From Computer Networks to Agent Networks

Guoqiang Zhong Ken'ichi Takahashi Satoshi Amamiya Daisuke Matsuno
Tsunenori Mine Makoto Amamiya

Department of Intelligent Systems
Kyushu University
Kasuga, Fukuoka, Japan 816-8580
{zhong, tkenichi, roger, matsuno, mine, amamiya}@al.is.kyushu-u.ac.jp

Abstract

From the 1990s on, one of the most important challenges facing computer science researchers has been the design and construction of software tools to exploit Internet computing. At the same time, the development of agent technology has gone hand in hand with the explosion of the Internet. As worldwide network computing environments become more and more complex, software agents are believed to have the potential to help present and manage the Internet in an autonomous or semi-autonomous way. Yet, to date, a number of fundamental questions about the theory and practice of this new software engineering paradigm have remained unanswered. Here we will explore the features that make the agent-based approach such an appealing and evolutionary computational model. In particular, we envision a global agent-based distributed computing architecture that provides a convenient programming abstraction and sufficient transparency.

This paper gives a general introduction to the underlying concepts of our research and development both at the level of design philosophy and in practical implementation techniques. It will be argued that the shift from computer networks *to* agent networks *is a significant extension of network programming technology because agents are well suited to modeling, designing and implementing scalable, flexible and secure distributed systems over a worldwide computing environment.*

1 Introduction

As early as 1980s, Sun Microsystems declared a singular vision — *The network is the computer*$^{\text{TM}}$. Over the past two decades, the performance of semiconductor products doubles approximately every eighteen months (commonly referred to as Moore's law) and the size of the Internet doubles approximately every year (more than one hundred and forty million hosts have been connected to the Internet by 2002). In contrast to the rapid growth of the global computer network, a global software infrastructure is still lacking.

Currently, TCP/IP is the *de facto* standard in the Internet, and services on the application layer are described in terms of transport protocols directly on top of the TCP/IP stack. Some well-known examples include World Wide Web (HTTP), electronic mail (SMTP), file transfer (FTP), virtual terminal (TELNET), etc. The drawback of this approach is that each application must implement all additional services themselves, such as naming service, message routing, fault tolerance, QoS (quality of service), security, and so on. Consequently, a lot of efforts are repeatedly spent on implementing common services, and interoperability between different applications is difficult, if not impossible. Even worse than this, those tailor-made protocols are hard to be enhanced or replaced because they require worldwide changing of independently written implementations.

Our solution lies in the development of a *global distributed computing architecture* based on agent-oriented programming paradigm. Rather than developing applications directly on top of the transport layer, agents are basic building blocks from which diverse application systems can be abstracted, organized and constructed. Through our work, we concentrate on enabling technologies for building a scalable, flexible and secure software agent layer that provides us with a convenient programming model, sufficient transparency and interoperability. So far both the design and implementation of the key abstraction models have been carried out. Furthermore, practical agent-based systems have been implemented and applied to real-world problems.

The remainder of this article is structured as follows. Section 2 addresses the core concept behind our programming model. Section 3 concentrates on the approaches we take to formalize the agent-based global software infrastructure. Section 4 covers some details of the implementation.

Section 5 presents a case study of an Internet-based faculty information retrieval system built for our university. Finally, we conclude with a vision of the future of our research.

2 Basic philosophy

Nowadays, *software agent* (or *agent* for short) is a hot buzzword that it has been advocated by many researchers and software developers as a promising and innovative way to model, design and implement complex distributed systems. The key notion behind this new paradigm is a *self-directed behavioral structure* [2]. In effect, it is a new programming tool that emphasizes the idea of interaction, as well as the idea of choice and options at the time of action, rather than at the time of programming. This shift from algorithms to online interaction is a consequence of the progression of programming technology because it reflects current computing reality more closely than *Turing Machines* [7].

Agent technology has been used in a wide range of areas from industrial and commercial applications to entertainment and medical applications [8]. The primary driving force behind our research is to integrate agent technology with the evolution of the Internet. In particular, we have noticed that the key concepts of *agent-oriented programming* (AOP) are *online communication* (both agent-to-agent and agent-to-user), the built-in *self-directed behavioral structure* and *society organization*. We believe that these three aspects of AOP make it suitable for developing applications in worldwide network computing environments.

However, building complex agent-based systems, which can operate properly in an open environment, requires not only new technologies but also new methodologies to support developers in analyzing, designing and developing such systems. This is especially important if agent technology is to break out of being a niche technology used by the few, to become something that will succeed as a mainstream software engineering paradigm [9].

Through our work on the KODAMA (Kyushu University Open & Distributed Autonomous MultiAgent) project [6], we have established a *separation* principle that mandates the separation of application-level logic from agent-level logic and the separation of agent-level logic from network-level logic. In this way, the inherent distribution and complexity that a network computing environment involves can be abstracted, encapsulated and tackled appropriately at different levels.

In an instance of the idea that "the network is the computer," agent technology emphasizes the ability to reach out, discover and interact with others. Such a emphasis raises a number of challenging issues that all are centered around an elementary question of *what*, *when* and *how* to interact with *whom*. The biggest contribution of the above-mentioned separation principle is to introduce a layered solution. With which logic at each layer specifies the way how services are offered, including some services performed for the layer above it. In particular, while higher-level (i.e. application-level and agent-level) logic deals with the problem of *what*, *when* and with *whom* to interact, low-level (i.e. network-level) logic concentrates on the problem of *how* to interact. Some examples of logic at different layers are given in Table 1.

Table 1. Examples of logic at each layer

application	problem solving
	choice and options
	self-learning
agent	interaction pattern
	social relationship
	name addressing (agent world)
	security policy (agent world)
network	agent name resolution (network world)
	agent message deliver
	quality of service management
	network security (network world)

By detaching data exchange activities from agent programmes, it is possible to build generic agent communication facilities in the network layer. This helps improve productivity in several ways.

- First, agent programmers can focus on higher-level abstractions only, and leave the low-level details of communication to network layer programmers. They do not need to write special software to move data between each possible pair of agents.
- Second, agent development is not restricted to a particular architecture, thus keeping the entire system flexible. Agents, for example, can remain unchanged while the network layer are reconfigured or updated and vice versa.
- Third, low-level logic handles agent communication traffic without understanding the applications that use it. The specification of the network layer can be publicly available and it accommodates a wide variety of underlying hardware, communication technologies and contents of agent communication.

It is important to note that both agent-layer logic and network-layer logic are application independent. The main purpose of having these two levels, therefore, is to integrate a set of common distribution services that forms a uniform developing platform upon which various applications can be efficiently built with inherent support of cooperation.

Table 2. Layered architecture

level	layer	protocol/language
application	*agent application unit*	open
agent	*agent kernel unit*	agent communication language
network	*agent infrastructure* (middleware)	agent platform protocol
	transport layer (the Internet)	TCP/IP protocol suite

3 Agent-based approach

From machine language and high-level languages in the 1950s to structured and object-oriented programming (OOP) in the 1970s and 1980s, and distributed objects, design patterns, and software components more recently, we can always see a struggle between free, open complexity and the need to control that complexity with structure. At this point, agents are the next step in the progression to a higher level of structure in programming technology.

On analogy with object-based concurrent programming (OBCP), our approach is based on *agent-based concurrent programming* (ABCP). Basically, agent-based systems are more difficult to correctly design and implement than other non-agent systems. This is mainly because agents are computational entities that perceive their environment through sensors and act upon their environment through efforts. In this diagram, three aspects of an agent are identified.

- An agent has to have a repertoire of possible actions available to it which constitute its ability to modify its environments.
- An agent has to have some interaction with the world around it and get feedback about its choice, whether it is successful or not. Interaction can take place indirectly through the environment (e.g., by carrying out an action that modifies the environmental state) or directly (e.g., by exchanging information with other agents) through a shared language.
- An agent has to have a continuous operating engine that persistently strives for success without any need to account at the outset for all possible conditions it will face or in what order it will face them. What programmers provide to an agent is a goal or a utility function and maybe a collection of building blocks for getting there and, in some advanced cases, maybe even a way to learn or remember something new that is needed.

3.1 Layered architecture

In accordance with the basic philosophy introduced in Section 2, a layered architecture has been adopted. Furthermore, a *plug-and-play* standard has been deployed to separate the application-level logic from the agent-level logic within an agent. More precisely, an agent is made up of a *kernel* unit, which encapsulates the common modules, and an *application unit*, which encapsulates the application-dependent modules [10]. In this way, the implementation aspects of data sharing, exchange and management among agents are made transparent to application programmers.

On the other hand, building agent-based systems in a network computing environment is challenging, time-consuming and costly. The network itself, for example, is full of latencies, congestion, overload and unforeseen failure. In the layered architecture, there is a specific layer, we call it *agent communication infrastructure* (or *agent infrastructure*, *infrastructure* for shot), on which agents running in different network spaces can communicate with each other easily, freely and without concern about the network issues. Actually, this is another separation between agent-level logic and network-level logic.

In sum, from the highest layer to the lowest level, the layered architecture consists of *agent application unit* layer, *agent kernel unit* layer, *agent infrastructure* layer. The relations among the separation principle (see Section 2), the layered architecture, the transport layer and examples of the underlying protocol or language for each layer are summarized in Table 2.

3.2 Agent layer

Not surprisingly, the *agent* layer stays on the top, which can be further divided into application unit sub-layer and kernel unit sub-layer. Typically, all agents speak one or more languages called agent communication language (ACL). ACL itself only specifies the format, or syntax, of the information being transfered. The meaning, or semantics of the information on the other hand, is specified by application logic. Such considerations are also reflected in our plug-and-play architecture of agents. That is, the basic support of ACL, such as message validating and message parsing is integrated into the kernel unit. On the other hand, message contents and message interpretation are open to different application units. As a consequence, once the kernel unit is finished, it can be used by all agents, whereas various application units need to be developed to meet the exact requirements of various applications.

Table 3. Agent/Infrastructure interface (registration and un-registration)

method	time	parameter	return value
`register`	creation, immigration	agent logical name (in agent layer)	message queue
			`false`
`remove`	deletion, emigration	agent logical name	`void`

Table 4. Agent/Infrastructure interface (agent message delivery)

method	time	parameter	return value
`put`	send message	agent message	`void`
`get`	receive message	`void`	agent message

However, agent interaction is more complex than interactions in conventional models, such as client-server or publish-subscribe. This is because data is transmitted among agents (sometimes users), regardless of whether a prior relationship exists. At the time of design, for example, nobody knows how many agents will be created, where agents will reside, or what agents will do. Rather, in a practical, worldwide, distributed agent-based system, interaction may occur at unpredictable times, for unpredictable reasons, between unpredictable components [4].

To support the scenario of agent communication, all agents are usually logically organized and located into various *agent communities*, which in turn may be linked together to form a unified agent society. As a consequence, an agent system can be divided into a number of top-level communities and each one may cover many agents or be partitioned into sub-communities. In this way, any agent can be uniquely located and named by giving its community position. It is also possible for agents to join two or more communities concurrently, so that they can be found in different communities. Agent communities have a dynamic membership because agents can join or quit from time to time. Social networks of agents, as a whole, can develop in an evolutionary fashion.

3.3 Agent infrastructure layer

Next comes the *agent infrastructure* layer, which plays a vital role in connecting the agent layer with the network layer. Such connection is guaranteed and realized by two well-defined interfaces, one between the agent layer and this infrastructure layer, and the other between this infrastructure and the network layer.

We consider this layer as a necessary extension of the network transport layer, which provides additional services (beyond transport service) tailor-made for agent communication over networks. Some examples of such services include *agent name resolution*, *agent message delivery*, *mobile computing support*, *quality of service management* and *information security*.

This layer, inserted between the technology-dependent communication mechanisms and agent systems, will absorb the variety and complexity of communication processes, and make the collection of agent systems appear to be a single large-scale and open system.

In practice, agent are self-contained entities that rely on the agent infrastructure layer to provide transparent support for network communication. Once agent messages are passed from agents to the agent infrastructure, they should be delivered to their destination without further interaction with agents. With the relationship and interaction between the two layers clear, it is natural to define the contents of their interface. Two examples are given in Table 3 and Table 4.

Till now, most existing agent systems have relied on some available protocols like CORBA, RMI or IIOP as their network communication protocols. Since those protocols are not tailor-made for the use of agent systems, it is up to the agents to deal with the details of network implementation. This is neither desirable nor allowable under the separation principle.

To systematically regulate the network-level communication processes within the agent infrastructure, a new network protocol, called *agent platform protocol* (APP), has been designed. APP plays a vital role in enabling the universal coordination among heterogeneous agents by realizing agent information transfer across the network independent of the data to be transmitted and the places where agents reside [5].

3.4 Transport layer

The underlying layer below the agent infrastructure is the transport layer in which the transport service is con-

Table 5. API for creating a new agent

method	parameter	return value
`newAgent`	`void`	a default agent with a random name
`newAgent`	`name`	a default agent with given name
		`null` (name conflict)
`newAgent`	`name, plugin`	an agent with given name and `plugin`
		`null` (name conflict)
`newAgent`	`au`	an agent with given `au` (application unit) and a random name
`newAgent`	`name, au`	an agent with given name and `au` (application unit)
		`null` (name conflict)

cretely realized. The interface between the agent infrastructure layer and this layer follows the *socket* API (Application Program Interface). Originally a part of the BSD UNIX operating system, the socket API is now available for many operating systems, and is the *de facto* standard for the interface between an application program and the network communication protocols [12].

In a layered system, each layer provides services to the layer above it and serves as a client to the layer below it, and interfaces between adjacent layers determine how layers will interact [13]. Usually, interfaces are well-defined so that higher layers are hidden from lower ones. This approach has several desirable properties.

- First, it supports designs based on increasing levels of abstraction.
- Second, it dramatically simplifies system enhancement and maintenance. Changes to the function or interface of one layer, for example, affect at most two other layers (below and above).
- Third, it supports component reuse. Whenever the interfaces are the same, different implementations can be built and used interchangeably.

4 Implementation

We have examined our design principle and agent-based approach in the previous sections. In this section, we briefly present our implementation details. In particular, the implementation of agents and an instance of agent infrastructure is discussed.

4.1 Agents

According to the plug-and-play standard, agents are implemented in two sections: the kernel unit section and the application unit section. Both kernel and application units consist of components. And they can be developed, extended and updated in relative isolation and then be merged into an agent. In particular, the basic building blocks of application units are various *plug-in*s which can be seamlessly plugged into kernel units. The API for creating a new agent is given in Table 5.

To keep agents acting concurrently, each of them runs on at least one independent thread and they do not have any hard-wired connection. Instead, agents keep exchanging messages with the outside through the pre-defined agent/infrastructure interface (see Table 4). In this way, agents have full control over both their behaviour and states. An agent, for example, can only ask other agents to perform certain actions by sending them messages. Whether or not they comply, however, is decided by the message recipients. This is intrinsically different from the object-oriented programming paradigm in which objects are generally passive and fixed.

4.2 Agent infrastructure

In this subsection, we discuss our implementation of the agent infrastructure, known as *agent communication zone* [11], as well as the implementation of two tables, namely *local agent table* and *remote agent table*.

4.2.1 Agent communication zone

In our implementation, agent communication zone (ACZ) must be installed on every machine which might have agents. For convenience, those individually installed ACZs are called *local ACZ*s. Every local ACZ maintains dynamic communication channel(s) through which remote ACZ(s) can be accessed. Conceptually, all local ACZs work together to form a unified ACZ, which actually is an instance of the agent infrastructure.

Specifically, a local ACZ is identified by an IP address of its host machine, a separator colon, and a port number. Let us look at an example. The ACZ identifier of a local ACZ installed on a host whose IP address is 52.14.159.78

and which is bound to port 7070 is defined as follows:

$$\overbrace{\underbrace{52.14.159.78}_{\text{IP address}} : \underbrace{7070}_{\text{Port}}}^{\text{ACZ Identifier}}$$

It is possible, however, for one host machine to have more than one local ACZ and for each of them to bind to a unique port number. In our example above, if there is another local ACZ installed on the same host machine but bound to a different port, say 7071, its ACZ identifier will be defined as:

52.14.159.78 : 7071

The merit of this host/port scheme is that once the ACZ identifier is known, it is possible to make a new communication channel to that local ACZ. It is worth noting here that two local ACZs are considered identical if and only if they have the same identifier.

With the ACZ identifier definition in place, it is possible — in the world of the ACZ — (i) to exchange network address information concerning host machines, and thus (ii) to build new network communication channels through which agent information and messages can be exchanged.

4.2.2 Local agent table

As we know already, the main function of the infrastructure layer is to route agent messages from the source agent to the destination agent. This makes it necessary to exchange address information among local ACZs. To meet this need, each local ACZ in our implementation has two tables, namely a *local agent table* (LAT) and a *remote agent table* (RAT), which are in charge of the management of local agents and remote agents, respectively.

According to the agent/infrastructure interface (see Table 3 and Table 4), when an agent is created (or immigrates from another machine), it registers its name with the agent infrastructure and in return receives a *message queue* (which is the only message I/O of the agent). Thus, it is necessary for a local ACZ to maintain a LAT that keeps records of this kind of registration information. Normally, the key to the table is an agent name and the value of the table is a message queue. Table 6 gives an example of local agent tables.

The records in a LAT should strictly reflect the state of local agents. That means that any change in local agents (e.g. joining or leaving of agents) will result in changes in the local agent tables. If agent f leaves ACZ1 and joins ACZ2, for example, then its record in ACZ1's LAT is deleted, and a new record is added to ACZ2's LAT.

Table 6. An example of local agent tables

local ACZ	key:agent name	value:message queue
ACZ1	g	`messageQueue@g`
	f	`messageQueue@f`
ACZ2	s	`messageQueue@s`
	t	`messageQueue@t`
ACZ3	m	`messageQueue@m`
	p	`messageQueue@p`

4.2.3 Remote agent table

The situation for remote agent tables, on the other hand, is different from that of local agent tables. Instead of the *message queue*s of agents, ACZ identifiers are recorded in RATs. Through these, remote agents can eventually be found. Normally, the key to the table is an agent name and the value of the table is an ACZ identifier. However, the records in a RAT are unpredictable because they depend on on-line exchange of agent addresses. Extending Table 6, Table 7 gives an example of remote agent tables.

Table 7. An example of Remote Agent Tables

local ACZ	key:agent name	value:ACZ ID
ACZ1 (`ACZ1-ID`)	t	`ACZ2-ID`
	s	`ACZ2-ID`
ACZ2 (`ACZ2-ID`)	m	`ACZ3-ID`
	p	`ACZ3-ID`
	g	`ACZ1-ID`
ACZ3 (`ACZ3-ID`)	f	`ACZ1-ID`
	g	`ACZ1-ID`
	t	`ACZ2-ID`
	s	`ACZ2-ID`

As discussed earlier, the implementation of agents can be separated from the implementation of agent infrastructure. The only constraint is that they both follow the predefined agent/infrastructure interface. Given this condition, developers are free to implement their own agents and agent infrastructure. While agent implementation focuses on key processes such as problem-solving, planning, decision-making, interacting and learning, agent infrastructure implementation has its focus on a infrastructure tool that provides a universal communication environment for agents.

5 A case study

To ensure that the new approach we propose is suitable for developing distributed applications, several experimen-

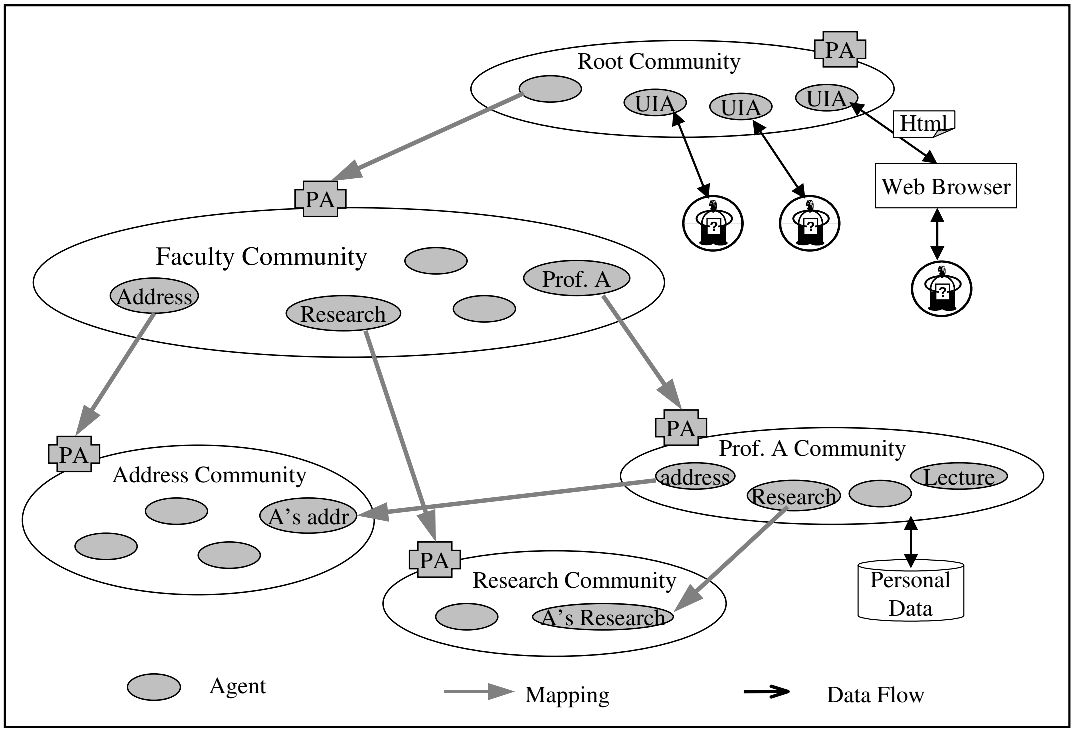

Figure 1. Agent-based faculty information retrieval system

tal systems have been developed and evaluated. One of them is a distributed Web searching system [3], and another one is a chat system [5] based on KODAMA and JADE [1] agents. As a case study of the application development, an Internet-based faculty information retrieval system is discussed in this section.

5.1 Faculty information retrieval system

There is an ongoing project in our university to publish the research reports of university faculty on the Web. Basically, the research report of a member of the faculty is a HTML file that contains a number of category/value pairs. A professor is free to write his or her research report by filling out the pre-defined set of category/value records with his or her own data. These research reports and related information resources are naturally distributed over the Web servers of laboratories, departments, schools, and the university administration. Therefore, anyone can access the on-line research information by using Internet browser.

Normally, a local search engine is needed to assist users to navigate as they browse organization so that they can find the desired information quickly by giving their queries to the search engine. Here, an agent-based information retrieval system has been designed, implemented and tested.

As illustrated in Figure 1, the basic components of the system are agents, who can enter or leave the agent communities by registering or un-registering themselves with portal agents (PA) at run-time. Without exception, all agents are organized in a hierarchical social structure and they work together to form a consistent, interoperable, robust environment in which intelligent information access is possible.

The *Faculty Community*, for example, is a gathering place for various kinds of agents concerned with the faculty. It has an agent for every member of the faculty, an agent for the address book, an agent for the research and so on. Any agent in the faculty community (e.g. the agent of professor A) is partitioned into a sub-community (e.g. Professor A Community). Furthermore, agents are allowed to join more than one community (but not sibling or descendant communities). The *address* agent of professor A, as a member of the *Professor A Community*, can join the *Address Community* to share its information with other address agents. The agent relationships, furthermore, can be changed. In the case of category agents, for example, their social connection can be used to reflect either explicit or implicit relations among various categories. Rather than relying on centralized control, the management of agents is accomplished by many distributed portal agents.

In our demonstration system, users can query various categories through JSP (JavaServer Pages) in which *user interface agents* (UIAs) are integrated. UIAs do not need to know either the location or the content of available online information resources. Indeed, they can join the root community and forward queries to the *faculty agent*, which in turn tries to find the appropriate agent with relevant information through run-time interaction. If one or more agents are found, they try to complete the query, calculate the confidence, and send the result back to the UIAs which in turn display the result with JSP. In the near future, we are going to load this faculty information retrieval system into our Web server so that the demonstration can be shown through the Internet.

This system can also develop in an evolutionary fashion. When a new agent-based system is implemented, it can easily be added to the current system by registering its portal agent with the *root community*. If the system becomes too big to be managed by one root community, several root communities can be created and they all can be gathered in a higher root community. It is also possible for heterogeneous agents to join this system, if the agent infrastructure they have installed is ACZ (the one used in our system) and the ACL those agents speak can be understood by the agents we implemented.

5.2 Experiments and results

To compare our system (agent-based) with the old system, a group of users have been asked to search faculty information on both systems and fill up a questionnaire.

Table 8. Experiment results (free keywords)

keyword	precision (top 10 answers)	
	agent-based	old system
agent	0.5	0.3
psychology	1.0	0.6
biotechnology	0.7	0.7
image processing	0.6	0.8
distributed processing, parallel processing	0.7	0.0
parallel and distributed processing	0.7	0.3
operating system	0.4	0.3
fine-grain multi-thread	0.2	0.2
artificial life	0.3	0.2
java, network	0.4	0.1
language processing	0.4	0.7
statistics	0.7	0.9
average	0.55	0.4

In particular, two sets of experiments have been carried out.

- First, users are free to give out their voluntary keywords and do same retrieval on both systems. Since the number of returned answers may be very large, users are asked to calculate the precision rate based on the top ten answers.

- Second, users are given a fix list of keywords and do same retrieval on both systems. Similarly, they are asked to calculate the precision rate based on the top ten answers.

Table 9. Experiment results (given keywords)

keyword	precision (top 10 answers)	
	agent-based	old system
electron	0.625	0.65
gene	0.925	0.85
creature	0.825	0.25
science	0.575	0.375
cell	0.95	0.775
capital	0.75	0.875
trade	0.8	0.6
pollution	0.5	0.275
breeding	0.725	0.625
grammar	0.625	0.575
average	0.73	0.585

The results of experiments are given in Table 8 and Table 9 respectively. As the result data shows, the agent-based system has higher precision rates than the old system either in the case of free keywords search or in the case of given keywords search. However, the comparison of recall rates of both systems has not been carried out according to the limited time and the large amount of faculty information.

Users' comments written in the questionnaire also give us some useful feedback from different aspects. Most of users, for example, think that some new functions added in our system are very helpful. Such functions include searching by categories, searching options setting and weighting answers. On the other hand, they also criticize that the system should be more efficient and robust.

6 Conclusions

This paper gives a general introduction to our work on a global agent-based software architecture. In particular, we have given a quick glance at the core motivation and objectives of our research, at the system design, and at some implementation techniques. At time of writing, we just completed the primitive research and development cycle. We expect that this cycle will be iteratively repeated, extending the work presented in this thesis to the solution of many real-world problems. Through our work, we are convinced that agent-based approaches to information management are both scalable and cost-effective.

7 Acknowledgments

This work is funded by the Telecommunications Advancement Organization of Japan (TAO).

References

[1] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE a FIPA2000 compliant agent development environment. In *Proc. of the 5th International Conference on Autonomous Agents*, pages 216–217, May 2001.

[2] Les Gasser. Agents and concurrent objects. *IEEE Concurrency*, 6(4):74–77, 81, October-December 1998. Interviewed by Jean-Pierre Briot.

[3] Tarek Helmy, Satoshi Amamiya, and Makoto Amamiya. Collaborative KODAMA agents with automated learning and adapting for personalized web searching. In *Proc. of Thirteenth Innovative Applications of Artificial Intelligence*, pages 65–72, August 2001.

[4] Nicholas R. Jennings. Agent-based computing: Promise and perils. In *Proc. of Sixteenth International Joint Conference on Artificial Intelligence*, pages 1429–1436, July 1999.

[5] Ken'ichi Takahashi, Guoqiang Zhong, Daisuke Matsuno, Satoshi Amamiya, Tsunenori Mine, Makoto Amamiya. Interoperability between KODAMA and JADE using agent platform protocol. In *Proc. of Autonomous Agents and Multi-Agent Systems 2002*, pages 90-94, July 2002.

[6] Guoqiang Zhong, Ken'ichi Takahashi, Satoshi Amamiya, Tsunenori Mine, Makoto Amamiya. KODAMA Project: from Design to Implementation of a Distributed MultiAgent System In *Proc. of Autonomous Agents and Multi-Agent Systems 2002*, pages 43-44, July 2002.

[7] Peter Wegner. Why interaction is more powerful than algorithem. *Communications of the ACM*, 40(5):80–91, 1997.

[8] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, USA, 2000.

[9] Franco Zambonelli, Nicholas R. Jennings, Andrea Omicini, and Michael Wooldridge. Agent-oriented software engineering for Internet applications. In *Coordination of Internet Agents*, pages 326–346. Springer-Verlag, New York, USA, 2001.

[10] Guoqiang Zhong, Satoshi Amamiya, Ken'ichi Takahashi, Tsunenori Mine, and Makoto Amamiya. The design and implementation of KODAMA system. *IEICE Transactions on Information and Systems*, E85-D(4):637–646, April 2002.

[11] Guoqiang Zhong, Tsunenori Mine, Tarek Helmy, and Makoto Amamiya. The design and implementation of agent communication in KODAMA. In *Proc. of the International Conference on Artifical Intelligence*, volume 2, pages 673–679, June 2000.

[12] Douglas E. Comer. *Computer Networks and Internets*. Prentice Hall, New Jersey, USA, 1997.

[13] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.