Enduser-Initiative Application Development based on Architecture of a Model, UI and Components

Takeshi CHUSHO, 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." 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. The application architecture is fixed as 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. The system behavior may be defined recursively in the manner of stepwise refinement. This implies that end-users define composite 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, componentware, software development process, object-orientation, domain modeling, end-user computing

[1]Tomoaki IWATA is with Sony, Co. since Apr. 1999.

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[13]. 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[3, 22, 24].

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[11, 17] which need defining an object model on static structure of objects prior to a dynamic model on interactive behavior among objects.

Our component-based development process has the following features:

- Application architecture is fixed.

- Behavior of a domain model is first constructed [5, 7].

- Composite components are constructed recursively.

That is, the application architecture is composed of a model, a user interface and componentware. 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. The system behavior may be defined recursively in the manner of stepwise refinement. This implies that end-users define composite 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, overview of M-base, feasibility study, and discussions about modeling process, application architecture and a component model, are described in Sec. 3, Sec. 4 and Sec. 5 respectively.

2. Overview of M-base

2.1. 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 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.

2.2. Formal modeling process

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 in this paper. These two layers are discriminated each other definitely in development process. In particular, domain-specific components, which are sometimes called as business objects, will contribute to easiness of development because a dynamic model with ready-made domain-specific components need not to require the construction of the corresponding static model. M-base promotes the growth of componentware.

The modeling process in M-base is formalized as shown in Figure 1. 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.”

The object-based analysis model is expressed as OAM = { O, M, T}. O denotes a set of objects. M denotes a set of messages. Min and Mout are subsets of M. That is, they denote a set of messages from the outside and a set of messages to the outside respectively. T denotes a set of behavior as a set of message transformation which implies that an object receives a message and then sends a sequence of messages.

In short, a domain model for a distributed system is constructed in accordance with a procedure shown in Figure 1. That is, Min and Mout are confirmed first. Then, while examining message flow processes, O, M and T are identified.

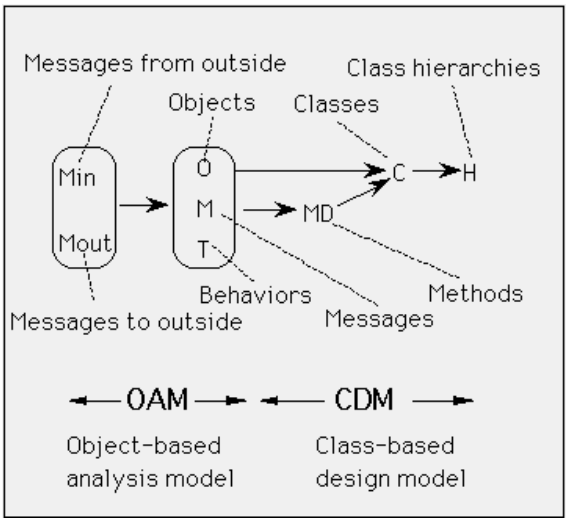

Figure 1. Modeling process based on the two-layer model.

Next, if necessary, 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. The formal modeling process is described in the previous paper[5, 7].

2.3. Metaphor-Base Modeling Process for End-users

Our conceptual framework is based on the object-oriented concepts. 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 workflow is essential in many 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 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.

2.4. Framework of Development Environment

Relations between an application architecture and M-base are shown in Figure 2. An application architecture to be developed by using M-base, is composed of the following three parts:

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

M-base provides the following tools :

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

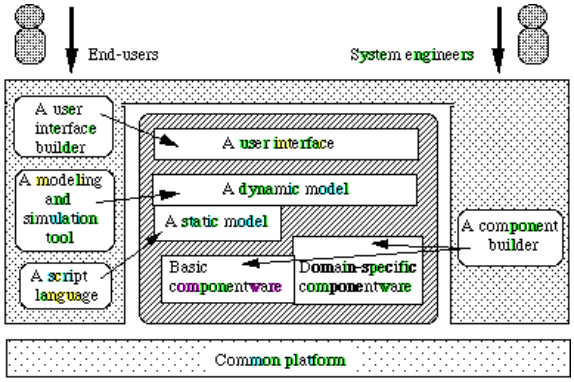

Figure 2. Application architecture (the inner part) and support tools (the outer part).

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 components. If it is not necessary to develop any new components, the application architecture is composed simply of the dynamic model, components and user interfaces.

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, 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 by the Working Group on Requirement Engineering, the Special Interest Group on Software Engineering, Information Processing Society of Japan. The tasks of a program chair are given as the eleven items.

The first lines of the first five items, which are used as initial requirements in this section, are described as follows:

(a) The schedule should be decided.

(b) CFP should be made and distributed.

(c) The program committee 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 as follows:

(a) Based on past experiences, the draft of the schedule is generated by entering the first day of a conference. This schedule can be modified.

(b) CFP is produced from a given prototype as a HTML document by entering the content corresponding to each item in the prototype. A plain text and a PDF file of CFP are automatically produced from this HTML document and distributed via email.

(c) The initial data of the program committee members are entered manually into the computer file.

(d) The initial data of the submitted paper are entered manually into the computer file also. For reduction of this task, the abstract of each paper should be required to be sent by email.

(e) The receipt of the paper is sent to the author(s) automatically when the submitted paper is registered into computer file.

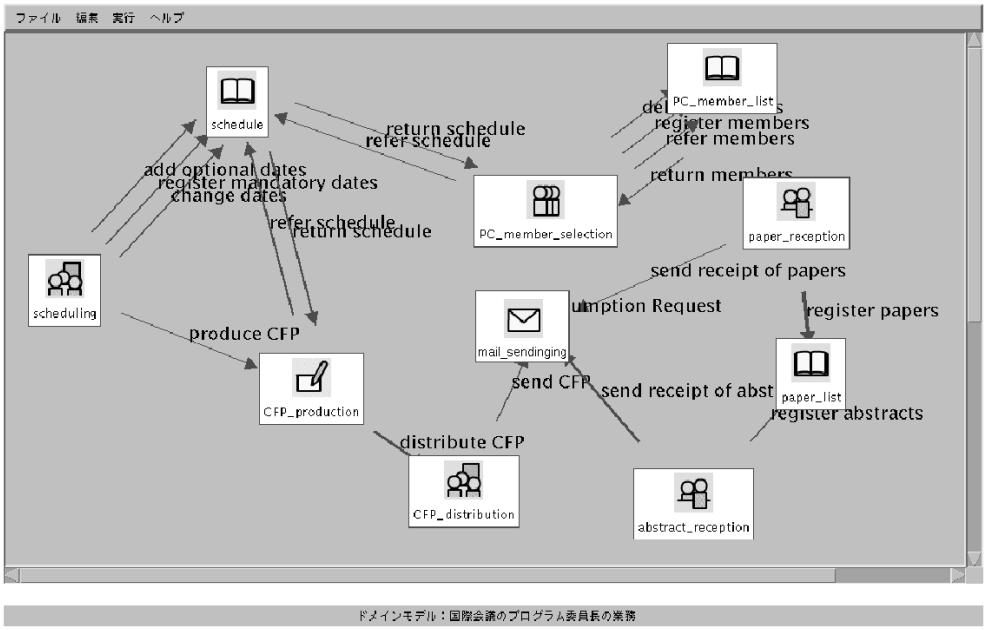

Figure 3. An example of domain model constructed by using the modeling tool.

3.3. Construction of a Dynamic Model

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 and to simulate the domain model for validation.

A dynamic model as shown in Figure 3 was constructed.

1. Identification of objects

   Ten kinds of objects were introduced, that is, scheduling, schedule, CFP_production, CFP_distribution, PC_member_selection, PC_member_list, paper_reception, paper_list, abstract_reception, mail_sending.

2. Identification of messages among those objects

   For example, the CFP_production object receives the “produce CFP” message and then sends a “distribute CFP” message to the CFP_distribution object.

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.

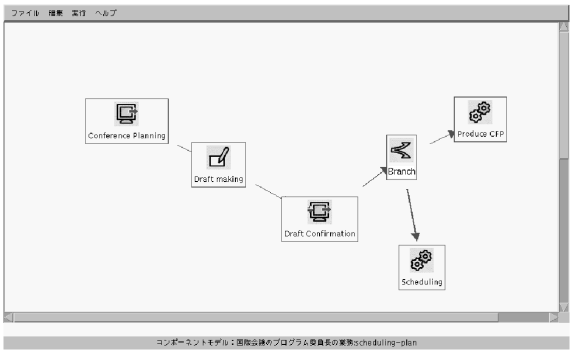

Figure 4. An example of a composite component defined recursively by using the modeling tool.

If an object is not given as a ready-made component, the object is refined. For example, the scheduling object was refined as shown in Figure 4. M-base supports 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. There are six control components of start, repetition start, repetition end, conditional branch, join and event occurrence.

There are four UI components of initial input, input on demand, data display, and display-and-input.

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 to be displayed as items, which information is sent from the modeling tool. Figure 5 shows an example of a window for input of the opening date and output of the draft schedule. The UI builder supports end-users to customize it for improving user friendliness if necessary.

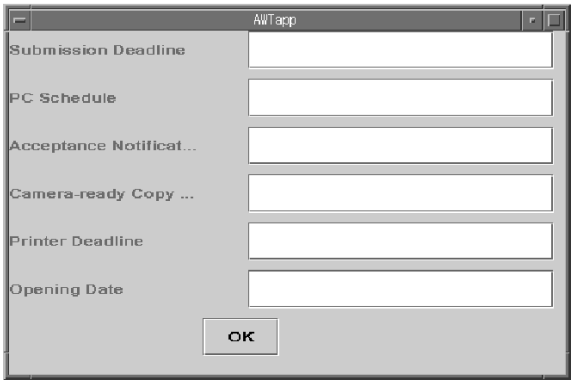

Figure 5. An example of a user interface generated automatically by the UI builder.

Next the UI builder generates the transition diagrams of user interfaces as shown in Figure 6. 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.

3.5. 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.

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.

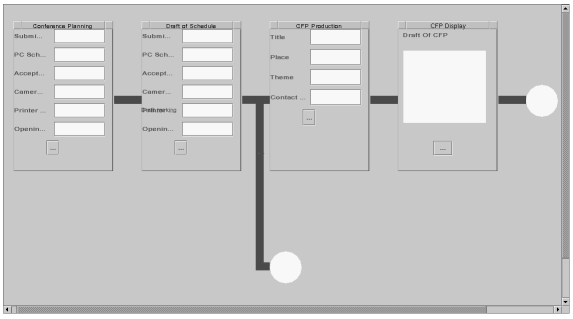

Figure 6. An example of a user interface transition diagram generated automatically by the UI builder.

4. Discussions

4.1. Object-Oriented Modeling Process

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[9]. 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[2, 8, 19, 21, 25].

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)[1] and VMT(Visual Modeling Techniques)[23] improve the imbalance.

Although UML does not define design process, it supports some kinds of diagrams for description of a dynamic behavior model such as a sequence diagram and collaboration diagram. These diagrams, however, are described based on classes.

In a distributed system for end-user computing, however, the dynamic model at a macro level based on instance objects, is required first since 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 [20]. 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 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 a DB management system. The difference is shown by classifying components into three categories:

- GUI components
- Business components
- DB components

In M-base, GUI components are included in UI, and Business components and DB components are included in componentware.

The feature of the application architecture in M-base, is a domain model which is expressed by a message passing model, and promotes enduser-initiative development of applications such as workflow systems and groupware.

In construction of these distributed system, correspondence of UIs and components to network nodes can be decided based on a physical environment because the application architecture in M-base is specified from the logical view, not from the physical view.

Furthermore, as an infrastructure of these distributed system, the ORB(Object Request Broker) architecture such as CORBA is applicable because the M-base engine, which controls execution of a domain model by deciding the execution sequence of components, supports network transparency of components. In OMG, business objects are defined as components of the information system that directly represent the business model, and construct an application with an application architecture based on CORBA [4].

4.3. Component Model

In M-base, domain-specific components are extracted easily from software architecture of a developed application system since the domain model is constructed based on an object-oriented model. One of issues on finding components is a granularity of a component. There are some ways for enlargement of granularity as follows:

1. An application framework [10, 14]
2. Design patterns [12, 16, 18]
3. Composite objects [15]

The application framework provides software architecture and a class library for developing an application system in the specific domain. If a domain is limited, not only the architecture but also implementation is almost fixed as the framework which is composed of components. The framework technology promotes enduser-initiative application development by reducing hot spots to be customized, as our experiences confirmed it [6].

The design pattern provides a set of classes which collaborate each other for solving a typical design problem. Design patterns are micro-architectures and smaller architectural elements than frameworks [22].

A composite object is composed of several objects and provides a high level component. 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 as an example has been shown in Figure 4 previously. 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.

JavaBeans supports the nested structure of objects by using a container. Communication among beans is performed by events. M-base support this event-based communication in order to use beans as M-base components in future.

4.4. Implementation

Tools of M-base have been implemented by using Java under JDK1.1.7. The modeling and simulation tool, the UI builder and the M-base engine are composed of 175 classes which program size is 25,000 lines. Data which are passed from the modeling and simulation tool to the UI builder, is described in XML for extensibility.

5. 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 enabled by the fixed architecture composed of a domain model, UIs and components. The feasibility study confirmed this component-based process by using tools of the environment, M-base, such as the modeling tool and the UI builder.

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

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

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

[4] C. Casanave. *Business-object architectures and standards*. OMG Business Object Domain Task Force, 1998.

[5] 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.

[6] T. Chusho and K. Fujiwara. wwhww : An application framework of distributed systems for enduser-initiative development. *Proc. APSEC'98*, pages 102–109, Dec. 1998.

[7] 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.

[8] P. Coad and E. Yourdon. *Object-oriented design*. Prentice-Hall, 1991.

[9] O. Dahl and C. A. Hoare. *Hierarchical program structures, Structured Programming*. Academic Press, 1972.

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

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

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

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

[14] R. E. Johnson. Frameworks = (components + patterns). *Commun. ACM*, 40(10):39–42, 1997 Oct.

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

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

[21] S. Shlaer and S. J. Mellor. *Object-oriented systems analysis : modeling the world in data*. Prentice Hall, 1988.

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

[23] D. Tkach, W. Fang, and A. So. *Visual modeling technique*. Addison-Wesley, 1996.

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

[25] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing object-oriented software*. Prentice Hall, 1990.