M-base : An Application Development Environment for End-user Computing based on Message Flow

Takeshi CHUSHO Yuji KONISHI Masao YOSHIOKA

Department of Computer Science, Meiji University
1-1-1 Higashimita, Tama-ku, Kawasaki 214-71, Japan
email : {chusho, yuji}@cs.meiji.ac.jp

Abstract

Explosive increase in end-user computing on distributed systems requires that end-users develop application software by themselves. One solution is given as a formula of "a domain model ≡ a computation model." This formula implies that one task in a domain model of cooperative work corresponds to one object in a computation model based on an object-oriented model. Application development environment, M-base[1]*, supports this formula for cooperative systems such as groupware and work flow systems. At the first stage, the system behavior at a macro level is expressed by using a modeling and simulation tool for constructing a message-driven model while focusing on message flow. At the second stage, static structure and detailed specifications of objects are expressed in a script language. Communication among objects is performed by a set of messages instead of a message, for implementation of flexible work flow.*

Key words :
end-user computing, software development environment, distributed system, object-orientation, domain modeling, visual programming

1. Introduction

Recently, hardware and software environments for information systems are rapidly changing on trends of downsizing, open architecture and distributed computing. An increasing number of untrained end-users began interacting with computers, and this number will continue to rise as communication infrastructure becomes popular. Then new software paradigms for such new fields with explosive increase in application software are required [2].

Generally, end-users are classified into the following three typical categories:

1. Clerks using terminals of a large-scale information system such as banking systems.

2. Office workers using application packages on personal computers.

3. Clients using public terminals such as ATMs on banking systems.

This paper primarily considers end-users of the second category. The users of the first category are supported by a department of information system development in a company or an organization. The users of the third category uses only application packages in a given way.

The users of the second category may use application packages for their individual task now. In addition, they are going to use other application packages for their cooperative work such as work flow systems and groupware. When these given application packages can not satisfy such end-users, they must customize these software or develop new ones. Furthermore, if there are no application packages for their work which they want to automate, they must develop their applications by themselves while being supported sometimes by system engineers.

For such end-users, it may be easy to understand a domain model, but it must be difficult to convert the domain model into a computation model which provides a framework of application software they require. One solution is given as a formula of

[1]This work has been supported in part by Engineering Adventure Group Linkage Program(EAGL).

* Masao Yoshioka is with NEC Informatec Systems Ltd., Kawasaki, Japan since Apr. 1996.

"a domain model ≡ a computation model."

This formula implies that one task in a domain model of cooperative work corresponds to one object in a computation model based on an object-oriented model. From this formula, the other formula of

"analysis ≡ design"

is derived since it is not necessary to convert a domain model into a computation model under this approach. This process requires necessarily a prototype approach with sufficient simulation of the domain model instead.

Application development environment, M-base, supports these formulas for developing cooperative systems such as groupware and work flow systems [4]. At the first stage, the system behavior at a macro level is expressed by using a modeling and simulation tool for constructing a message-driven model while focusing on message flow. At the second stage, static structure and detailed specifications of objects are expressed in a script language. Communication among objects is performed by a set of messages instead of a message, for implementation of flexible work flow.

Our basic idea is based on an object-oriented model since the model may satisfy these two formulas. However, our approach is different from most conventional object-oriented analysis and/or design methods [16] which need defining an object model on static structure of objects prior to a dynamic model on interactive behavior among objects. In our approach, behavior of a domain model is first constructed by focusing on message flow. Next the message flow is defined strictly by message sets. Then specifications of each object is defined by message transformation from input messages to output messages.

In the next section, the modeling process is described. The framework and tools of M-base are described in Sect. 3. In Sect. 4, the results are discussed.

2. Modeling process

2.1. Previous studies

For the past few years, the greatest attention in software engineering has been focused on object-oriented software development. This technology seems to promote paradigm shift of software for coming generation information systems. Essential concepts of object-oriented technologies came out around 1970 [6, 13] and were expanded into programming methodologies in 1970's [15]. Smalltalk-80 [10] triggered off developments of various object-oriented programming languages and trials of their applications in 1980's. Object-oriented programming has been already used in practice into various software fields, especially in middleware such as graphical user interface builders and object management platforms. However, these successes in object-oriented programming(OOP) do not necessarily imply successes in object-oriented analysis(OOA) and design(OOD) yet although many methodologies of OOA and OOD came out around 1990 [8, 16].

Most of conventional OOA/OOD methodologies are suitable for large-scale database-centered systems such as banking systems, but they are not suitable for application software of such new fields as distributed office information systems with end-user computing and cooperative work [11]. This is because these conventional techniques are based on a data model rather than a dynamic behavior model of the whole system and promote such design process as objects are defined prior to their behavior by using various notations of static relationships between objects [1, 5, 17, 18, 20], although there are a few notations for system behavior among objects such as an event trace diagram.

2.2. Research goals

A new approach and its support tools have been developed for satisfying the following requirements and are described as facilities of M-base in this paper:

1. The target software is a distributed office information system for cooperative work such as a work flow system and groupware.
2. The end-users are office workers who are professionals of office work but are not professionals of information technologies.
3. The system designers are mainly the end-users themselves although system engineers may support the end-users.
4. The maintenance is performed by the end-users themselves since the system specifications will modified frequently after running and the system must be changed quickly.

2.3. A two-layer model

Object-oriented technologies are primarily considered as a computation model since the essence of object-oriented technologies must be a message-driven model which is suitable to express a whole behavior of a system or a subsystem. This paper proposes the following paradigm for software development based on object-oriented modeling:

1. A dynamic model corresponding to system behavior, is expressed in a message-driven model.
2. A static model corresponding to both specifications and static relations of objects, is expressed in classes and its hierarchies.

This paradigm is called a two-layer model [3] in this paper and the conceptual framework is shown in Figure 1. These two layers are discriminated each other definitely in development process. The dynamic model expresses a domain model and almost satisfies the following two formulas:

1. A domain model $\equiv$ a computation model
2. Analysis $\equiv$ design

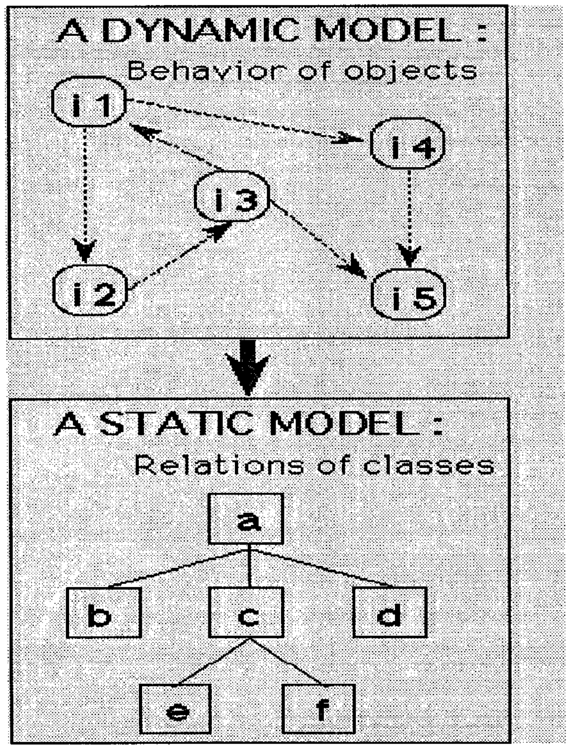

Figure 1. Conceptual framework of the two-layer model. The upper dynamic model expresses behavior of objects. The lower static model expresses relations of classes.

In the static model, however, satisfaction degree of the requirements depends on a script language and/or a class library to be used. In particular, domain-specific componentware will contribute to easiness of development. M-base promotes the growth of componentware [19].

2.4. Domain modeling

2.4.1 Modeling process

The modeling process in M-base is formalized as shown in Figure 2. A domain model is composed with an object-based analysis model(OAM) and a class-based design model(CDM), where these two models correspond to the dynamic model and the static model of the aforementioned two-layer model respectively. In the remainder of this paper, "object" implies "instance" and is discriminated from "class."

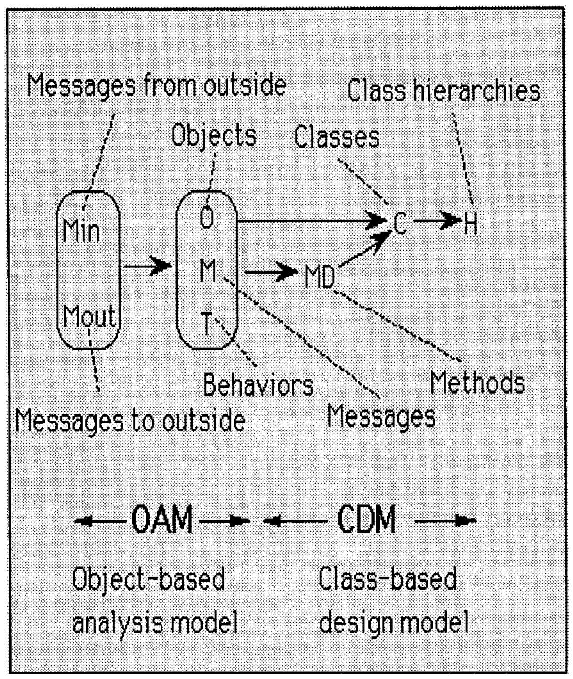

Figure 2. Modeling process is based on the two-layer model. The first step is called an object-based analysis model(OAM) and corresponds to the dynamic model. The second step is called a class-based design model(CDM) and corresponds to the static model.

2.4.2 Object-based analysis model

The object-based analysis model is expressed as follows:

$$OAM = \{ O, M, T\}.$$

O denotes a set of objects as

$$O = \{o[i]\},$$

where o[i] is the i-th object. M denotes a set of messages as

$$M = \{m[i,j,n]\},$$

where m[i,j,n] is the n-th kind of a message from o[i] to o[j]. Two functions of "sender" and "receiver" are introduced for getting a sender object and a receiver object of the message as

$$\text{sender}(m[i,j,n]) = o[i]$$

and

$$\text{receiver}(m[i,j,n]) = o[j]$$

respectively. Assuming that the outside of the system is regarded as an object with the subscript number of zero, o[0], a set of messages from the outside and a set of messages to the outside can be denoted by

$$\text{Min} = \{m \mid \text{sender}(m) = o[0],\ m \in M\}$$

and

$$\text{Mout} = \{m \mid \text{receiver}(m) = o[0],\ m \in M\}$$

respectively. T denotes a set of behavior as

$$T = \{t[r]\}$$

where t[r] is the following message transformation:

$$t[r] : m[i,j,n] \rightarrow \{m[j,k1,n1],\ m[j,k2,n2],\ \ldots\}$$

This expression implies that the object of o[j] receives the message of m[i,j,n] and then sends a sequence of messages, m[j,k1,n1], m[j,k2,n2], ...

In short, a domain model for a distributed system is constructed in accordance with a procedure shown in Figure 2. At the first step, Min and Mout are confirmed. Then, while examining message flow processes, O, M and T are identified.

2.4.3 Class-based design model

Next, this model is refined into the class-based design model:

$$\text{CDM} = \{\text{MD}, C, H\},$$

where MD, C and H denote a set of methods, a set of classes and a set of class hierarchies respectively.

1. External specifications
External specifications of each object are represented by a set of methods corresponding to messages which are received by the object. Suppose M(o[j]) denotes a set of messages which o[j] receives, and must be a subset of M. A set of methods of o[j], MD(o[j]), is obtained by operations that a subset of M(o[j]) is extracted from M(o[j]) as each message in the subset is equivalent to one another in the function and that the subset is corresponded to a method of MD(o[j]). That is, o[j] has methods which number is equal to the number of such equivalent sets. Consequently,

$$\text{MD} = \cup_j \text{MD}(o[j]).$$

2. Class identification
A set of objects which are equivalent to one another in a set of methods, can be generated from the same class. That is, a set of classes, C, is obtained by operations that a subset of O is extracted from O as each object in the subset is equivalent to one another in the set of methods and that the subset is corresponded to a class of C. That is, if MD(o[i]) = MD(o[j]), o[i] and o[j] are generated from the same class. The other objects are corresponded to different classes respectively.

3. Class hierarchies
A set of classes which are similar to one another in a set of methods, can compose a class hierarchy with inheritance. Suppose MD(c[i]) denotes a set of methods for a class of c[i]. The following hierarchical relation is introduced:

$$h[r] : c[i] \rightarrow c[j] \text{ if } \text{MD}(c[i]) \subset \text{MD}(c[j])$$

That is, if MD(c[i]) is a true subset of MD(c[j]), c[i] is able to become a superclass of c[j] by the following operation:

$$\text{MD}(c[j])/\text{new} = \text{MD}(c[j])/\text{old} - \text{MD}(c[i])$$

Furthermore, if MD(c[i]) and MD(c[j]) share a true common subset, a new class corresponding to this common subset, c[k], is able to become a superclass of both c[i] and c[j] as follows:

$$h[s] : c[k] \rightarrow c[i]$$
$$h[t] : c[k] \rightarrow c[j]$$

At the same time, the following operations are performed:

$$\text{MD}(c[k]) = \text{MD}(c[i])/\text{old} \cap \text{MD}(c[j])/\text{old}$$
$$\text{MD}(c[i])/\text{new} = \text{MD}(c[i])/\text{old} - \text{MD}(c[k])$$
$$\text{MD}(c[j])/\text{new} = \text{MD}(c[j])/\text{old} - \text{MD}(c[k])$$

Consequently,

$$H = \{h[i]\} = \cup_i h[i].$$

2.5. Metaphor-base modeling process for end-users

Our conceptual framework is based on the two-layer model and object-oriented concepts as mentioned in subsections of 2.3 and 2.4. However, since end-users are not familiar with these technologies, practical development process has been provided based on metaphors of an office as described below.

Since work flow is essential in most cases of developing a distributed system, it is natural to model the system behavior in message flows expressing dynamic relationships among objects. Cooperative work at an office is expressed by using a message-driven model as follows:

1. A person or a group to whom one or more tasks are assigned, is considered as an object.
2. Communication means such as forms, memos, telephone calls, mails, verbal requests, etc. between persons or groups, are considered as messages.
3. Cooperation of persons or groups is performed by message flow.

For support of such metaphor-base modeling, each task is often personified, and then is considered as an object as follows in M-base:

1. If one task is assigned to a person in the real world, an object corresponding to the person is introduced for assignment of the task in the domain model. This mapping is very natural personification.
2. If one task is assigned to a group in the real world, an object corresponding to the group is introduced for assignment of the task in the domain model. The group, that is, the task is personified as if the task were assigned to one person in the real world.
3. If some tasks are assigned to a person or a group in the real world, an object corresponding to each task is introduced. The task is personified as if each task were assigned to a different person in the real world.

This paper gives an example of the object-oriented office system, OOOffice, for convenience of explanation of a metaphor-base modeling. OOOffice is a system for meeting arrangement as shown in Figure 3, which was originally introduced in [7]. This system is similar to a scheduling function in an application package of a groupware product, and then is considered a typical example of a distributed system. This paper pays attention to software development process for end-users instead of application itself.

In Figure 3, objects of staffs may correspond to the item 1. An object of a secretary may correspond to the item 2 if secretaries in a group are in charge of the same task in the real world. Objects of room managers and objects of instrument managers may correspond to the item 3 if a person or a group manage all meeting rooms and/or all instruments for meetings in the real world.

In M-base, the principle of object decomposition is very simple as follows:

"Assign one task to an object."

It must be easy for end-users to apply this principle because they can assign each task to an object as if to assign each task to each person under the condition that the sufficient number of able persons exist.

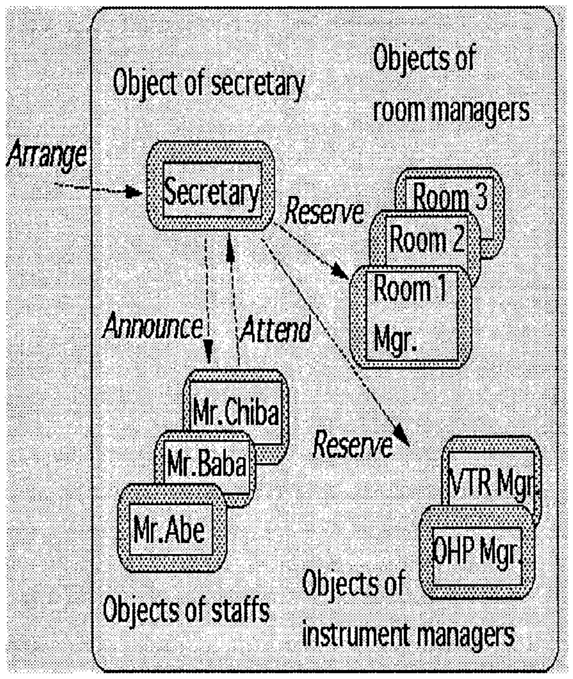

Figure 3. An example of a domain model of a distributed office system, OOOffice. There are four kinds of objects : a secretary, three staffs of Mr.Abe, Mr.Baba and Mr.Chiba, three room managers of meeting-rooms and two instrument managers of OHP and VTR.

3. Modeling tools

3.1. Framework of M-base

M-base provides the following four tools :

1. A modeling and simulation tool
2. A script language
3. A component builder
4. A user interface builder

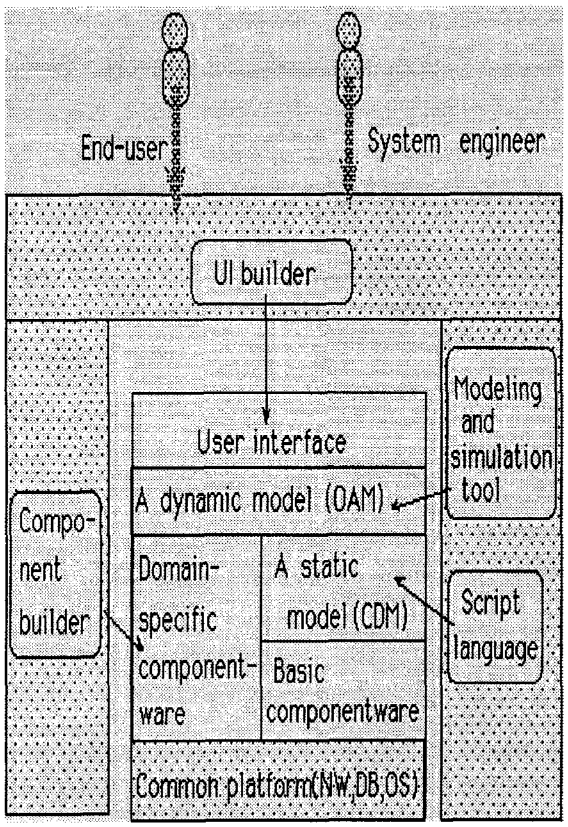

Figure 4. Application architecture and support tools. The inner box implies application architecture which is composed of user interface, a dynamic model, a static model, domain-specific componentware, and basic componentware based on a common platform. The outer part implies support tools of a modeling and simulation tool, a script language, a component builder and a UI builder.

The relations between these tools and application architecture which is supported by these tools, are shown in Figure 4. A dynamic model, OAM, is constructed by using the modeling and simulation tool while referring to the domain-specific componentware if necessary. A static model, CDM, is defined in the script language while referring to the basic componentware as class libraries.

3.2. A modeling and simulation tool

The modeling and simulation tool is called OAM-designer, and is used for constructing OAM by mouse manipulation as a kind of visual programming. The typical procedure is shown as follows :

1. Objects are defined by drag-and-drop from the palette of icons.
2. Messages from/to outside are defined by drawing arrow lines from outside to drain objects or from source objects to outside and by giving its name and attributes as parameters.
3. Messages between inner objects are defined by drawing arrow lines from source objects to drain objects and by giving its name and attributes as parameters.
4. Actual values are given to attributes of the input message from outside.
5. Message flow of OAM is simulated while displaying messages which move successively from one object to another object.

Examples are shown in Figure 5 and Figure 6 as a part of the domain model of OOOffice shown in Figure 3, which is composed of three objects of Office, Room and Abe. The Office object receives an Arrange message for meeting arrangement, sends a Reserve message for meeting room reservation to the Room object, and sends an Announce message for notice of the meeting to the staff objects such as Abe.

At the beginning of simulation, the arrow line of the Arrange message is clicked and 'attribute setting' is selected from a command menu. Then a balloon is displayed on the left side. This balloon requires a user to fill actual values in attributes of the Arrange message. After giving values to the three attributes of the date, time and the meeting name, the balloon in Figure 5 shows "Arrange a meeting of 5th 11:00."

Then, let's select 'message sending' from a command menu. The screen changes to Figure 6. The balloon of the Office object shows the Reserve message to be sent to the Room object as "Reserve a meeting-room of 5th 11:00.". The actual values of attributes of the Reserve message have been already set. At this time, a user can confirm actions of the Office object which received the Arrange message. Simulation will continue by selecting 'message sending' of a command menu while confirming message flow.

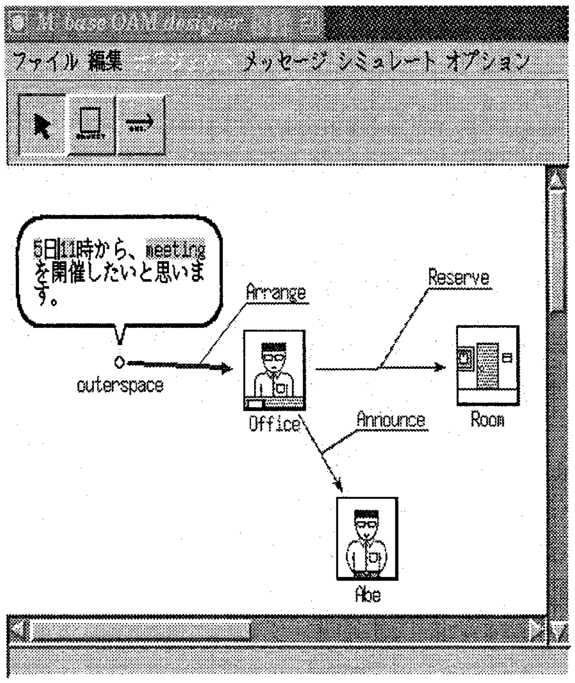
M-base OAM designer
ファイル 編集 オブジェクト メッセージ シミュレート オプション
5日11時から、meeting を開催したいと思います。
Arrange
Reserve
outerspace
Office
Announce
Room
Abe

Figure 5. Input of actual values from outside to attributes of the Arrange message in OOOffice. Three blanks in the Arrange message are filled as the date = '5', the start time = '11' and the meeting name = 'meeting'.

3.3. A script language

3.3.1 Design concepts

Since it is not suitable to define complex logic for message flow control by iconic programming, it is almost inevitable to make end-users use a script language. In M-base, the script language, Hoop, is used for defining the static model of CDM with class definitions, although basic frameworks of class definitions are generated from the dynamic model of OAM.

In comparison with conventional object-oriented languages, the significant feature of Hoop is that communication among objects is performed by a set of messages instead of a message, for implementation of flexible work flow. For example, let's consider a work flow system. Generally, the whole work flow is controlled by the meta-system. Therefore, it is difficult to understand system behavior because both the object-level and the meta-level must be considered. The message sets omit this difficulty and enable end-users to consider system behavior of only the object-level and to construct pure cooperative systems based on metaphors of communication at offices as described bellow.

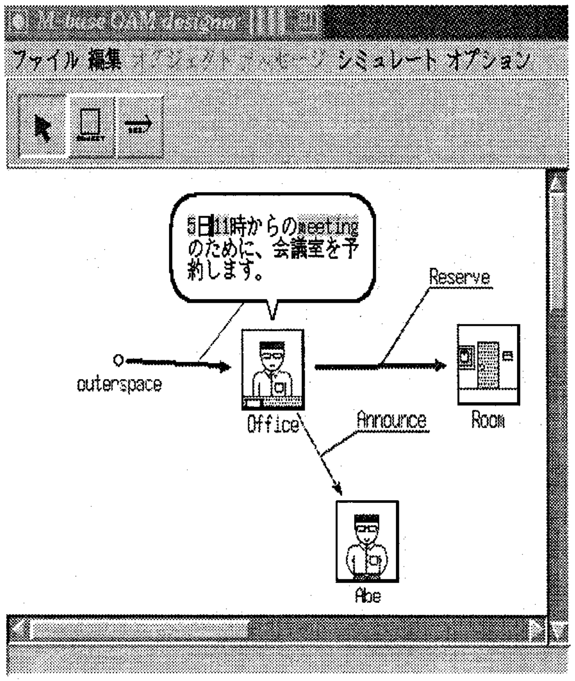
M-base OAM designer
ファイル 編集 オブジェクト メッセージ シミュレート オプション
5日11時からのmeeting のために、会議室を予約します。
Reserve
outerspace
Office
Announce
Room
Abe

Figure 6. Message sending. After the Office object received the Arrange message, it transformed the message to the Reserve message for meeting-room reservation and will send the message to the Room object.

3.3.2 Syntactic rules

The syntax of the message set in Hoop is as follows :

M ::= M' || [cond] M'
M' ::= Ms || Mp || X
Ms ::= {M, M, ... , M}
Mp ::= {M | M | ... | M}
X ::= (obj, msg)

where meta-symbols of '::=' and '||' imply 'equal to' and 'or' respectively. 'msg' and 'obj' imply a message and its receiver object respectively. 'cond' implies a condition for sending the following message. Semantics is described in remainder of this subsection and the details of specifications are shown in [14].

3.3.3 Sequential message sets

When each message among a message set should be executed sequentially, the message set is expressed as follows :

{M, M, ... M}

The object which received this message set, executes the first message and passes the remainder of the message set to the next object.

For example, consider the following message set is sent to the obj1 object :

{(obj1, msg1), (obj2, msg2), (obj3, msg3)}

The obj1 object executes the msg1 message, and then send the following remainder of a message set to the obj2 object :

{(obj2, msg2), (obj3, msg3)}

The obj2 object executes the msg2 message, and then send the following remainder of a message set to the obj3 object :

(obj3, msg3)

Finally, the obj3 executes the msg3 message.

This example may correspond to the following work flow in three sections of a mail-order firm when an order is received :

1. The stock of the ordered goods is checked.

2. The accounts are calculated.

3. The invoice is made.

3.3.4 Concurrent message sets

When all messages among a message set can be executed concurrently, the message set is expressed as follows :

{M | M | ... | M}

For example, consider the following message set is sent from some object :

{(obj1, msg1) | (obj2, msg2) | (obj3, msg3)}

The three objects of obj1, obj2 and obj3 execute the messages of msg1, msg2 and msg3 respectively.

As shown in the syntactic rule, arbitrary combination of the sequential message sets and concurrent message sets are admitted. For example, when the aforementioned work flow of the mail-order firm is changed to the flow that tasks of the item 1 and the item 2 are executed concurrently, the following message set is sent:

{ {(obj1, msg1) | (obj2, msg2)} , (obj3, msg3)}

3.3.5 An example of a Hoop program

Consider OOOffice of Figure 5 and Figure 6 again as a typical example of groupware. M-base generates the message sets such as (Room , Reserve) and (Abe , Announce) automatically after receiving (Office , Arrange). A user may want to be informed the result of failure if the reservation of a meeting room is failed. At that time, The user modifies the message set to the following message set:

{ (Room , Reserve) , [fail] (Office , NoReserve) }

Next, when the user wants to get the reply of the Announce message, the message set is modified to the following message set:

{ (Abe , Announce) , (Office , Attend) }

Furthermore, since the Announce message is sent to three staff objects of Abe, Baba and Chiba concurrently, the message set is modified to the following message set:

```
{ { (Abe , Announce) , (Office , Attend) } |
    { (Baba , Announce) , (Office , Attend) } |
    { (Chiba , Announce) , (Office , Attend) } }
```

4. Discussions

4.1. Three kinds of message expression

M-base supports the following three kinds of message expression:

1. A message flow diagram

2. A message set

3. A message transformation

A message flow diagram is drawn by the modeling and simulation tool. This is the easiest way for end-users since the system generates the corresponding codes in the script language based on simulation.

However, it is not suitable to define complex logic for message flow control by iconic programming.

A message set is described in the script language. This is an easier way for end-users since the message sets correspond to work flow at their office. Sometimes, however, a common object which is included among many message sets, may be required to add exception handling for one of them. It is not easy for end-users to solve the problem of how to assign a new task to the common object.

A message transformation is derived from the message flow diagram and the message sets. Sometimes, however, end-users may define message transformations of some objects directly in the script language. This is because specifications of common components must be defined independent of the individual work flows.

4.2. Control of message flow

These three ways of message expression can be considered from the viewpoint of message flow control. M-base supports the following two ways of message flow control:

1. Integrated control
2. Distributed control

Integrated control of massage flow is performed by a message set which specifies one or more paths of message flow. This way corresponds to cases where work flow of a task is decided at the starting point in the real world. It is easy to modify work flow by rewriting the message set.

Distributed control of message flow is performed by a message transformation which specifies only the relation of an input message and one or more output messages. A path of message flow is expressed by a sequence of message transformations. It may be risky to modify work flow by changes of message transformation because the change may cause side effects of unintentional change of other work flows.

Consequently, M-base recommends users to specify work flow by message sets if possible. On the other hand, objects in high commonality of a domain should be provided as domain-specific component which specifications are defined by message transformations.

4.3. Componentware

In M-base, domain-specific componentware is extracted easily from software architecture of a developed application system since the domain model is constructed based on an object-oriented model. The componentware is classified into the following three categories on granularity of components:

1. An application framework
2. A design pattern
3. A class library

The application framework shows software architecture of an application system. The class library supplies components of classes. The design pattern shows how to combine several related classes for embedding into the application framework. Typical examples of general design patterns are given by E. gamma et al.[9, 12].

4.4. Modeling process

Most of conventional OOA/OOD techniques propose to identify objects or classes from the real world at the first step as Rumbaugh's object model [17] or Shlaer's information model [18]. By emphasizing benefits of data abstraction and encapsulation, object-oriented technologies are apt to be considered as data-oriented approach which is an antithesis against the conventional thesis of function-oriented approach such as structured analysis.

For example, some of conventional OOA/OOD techniques propose to consider nouns in problem specifications as objects and to consider verbs as methods. This idea may be useful for banking systems in which problem domain has been refined enough and a data model has been defined also in conventional systems. If not, too many objects and methods will be selected in vain, especially in an office information system.

In a distributed system for end-user computing, however, the dynamic model at a macro level is required first since the requirements can not be specified exactly at the initial stage. In M-base, modeling and simulation of work flow are repeated first for constructing the dynamic model based on the very simple principle of "Assign one task to an object."

5. Conclusions

One solution was given for two indispensable requirements of new fields with explosive increase in application software on distributed systems, namely, "a domain model $\equiv$ a computation model" and "analysis $\equiv$ design." The practical development process was derived from the two-layer model. That is, the dynamic model corresponding to system behavior, is expressed in a message-driven model, and the static model corresponding to both specifications and static relations of objects, is expressed in classes and its hierarchies. This modeling process is supported by the modeling and simulation tool and the script language. Since untrained end-users are increasing still, further study is needed to enrich domain-specific componentware.

References

[1] G. Booch. Object-oriented design with applications. *Benjamin/Cummings*, 1991.

[2] T. Chusho. End-user computing,(in japanese). *Journal of Information Processing Society of Japan*, 32(8):950–960, Aug. 1991.

[3] T. Chusho and H. Haga. A multilingual modular programming system for describing kn owledge information processing systems. *Proc. the 10th World Computer Congress IFIP'86*, pages 903–908, 1986.

[4] T. Chusho, Y. Konishi, N. Hama, and M. Yoshioka. Application software development environment, m-base, based on the concept of "domain model a computation model," (in japanese). *Information Processing Society of Japan, SIG on Software Engineering*, 96(41):9–16, May 1996.

[5] P. Coad and E. Yourdon. *Object-Oriented Design.* Prentice Hall, 1991.

[6] O. Dahl and C. A. Hoare. Hierarchical program structures. *Structured Programming, Academic Press*, pages 175–220, 1972.

[7] M. Ejiri, Y. Nakano, and T. Chusho. *Artificial Intellignce, (in Japanese).* Shokodo, 1988.

[8] R. G. Fichman and C. F. Kemerer. Object-oriented and conventional analysis and design methodologies. *IEEE Trans. Computer*, 25(10):22–39, Oct. 1992.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns.* Addison Wesley, 1995.

[10] A. Goldberg and D. Robson. *Smalltalk-80 : The Language and its Implementation.* Addison Wesley, 1983.

[11] J. Grudin. Computer-supported cooperative work : history and focus. *IEEE Trans. Computer*, 27(5):19–26, May 1994.

[12] R. Helm. Patterns in practice. *OOPSLA'95, ACM SIGPLAN Notices*, 30(10):337–341, Oct. 1995.

[13] C. Hewitt and H. Baker. Laws for communicating parallel processes. *Proc. the 7th World Computer Congress IFIP'77*, pages 987–992, 1977.

[14] Y. Konishi and T.Chusho. Design of an object-oriented language for analysis and design of cooperative systems, and its feasibility study (in japanese). *Proc. Object-Oriented Software Technology '96 Simposium*, pages 89–94, July 1996.

[15] B. Liskov and S. Zills. Programming with abstract data types. *ACM SIGPLAN Notices*, 9(4):50–59, Apr. 1974.

[16] D. E. Monarchi and G. I. Puhr. A research typology for object-oriented analysis and design. *Comm. ACM*, 35(9):35–47, Sept. 1992.

[17] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, 1991.

[18] S. Shlaer and S. J. Mellor. *Object-Oriented Systems Analysis : Modeling the World in Data.* Prentice Hall, 1988.

[19] J. Udell. Componentware. *BYTE*, pages 46–56, May 1994.

[20] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software.* Prentice Hall, 1990.