    - Agent communication language.
    - Agent interaction protocols.
    - Agent knowledge.

- Agent programming languages.
- Agent methodology.

The KSE offers a four-part methodology for developing complex agent-based systems:

- Collect/construct necessary ontologies.
- Use standard, published ontologies if possible.
- Develop (and publish) new components as needed.
- Use common tools, e.g. Ontolingua, *GFP etc.*
- Choose common representation language(s) e.g., SQL or KBMS with KIF as a recommended default.
- Use an ACL like KQML as communication language extended with new performatives and protocols as needed.
- Identify and define new higher-level protocols e.g., for negotiation, purchasing, cataloging, etc.

What's needed tomorrow?

- Further develop semantics of ACLs.
- Common content languages and ontologies.
- A language for describing agent actions, beliefs, intentions, and so on.
- Better handle on metadata. An abstraction and applicable to many content languages.
- Declarative and learnable protocols. Languages for defining higher-level protocols based on more primitive ones.
- Practical agent knowledge sharing.
- "Social" mechanisms for distributing information and knowledge.
- Frameworks for controlling collections of agents, e.g., artificial markets, natural selection, etc.

I agree fully with these sentiments, which is why we believe agent technology is not a passing fad. People have now overviewed a broad range of work, which goes under the banner of 'agents'. They have outlined their various promises as well as their challenges. However, apart from such technical issues, there is also a range of societal (i.e. social) and ethical problems, which are looming. Donald Norman writes: "Probably all the major software manufacturers are exploring the use of intelligent agents. Myths, promises, and reality are all colliding. But the main difficulties I foresee are social, not technical. How will

intelligent agents interact with people and perhaps more importantly, how might people think about agents?" (Norman, 1994). Many people disagree with Norman regarding the major technical hurdles ahead. As shown in the previous section, there are some extremely demanding technical issues to be resolved in most of the agent classes reviewed. However, he did pose an extremely pertinent question.

There are issues, which society would have to grapple with through various legislations and they would be very thorny. They include the following:

- Privacy: how do you ensure your agents maintain your much needed privacy when it acts on your behalf?
- Responsibility: which goes with relinquished authority: when you relinquish some of your responsibility to software agent(s) (as you would do implicitly), be aware of the authority that is being transferred to it/them. How would you like to come back home after a long hard day at work being the proud owner of a used car negotiated and bought for, courtesy of one of your (Kasbah) software agents? How do you ensure the agent does not run amok and run up a huge credit card bill on your behalf?
- Legal issues: following on from the latter, imagine your agent (which you probably bought off-the-shelf and customized) offers some bad advice to other peer agents resulting in liabilities to other people, who is responsible? The company who wrote the agent? You who customized it? Both? We envisage a new raft of legislation, which should be developed in the future to cover software agents.
- Ethical issues: these would also need to be considered. We already concerned enough about the ethics of software agents that an agent etiquette has been proposed for information service and user agents as they gather information on the WWW. They include the following:
    - Agents must identify themselves;
    - They must moderate the pace and frequency of their requests to some server;
    - They must limit their searches to appropriate servers;
    - They must share information with others;
    - They must respect the authority placed on them by server operators;
    - An agent's services are accurate and up-to-date.

Etzioni (1994) has proposed others including:

- Safety - the agent should not destructively alter the world;
- Tidiness - the agent should leave the world as it found it;
- Thrift - the agent should limit its consumption of scarce resources;
- Vigilance - the agent should not allow client actions with unanticipated results.

However, such issues are not that critical immediately, but would be in the medium to long terms. In the short term, we expect some basic agent-based software to be rolled out e.g. some basic interface agents such as mail filtering or calendar scheduling agents. More basic mobile agent services would also be provided in the short term. We can also predict comfortably that many vendors would claim that their products are agent-based even though they most certainly are not. For example, we are already hearing of 'compression agents' and 'system agents' when 'disk compressors' and 'operating systems' would do respectively, and have done in the past. As Guilfoyle (1995) warns: "there is a danger, however, that customers may be disappointed by the gap between colorful press reports about smart agents handling half the work, and the reality of 'if-then' rules for message routing". In the medium term (3 to 5 years), some more decent agent applications would be rolled out for most of the classes of agents overviewed in this paper. Perhaps, there would be collaborative agents and/or integrated heterogeneous agent applications doing limited, but *real* workflow or air traffic management or controlling real telecommunication networks etc., rather than just simulations. Useful, but limited, interface agents should be available which perform roles including the following: eager assistants, WWW guides, memory aids, entertainment and WWW filters/critics. Indeed, Agents Inc. - a Boston-based agents company, is already marketing the HOMR/Ringo system as Firefly. Open Sesame – another interface agent that employs a neural network learning technique is already on the market from Charles River Analytics (Cambridge, MA). Vendors would soon roll out more mobile and information agent applications and languages. We expect reactive and/or hybrid agent technology to start delivering some real everyday industrial applications during this period. Furthermore, during this medium term, we expect the WWW to be commercialized to some degree, enabling agents of different classes to play a role in paying for services and performing some restricted buying and selling on our behalf, as Kasbah agents propose to do. We would also start seeing a proliferation of specialist agents' conferences: agents in the aviation industry, agents in law, etc. We also envisage that the new domain of

agent-based software engineering will grow from strength to strength. In the long term (> 7 years), we would expect to see agents which *approximate* true 'smartness' in that they can collaborate and learn, in addition to being autonomous in their settings. They would possess rich negotiation skills and some may demonstrate what may be referred to, arguably, as 'emotions'. We caution against the usage of words such as the latter, not least because the agent literature does not work with such vocabulary, which have subtle and complex meanings in the human context (Smith, 1996a). However, it is also at this stage society would need to begin to confront some of the legal and ethical issues which are bound to follow the large scale fielding of agent technology. In the long term too, agents would also provide another design approach to constructing complex pieces of software.

The MAS/MASS think the agent that "they can't fly yet, but intelligent agents and smart software are beginning to walk". Our view is different: using Indermaur's metaphor, we argue that limited software agents are just to about to 'crawl' out of research laboratories; apart from a few interface and information agents. In summary, we believe, like Greif (1987, 1988a, 1988b), that agents can have an enormous effect, but that this will appear in everyday products as an *evolutionary* process, rather than a *revolutionary* one predicted by much press coverage. For example, the Ovum figures quoted in Guilfoyle's (1995) paper suggest a seven to eight fold increase in the agents market, in Europe and USA alone by the year 2002 - this suggests a revolution! But today's market does not match this prediction! Greif notes correctly that agents would initially leverage simpler technologies available in most applications (e.g. word processors, spreadsheets or knowledge-based systems). Then agents would gradually be evolved into more complicated applications. We hope the over optimistic claims about agent technology are moderated. It is a sad fact that the only jarring voice in this hymn of confidence about agent technology comes from those doing the *real* research, and who know what some of the real technical, social and ethical challenges are.

Finally, we should ask what has to be taken for agents to remain as an important distributed paradigm in the next 10 years. If we could list some critical requirements for this, can this be for agent paradigm? This is a very important question for software engineering community, and I have no sufficient knowledge to answer it based on what I know of the current MAS/MASS status.

References

C. Abrami, "An Agent-Group-Role based modeling framework for participative water management support", http://www.mmrg.ecs.soton.ac.uk/publications/papers/Voyager/papers/, (2002).

N. Anil and F. Shatz "A Brief Discussion on Current Agent Architecture", http://www.mmrg.ecs.soton.ac.uk/publications/papers/Voyager/papers/(2001).

Y. Aridor, and Lange, D.B., "Agent Design Patterns: Elements of Agent Application Design", Proc. 2nd Int'l Conf. on Autonomous Agents (Agents '98), ACM Press, pp. 108-115 (1998).

M. Balabanović and Y. Shoham, "Fab: Content-Based, Collaborative Recommendation", Communications of the ACM 40(3), 67-72 (1997).

O. Barreteau and B, F. Shadoc, "A multi-agent model to tackle viability of irrigated system", Annals of Operational Research, 94, 139-162 (2000).

R. Barrett, P. P. Maglio and D. C. Kellem, "WBI: A Confederation of Agents that Personalize the Web", in *Autonomous Agents 97*, (Marina Del Rey, California, USA, 1997).

Barbuceanu, M., T. Gray, and S. Mankovski, "Co-coordinating with obligations", in *Autonomous Agent 98*, (Minneapolis, USA, 1998), pp. 62-69.

J. Bates, "The Design of Databases and Other Information Resources for Humanities Scholars: The Getty Online Searching Project Report No. 4." Online and CD-ROM Review 18 (December 1994): 331-340., (1994).

J. Bates, "The role of emotion in believable agents", Communications of the ACM 37(7), 122-125 (1994).

B. Bauer, "Extending UML for the specification of interaction protocols", FIPA, (1999).

Bauer, B., "UML class diagrams revisited in the context of agent-based systems", in *AOSE 2001*, (Montreal, Canada 2001), pp.125-141.

B. Bauer, "AUML – first steps", in *Proceedings of SCI/ISAS 2000*, (Orlando, FL, USA, 2000).

B. Bauer, "An extension of UML by protocols for multiagent interaction", in *Proceedings, Fourth International Conference on MultiAgent Systems. ICMAS 2000*, (Boston, IEEE Computer Society 2000b).

B. Bauer, "Agent UML: a formalism for specifying multiagent software systems", Int. Jour. Software Eng. and Knowledge Eng. 11(3), pp. 1-24 (2001).

F. Bergenti and A. Poggi, "Exploiting UML in the design of multi-agent system", Int. Jour. Software Eng. and Knowledge Eng. 11(3), pp.56-74 (2001).

D. Billsus and M. Pazzani, "Learning Probabilistic User Models", in *Workshop Notes of Machine Learning for User Modeling, Sixth International Conference on User Modeling*, (Chia Laguna, Sardinia, 1997), pp. 136-139.

D. Billsus and M. J. Pazzani, "A Personal News Agent that Talks, Learns and Explains", in *Autonomous Agents 98*, (Minneapolis, USA, 1998), pp.216-219.

E. Bloedorn and J. Wenk, "Constructive Induction-based Learning Agents: An Architecture and Preliminary Experiments", in *Proceedings of the First International Workshop on Intelligent Adaptive Systems (IAS-95)*, (Melbourne Beach, Florida, 1995), pp. 38-51.

K. Bollacker, "Cite: an autonomous web agent for automatic retrieval and identification of interesting publications", in *Proceedings of the First International Conference on Autonomous Agents*, (Marina del Rey, CA, USA 1998), pp. 116-123 .

Boloni, F. and J. Marinescu, "An agent behavior model", http://www.trl.ibm.co.jp/papers (2000).

A. H. Bond and L. Gasser, *"Readings in Distributed Artificial Intelligence"*, (San Mateo, CA: Morgan Kaufmann, 1988).

G. Boone, "Concept Features in Re :Agent, an Intelligent Email Agent", in *Autonomous Agents 98*, (Minneapolis MN USA, 1998).

R. A. Brooks, *"A robot that walks: Emergent behaviors from a carefully evolved network"*, Memo 1091, MIT AI Lab (1998).

R. A. Brooks, "Intelligence Without Reason", in *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, (USA, 1991), pp. 569-695.

R. A. Brooks and L. A. Stein, "Building brains for bodies", (Memo 1439, MIT AI Lab, 1988).

S. M. Brown, E. Santos and S. B. Banks, "A Dynamic Bayesian Intelligent Interface Agent", Dept of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH 45433-7765.

R. Brugali, "The framework life span", Communications of the ACM 40(10), pp. 65-68(1993).

F. Buschmann, Pattern-oriented software architecture: A system of Patterns. (Wiley and Sons, 1996).

N. Carver and V. Lesser, "The DRESUN Testbed for Research in FA/C Distribution Situation Assessment: Extensions to the Model of External Evidence", in *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95*, San Francisco, USA, 1995), pp. 33-40.

N. Carver, Z. Cvetanovic and V. Lesser, "Sophisticated Cooperation in Distributed Problem Solving", in *Proceedings of the 9th National Conference on Artificial Intelligence* **1**, (Anaheim, 1995), pp.191-198.

C. Castelfranchi and J. Müller, "From Reaction to Cognition", 5th European Workshop on Modelling Autonomous Agents, MAAMAW '93, Neuchatel, Switzerland, August 25-27, Selected Papers. Springer 1995 (1995).

A. Cesta, "Mixed-Initiative Agenda management with Software Agents", in *Proceedings of the First International Conference on Autonomous Agents*. (Marina del Rey, CA, USA 1997), pp. 524-526.

Chaib-draa, B., B. Moulin, R. Mandiau, and P. Millot, "Trends in Distributed Artificial Intelligence", Artificial Intelligence Review 6, pp. 35-66 (1992).

L. Chen and K. Sycara, "WebMate: a personal agent for browsing and searching", in *Autonomous Agents 98.* (Minneapolis MN USA, 1998), pp. 512.

L. Chen, "A logical approach to high-level agent control", *in AGENTS 01*. (Montreal, Quebec, Canada, 2001).

J. C. Collis and L. C. Lee, "Building electronic marketplaces with the ZEUS agent toolkit", http://www/labs.bt.com/projects/agents (2001).

Concordia White Paper.
http://www.merl.com/projects/concordia/WWW/MobileAgentsWhitePaper.pdf, 2002.

A. Cypher, "Eager: Programming repetitive tasks by example", in *Human factors in computing systems conference proceedings on Reaching through technology*. (New Orleans, LA USA, 1991), pp. 33-39.

A. O. Damasio, Error de Descartes. (Publicacoes Europa-America, 1990).

R. Davis and R. G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving", *Artificial Intelligence* 20, 63-109 (1983).

J. Davies, R. R. Weeks and A. McGrath, "Using Clustering in a WWW Information Agent", in *18th BCS IR Colloquium.* (Manchester, UK, 1996).

K. S. Decker, "Distributed Artificial Intelligence Testbeds", in *Foundations of Distributed Artificial Intelligence*, edited by G. O'Hare, and N. Jennings (London Wiley, 1995).

K. S. Decker and V. R. Lesser, "*Designing a family of coordination algorithm*", in: Proceedings of the First International Conference on Multi-Agent Systems. (San Francisco: AAAI Press, 1993), pp. 73-80.

K. S. Decker, "Designing behavior for information agents", in Proceedings of the First International Conference on Autonomous Agents. (Marina del Rey, CA, USA, 1997), pp. 404-412.

D. Dennet, "Intentionality", http://ase.tufts.edu/cogstud/papers/intentio.htm (1987).

R. Depke, R. Heckel and J. M. Kuster, "Formal agent-oriented modeling with UML and graph transformation", submitted to Elsevier Science (2001).

J. Doran, H. Carvajal, Y. Choo and Y. Li, "The MCS Multi-agent Testbed: Developments and Experiments", in *Cooperating Knowledge based Systems*, edited by S. Deen, (Heidelberg: Springer-Verlag, 1991), pp. 240-251.

E. H. Durfee and T. A. Montgomery, "MICE: A Flexible Testbed for Intelligent Coordination Experiments", *in Proceedings of the 1989 Distributed Artificial Intelligence Workshop*, 25-40 (1989).

K. Erol and J. Lang, "Designing agents from reusable components", in *Proc. the 4th Ann. Conf. On Autonomous Agent*, edited by C. Sierra and et al.,(2000), pp. 76 – 77.

O. Etzioni, "A softbot-based interface to the internet", Communications of the ACM 37(7), 72-76 (1994).

J. R. Falcone and J. S. Emer, "*A programmable interface language for heterogeneous distributed system*"', ACM Trans. Computer Systems, 5, (4), pp. 330--351 , (1987).

J. Ferber, "Simulating with Reactive Agents", on *Many Agent Simulation and Artificial Life*, edited by E. Hillebrand and J. Stender. (Amsterdam: IOS Press, 1994), pp. 8-28.

J. Ferber and O. Gutknecht, "A meta-model for the analysis and design of organizations in multi-agent system", in *ICMAS98*, edited by Y. Demazeau, pp. 128-135 (1998).

The FIPA home: http://www.fipa.org/

FIPA, "FIPA '99 Specification Part 2: Agent Communication Language". http://www.fipa.org (1999).

FIPA Technical Committee C, "Extending UML for the Specification of Agent Interaction Protocols", response to the OMG Analysis and Design task Force UML RTF 2,0 Request for Information (1999).

T. Finin, "KQML as an agent communication language", In Software Agents, edited by J. M. Bradshaw. (MIT Press, 1997).

M. Fowler, "Dealing with roles", in *4th Annual Conference on the Pattern Languages of Programs.* (Monticello, Illinois, USA, 1997).

S. Franklin and A. Graesser, "Is it an agent, or just a program?: A Taxonomy for autonomous agents", in *Proc. ECAI'96 Workshop on Agent Theories, Architectures, and Languages: Intelligent Agents III* , edited by (J.P. Muller, M.J. Wooldridge, and N.R.. Jennings. (LNAI, vol. 1193, Spriner-Verlag, 1997), pp.21-36.

L. N. Foner, "What's an Agent, Anyway? A Sociological Case Study", in *Agents Memo 93-01*, MIT Media Lab. (Cambridge, MA, 1993).

L. N. Foner, "Yenta: A Multi-Agent, Referral-Based Matchmaking System", in *Autonomous Agents 97*, (Marina Del Rey, California USA, 1997).

K. Franklin and T. Kraesser, "A Preliminary Study on Multi-Agent System for Internet Info Search", in *Autonomous Agents 97*, (Marina Del Rey, California USA, 1997).

D. Friha, "Multi-agent system specification using CO-OPN", in *University of Geneva, Technical Report CUI No. TR-93/80* (1991).

N. Fukuta, T. Ito and T. Shintani, "A logic-based framework for mobile intelligent information agents", ", in *Proc. Of the First Pacific Rim International Workshop on Intelligent Information Agents*, (PRIIA' 2000a).

N. Fukuta, T. Ito and T. Shintani, "MiLog: a mobile agent framework for implementing intelligent information agents with logic programming", in *Proc. Of the First Pacific Rim International Workshop on Intelligent Information Agents*, (PRIIA' 2000b).

E. R. Gamma, R. Helm, R. Johnson and J. Vlissides, Design patterns: *Elements of Reusable Object-Oriented Software*. (Addison-Wesley, 1994).

L. Gasser, C. Braganza and N. Herman, "MACE: A Flexible Testbed fo Distributed AI Research", in *Distributed Artificial Intelligence*, Research Notes in Artificial Intelligence, edited by M. Huhns. (London: Pitman, 1987), pp. 119-152.

L. Gasser and M. Huhns, *Distributed Artificial Intelligence* 2. (San Mateo, CA: Morgan Kaufmann, 1989).

L. Gasser, "Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems", *Artificial Intelligence* 47, 107-138(1991).

M. Genesereth and S. Ketchpel, "*Software Agents*". Communication of the ACM 37(7), pp. 48-53(1994).

M. Genesereth, R. Keller and A. M., Duschka, O., ``*Infomaster: An Information Integration System*", *Proceedings of 1997 ACM SIGMOD Conference*, (1997).

R. S. Gray, David Kotz, Ronald A. Peterson, Joyce Barton, Daria A. Chacón, Peter Gerken, Martin O. Hofmann, Jeffrey Bradshaw, Maggie R. Breedy, Renia Jeffers, Niranjan Suri, "Mobile-Agent versus Client/Server Performance: Scalability in an Information-Retrieval Task". Mobile Agents 2001: pp. 229-243 (2001).

R. S. Gray, G. Cybenko, D. Kotz, R. A. Peterson and D. Rus: D'Agents: Applications and performance of a mobile-agent system. Software - Practice and Experience 32(6): pp. 543-573 (2002).

C. L. Green and P. Edwards, "Using Machine Learning to Enhance Software Tools for Internet Information Management", in *AAAI-96 Workshop on Internet-Based Information Systems, WS-96-06*. (AAAI Press, 1996), pp.48-55.

I. Greif and S. Sarin, "Data Sharing in Group Work", *ACM TRansactions on Office Information Systems*, vol. 5, no. 2, pp. 187-211 (1987).

I. Greif (ed.), *Computer-Supported Cooperative Work: A Book of Readings*, Morgan Kaufman, San Mateo, California (1988a).

I. Greif , Remarks in panel discussion on "CSCW: What does it mean?", *CSCW '88, in "Proceedings of the Conference on Computer-Supported Cooperative Work, September 26-28, 1988, Portland, Oregon"*, ACM, New York, NY, (1988).

C. Guilfoyle, "Vendors of Agent Technology", *in UNICOM Seminar on Intelligent Agents and their Business Applications*. (London, 1995), pp. 135-142.

M. Hannebauer and S. Muller, "Distributed constraint optimization for medical appointment scheduling", Communications of the ACM 53(6), pp. 165-178(1998).

C. Hewitt, "Viewing Control Structures as Patterns of Passing Messages", *Artificial Intelligence* 8(3), pp. 323-364 (1977).

M. N. Huhns and M. P. Singh, "Distributed Artificial Intelligence for Information

Systems", in *CKBS-94 Tutorial*, (University of Keele, UK, 1994)

D. Hotz and R. S. Gray, "Mobile Agents and the Future of the Internet", http://www.cs.dartmouth.edu/~dfk/papers/kotz:future2/ (1999).

M. A. Hoyle and C. Lueg, "Open Sesame !: A Look at Personal Assistants", in *Proceedings of the International Conference on the Practical Application of Intelligent Agents (PAAM97).* (London, 1997), pp. 51-60.

C. A. Iglesias, M. Garrjo, J. Gonalez and J. R. Velasco, "A methodological proposal for multiagent systems development extending common KADS", in *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop 1*, edited by B. Gaines and T. Musen, (1996).

C. A. Iglesias, M. Garrjo, J. Gonalez and J. R. Velasco, "Analysis and design of multiagent systems using MAS-CommonKADS. In Fourth International Workshop on Agent Theories, Architectures and Languages", (ATAL 97. Rhode Island, USA, Springer, 1997).

Information on KQML, http://agents.umbc.edu/kqml, 2003.

Information on KIF, http://agents.cs.umbc.edu/kif, 2003.

Information on Ontology, http://agents.umbc.edu/ontology/, 2003.

Information on Aglets, http://www.trl.ibm.co.jp/aglets/, 2002.

Information in Agent Communication Languages, http://agents.umbc.edu/acl/ (see also http://www.fipa.org/), 2003.

Information on IBM Aglets: http://www.trl.ibm.co.jp/aglets/api/com.ibm.aglet.AgletInfo.html, 2003.

T. Ito, "Cooperative bidding mechanisms among agents in multiple online auctions", in *Proc. of PRICAI2000, 2000).* pp. 810.

N. Jennings, J. M. Corera, L. Laresgoiti, E. Mamdani, F. Perriollat, P. Skarek, and L. Varga, "Using ARCHON to Develop Real-World DAI Applications for Electricity Transportation and Particle Accelerator Control", *IEEE Expert*, Special Issue on Real World Applications of DAI systems (1995).

S. Khanna, "Realtime Scheduling in SunOS 5.0," in *Proceedings of the USENIX Winter Conference*, pp. 375–390, USENIX Association, (1992).

T. Kamba, K. Bharat, and M. C. Albers, "The Krakatoa Chronicle: An Interactive, Personalized Newspaper on the Web", in *Proceedings of WWW. (* Boston, USA, 1995).

H. Kautz, B. Selman, and M. Shah, "Referral Web: Combining Social Networks and Collaborative Filtering", Communications of the ACM 40(3), pp.63-65 (1997).

A. Kay, "User interface: A personal view", in: The art of Human-Computer Interface Design, edited by B. Laurel. (Addison-Wesley, 1990), pp.191-207.

E. A. Kendall and M. Malkoun, " The layered agent pattern", in *Proceedings of the Conference on Pattern Languages of Programs.* (PloP'96, 1996).

E. A. Kendall, "The layered agent pattern language", in *Proceedings of the Conference on Pattern Languages of Programs* (PloP'97, 1997).

E. A. Kendall, "Patterns of Intelligent and mobile agents", in *Autonomous Agents, 98.* (Minneapolis, MN, USA 1998).

E. A. Kendall, "Role model designs and implementations with aspect-oriented programming", ACM Computing Survey, 31(3). (1999).

E. A. Kendall, "An application framework for intelligent and mobile agents", ACM Computing Survey, 32(1), 2000.

J. A. King, "Intelligent Agents: Bringing Good Things to Life", *AI Expert*, 17-19(1995).

D. Kinny, M. Georgeff, and A. Rao, "A methodology and modeling technique for system of BDI", in *MAAMAW 96, LNAI 1038* (The Netherland, 1996), pp 56-71.

A. Kobsa, "User Modeling: Recent work, prospects and Hazards", in *Adaptive User Interfaces: Principles and Practice, edited by Schneider-Hufschmidt*, M. Kühme, and T. Malinowski, (North-Holland 1993).

J. A. Konstan, B. N. Miller, D. Maltz, and J. L. Herlocker, L. R. Gordon, and J. Riedl, "GroupLens: Applying Collaborative Filtering to Usenet News", Communications of the ACM 40(3), pp. 77-87(1997).

D. Kotz and R. Gray, " Mobile Agents and the Future of the Internet. Operating Systems Review 33(3): 7-13 (1999).

Y. Labrou, "Agent Coomunication Languages: Past, Present and Future", http://www.objs.com/agent/agent.pdf (2000).

K. Lang, "NewsWeeder: Learning to Filter NetNews", in *ICML95 Conference Proceedings*, pp. 331-339 (1995).

V. Lesser and D. Corkill, "Functionally Accurate, Cooperative Distributed Systems, *IEEE Transactions on Systems, Man, and Cybernetics* C-11(1), pp.81-96 (1981).

R. Levitt, P. Cohen, J. Kunz, C. Nass, T. Christiansen, and Y. Jin, "The Virtual Design Team: Simulating how Organisational Structure and Communication Tools affect Team Performance", in *Computational Organisation Theory*, edited by K. Carley and M. Prietula (San Francisco: Lawrence Erlbaum.1994).

H. Lieberman and D. Maulsby, "Instructible agents: Software that just keeps getting better", IBM systems Journal 35(3&4), 1996.

H. Lieberman, P. Maes and N. W. Van Dyke, "Butterfly: A Conversation-Finding Agent for Internet Relay Chat", in *Proceedings of the 1999 International Conference on Intelligent User Interfaces (*1999).

H. Lieberman, B. A. Nardi and D. Wright, "Training Agents to Recognize Text by Example", in *Autonomous Agents 99* (Seattle WA USA, 1999).

J. Lind, "Specifying agent interaction protocols with standard UML", http://www.agentlab.de/protocol_specification.html (2001).

A. Lingnau, Infrastructure for mobile agents, http://mars.tm.informatik.uni-frankfurt.de/~lingnau/ (1995a).

A. Lingnau, "An HTTP-Based infrastructure for mobile agents", in: *Proceedings of the Fourth Int'l WWW Conference* (1995b).

J. Liu and H. Qin, "Organizing synthetic agent behavior based on a motif architecture", in *Proceedings of the Third Annual Conference on Autonomous Agents* (Seattle, WA USA. 1999), pp 340-341.

J. Liu, "Autonomous Agents and Multi-Agent Systems" (World Scientific, Singapore, New Jersey, London and Honkong, 2001).

J. Liu and K. P. Sycara, "Exploiting problem structure for distributed constraint optimization", in "*Proceedings of the First International Conference on Multi--Agent Systems*", (1995) (or http://citeseer.nj.nec.com/cache/papers/cs/1657/http:zSzzSzwww.cs.cmu.eduzSzafszSzcszSzusrzSzkatiazSzwwwzSzcora-paperszSzicmas95.pdf/liu95exploiting.pdf).

A. Loyall, "Real-time control of animated broad agents", in *Proceedings of the 15th Annual Conference of the Cognitive Sciences Society*" (1995).

P. Maes and R. Kozierok, "A learning interface agent for scheduling meetings", *in Proceedings of ACM SIGCHI International Workshop on Intelligent User Interfaces*, (ACM Press, NY, 1993), pp. 81-88

P. Maes, "Agents that reduce work and information overload", Communications of the ACM 37(7) (1994b).

P. Maes, "Intelligent Software", *Scientific American* 273 (3), 187-195(1995a).

P. Maes., "Artificial Intelligence meets Entertainment: Lifelike Autonomous Agents", *Communications of the ACM* **38** (11), 108-114 (1995b).

P. Maes and A. Chavez, "Kasbah: An Agent Marketplace for Buying and Selling Goods", in *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology* (London, UK, 1996).

P. Maes, T. Darrell, B. Blumberg and A. Pentland, "The ALIVE System: Wireless, Full-Body Interaction with Autonomous Agents", *ACM Multimedia Systems, Special Issue on Multimedia and Multisensory Virtual World,* (ACM Press, 1996).

P. Maes and J. D. Velásquez, "Cathexis: A Computational Model of Emotions", in *Autonomous Agents 97*, pp. 678-690 (1997).

T. Maria and K. Oliveira. "ARA: An Agent Robotic Architecture", in *Autonomous Agents 98*, pp. 271-285 (1998).

J. McCarthy, "Recursive Functions of Symbolic Expressions", CACM 3, pp. 184-195(1960).

J. McCarthy and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence", Machine Intelligence 4, 463-502(1969).

J. McCarthy, "Some expert systems need common sense", in Computer Culture: The Scientific, Intellectual and Social Impact of the Computer, 426, edited by P. Heinz (Annals of the New York Academy of Sciences, 1983), pp. 129-137.

F. Menczer and R. K. Belew, "Adaptive Information Agents in Distributed Textual Environments", in *Autonomous Agents 98*, (Minneapolis, MN, USA, 1998).

T. M. Mitchell, "Machine Learning" (McGraw-Hill, 1997).

T. M. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski. "Experience with a learning personal assistant", Communications of the ACM, 37(7), pp.81-91(1994).

Minsky, M., "The Society of Mind", (Simon and Schuster, New York, NY 1986).

D. Mladenić, "Personal WebWatcher: design and implementation", Technical Report IJS-DP-7472, (Department for Intelligent Systems, J. Stefan Institute, 1996).

D. Mladenić and J. Stefan, "Text-Learning and Related Intelligent Agents: A Survey", IEEE Intelligent Systems, pp. 44-54(1999).

A. Moukas, "User modelling in a Multi-Agent Evolving System", in *International Conference on User Modelling '97, Machine Learning in User Modelling Workshop Notes*, (Chia Laguna, Sardinia, 1997).

H. Moldt and T. Wienberg, Multi-agent-systems based on colored Petri Nets, in *Proceedings of the 18th International Conference ICATPN'97*, (Toulouse, France, 1997).

J. Morris and P. Maes, "Negotiating Beyond the Bid Price", in *Workshop Proceedings of the Conference on Human Factors in Computing Systems* (CHI ECS Technical Reports. 26 2000, The Hague, The Netherlands, 2000), pp. 1-6.

T. Murata, "Petri Nets: Properties, Analysis and Applications", in Proceedings of the Institute of Electrical and Electronics Engineers, 77, pp. 541-580 (1989).

D. J. Musliner, E. H. Durfee, and K. G. Shin, "CIRCA: A Cooperative Intelligent RealTime Control Architecture", http://citeseer.nj.nec.com/musliner93circa.html, (1993).

D. Nergenti and A. Poggi, Exploiting UML design of multi-agent system (2001).

N. Negroponte, "The Architecture Machine; Towards a more Human Environment" (MIT Press, Cambridge, Mass. 1970)

A. Newell, J. C. Shaw and H. Simon, "A General Problem-Solving Program for a Computer", Computers and Automation 8(7), 10-16 (1959).

M. Neves and E. Oliveira, "A control architecture for an autonomous mobile robot", in *Proceedings of the First International Conference on Autonomous Agents* (Marina del Rey, CA, USA. 1997), pp193 –195.

N. J. Nilsson, "Problem-Solving Methods in Artificial Intelligence" (McGraw-Hill, New York, NY, 1971).

N. J. Nilsson, "Shaky the Robot", (SRI A.I. Center Technical Note 323, April 1984).

H. S. Nwana, "Software Agents: An Overview", Knowledge Engineering Review, pp. 125-154 (1996).

H. S. Nwana, "An advanced tool kit for engineering distributed multi agent system", Prac. App. Agent Multi-agent Sys (PAAM), pp.377-391 (1998).

H. S. Nwana, ZEUS: "A toolkit and approach for building distributed multi-agent systems", in *Proc. the Third Ann. Conf. On Autonomous Agent.* edited by O. Etzioni et al., pp 360 – 362 (1999).

T. Oates, "A distributed problem solving approach to cooperative information gathering", in *AAAI Spring Symposium on Information Gathering in Distributed Environment*(1995).

Object Management Group, "UML specification version 1.3, June". http://www.omg.org (1999).

J. Odell, "Agent Technology", OMG, green paper produced by the OMG Agent Working Group (1995).

J. Odell and C. Bock, "Suggested UML Extension for Agents, response to the OMG Analysis and Design", Task Force UML RTF 2.0 Request for Information (1999).

J. Odell, "Representing agent interaction protocols in UML", in *Proceedings of Agents 2000* (2000).

J. Ousterhout, "Tool Command Language", http://whatis.techtarget.com/definition/0%2C%2Csid9_gci214196%2C00.html (1994).

Ovum, Ovum Report on 'Intelligent Agents: The New Revolution in Software" , http://www.ovum.com/, (1994).

A. J. Pannu and K. Sycara, "A Learning Personal Agent for Text Filtering and Notification", in *Proceedings of the International Conference of Knowledge Based Systems*(1996).

Y. van Parunak, "Distributed component-centered design as agent-based distributed constraint optimization", AAAI'97 Workshop (1997).

S. Philipps and J. Lind, "Ein System zue Definition und Ausfuhrung von Protokollen fur Multi-Agenten system", Technical Report RR-99-01, DFK1 (1999).

D. Poole, "Application of ICL for multiple agents under uncertainty", Artif. Intell. 94, pp. 7-56 (1997).

S. Rao and M. P. Georgeff, "BDI Agents: From Theory to Practice", in *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95)* (San Francisco, USA, 1995), pp. 312-319.

B. J. Rhodes and T. Starner, "Remembrance Agent: A continuously running automated information retrieval system", in *The Proceedings of the First International Conference on The Practical Application of Intelligent Agents ECS Technical Reports. 27 (PAAM96),* 487-495 (1996).

C. Rich and C. L. Sidner, "COLLAGEN: When Agents Collaborate with People", in Autonomous Agents 97 (Marina Del Rey, California USA, 1997).

A. Reinhardt, "The Network with Smarts", *Byte*, pp.51-64(1994).

J. S. Rosenschein, "*Rational Interaction: Cooperation Among Intelligent Agents*", (PhD Thesis, Stanford University, 1985).

J. S. Rosenschein and G. Zlotkin, "Rules of Encounter: Designing Conventions for Automated Negotiation among Computers" (Cambridge, MIT Press, 1994).

J. Rucker and M. J. Polanco, "Siteseer: Personalized Navigation for the Web", Communications of the ACM 40(3), 73-75(1997).

D. E. Rumelhart, G. E. Hilton and R. J. Williams,"Learning Internal Representations by Error Propagation", in *Parallel Distributed Processing*, edited by D. E. Rumelhart and J. L. McClelland (MIT Press, Cambridge, MA, 1986).

L. E. Sagula, M. F. Puricelli, G. J. Bobeff, G. M. Martin and E. P. Carlos, "GALOIS: An Expert-Assistant Model", in *Autonomous Agents 97* (Marina Del Rey, California USA, 1997).

S. Sakagami, T. Kamba and Y. Koseki, "Learning personal preferences on online newspaper articles from user behaviors", in *6th International World Wide Web Conference*, pp. 291–300 (1997).

R. B. Segal and J. O. Kephart, "MailCat: An Intelligent Assistant for Organizing E-Mail", in "*Autonomous Agents 99*", (Seattle WA USA, 1999).

T. Selker, "Coach: A Teaching Agent that Learns", Communications of the ACM 37(7), 92-99 (1994).

Shoham, Y. and Moshe Tennenholtz: "On Social Laws for Artificial Agent Societies: Off-Line Design", Artificial Intelligence 73(1-2): 231-252 (1995).

A. Silva, M. M. Silva and J. Delgado, "An overview of AgentSpace: a next-generation mobile agent system",

http://citeseer.nj.nec.com/cache/papers/cs/11498/http:zSzzSzberlin.inesc.ptzSzalbzSzpaperszSz1998zSzma98-final-lncs-all-rtf.pdf/silva98overview.pdf (2000). q

A. Sloman, "What sort of architecture is required for a human-like agent?", in Foundation of rational Agency, edited by M. Wooldridge, J. Mueller, and M. Tambe (Kluwer Academic, 1998).

R. G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", *IEEE Transactions on Computers* **C29** (12), 1980.

J. W. Stamos and D. K. Gifford, Remote Evaluation. TOPLAS 12(4): 537-565 (1990)

A. D. Stoyenko, `***SUPRA-RPC***: "SUbprogram PaRAmeters in Remote Procedure Calls", ***Proc. 3rd IEEE Symposium on Parallel and Distributed Processing, Dallas, TX, December (1991).***

K. Sycara, "Intelligent Agents and the Information Revolution", *UNICOM Seminar on Intelligent Agents and their Business Applications* (London, 1995), pp. 143-159.

K. Sycara, "Distributed intelligent agents", IEEE Expert, Special Issue on Intelligent System and Their Applications. 11, 6, 36-46 (1996).

L. Terveen, W. Hill, B. Amento, D. McDonald, J. Crester, "PHOAKS: A System for Sharing Recommendations", Communications of the ACM 40(3), 59-62 (1997).

R. Titmuss, C. S. Winter and B. Crabtree, "Agents, Mobility & Multimedia Information", in *Proceedings the First International Conference on the Practical Application of Intelligent Agents and Mullti-Agent Technology (PAAM '96,* London, 1996), pp. 693-708.

A. M. Turing, "On Computable numbers with an application to the Entscheidungs problem", Proc. London Math. Soc. 42, 230-65 (1937).

A. M. Turing, "Computing Machinery and Intelligence", Mind 59, 433-460 (1950).

M. van Hilst and D. Notkin, "Using role components to implement collaboration-based design", OOPSLA'96, Proceedings of the 1996 Conference on Object-oriented Programming System, Languages, and Applications (ACM Press, 1996), pp. 359-369.

R. van Renesse, K. P. Birman and S. Maffeis, "Horus: A Flexible Group Communications System", http://wilma.cs.brown.edu/courses/cs275/horus.ps, (1996).

J. D. Velásquez and P. Maes: Cathexis: A Computational Model of Emotions. Agents 1997: pp. 518-519 (1997).

M. Veloso, P. Stone, and K. Han, "The CMUnited-97 robotic soccer team: perception and multiagent control", in Autonomous Agents 98, (Minneapolis MN USA, 1998), pp 78-86.

A. Vivacqua, "Agents for Expertise Location", in Autonomous Agents 98, (Minneapolis MN USA, 1998).

G. L. von Trzebiatowski, "The role concept for agent in multi-agent system", in *Fundamenta Informaicae* 43 (1-4), pp. 215-226 (2001).

T. Walsh, N. Paciorek, and D. Wong, "Security and Reliability in Concordia", in *Hawaii Int. Conf. Sys. Sci. (HICSS'31),* (Hawaii, USA, 1998).

J. E. White, Telescript technology: The foundation for the electronic marketplace. White paper, General Magic, Inc., 2465 Latham Street, Mountain View, CA 94040. (1994)

J. E. White, "The Telescript Language Reference", http://www.science.gmu.edu/~mchacko/Telescript/docs/TSLangRef_TableofContents.html. (1995).

T. Wittig, *ARCHON: An Architecture for Multi-Agent Systems*, (London: Ellis Horwood, 1992).

M. Woodldridge and N. R. Jennings, "Intelligent Agents: Theory and Practice. Knowledge Engineering", Review. 10(2), (Cambridge University Press, 1995a), pp115-152.

M. Wooldridge and N. Jennings, N, *Intelligent Agents*, Lecture Notes in Artificial Intelligence 890, (Heidelberg: Springer Verlag, 1995b).

M. Wooldridge, "Conceptualising and Developing Agents", In *Proceedings of the UNICOM Seminar on Agent Software,* (London, 1995), pp. 40-54.

M. Wooldridge, J. P. Mueller and M. Tambe, "*Intelligent Agents II, Lecture Notes in Artificial Intelligence 1037*", (Heidelberg, Springer Verlag, 1996).

M. Woodldridge and N. R. Jennings, "Pitfall of agent-oriented development", in *Proceedings of the Second International Conference on Autonomous Agents*, (Minneapolis/St. Paul, MN, USA, 1998), pp. 385 – 391.

M. Woodldridge, N. R, Jennings and D. Kinny, "The gaia methodology for agent-oriented analysis and design", Journal of Autonomous Agents and Multi-Agent Systems 3(3), 285-312 (2000).

D. Xu, "A formal architectural model for logical agent mobility", in *Proc. of IEEE Inter. Conf. Systems, Man., and Cyber.,* pp. 3177-3182, 2000.

D. Xu, and S. M. Shatz, "A framework for modeling agent-oriented software", in *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS),* (Phoenix, Arizona, 2001).

D. Xu, "Modeling and verifying multi-agent behavior using predicated/transition nets", *SEKE'02*, (Ischia, Italy, 2002).

G. Zlotkin and J. S. Rosenschein, "Negotiation and Task Sharing among Autonomous Agents in Cooperative Domains", in *Proceedings of the 11th IJCAI* (Detroit, Michigan, 1989), pp. 912-917.

Appendix

AgentSpace: A Next Generation Mobile Agent System

Nowadays there are many MAS/MAsS proposals (Table 3.1). Among of them, Telescript, IBM Aglet, AgentOS, Agent-Tcl, Odeyssey, Concordia, and Tacoma are the most representative ones. They serve roughly the same purpose and present a common set of functionalities. They also present some important technical and even conceptual differences. For instance, Telescript used to be the usual reference for MAS developed at a highly technological level (White, 1994). However, it is a proprietary system with a difficult-to-learn programming language, and badly suited for a dynamic and open environment such as the Internet. On the other hand, some systems such as ffMAIN are language and system independent. However, it shows severe limitations on the overall performance and difficulties in developing complex applications. IBM Aglets provides the current reference on Java-based MAS, but it lacks an elaborate object model, e.g., it does not have the notion of execution hierarchically and management of agent families and clusters. The rest of the MAS/MAsS in Table 3.1 are built for specific applications, or limits on particular platforms. Thus they lack a common base to be flexible, open, reliable, secure and efficient agent-based applications (ABA).

In order to support the development and execution of such a MAS/MAsS, they should incorporate a good combination of features from existing MAS/MAsS. For example, they should support the independence protocol of TCP/IP and/or HTTP-based MAS/MAsS. AgentSpace is one of such systems, which has been developed to meet these requirements.

AgentSpace Concept

AgentSpace is a Java-based agent framework. It has been built on the top of Voyager from ObjectSpace, which is the agent platform built in 1997 for a MAsS system. As a MAS/MAsS framework, the AgentSpace has it's own conceptual model/architecture. It is composed by three complementary components: AES, for Agent Execution System; ACS, for Agent Class System; and AEE, for Agent Execution Environment. These three sub-systems provide the "building blocks" of any type of

MAS/MAsS. The AgentSpace group is now working on other relevant sub-systems / components to develop, manage and monitor agent-based applications. Inside the ACS, it contains basic agent class, which fits the features of the basic agent abstraction provided in Figure 1.1. This means that it can be used to build agents required by both the AI and SW engineering societies.

AgentSpace Features

The AgentSpace defines ABA as a dynamic, potentially large-scale distributed application in an open and heterogeneous context such as the Internet. The basic conceptual unit for designing and building ABAs is the agent (atomic unit). The AgentSpace understands the notion of ABA.

ABA applications, built/run on the AgentSpace, have a number of characteristics and requirements that have been dealt with independently as early as at its design phase. It is their combination that poses problems;

- *Autonomous*: Each user creates and maintains his/her own agents using his/her own resources and/or using resources from others.
- *Heterogeneous*: Each user has bought, got used to and used different interfaces, machine architectures, programming languages, database systems, communication packages, operating systems and so on.
- *Open*: Some agents may depend on other agents and applications, even from external organizations. This means agents will have to inter-operate with other (legacy) information systems (applications, databases and so on).
- *Dynamic*: Agents are added, updated and removed at any time without prior notice. They will have to cope with unavailability, new interfaces, oscillating bandwidths, and other variable characteristics.
- *Robust*: Agents can tolerate different kinds of failures on machines, networks or at any level of software applications.
- *Secure*: The system provides different levels of security depending on each particular part of the whole application. There is public, place-specific, and administrative access control unit.

NOTE TO USERS

Page(s) not included in the original manuscript and are unavailable from the author or university. The manuscript was scanned as received.

pages #176

This reproduction is the best copy available.

UMI®

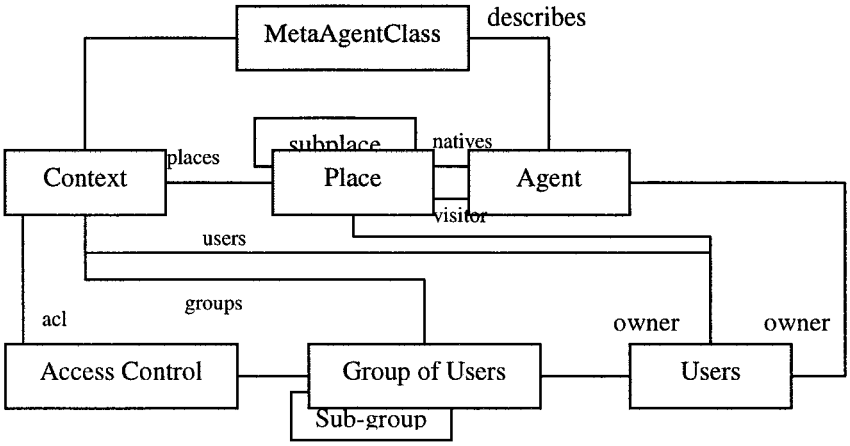

Figure 1. UML class diagram of AgentSpace

Agent Transport Protocol

AgentSpace supports multiple transport protocols that mean the TCP and/or HTTP. It is well suited for applications running on an Internet.

Agent Communication

AgentSpace applies the agent communication architecture from that used in Voyager framework, which is very flexible. It provides for asynchronous, synchronous, and in the future, remote methods. References to remote virtual objects can be passed as parameters to methods and can be serialized. Method calls across address spaces have the same semantics as local method calls; they are fully polymorphic, allowing access to local and remote objects through virtual object proxies using exactly the same syntax.

AgentSpace Help and Tools

AgentSpace provides good help and development tool kits, which make the framework easy to learn and use. A good help example is the "VirtualShop" application, which is composed by an open and dynamic set of shoppers. These shoppers want to advertise and sell their specific products. A trusted company manages the VirtualShop. In this sample application, the perspective programmer is guided through the following steps:

- How to create agents;
- Agent creation from another agent;

- Obtain reference to agents;
- Agents navigate;
- Agents communicate.

Summary

AgentSpace is a Java framework to support, manage and develop future ABAs. This MAS/MAsS framework provides most of the features required by current agent framework:

- It supports dynamic and distributed applications. Creation of agents is in a transparent, clean and easy process;
- It provides a very extensible and elegant way to handle security policies related to the access and interactions between the agent and host users, and between the agents themselves;
- It provides a well-integrated association between users and agents/locality;
- It support subject concepts, which is inherited from the technology of Voyager;
- It supports multiple platforms due to Java itself with its recent features, namely reflection, dynamic class loaders and object serialization.
- It provides a generic "Agent Manager Tool" with GUI and a prototypical electronic payment system based on SET.

Another good thing for AgentSpace is the fact that it is built on the top of Voyager Project. The Voyager project is a well-established agent framework project, which started in the mid-1996. It provided the first ObjectSpace press release on 2 April 1997, and shipped their first evaluation release later in 1997. It supplies both source code and tools, which are very helpful. A great deal of documentation (User Guide, Technical Overview and comparison white paper) is available for Voyager via both the distribution and it's web site, and most of them are referential to AgentSpace. Voyager comes also with several dozens examples, most of which are trivial, but well focused, short pieces of code that highlight a particular aspect of the system. These samples are also applicable to AgentSpace.

There are many research groups and companies riding the current wave of popularity of MAS/MAsS. Each wants to be the one that provides the breakthrough system we all end up using. It is hard to be there

since the ABA is diverse, and the MAS/MAsS is still at the stage of development. I name the AgentSpace as the "most representative" one based on the following:

- Availability: The AgentSpace is commercially available now, and it shows some industry strength with applications in the E-Commerce;
- The feature set is almost completed, which meets the ABA requirement;
- It is well documented for its architecture, model and communication and installation, and it is relatively easy to operate.

There are two aspects, in which AgentSpace is a bit weak: 1) agent interactions; 2) system security when an agent migrates across organizations (inter-organization migration) in a corporate network. These are two common issues faced by all MAS/MAsS.

Reference

ObjectSpace: Voyager Core Package Technical Overview (1997)

ObjectSpace: Voyager and Agent Platforms (1997).

White, J., "Mobile Agent System White Paper", (General Magic Inc., 1994). http://www.genemagic.com/agents/Whitepaper.html