Component-Based Application Development on Architecture of a Model, UI and Components

Takeshi CHUSHO, Hisashi ISHIGURE, Naoyuki KONDA and Tomoaki IWATA

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 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," which implies that one task in cooperative work corresponds to one object in an object-oriented model. Application development environment, M-base[1]*, supports this formula. The application architecture is fixed and is composed of a model, a user interface and components. At the first stage, the system behavior is expressed as a message-driven model by using a modeling tool while focusing on message flow and components. At the second stage, a user interface is generated automatically and may be customized if necessary. Then transition diagrams of user interfaces are generated automatically and used for confirmation of external specifications of the application. Finally, the system behavior is verified by using a simulation tool. This component-based development process is confirmed by feasibility study on a given problem of IPSJ sigRE group.*

Key words :
software architecture, components, software development process, object-orientation, domain modeling, end-user computing

1. Introduction

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

These end-users began using application packages not only for their individual task, but also for their cooperative work such as workflow systems and groupware. When these application packages can not satisfy such end-users, 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 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 an architecture of application software.

One solution for enduser-initiative application development 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. 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 with application architecture. This process requires necessarily a fixed architecture and ready-made components as business objects for component-based development[2, 14].

Application development environment, M-base, supports these formulas for developing cooperative 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[11] which need defining an object model on static structure of objects prior to a dynamic model on interactive behavior among objects.

[1]N. Konda and T. Iwata are with Sony, Co. at present.
This work is supported in part by The Okawa Foundation for information and telecommunications.

In our component-based development process, an application architecture is fixed and the behavior of a domain model is first constructed. That is, the application architecture is composed of a model, a user interface and components. At the first stage, the system behavior is expressed as a message-driven model by using a modeling tool while focusing on message flow and components. At the second stage, a user interface is generated automatically and may be customized if necessary. Then transition diagrams of user interfaces are generated automatically and used for confirmation of external specifications of the application. Finally, the system behavior is verified by using a simulation tool. This component-based development process is confirmed by feasibility study on a given problem of IPSJ sigRE group.

In this paper, an overview of M-base and feasibility studies are described in Sec. 2 and Sec. 3 respectively. In Sec. 4, technical issues on the component-based development are discussed.

2. Overview of M-base

2.1. Research Goals

M-base was developed for satisfying the following requirements :

1. The target software is a distributed office information system for cooperative work.
2. The end-users are office workers who are professionals of office work but are not professionals of information technologies.
3. The system is mainly designed by 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.

2.2. Metaphor-Base Modeling Process

Our conceptual framework is based on the object-oriented concepts. The formal modeling process that a dynamic model is designed prior to a static model, is described in the previous papers[3, 5].

However, since end-users are not familiar with these technologies, practical development process has been provided based on metaphors of an office. 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. Various means of communication 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. 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 objects as if to assign each task to each person under the condition that the sufficient number of able persons exist.

2.3. Development Environment

Relations between an application architecture and M-base are shown in Figure 1. An application architecture to be developed by using M-base, is composed of the three parts of a model, components and a user interface

The model is inherent 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 components. If it is not necessary to develop any new components, the application architecture is composed simply of the dynamic model, components and a user interface. If necessary, the static model is defined. After the skeleton codes of the static model is generated automatically from the dynamic model, the method definitions are refined by using the script language. Basically, however,

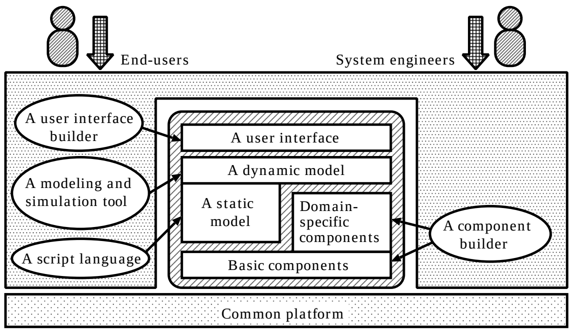


Figure 1. Application architecture (the inner part) and support tools.

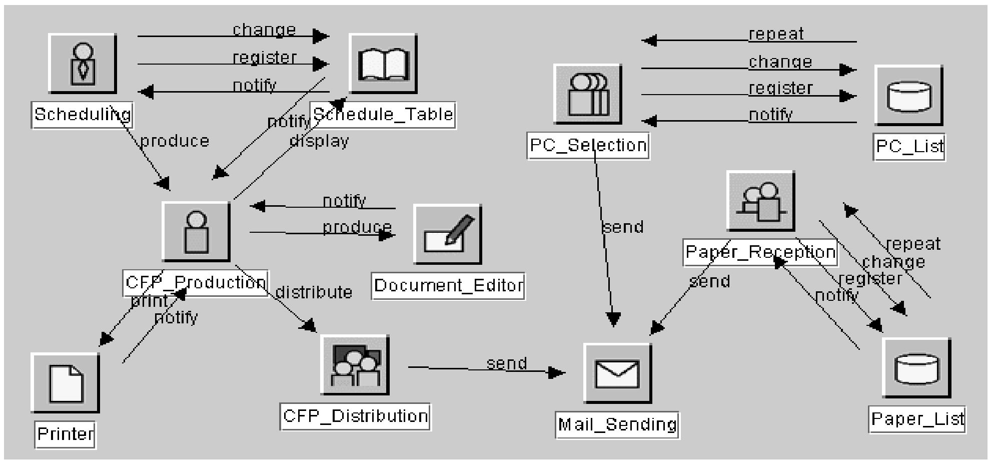


Figure 2. An example of domain model constructed by using the modeling tool.

a static model is an internal form which end-users need not to understand.

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. If necessary, components are developed by using the component builder though system engineers may support it. The common platform plays an important role in an open system including an distributed object management.

3. Feasibility Study

3.1. Modeling Procedure

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

1. Definitions of external specifications
2. Construction of a dynamic model of a domain
3. Refinement of user interfaces
4. Simulation of behavior

We use a given example of “tasks of a program chair for an international conference” which is defined and used since 1998 by the Working Group on Requirement Engineering, the Special Interest Group on Software Engineering, Information Processing Society of Japan [12]. The tasks of a program chair are given as the eleven items. The first sentences of the first five items are described as follows:

(a) The schedule should be decided.

(b) CFP should be made and distributed.

(c) The program committee(PC) members should be selected and registered.

(d) The submitted paper should be given the number to and be registered.

(e) The receipt of the paper should be acknowledged to the author(s).

3.2. Definitions of External Specifications

The initial requirements are refined for definitions of external specifications. Examples on the five items are given. That is, (a) Based on past experiences, the draft of the schedule is generated by entering the first day of a conference, and may be modified. (b) CFP is produced as a HTML document by entering the content corresponding to each item in a given prototype. A plain text and a PDF file are automatically produced and distributed via email. (c) The initial data of the PC members are entered manually into the computer file. (d) The initial data of the submitted paper are entered manually into the computer file also. (e) The receipt of the paper is sent to the author(s) automatically.

3.3. Construction of a Dynamic Model

The modeling and simulation tool is used for constructing the dynamic model by mouse manipulation as a kind of visual programming tool. Especially, this tool supports to express a domain model as message-driven model first and to simulate the domain model for validation.

A dynamic model was constructed as shown in Figure 2 while introducing eleven kinds of objects. Objects are defined by drag-and-drop from the palette of icons. A message between objects is defined by drawing an arrow line from the source object to the drain object.

If an object is not given as a ready-made component, the object must be refined. M-base supports three ways for component refinement. The script language was first implemented. Although complicate workflows was expressed precisely in message sets [5], it was not easy for end-users to use it.

Next, M-base supported the nested structure of objects for recursive definition of user-defined components by providing the four kinds of components, namely, control components, UI components, domain-specific components and other primitive components.

Recently, the rule expression has been implemented, for easy description of branch conditions. For example, in the "produce" method of the CFP_Production object, the following rules are described for branch conditions:

- if Printer = yes then print;
- if CFP_Distribution = yes then distribute;

3.4. Refinement of User Interfaces

After the instance-based domain model is constructed, the UI builder generates user interfaces automatically by using the information which is sent from the modeling tool. Figure 3 shows an example of a window for input of contents of the CFP. The UI builder supports end-users to customize it for improving user friendliness if necessary.

Next the UI builder generates the transition diagrams of user interfaces. An end-user can validates external specification by using these diagrams. As a result, two errors of missing the necessary message flows were found and specifications of five items were improved in our experience.

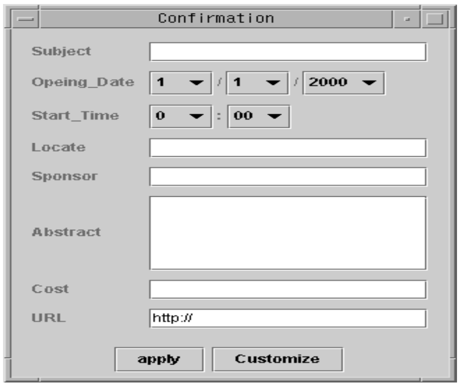
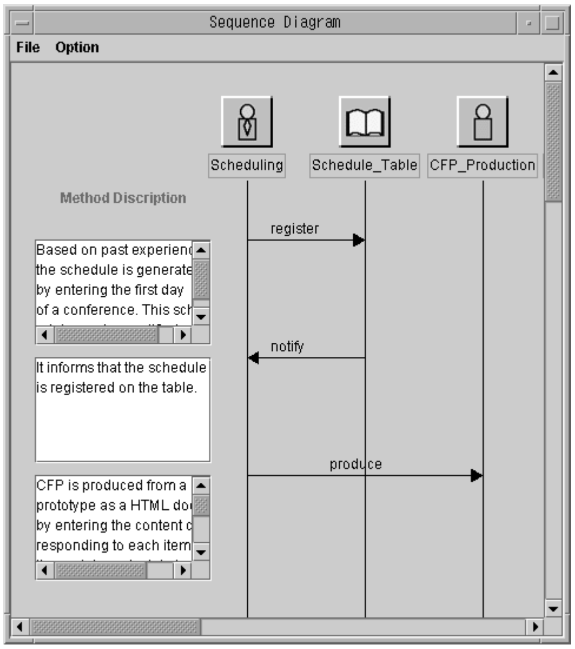


Figure 3. An example of a user interface generated automatically.

3.5. Simulation of Behavior

Simulation is executed for validation of the application, both on the domain model in Figure 2 and on the sequence diagram in Figure 4, while displaying traces of the message flow.

For execution of each scenario, one of methods to be invoked from outside, is first selected. Then the simulation is started by a click of the "start" button. Message passing is executed one by one while clicking the "next" button.



Based on past experien
the schedule is generate
by entering the first day
of a conference. This sch

It informs that the schedule
is registered on the table.

CFP is produced from a
prototype as a HTML do
by entering the content c
responding to each item

Figure 4. An example of simulation on a sequence diagram.

4. Discussions

4.1. Object-Oriented Modeling Process

Many of conventional OOA/OOD techniques propose to identify objects or classes from the real world at the first step. 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)[1] and its related approaches [9, 6] improve this imbalance and become more flexible, although UML does not define design process.

For end-user computing, however, the dynamic model at a macro level based on instance objects, is required first because the domain model is specified while corresponding to objects with tasks of the real world. 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."

4.2. Application Architecture and Components

In general, software architecture is defined in terms of a collection of components and interactions among those components [13]. M-base supports application architecture as a model, components and a user interface. That is, interactions among components are expressed as a domain model, and the user interface is separated from the domain model.

This architecture is similar to the 3-tier architecture of presentation, function and data but not the same. A presentation layer corresponds to UI. A function layer corresponds to a domain model and may include business components. A data layer implies DB components.

One of issues on components is a granularity gap between an application and components. There are some ways for enlargement of granularity. That is, an application framework [7] provides software architecture and a class library. Design patterns [8] are micro-architectures and smaller architectural elements than frameworks [14]. A composite object [10] is composed of several objects and provides a high level component. For example, the component-base technologies of frameworks and design patterns were applied into enduser-initiative agent-based application development [4].

The recursive definition of the component is essential for large-scale applications, for top-down development by stepwise refinement and/or for bottom-up development by building block approach, as follows:

$<$A$>$ ::= a set of $<$A$>$ | $<$a$>$

where $<$a$>$ is a primitive component and $<$A$>$ is an application or a composite component.

M-base supports the nested structure of objects for recursive construction of components. However, the best case for end-users is that they produce applications by a dynamic model, UIs and ready-made domain-specific components which are called as business objects.

5. Conclusions

One solution for enduser-initiative application development was given based on the formulas of "a domain model $\equiv$ a computation model" and "analysis $\equiv$ design." The architecture was fixed and was composed of a domain model, UIs and components. The feasibility study confirmed the component-based process by using tools of the environment, M-base, such as the modeling tool and the UI builder. These tools were implemented by using Java and are composed of 185 classes which program size is 18,800 lines.

References

[1] S. S. Alhir. *UML*. O'reilly, 1998.

[2] A. W. Brown(Ed.). *Component-based software engineering*. IEEE CS Press, 1996.

[3] T. Chusho. M-base : Object-based modeling of application software as "a domain model $\equiv$ a computation model," (in japanese). *Information Processing Society of Japan, SIG on Software Engineering*, 95(104-4):25–32, May 1995.

[4] T. Chusho and K. Fujiwara. Facl : A form-based agent communication language for enduser-initiative agent-based application development. *Proc. COMPSAC2000*, Oct. 2000.

[5] T. Chusho, M. Matsumoto, and Y. Konishi. M-base : Enduser-initiative application development based on message flow and componentware. *Proc. COMPSAC98*, pages 112–120, Aug. 1998.

[6] D. F. D'Souza and A. C. Wills. *Objects, components and frameworks with UML*. Addison-Wesley, 1999.

[7] M. Fayad and D. C. Schmidt. Object-oriented application frameworks. *Commun. ACM*, 40(10):32–38, Oct. 1997.

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

[9] I. Jacobson, G. Booch, and J. Rumbaugh. *The unified software development process*. Addison-Wesley, 1999.

[10] D. Krieger and R. M. Adler. The emergence of distributed component platforms. *IEEE Computer*, 31(3):43–53, Mar. 1998.

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

[12] A. Ohnishi. Requirements engineering working group (in japanese). *Winter Workshop in Kouchi*, IPSJ Simposium series Vol.99(1):21–26, Jan. 1999.

[13] M. Shaw and D. Garlan. *Software architecture*. Prentice Hall, 1996.

[14] C. Szyperski. *Component Software*. Addison-Wesley, 1997.