OOPSLA 1993

Addendum to the Proceedings

Washington, DC—26 September-1 October, 1993

Invited Address

The Society of Objects

Mario Tokoro
Keio University &
Sony Computer Science Laboratory Inc.

Abstract

In this paper, I will first review the notions of objects and concurrent objects and discuss their main roles. Then, I will introduce two observations on our current computer systems and explain why we need an evolved notion of objects, which we call *autonomous agents,* to describe *open* and *distributed* systems. An autonomous agent is a software *individual* that reacts to inputs according to its situation and its goal of survival. A collection of such autonomous agents shows emergent behaviors which cannot be ascribed to individuals, eventually forming a *society.* Research into achieving a society of autonomous agents being carried out at Sony Computer Science Laboratory and Keio University will then be presented. In the last section, I will speculate about yet-to-be-realized computational modules called *volitional agents,* that could be used to create safe, evolutionarily stable, cohabitating society.

1 Introduction

Human society is characterized by a dichotomy between individuals with their goals and aspirations, and the emergence of collective behavior that cannot be ascribed to individuals. Intriguingly, computational systems and environments are beginning to exhibit some of the collective behavior that is characteristic of society.

As high speed communication networks proliferate, every computer and, thus, every software module is connected with every other. An enormous variety of software, with many variants for each type, has already been produced, and will continue to be produced by a great many software manufacturers. Hence, future software systems will consist of multi-vendor software, often dynamically integrated, residing at multiple sites as servers. Software modules, or servers, are dynamically shared by multiple users, and may be changed from time to time. Future software systems will also exhibit dynamic resource discovery. It will be useful to view such systems as forming a society that is analogous to human society and interleaved with it.

Hence, we need an evolved notion of objects, derived from a dynamic and interactive viewpoint. In this context, I will speculate on what the evolved notion of objects is, and how future software should be composed.

I will first review the notions of objects and concurrent objects. Then, I will present two observations on our current computing systems operating in open, distributed environments, and illustrate the necessity for higher-level software modules called *autonomous agents.* Then, I will present some of our research work, being carried out at Sony Computer Science Laboratory and Keio University. This research is aimed at achieving a society of autonomous agents that cohabit with human society. The presentation features a personal, personified computer with a face and capable of vocal communication, called the *Intimate Computer;* an infrastructure model that abstracts future open, distributed computing environments, called the *Computational Field Model (CFM);* the *Apertos* distributed, real-time operating system; and the *Virtual Internet Protocol (VIP)* mobile host protocol. In the last section, I will be a little provocative in advocating an even more advanced notion of autonomous agents, called *volitional agents,* that could be used to create safe, evolutionarily stable, cohabitating society. I will conclude this paper with a description of recent developments toward the understanding of collective behavior in terms of dynamic, non-deterministic, stochastic, and irreversible processes taking root in various scientific fields.

2 From Objects to Concurrent Objects

The world in which we live is *concurrent* in the sense that there are multiple active entities; *distributed* such that there is a distance between entities that yields a propagation delay in communication between them; and *open,* meaning that the entities and their environment are always changing. Computation can be considered as a simulation of part of the real or an imaginary world. In doing a simulation, you can model your problem in terms of sequential computing, concurrent computing, distributed computing or open computing. To solve a simple, small problem, sequential computing is usually sufficient. However, when the problem becomes larger and more realistic, it is much easier to model it as concurrent, distributed, or open computing. For example, if you have multiple users at a time, such as in banking or airline reservation systems, you would naturally model the problem in the form of concurrent or distributed computing.

2.1 Objects

The notion of *objects* provides a very convenient way of describing problems in any of sequential to concurrent, distributed, or open computing. An object is usually considered as being a physical or logical entity with a unified communication protocol, which is usually message passing [44]. It is composed of a local storage and a set of procedures, as shown in Figure 1.

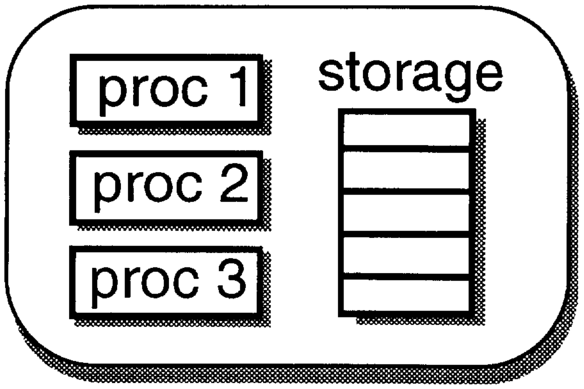

Figure 1: An Object

However, if we examine more closely what an object is, it can be seen to be an abstraction of computation as things. Chairs, pens, books, ... these are *things.* The sky, air, and water are not usually considered as being things. Things can be distinguished from others. An apple is distinguished from other apples. Water in a glass can now be distinguished from other glasses of water. This is the external view of things. The other characterization is that a thing has both an inside and an outside. This is the internal view of things. Programming and computing a problem in terms of the interaction between things is the true benefit of object orientation. It is for this reason that the notion of objects is applicable equally to concurrent, distributed, and open computing.

Object-oriented computing can be understood as being a movement from a microscopic view of computing to macroscopic view, where microscopic corresponds to computation done by executing an algorithm, and macroscopic is computation done using the mutual effects of objects.

However, objects in most existing languages and systems are sequential, and therefore, static, or passive. This is a remnant of programming styles from when the computer was centralized and based on a uniprocessor. Using this kind of object abstraction, programmers have to write execution control, or processor allocation, for objects, if they need to write a concurrent program. That is, we need the notion of *processes* on top of the notion of objects. This is very inconvenient and is a common source of errors. The fundamental reason for this is that real *things* are not like this. Every thing exists and behaves simultaneously on its own right. Analogically, every object should exist and behave simultaneously. Therefore, it should have its own processor.

2.2 Concurrent Objects

The notion of *concurrent objects* is an extremely significant development. A concurrent object contains a (virtual) processor, as shown in Figure 2. Here, we can eliminate the notion of processes which is necessary in concurrent programming using sequential objects. Programmers don't have to describe execution control. Concurrent objects are executed in the same way as in time-sharing systems.

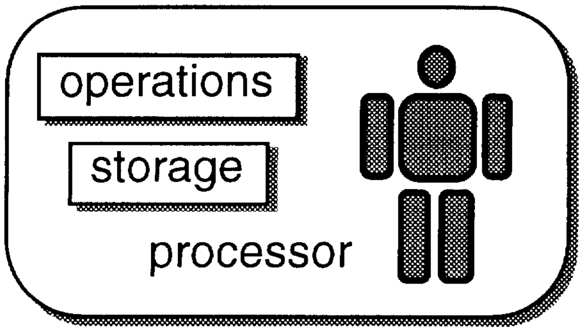

Figure 2: A Concurrent Object

We can trace the history of concurrent objects back to the early 70's when Carl Hewitt proposed Actors [16, 1]. Since then, many concurrent object-oriented languages and systems have been proposed and used. Concurrent Smalltalk [56] and Orient84/K [24] are languages which I designed with my co-researchers. ABCL [59], POOL [3], Concurrent Eiffel [7], Concurrent C++ [15], Active Objects [34], and many other languages have also been designed. It was almost ten years later when I edited the book titled, "Object-Oriented Concurrent Programming" with Aki Yonezawa [60]. The notion of concurrent objects can be found in the field of operating systems, too. Examples include Eden [5] and Apertos [57, 58]. Theoretical investigations on concurrent objects have also been pursued, such as π-calculus [30] and ν-calculus [19, 20]. Some recent accomplishments are detailed in [2].

Although the notion of concurrent objects is a more natural means of modeling things in programming, it has not yet found practical or commercial use. This is probably just a matter of the notion not yet being well known. It takes some time, say 10 years or so, until people actually feel comfortable with a new notion. Virtual memory is one such example, and (sequential) object-oriented programming is another. The notion of concurrent objects will become much more important when we need to migrate objects in a widely distributed environment. I fully expect the notion of concurrent objects to be accepted and widely utilized in the near future.

In summary, the most important role of objects (for both sequential and concurrent objects) is modularity: that is, to enable the writing of a program as a thing with interface. This affords us various benefits, such as macroscopic programming, analysis and design, classes and instances, class hierarchies, concurrency, and so on.

3 Computation in Open, Distributed Environments

I have been doing research on concurrent objects for more than ten years. The notion is neat, and provides us with a very appropriate level of abstraction. As such, it seemed as though it would be enough for describing distributed and even open systems. I tried to convince myself that concurrent objects would be enough. And, I was almost fully convinced. But something had annoyed me for a long time and prevented me from being fully convinced. In the following subsections, after presenting the essential characteristics of open, distributed systems, I will explain what it was that annoyed me by observing two example systems.

3.1 Essential Characteristics

The technical characteristics of open and distributed systems can be summarized as follows. Distributed systems are characterized by there being distance between objects, which results in communication delays. Distance, and therefore delay, has an inherent consequence that there is no unique global view of the system. The state-space of the system that one observer sees is different from the state space a distant observer sees. Since there is delay in communication, we use asynchronous communication for the sake of efficiency. Asynchronous communication means that the timing when the sender sends a message and that when the receiver receives it are different. By using asynchronous communication, we can exploit the concurrency between computation and communication.

Open systems are characterized by their entities and environments constantly changing. Widely distributed systems are usually open systems, since the topology of the networks, the component computers, and the functions, quality, and locations of the services are dynamically changing. Thus, our future computing environment will be modeled well as an open, distributed environment.

3.2 Two Observations

Here, I would like to describe two observations on our computing systems, which offer a good prediction of future computing environments.

The first observation is as follows: Assume you need a system. You write a specification, you program it or have somebody else program it, and you use it. Then, after a while, or perhaps even before the completion of the system, you need to change it. And at a later time, you need to change it again. This is the problem of version management. As everybody knows, it is not easy to follow how the current system is composed, and how it works.

Version management is much more complex in open, distributed environments. Assume your friends or colleagues happen to know that you have a good system. So, they ask you for permission to use it. They are usually at remote sites, so they want to use your system through remote procedure calls (or remote object calls). However, they also request revision of the system to tailor the system to their particular needs. This tends to be repeated over and over, in a distributed manner. Thus, after a while, nobody understands the inner workings of the system, even though the users are using the system at a reasonably satisfactory rate. However, a problem arises when somebody finds a bug. How can the system be debugged and maintained without interfering with the other users? What should be done if the system stops? Can you, in fact, depend on the system at all?

The second observation is of almost the same problem, but from a different viewpoint. Nowadays, programs with the same functions are provided by many different vendors, so that users can choose those that best fit their needs and budget. For example, user #1 buys an OS from vendor #1, and a windowing system from vendor #2, etc. This is very good situation for the users.

Vendors request software houses to develop program modules, and buy complete modules from other vendors, and combine those modules into software products for release under the vendor's own brand. I call this *nested multi-vendor software* in the sense that software vendors use other software vendors' software as their components. Up to this point, users have a *physical* copy of the software on their machines.

However, the next step is a distributed version. Here, the users don't have a copy on their machines. Instead, they only have a calling program and the right to access programs on the vendors' machines. In turn, the software on the vendors' machines will call program modules residing on machines located at software houses, and so on. I would like to call this *distributed, nested multi-vendor software.* Here, no physical copy of the programs are made. Instead, they perform *remote procedure calls,* or *remote object calls* to each other. In fact, this is already happening. You are using a program on your network through a license server. The program may use other programs remotely (Figure 3).

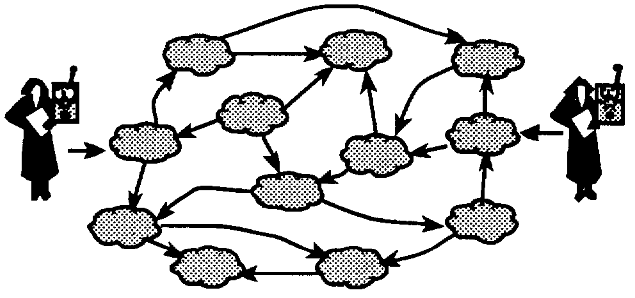

Figure 3: Distributed, Nested Servers

Once again, assume that somebody finds a bug. How can the system be debugged and maintained without interfering with the other users? What should be done if the system stops? Can you depend on the system at all?

I would like to summarize that, in open and distributed systems, we use a software module without having complete knowledge of it. Thus, it would appear to be like it is changing by itself. And, you need to discover the services you need. Then, you have to write your program in such a way that it protects the users or customers, by protecting the services and defending the computational resources you provide.

I don't think that concurrent objects provide a suitable framework for such defensive programming. First, I would like to claim that we need the notion of time for programming open, distributed systems. This necessity is derived from the fact that the essential difference between a distributed system and a concurrent system is the existence of *distance* in the system, which is equivalent to *time*. Second, we need a higher level module than a concurrent object for constructing a larger system. Hence, the above situations can be naturally modeled as a society of such modules. Let's call such a higher level module an *autonomous agent*.

4 Autonomous Agents

Now, I would like to give a rough definition of *autonomous agents*. First, I would like to clarify that the notion of autonomous agents does not conflict with the notion of objects or concurrent objects. In fact, an agent will be composed of concurrent objects, in much the same way as a person is composed of cells living concurrently. An autonomous agent is the unit of *individual* software, that interfaces with humans, other agents, and the real world in real-time. Each autonomous agent has its own goal, and reacts to stimuli, based on its situation. It behaves to survive. The collection of autonomous agents forms a society.

The definition of an individual is most important in thinking about autonomous agents. This is one kind of granularity argument, but taken from a completely different viewpoint, i.e., not for parallelism or efficiency, but for robustness or defensive programming. According to recent findings in biology and the theory of evolution, definition is very difficult, almost impossible, in fact. But we will not take such a serious approach. We instead use a naive, intuitive definition:

An individual autonomous agent is a collection of component objects (or cells) that are not physically shared with other individuals.

We assume an individual is a unit of feedback for utility or reward. An individual can be considered as being the unit of security, which corresponds to our bodies' immune system. It can also be regarded as being the unit of reliability and maintenance which corresponds to homeostasis. That is to say, security and reliability have to be provided and maintenance has to be done on an individual basis, not as a whole system. Each individual autonomous agent should provide such abilities per se.

For its functionality, each autonomous agent has an individual goal. It is reactive, in the sense that it responds to a stimulus, taking the situation or environment into consideration, in real-time. This implies that an autonomous agent is not just an object that responds to an input, but also needs to be able to learn the situation, and to have the ability to make timely decisions in real-time.

Survival is yet another important property of each individual autonomous agent. This property is, in fact, the result of only autonomous agents with a higher survivability surviving. To survive, an autonomous agent has to make its *best effort* to satisfy the users, in terms of response time and functionality; or the quality of services in general, so that it can maximize its utility. Restaurants with bad food or those that makes you wait one hour for "today's special" would never survive. To survive, an autonomous agent has to keep its losses to a minimum. This is called the *least suffering* strategy. A simple example is that, if your order doesn't come within one hour, you should decide whether to wait longer or move to a different restaurant. You have to monitor the situation and make a decision on *time-out*. That is to say, time-out is the last resort for survival in open, distributed systems. A simple programming language [47] and a formal system [42], which provide for the agents' survivability by incorporating a time-out notion, were presented at OOPSLA'92 by myself and my co-authors.

Agents should provide the facility of reflection to allow their adaptation to environments [28, 25]. Agents with negotiation ability are advantageous. Agents that can maintain a cooperative relationship for a longer duration are more profitable. They can form a group, and thus, society. Research on negotiation, cooperation, and group formation can be seen in one area of Artificial Intelligence, called "Distributed AI" [23, 13] or "Multiagent Systems" [10].

5 Current Research Activities

Here, I would like to present some of our research activities into open and distributed systems, currently being carried out at Sony Computer Science Laboratory and Keio University. The work is mainly based on concurrent objects and, as a whole, on autonomous agents.

I believe that a future computer system must be:

- *ubiquitous*, so that you can use it any time and anywhere;
- *portable and mobile*, so that you can carry it and use it on the move;
- *reliable and secure*, so that you can depend on it; and
- *friendly*, so that it is comfortable and easy to use.

With the ultimate goal of realizing such a computer system, I have been proposing two notions: *intimate computers* and *computational field* [52, 53].

5.1 Intimate Computers

Intimacy implies security, peace of mind, trustworthiness, reliability, and respect. The intimate computer is intended to inspire users with such a feeling. It has a face, and it understands natural languages, so that it presents you with a completely different user-computer interface from those we are used to today (Figure 4). An intimate computer can be seen as an autonomous agent overall, whereas it is composed of a collection of autonomous agents.

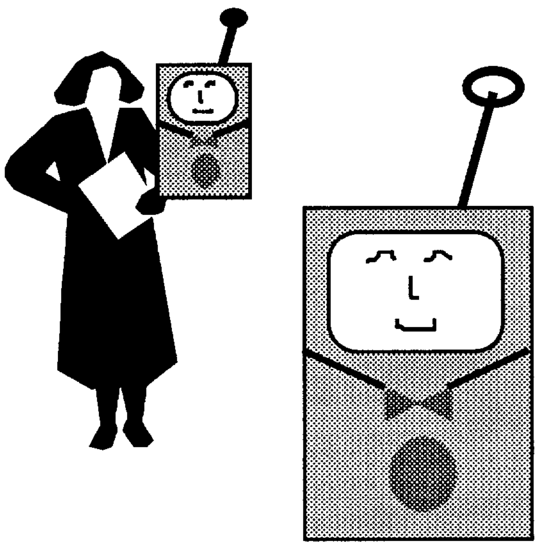

Figure 4: An Intimate Computer

An intimate computer can be thought of as an evolved version of a Personal Digital Assistant. It can be used as an access terminal to distributed computing facilities. It can be used as a communication terminal to access other intimate computers and their users. But, the ultimate purpose is the dialog itself; understanding each other and recognizing each other, rather than an interchange of ordering and inquiring.

Unfortunately, intimate computers are not yet available, but the following is an example of a possible conversation with an intimate computer in the future:

> *"Hey buddy, could you arrange a dinner meeting with Ralph?"*

My intimate computer understands who I mean by Ralph, asks Ralph's intimate computer when he is available, what kind of food he likes, makes a reservation for a restaurant, then comes back to me saying,

> *"It's done."*

On another occasion, my intimate computer suddenly talks to me

> *"Hey, Mario, how're you doing?"*

And I respond

> *"Don't bother me now!"*

Then, my intimate computer exultantly says to me

> *"Sorry, but I guess you forget something. It's your daughter's birthday. You should go back home right now!"*

5.2 Computational Field Model

To make intimate computers usable in a distributed environment, we need an infrastructure. I am proposing a higher-level abstraction of distributed computing than that of computers connected by networks. Forget about computers and networks; let's consider the field of computers. It is like a sea of computers. Concurrent objects are floating on the sea (Figure 5). The sea, the Computational Field, yields various forces between objects for the suboptimal placement of objects for moving users, grouping objects, balancing loads, and avoiding faults, such as:

- *Gravitational force* is defined for grouping objects. Frequent communication between objects yields a stronger force.
- *Repulsive force* is defined for load balancing. If two objects come very close, the repulsive force increases between them.
- *Friction* is defined for stability. It is proportional to the size, or weight of each object, so that a large object tends not to move.

Significance of the Computational Field Model is that it integrates load balancing and object grouping.

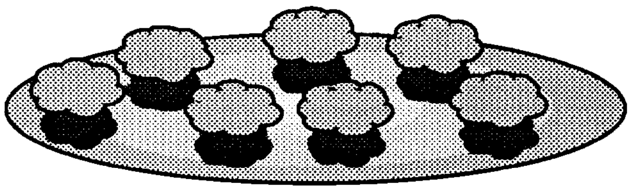

Figure 5: Computational Field Model

I will explain how the Computational Field works (Figure 6). If you place a task in the Computational Field, a mountain is formed which is a collection of concurrent objects for the given task. Then, a repulsive force between the objects arises, so that the mountain becomes lower and lower. At the same time, the gravitational force between concurrent objects increases, as they send messages to each other. Thus, they form a hill, as the two forces balance. If the user moves, the mountain follows.

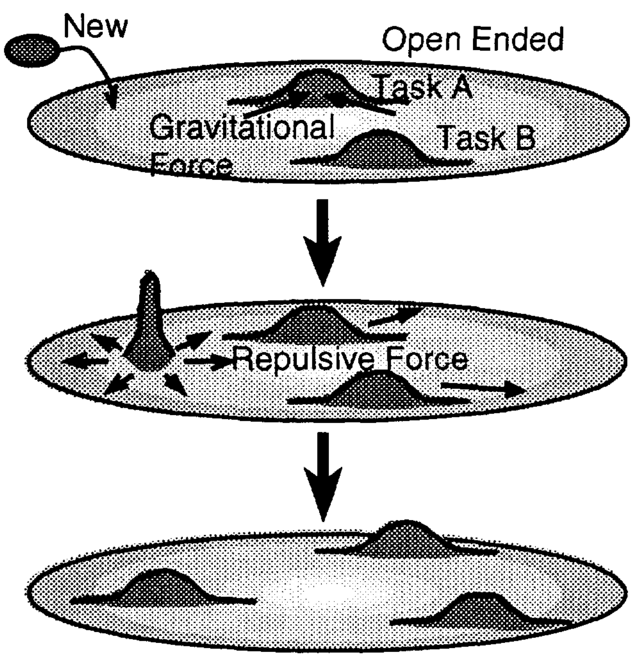

Figure 6: Dynamic Object Placement

All those properties are actually realized by an underlying distributed operating system. Also, the notion of concurrent objects is indispensable for object migration, as pointed out above.

5.3 Essential Technologies

To realize intimate computers and the computational field, many points demand our attention. We are currently concentrating our efforts on the following five topics:

- Multi-Modal Interaction
- Operating Systems
- Computer Networks
- Programming Languages, and
- Multi-Agent Systems

Demonstration videos are available for the first three topics.

Speech Dialog with Facial Displays

The first demonstration is of the speech dialog system with facial displays [48, 33]. The system was developed to verify the idea of bringing facial displays into human computer interaction as a new modality to make computers more communicative and sociable. It consists of two subsystems. One is a speech dialog subsystem. The other is a facial animation subsystem.

The speech dialog subsystem consists of a speech recognition module, a syntactic and semantic analyzer, a plan recognition module, a response generation module, and a voice synthesis module. It realizes speaker-independent speech recognition and handles the speaker's intentions. A facial animation subsystem generates a facial display by the local deformation of the polygons representing the 3D face. We adopted Keith Waters method [55] for our deformation scheme. Lip and speech synchronization was also implemented.

The speech dialog subsystem recognizes a number of typical conversational situations that are important in dialog. These situations are associated with specific conversational facial displays categorized by Nicole Chovil [8]. Upon detecting a prescribed situation, each module in the speech dialog subsystem sends a request for a specific facial display to the facial animation subsystem. An empirical study of the system with 32 subjects indicated that the speech dialog system with facial displays is helpful, especially in the first interaction with the system. An example of a session is shown in Figure 7.

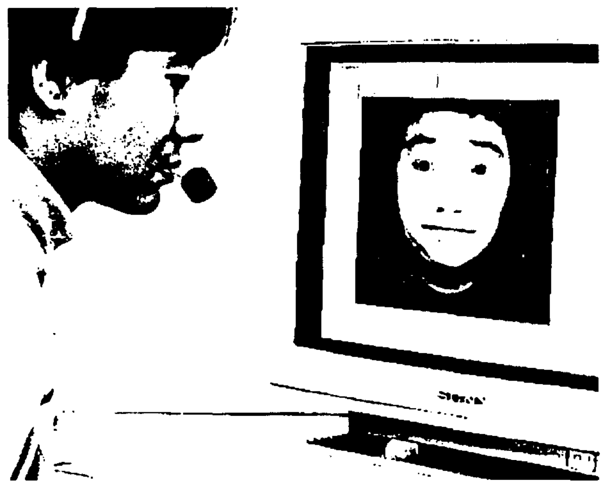

Figure 7: Speech Dialog System with Facial Displays

The system uses two workstations, one for speech dialog and the other for facial display, running in real-time mode. The speech dialog subsystem is designed as a multi-agent system, whereas the facial display subsystem is currently a collection of C programs. We plan to introduce more modalities, such as reading the user's face. We are also interested in investigating the relationship between the framework of our work with social knowledge and social actions, as presented by Les Gasser [14].

Apertos Distributed Real-Time OS

The second demonstration is on the Apertos operating system [57, 58]. The Apertos OS is a pure object-oriented, real-time OS, based on concurrent objects. It clearly separates objects and meta-objects, and it can evolve by itself by using the mechanism of reflection, without stopping. Object migration and distributed naming mechanisms are provided at the system level. That is, this supports the Computational Field. The Apertos OS has been stably operating on Sony's 68030-based workstations since April 1991, and was recently ported to Sony's R3000-based workstations and 486-based IBM PC-compatible computers.

Virtual Internet Protocol

The last demonstration is of the computer network protocol that supports mobile hosts. It is called the *Virtual Internet Protocol,* or VIP for short [50, 49]. By using this protocol, you can hook-off your portable workstation from the current network, move with it, and hook it into any interconnected network. You can obtain the same computing environment there, and all messages are redirected to the new location, taking their optimal routes. You can even move with your portable workstation while preserving communication channels. The mechanism we designed is analogue to that of virtual memory. Virtual to physical address translation is done in a distributed manner by using cached mapping information. Since this protocol is implemented as a sublayer of IP, it is transparent to application programs. It has been running since spring of 1992, and has been proposed to the Internet Engineering Task Force (IETF) for standardization.

VIP protocol is running on UNIX and MS-DOS machines, but is not described in an object-oriented style at this moment. Porting to Apertos OS in an object-oriented fashion will be done very soon. We are also interested in combining this technology with real-time communication facilities [43].

Programming Languages and Multi-Agent Systems

We are intensively doing research on Programming Languages and Multi-Agent Systems. Regarding the former field, we are especially interested in persistent object programming languages [54, 31], distributed transactions [18] and the applicability of the notion of reflection to distributed and real-time programming [37, 21]. For the latter, we are interested in collaboration [38, 29].

6 Volitional Agents

I have proposed concurrent objects and autonomous agents, and have presented some of our research at Sony Computer Science Laboratory and Keio University. Here, I would like to raise the final question, that is:

Are autonomous agents sufficient for future computing? Are they safe? Are they stable? Are they cooperative to humans?

In fact, I don't have any answer to these questions. But, I would like to be a little provocative and controversial in saying that the definition of autonomous agents given in section 4 may not be sufficient, safe, or cooperative to humans, and that a society composed of such autonomous agents would not be stable. Hence, I would like to propose *volitional agents.* Volition means "actions with will" or "actions of will." So, a volitional agent is an autonomous agent with will, or a spontaneous autonomous agent. A volitional agent is more active than an autonomous agent which is "reactive." It has desire, or it is aggressive [27].

I am saying that volitional agents are safer, more stable, and more cooperative with humans, compared to autonomous agents that are reactive. Reactive implies passive in a sense, since the agent doesn't perform any action unless it receives input. A society of autonomous, reactive agents may seem safe and stable, because they are passive. But that can be the very source of danger. You cannot know anything unless you give an input to the society, which may eventually result in a fatal damage.

Volitional agents are active and dynamic, and are doing something all the time. Internally, they will have antagonistic desires. Externally, they will have contentions with other agents. They will cooperate to achieve higher utility, and they will compete with each other to survive through natural selection (since computer environments are rather artificial than natural, we may need minimal legislation to ensure fair competition). They might behave selfishly [9].

Since volitional agents are active, and society is living, we can observe the behavior of the society. And, we can obtain even a higher stability of the society. Of course, it is impossible to predict the precise behavior in any ways, since the system is very large and complex. However, we can take advantage of recent developments in the study of complex dynamical systems. For example, according to the theory of *chaos* [11], it is given that under a certain condition, a system of active or dynamic components give a higher stability than that of passive or static components.

A society of agents, as well as our own society, should be evolutionarily stable. This means that society is stable for a while, but the environment changes, so that it rather quickly moves to the next stable state. This phenomenon can also be explained for a system of active or dynamic components as a phase transition by taking the same approach. Hence, we can conclude that volitional agents can provide a higher stability without sacrificing flexibility of the society than autonomous agents. This will lead to a society that is safer and more cooperative with humans.

The importance, and the necessity for aggressiveness in forming a stable society has been studied in the field of biology and ethology, such as in the work of Nikolass Tinbergen [51] and Konrad Lorenz [27]. The stability of society has also been intensively studied by political scientist Axelrod [4], biologist Maynard-Smith [45], and other researchers, taking game-theoretic approaches. Study for the behavior of society taking dynamical systems approaches are found in the new area called "Ecology of Computation" [22] or "Emergent Computation" [12]. We have also started research in this direction, particularly on chaos and collective behavior [35, 46, 36]. Distributed and massively parallel computing are expected to be powerful computing platforms [26].

7 Conclusions

I have discussed a couple of things in this paper. Objects are things which can be distinguished from others. This notion brought us "macroscopic programming." Concurrent objects are the "real" self-contained objects including virtual processors. This provided us with easy concurrent programming. Objects and concurrent objects are the cells for autonomous agents. Autonomous agents are based on the notion of the individual, are reactive, and try to survive. They form a society.

Then, I described and showed our recent research accomplishments toward the society of agents: the Intimate Computer, the Computational Field Model, the Apertos object-oriented OS and the VIP mobile host protocol. Finally, I raised a controversial proposal for volitional agents, that would provide safer and more cooperative interaction with humans and other agents, and that would provide an evolutionarily stable society, with which we can cohabit.

The notion of volitional agents is a conjecture, without any proof. We don't know how to make a volitional agent. We don't even know what "desire," or "aggressiveness" mean. But a new way of understanding collective behavior in terms of dynamic, nondeterministic, stochastic, and irreversible processes is taking root in various scientific fields. I have already mentioned this trend in biology, ethology, and the theory of evolution. We can also see similar movements in AI, such as the Society of Mind by Marvin Minsky [32] and the Subsumption Architecture by Rodney Brooks [6]. It is also happening in Chemistry and Physics. For example, a new view is given based on Thermodynamics by Ilya Prigogine [41]. It is giving us the sign of departing from the "reductionist attitude" or the "Cartesian attitude" in science.

The notion of Open Systems was advocated by philosopher Karl Popper [39, 40] and brought in to computer science by Carl Hewitt [17]. We must inevitably see Distributed and Open Systems as societies. This is already coming. In this paper, I proposed the notion of autonomous agents and volitional agents as individuals of societies. Volition might be the true meaning of autonomy, and may realize a safe, stable, cooperational society with computers.

Acknowledgment

I would like to thank Akikazu Takeuchi, Yasuhiko Yokote, Fumio Teraoka, Katashi Nagao, Hiroaki Kitano, Jun Tani, Toru Ohira, Chisato Numaoka, Shigeru Watari, Tatsumi Nagayama, and other researchers at Sony Computer Science Laboratory; Kohei Honda, Ichiro Satoh, Vasco Vasconcelos, and many other students and ex-students at Keio University; Aki Yonezawa, Satoshi Matsuoka, Gregor Kiczales, Eric Manning, Koiti Hasida, Jörg Kaiser, and many other friends of mine in computer science, for their inspiring discussions and help in preparing for the speech and this manuscript. Finally, I would like to thank the many members of the audience at the OOPSLA'93 conference who gave me useful comments and encouraged me to prepare this manuscript.

References

[1] Agha, G., *ACTORS: A Model of Concurrent Computation in Distributed Systems,* MIT Press, 1986.

[2] Agha, G., Wegner, P., and Yonezawa, A., eds, *Research Directions in Concurrent Object-Oriented Programming,* MIT Press, 1993.

[3] America, P., *POOL-T: A Parallel Object-Oriented Language, in Object-Oriented Concurrent Programming,* eds. Yonezawa, A., and Tokoro, M., MIT Press, 1987.

[4] Axelrod, R., *The Evolution of Co-operation,* Basic Books, Inc., 1984.

[5] Black, A. P., "Supporting Distributed Applications: Experience with Eden," *Proceedings of ACM Symposium on Operating System Principles,* p. 39–51, December, 1985.

[6] Brooks, R., "Intelligence Without Representation," *Artificial Intelligence,* Vol. 47, p. 139–160, 1991.

[7] Caromel, D., "Concurrency: An Object-Oriented Approach," *Proceedings of TOOL2,* p. 183–198, June, 1990.

[8] Chovil, N., "Discourse-Oriented Facial Displays in Conversation," *Research on Language and Social Interaction,* Vol. 25, p. 163–194, 1991

[9] Dawkins, R., *The Selfish Gene (2nd Edition),* Oxford University Press, 1989.

[10] Demazeau, Y., Muller, J.-P., and/or Werner, E, *Decentralized A.I. 1, 2, and 3,* North-Holland, 1990, 1991, and 1992.

[11] Devaney, R. L., An Introduction to *Chaotic Dynamical Systems (2nd Edition),* Addison-Wesley, 1989.

[12] Forrest, S. (ed.), *Emergent computation,* MIT Press, 1991.

[13] Gasser, L. and Huhns, M. (eds.), *Distributed Artificial Intelligence Vol.2,* Pittman, London, 1989.

[14] Gasser, L., "Social Knowledge and Social Action: Heterogeneity in Practice," *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'93),* p. 751–758, 1993.

[15] Gehani, N. H., "Concurrent C++: Concurrent Programming with Class(es)," *Software Practice and Experience,* Vol.16, No.12, Dec, 1988.

[16] Hewitt, C. E., "A Universal, Modular Actor Formalism for Artificial Intelligence," *Proceedings of International Joint Conference on Artificial Intelligence,* 1973.

[17] Hewitt, C. E., "The Challenge of Open Systems," *Byte,* April 1985, p. 223–242, 1985.

[18] Hirotsu, T., "A Flexible Transaction Facility for Distributed Object-Oriented Systems," *Proceedings of IEEE Workshop on Object-Orientation in Operating Systems,* September, 1992.

[19] Honda, K. and Tokoro, M., "An Object Calculus for Asynchronous Communication," *Proceedings of ECOOP'91,* LNCS 512, p. 133–147, June, 1991.

[20] Honda, K. and Tokoro, M., "Combinator Representation of Mobile Processes," *Proceedings of Symposium on Principle of Programming Languages,* January, 1993.

[21] Honda, Y. and Tokoro, M., "Soft Real-Time Programming through Reflection," *Proceedings of IMSA'92 International Workshop on Reflection and Meta-level Architectures,* 1992

[22] Huberman, B. A. (ed), *The Ecology of Computation,* North-Holland, 1988.

[23] Huhns, M. N. (ed), *Distributed Artificial Intelligence, Vol. 1,* Pitman, London, 1987.

[24] Ishikawa, Y. and Tokoro, M., "A Concurrent Object-Oriented Knowledge Representation Language Orient84/K: Its Features and Implementation," *Proceedings of OOPSLA'86,* p. .232–241, September, 1986.

[25] Kiczales, G., "Towards a New Model of Abstraction in Software Engineering," *Proceedings of the IMSA'92 International Workshop on Reflection and Meta-level Architectures,* 1992.

[26] Kitano, H. and Hendler, J., eds., *Massively Parallel Artificial Intelligence,* The MIT Press, 1994.

[27] Lorenz, K., *Das Sogenannte Böse,* Dr. G. Borotha-Schoeler Verlag, 1963.

[28] Maes, P., "Concepts and Experiments in Computational Reflection," *Proceedings of OOPSLA'87,* p. 147–155, 1987.

[29] Matsubayashi, K., "A Collaboration Mechanism on Positive Interactions in Multi-Agent Environments," *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'93),* p. 346–351, August, 1993.

[30]Milner, R., Parrow, J., and Walker, D., "A Calculus of Mobile Processes, Part 1 & 2," Technical report ECS-LFCS-89-85 & 86, University of Edinburgh, 1989.

[31] Minohara, T. and Tokoro, M., "Providing Dynamic Abstractions and Type Specifications for Persistent Information," *Proceedings of Int. Conf. on Deductive and Object-Oriented Databases,* December, 1991.

[32] Minsky, M., *The Society of Mind,* Simon and Schuster, New York, 1987.

[33] Nagao, K. and Takeuchi, A., "A New Modality for Natural Human-Computer Interaction: Integration of Speech Dialogue and Facial Animation," *Proceedings of the International Symposium on Spoken Dialogue (ISSD'93),* p. 129–132, 1993.

[34] Nierstrasz, O. M., "Active Objects in Hybrid," *Proceedings of OOPSLA'87,* p. 243–253, September, 1987.

[35] Numaoka, C. and Takeuchi, A., "Collective Choice of Strategic Type," *Proceedings of International Conference on Simulation of Adaptive Behavior (SAB92),* December. 1992.

[36] Ohira, T. and Cowan, J. D., "Feynman Diagrams for Stochastic Neurodynamics," *Proceedings of Australian Conference of Neural Networks,* January 1994.

[37] Okamura, H., Ishikawa, Y., and Tokoro, M., "Metalevel Decomposition in AL-1/D," *Proceedings of Object Technologies for Advanced Software,* LNCS No. 742, p. 110–127, November, 1993.

[38] Osawa, E., "A Scheme for Agent Collaboration in Open Multiagent Environment," *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'93),* p. 352–358, August, 1993.

[39] Popper, K. R., *The Open Society and its Enemies,* Princeton University Press, 1945.

[40] Popper, K. R. and Lorenz, K., *Die Zukunft ist Offen (The Future is Open),* R. Piper GmbH & Co., 1985.

[41] Prigogine, I. and Stengers, I., *Order out of Chaos,* Bantam Books, 1984.

[42] Satoh, I. and Tokoro, M., "A Formalism for Real-Time Concurrent Object-Oriented Computing," *Proceedings of OOPSLA'92,* p. 315–326, 1992.

[43] Shionozaki, A. and Tokoro, M., "Control Handling in Real-Time Communication Protocols," *Proceedings of SIGCOMM'93,* 1993.

[44] Shriver, B. and Wegner, P, eds., *Research Directions in Object-Oriented Programming,* MIT Press, 1987.

[45] Maynard Smith, J., *Evolution and the Theory of Games,* Cambridge University Press, 1982.

[46] Tani, J. and Fukumura, N., "Learning Goal-directed Sensory-based Navigation of a Mobile Robot," *Neural Networks,* in press.

[47] Takashio, K. and Tokoro, M., "DROL: An Object-Oriented Programming Language for Distributed Real-time Systems," *Proceedings of ACM OOPSLA'92,* p. 276–294, October, 1992.

[48] Takeuchi, A. and Nagao, K., "Communicative Facial Displays as a New Conversational Modality," *Proceedings of ACM/IFIP INTERCHI, 1993.*

[49] Teraoka, F., Yokote, Y., and Tokoro, M., "A Network Architecture Providing Host Migration Transparency," *Proceedings of ACM SIGCOMM'91,* p. 209–220, 1991.

[50] Teraoka, F., Claffy, K, and Tokoro, M., "Design, Implementation, and Evaluation of Virtual Internet Protocol," *Proceedings of 12th International Conference on Distributed Computing Systems,* p. 170–177, 1992.

[51] Tinbergen, N., *Social Behaviour in Animals,* Methuen & Co. Ltd., 1953.

[52] Tokoro, M., "Computational Field Model: Toward a New Computing Model/Methodology for Open Distributed Environment," *Proceedings 2nd IEEE Workshop on Future Trends in Distributed Computing Systems,* September, 1990.

[53] Uehara, M. and Tokoro, M., "An Adaptive Load Balancing Method in the Computational Field Model," *OOPS Messenger, Vol. 2, No. 2,* April, 1991.

[54] Watari, S., Honda, Y., and Tokoro, M., "Morphe: A Constraint-Based Object-Oriented Language Supporting Situated Knowledge," *Proceedings of International Conference on Fifth Generation Computer Systems,* 1992.

[55] Waters, K., "A Muscle Model for Animating Three-Dimensional Facial Expression," *Computer Graphics,* Vol. 21, No. 4, p. 17–24, 1987.

[56] Yokote, Y. and Tokoro, M., "The Design and Implementation of Concurrent SmallTalk," *Proceedings of OOPSLA'86,* p. 331–340, September, 1986.

[57] Yokote, Y., Teraoka, F., and Tokoro, M., "A Reflective Architecture for an Object-Oriented Distributed Operating System," *Proceedings of ECOOP'89,* p. 89–108, July, 1989.

[58] Yokote, Y., "The Apertos Reflective Operating System: The Concept and its Implementation," *Proceedings of OOPSLA'92,* p. 397–413, October, 1992.

[59] Yonezawa, A., eds., *ABCL An Object-Oriented Concurrent Systems,* MIT Press, 1990.

[60] Yonezawa, A. and Tokoro, M., eds., *Object-Oriented Concurrent Programming,* MIT Press, 1987.

Contact Information:

Mario Tokoro
Department of Computer Science
Keio University
3-14-1 Hiyoshi
Yokohama 223 Japan
E-mail: *mario@keio.ac.jp*

Mario Tokoro
Sony Computer Science Laboratory Inc.
Takanawa Muse Building,
3-14-13 Higashi Gotanda, Shinagawa-ku
Tokyo, 141 Japan
E-mail: *mario@csl.sony.co.jp*