M-base : Enduser-Initiative Application Development based on Message Flow and Componentware

Takeshi CHUSHO, Mitsuyoshi MATSUMOTO and Yuji KONISHI

Department of Computer Science, Meiji University
1-1-1 Higashimita, Tama-ku, Kawasaki 214-8571, Japan
e-mail : chusho@cs.meiji.ac.jp

Abstract

Explosive increase in enduser computing on distributed systems requires that endusers develop application software by themselves. One solution is given as a formula of "a domain model $\equiv$ 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 workflow 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 and componentware. At the second stage, a source program in a script language is generated automatically from the message-driven model. Furthermore, if necessary, static structure and detailed specifications of objects are expressed in the script language. Communication among objects is performed by a set of messages instead of a message, for implementation of flexible workflow.*

Key words :
enduser computing, software development environment, distributed systems, object-orientation, domain modeling, visual programming, componentware

1. Introduction

Recently, computer networks for information systems are rapidly spreading on trends of the Internet and intranets. An increasing number of untrained endusers began interacting with computers. Then new software paradigms for such new fields with explosive increase in application software are required.

[1]Mitsuyoshi MATSUMOTO is with Hitachi, Ltd. since Apr. 1998 and Yuji KONISHI is with Hitachi Software Eng. Co., Ltd. since Apr. 1997.

Generally, endusers are classified into the following three typical categories:

1. Clerks using terminals of a large-scale mission-critical information system such as banking systems.
2. Office workers using application packages on personal computers for their own tasks.
3. Clients using public terminals such as ATMs on banking systems.

This paper primarily considers endusers 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 began using application packages not only for their individual task, but also for their cooperative work such as workflow systems and groupware[11]. When these application packages can not satisfy such endusers, they must customize these packages or find new ones. Finally, if there are no 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 endusers, 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. One solution for enduser-initiative application development is given as a formula of "a domain model $\equiv$ 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 $\equiv$ design" is derived since it is not necessary to convert a domain model into a computation model. 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 workflow systems. The 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[9, 17] 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 and then a static model is defined strictly[4, 5]. That is, 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 and componentware[15]. At the second stage, a source program in a script language is generated automatically from the message-driven model. Furthermore, if necessary, static structure and detailed specifications of objects are expressed in the script language[14]. Communication among objects is performed by a set of messages instead of a message, for implementation of flexible workflow. 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 in comparison with conventional modeling methods. The framework of M-base, the modeling and simulation tool, and the script language are described in Sec. 3, Sec. 4 and Sec. 5 respectively.

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 and were expanded into programming methodologies[7]. Object-oriented programming has been already used in practice into various software fields, especially in middleware such as graphical user interface builders and distributed 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[1, 6, 19, 20, 23].

Many of conventional OOA/OOD techniques propose to identify objects or classes from the real world at the first step. For example, some methodologies propose to consider nouns in problem specifications as objects and to consider verbs as methods. This idea may be suitable for large-scale database-centered systems such as 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.

This is because these 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. Recent modeling techniques such as UML (Unified Modeling Language)[2] and VMT(Visual Modeling Techniques)[21] improve the imbalance.

In a distributed system for enduser 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 workflow are repeated first for constructing the dynamic model based on the very simple principle: “Assign one task to an object.”

2.2. Research Goals

A new approach and its support tools were developed for satisfying the following requirements :

1. The target software is a distributed office information system for cooperative work such as a workflow system and groupware.
2. The endusers are office workers who are professionals of office work but are not professionals of information technologies.
3. The system designers are mainly the endusers themselves although system engineers may support the endusers.
4. The maintenance is performed by the endusers 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.

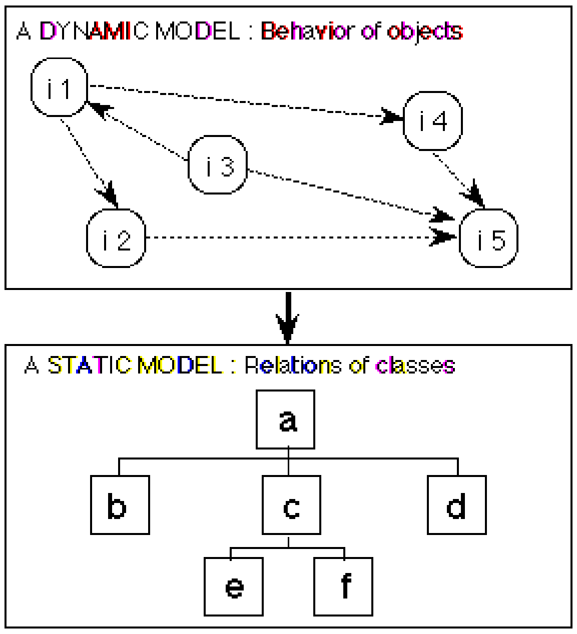
A DYNAMIC MODEL : Behavior of objects
i 1
i 2
i 3
i 4
i 5
A STATIC MODEL : Relations of classes
a
b
c
d
e
f

Figure 1. Conceptual framework of the two-layer model.

This paradigm is called a two-layer model in this paper and the conceptual framework is shown in Figure 1. These two layers are discriminated each other definitely in development process. A source program in a script language is generated automatically from the dynamic model, and is considered as a base of a static model.

In the static model, however, satisfaction degree of the requirements depends on a class library to be used. In particular, domain-specific components, which are sometimes called as business objects, will contribute to easiness of development. M-base promotes the growth of componentware. Furthermore, if necessary, static structure and detailed specifications of objects are expressed in the script language.

2.4. Domain Modeling

2.4.1. Formal 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."

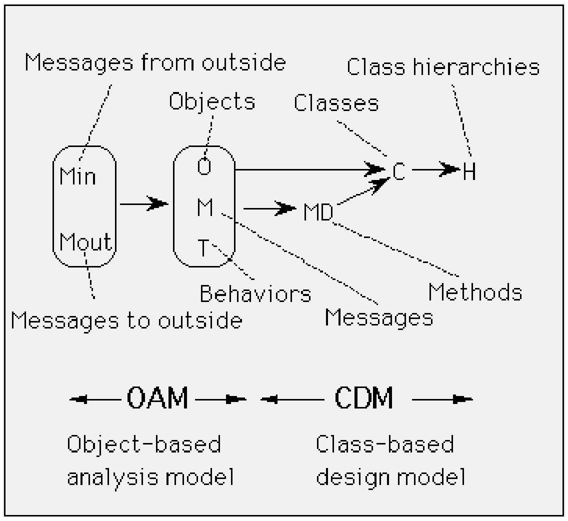

Figure 2. Modeling process based on the two-layer model.

2.4.2. Object-based analysis model

The object-based analysis model is expressed as 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 sender(m[i,j,n]) = o[i] and 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], then a set of messages from the outside and a set of messages to the outside can be denoted respectively as follows:

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

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

T denotes a set of behavior as T = {t[r]} where t[r] is the following message transformation:

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

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 as CDM = {MD, C, H}, where MD, C and H denote a set of methods, a set of classes and a set of class hierarchies respectively.

External specification of each object is 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 in MD(o[j]). That is, o[j] has methods which number is equal to the number of such equivalent sets. Consequently, $MD = \cup j\ MD(o[j])$.

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 in C. That is, if MD(o[i]) = MD(o[j]), o[i] and o[j] are generated from the same class.

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:

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

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:

$MD(c[j])/new = MD(c[j])/old - MD(c[i])$

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:

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

Consequently,

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

2.5. Metaphor-Base Modeling Process for Endusers

Our conceptual framework is based on the object-oriented concepts. However, since endusers are not familiar with these technologies, practical development process has been provided based on metaphors of an office as described below.

Since workflow is essential in most cases of developing a distributed system, it is natural to model the system behavior in message flow 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 in M-base as follows:

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.

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

"Assign one task to an object."

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

3. Framework of Development Environment

Relations between M-base and an application architecture to be developed by using M-base, are shown in Figure 3. M-base provides the following tools :

1. A modeling and simulation tool

2. A script language

3. A user interface builder

4. A component builder

An application to be developed by using M-base, is composed of the following three parts:

1. A model
2. Componentware
3. A user interface

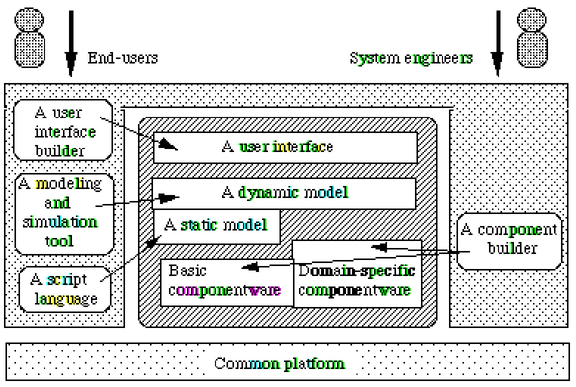

Figure 3. Application architecture (the inner part) and support tools (the outer part).

The model is a body for inherent process in the application, and is partitioned into two parts. The dynamic model is constructed by using the modeling and simulation tool while referring to the domain-specific componentware. The static model is defined in the script language while referring to the basic componentware although a base of the static model is generated automatically from the dynamic model. If necessary, components are developed by using the component builder though system engineers may support it. The user interface is separated from the model for a client/server or 3-tier system configuration, and is constructed by using the user interface builder. The common platform plays an important role in an open system including an distributed object management.

4. Modeling and Simulation

4.1. A Modeling and Simulation Tool

The modeling and simulation tool is used for constructing the dynamic model by mouse manipulation, and is a kind of visual programming tool which supports application development by connecting icons. Conventional tools, however, support only such typical procedures as retrieving data from database, making a table of the data and then displaying its bar chart. On the other hand, M-base supports to express a domain model as message-driven model first, to define semantics of messages in detail if necessary, and to simulate the domain model for validation.

In this section, the following modeling procedure is described while giving an example:

1. Definitions of external specifications
2. Construction of a dynamic model
3. Refinement of each object
4. Simulation of behavior

Let's consider development of the object-oriented office system, OOOffice, for convenience of explanation of a metaphor-base modeling. OOOffice is a system for meeting arrangements. 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 endusers instead of application itself.

4.2. Definitions of External Specifications

The external specifications of OOOffice are defined in the same way as a context diagram of structured analysis or a usecase of OOSE[12] as follows:

1. Identification of endusers

   The enduser of OOOffice is a chairperson.

2. Specifications of functions

   The usecases of OOOffice are meeting arrangements and the cancellation.

4.3. Construction of a Dynamic Model

A dynamic model of OOOffice as shown in Figure 4 is constructed.

1. Identification of objects

   Four types of objects are introduced , that is, a secretary, rooms, instruments and members.

2. Identification of messages among those objects

   The secretary object receives an Arrange message for meeting arrangements and then sends a Reserve message for meeting room reservation to the room object, a Reserve message for projector reservation to the instrument object and Announce messages for notice to the member objects. The member objects return Attend messages.

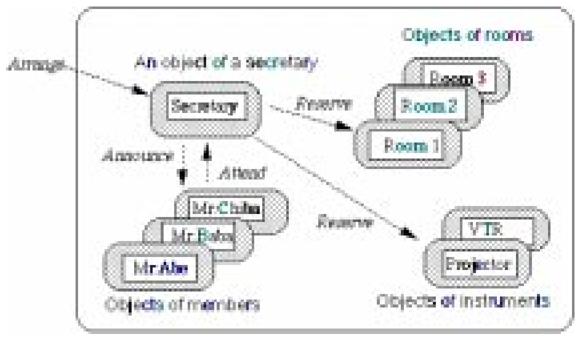

Figure 4. An example of a domain model of a distributed office system, OOOffice.

An actual example is shown in Figure 5 as a part of the domain model of OOOffice shown in Figure 4. Objects are defined by drag-and-drop from the palette of icons. Messages between objects are defined by drawing arrow lines from source objects to drain objects while numbering the messages in order of execution. An enduser is called an actor.

4.4. Refinement of Each Object

After the instance-based domain model is constructed, a class-based static model will be defined. If objects are generated from given components, the following refinement is not necessary. If not, a source program in the script language is generated automatically as a basic part of class definition for each object in the dynamic model. When the role of the object is only transformation from an input message into output messages, the following refinement is not necessary also.

1. Specifications of semantics of methods

   The method semantics specifications play an important role in the modeling process because they bridge a semantic gap between a dynamic model and a static model. Therefore, they can be described under the domain model construction. An example of method semantics specifications for the Arrange method of the secretary object is shown as follows:

   ⟨A⟩ Reserve a meeting room.

   ⟨a1⟩ If not, ask a user the next action.

   ⟨B⟩ Reserve a projector.

   ⟨b1⟩ If not, inform a user.

   ⟨C⟩ Announce a meeting to members.

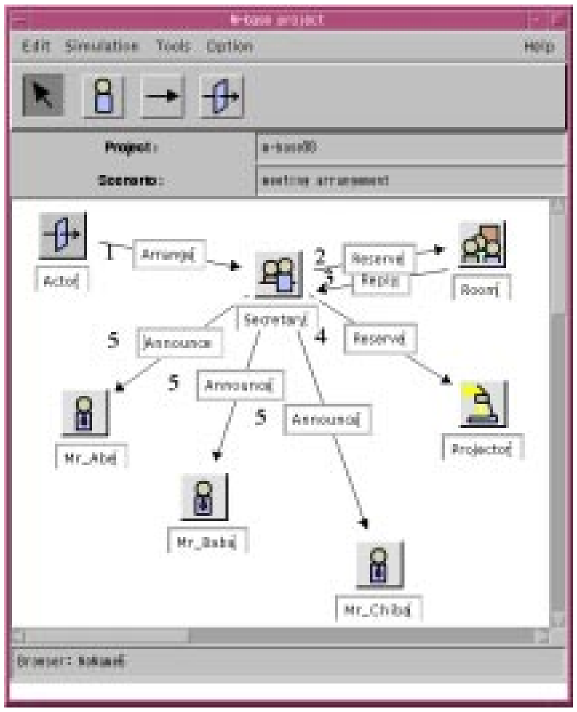

Figure 5. An example of domain model construction by using M-base.

   The main procedure is described in the sequence of ⟨A⟩, ⟨B⟩ and ⟨C⟩. Exception handling is described in the supplements of ⟨a1⟩ and ⟨b1⟩.

2. Scripting of method bodies

   Furthermore, if necessary, detailed specifications are expressed in the script language which is described later.

4.5. Simulation of Behavior

Simulation is executed for validation of the application, both on the domain model and on the event trace diagram, while displaying trace of message flow with method semantics specifications.

1. Selection of a scenario

   In the simulation mode, one of methods to be invoked from outside, is selected.

2. Execution of the scenario

   Simulation is started by a click of the “start” button. Message passing is executed one by one while clicking the “next” button.

An example of simulation on an event trace diagram is shown in Figure 6. After an Arrange message is arrived at the secretary object, the method semantics specifications of the Arrange method is shown in a box. By clicking the "next" button, the reserve message is sent from the secretary object to the room object. Simulation continues while confirming message flow.

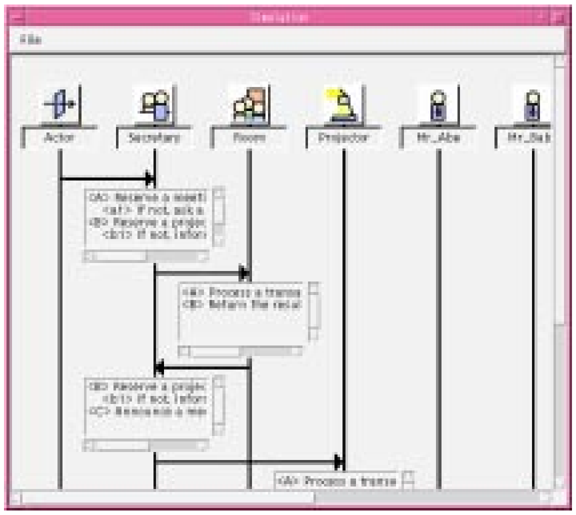

Figure 6. An example of simulation on an event trace diagram.

5. A Script Language

5.1. Design Concepts

Since it is not suitable to define complex logic for message flow control by iconic programming, it is almost inevitable to make endusers use a script language. In M-base, the script language, Hoop [14], is used for defining the static model with class definitions, although basic frameworks of class definitions are generated from the dynamic model.

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 workflow. Generally, the whole workflow is controlled by the meta-system. However, 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 endusers to consider only system behavior of the object-level and to construct pure cooperative systems based on metaphors of communication at offices.

5.2. Syntactic Rules

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

$$\begin{array}{ll} M & ::= Ms||Mp||X \\ Ms & ::= [M, M, ..., M] \\ Mp & ::= [M|M|...|M] \\ X & ::= obj.msg \end{array}$$

where meta-symbols of '::=' and '||' imply 'equal to' and 'or' respectively. 'msg' and 'obj' imply a message and its receiver object respectively. Semantics is described in remainder of this section.

5.3. Sequential Message Sets

When each message in a message set should be executed sequentially, the message set is expressed as [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 obj1 :

$[obj.msg1, obj2.msg2, obj3.msg3]$

Obj1 executes msg1, and then send the following remainder of a message set to obj2 :

$[obj2.msg2, obj3.msg3]$

Obj2 executes msg2, and then send the following remainder of a message set to obj3 :

$obj3.msg3$

Finally, obj3 executes msg3.

This example may correspond to the following workflow 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.

5.4. Concurrent Message Sets

When all messages in 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 :

$[obj.msg1|obj2.msg2|obj3.msg3]$

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 workflow of the mail-order firm is changed to the flow that tasks of (1) and (2) are executed concurrently, the following message set is sent:

$[[obj1.msg1|obj2.msg2], obj3.msg3]$

5.5. An Example of a Hoop Program

Consider OOOffice of Figure 5 again as a typical example of groupware. M-base generates the following Arrange method:

```
public Arrange(){
    [ Room.Reserve(), this.Reply(boolean),
        Projector.Reserve()];
    [ Mr_Abe.Announce() | Mr_Baba.Announce() |
        Mr_Chiba.Announce()];
}
```

This method has two message sets. The first sequential message set corresponds to the message flow of 2, 3 and 4 in Figure 5, which sequence implies a picture drawn with a single stroke of the brush. The second concurrent message set corresponds to the three message flows of 5 in Figure 5.

6. Discussions

6.1. Message Expression and its Flow Control

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 endusers since the system generates the corresponding codes in the script language based on simulation. However, it is not suitable to define complex logic.

A message set is described in the script language. This is an easier way for endusers since the message sets correspond to workflow 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 endusers 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, endusers 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 workflows.

For these three kinds of message expression, M-base supports the following two ways of message flow control:

1. Integrated control
2. Distributed control

The integrated control of massage flow is performed by a message set which specifies one or more paths of message flow. This way corresponds to a case where the workflow of a task is decided at the starting point in the real world. It is easy to modify workflow by rewriting the message set. The integrated control is also performed by the message flow diagram with numbers for execution order since the message flow implies the message set.

On the other hand, the distributed control of message flow is performed by a message transformation which specifies only the relation between 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 workflow by changes of message transformation because the change may cause side effects of unintentional change of other workflows.

Consequently, M-base recommends users to specify workflow by the message flow diagram with numbers for execution order or 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.

6.2. 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[3, 22] may be classified into the following categories though classification is dependent on a viewpoint :

1. An application framework[8, 13]
2. Design patterns[10, 16, 18]
3. A class library
4. Composite objects
5. Black-box components

The application framework provides software architecture and a class library for developing an application system in the specific domain. The design pattern provides a set of classes which collaborate each other for solving a typical design problem. A composite object is composed of several objects and provides a high level component. A black-box component is easy-to-use since it is not necessary for a user to know the source code. M-base supports these componentware. Especially, The script language supports the nested structure of objects for recursive construction of componentware.

6.3. Implementation

Tools of M-base have been implemented by using Java under JDK1.1.5. The modeling and simulation tool is composed of 63 classes whose program size is 7,300 lines. The processor of the script language is composed of 55 classes whose program size is 6,000 lines. Communication protocol for a distributed system was implemented by JavaRMI. The UI builder which is a kind of a draw tool with GUI components, has been almost developed and will be linked to the modeling and simulation tool for exchange of input and output data soon. The component builder under construction is based on JavaBeans.

7. Conclusions

One solution was given for two indispensable requirements of new fields with explosive increase in application software on distributed systems, that is, "a domain model $\equiv$ a computation model" and "analysis $\equiv$ design." The practical enduser-initiative development process was derived from the two-layer model. This modeling process is supported by the modeling and simulation tool and the script language. Since untrained endusers are increasing still, further study is needed to enrich domain-specific componentware.

Acknowledgment

This work on the modeling and simulation tool and the script language has been supported in part by Engineering Adventure Group Linkage Program(EAGL) and by The Telecommunications Advancement Foundation (TAF) respectively. The authors wish to express their gratitude to Mr. Katsuya Fujiwara for icon designs.

References

[1] G. Booch. *Object-oriented design with applications*. Benjamin/Cummings, 1991.
[2] G. Booch, I. Jacobson, and J. Rumbaugh. The unified modeling language. *OOPSLA'97 Tutorial Note*, (35), 1997.
[3] A. W. Brown(Ed.). *Component-based software engineering*. IEEE CS Press, 1996.
[4] T. Chusho. M-base : Object-based modeling of application software as "domain model $\equiv$ a computation model," (in japanese). *Information Processing Society of Japan, SIG on Software Engineering*, 95(104-4):25–32, May 1995.
[5] T. Chusho, Y. Konishi, and M. Yoshioka. M-base : An application development environment for end-users computing based on message flow. *Proc. APSEC'96*, pages 366–375, Dec. 1996.
[6] P. Coad and E. Yourdon. *Object-oriented design*. Prentice-Hall, 1991.
[7] O. Dahl and C. A. Hoare. *Hierarchical program structures, Structured Programming*. Academic Press, 1972.
[8] M. Fayad and D. C. Schmidt. Object-oriented application frameworks. *Commun. ACM*, 40(10):32–38, Oct. 1997.
[9] R. G. Fichman and C. F. Kemerer. Object-oriented and conventional analysis and design methodologies. *IEEE Computer*, 25(10):22–39, Oct. 1992.
[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, 1995.
[11] J. Grudin. Computer-supported cooperative work : history and focus. *IEEE Trans. Computer*, 27(5):19–26, May 1994.
[12] I. Jacobson and et al. *Object-oriented software engineering*. Addison-Wesley, 1992.
[13] R. E. Johnson. Frameworks = (components + patterns). *Commun. ACM*, 40(10):39–42, 1997 Oct.
[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 Technology '96 Simposium*, pages 87–94, July 1996.
[15] M. Matsumoto, Y. Konishi, and T.Chusho. Analysis and design of a software development, m-base, based on "a domain model $\equiv$ a computation model," (in japanese). *Proc. Object-Oriented Technology '97 Simposium*, pages 128–135, July 1997.
[16] S. J. Mellor and R. Johnson. Why explore object methods, patterns, and architectures ? *IEEE Software*, 14(1):27–30, Jan./Feb. 1997.
[17] D. E. Monarchi and G. I. Puhr. A research typology for object-oriented analysis and design. *Comm. ACM*, 35(9):35–47, Sep. 1992.
[18] W. Pree. *Design patterns for object-oriented software development*. Addison-Wesley, 1994.
[19] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design*. Prentice-Hall, 1991.
[20] S. Shlaer and S. J. Mellor. *Object-oriented systems analysis : modeling the world in data*. Prentice Hall, 1988.
[21] D. Tkach, W. Fang, and A. So. *Visual modeling technique*. Addison-Wesley, 1996.
[22] J. Udell. Componentware. *BYTE*, pages 46–56, May 1994.
[23] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing object-oriented software*. Prentice Hall, 1990.