객체지향 소프트웨어 개발 방법론의 객체지향 모델링 : OMOS(Object-oriented software development Method for Object-oriented software System)

최 성 운†

요 약

객체지향 모델은 구조적 모델에 비해 모듈성, 재사용성, 유지보수성, 확장성 등의 장점을 제공한다. 이러한 이유로 객체지향 소프트웨어 개발 방법들은 정보 시스템 개발에 광범위하게 적용되고 있다. 그러나 현재의 객체지향 소프트웨어 개발 방법들은 목적 소프트웨어 시스템의 객체지향 모델링을 지원할 뿐, 방법론 그 자체는 구조적 및 절차적 체계를 기반으로 하고 있다. 본 논문에서는 객체지향 소프트웨어 개발 방법론을 객체지향적으로 모델링한 OMOS(Object-oriented software development Method for Object-oriented software System)를 제시한다. OMOS에서는 개발 산출물과 관련된 활동이 객체로써 모델링 되며, 개발 생명주기 프로세스가 객체간의 상호작용으로 모델링 된다. 소프트웨어 개발 방법론의 객체지향적 모델링은 방법론 자체의 재사용성, 유연성, 확장성, 유지보수의 용이성 등의 장점을 제공한다.

An Object-Oriented Modeling of Object-Oriented Software Development Methods : OMOS(Object-oriented software development Method for Object-oriented software System)

Sungwoon Choi†

ABSTRACT

Object-oriented software development methods are used to develop object-oriented software systems. Object-oriented systems are believed to have better modularity, reusability, maintainability, and extensibility than systems modeled in conventional methods. Current object-oriented software development methods, however, are modeled in terms of procedural, functional, and structural models. Theses models cause problems such as tight coupling among activities, and uncontrolled access to global artifacts. In this paper, we introduce OMOS (Object-oriented software development Method for Object-oriented software System), an object-oriented modeling of object-oriented software development methods. Artifacts and their related activities are modeled as classes and objects. Development lifecycles are modeled as interactions among the objects. By modeling the software development method in an object-oriented way, OMOS achieves better reusability, flexibility, extensibility, and maintainability.

키워드 : 객체지향 소프트웨어 시스템(Object-oriented software system), 객체지향 소프트웨어 개발 방법론(object-oriented software development method), 생명주기 모델링(lifecycle modeling), 재사용성(reusability), 유연성(flexibility)

1. Introduction

A software development method is a planned procedure by which a specified goal is approached. Software development methods consist of activities, artifacts, and processes. Activities are certain phases in software development and define the functionality that should be performed by a developer. Artifacts are results of related activities and defined based on either formal models or informal documents. A process is a group of activities to achieve a goal of software development. Waterfall, incremental and evolutionary processes are typical processes [1-3].

Conventional development methods such as Method/1 [9] and IEM (Information Engineering Method) [10] develop software systems based on functional, procedural, and structural models. Object-oriented development methods develop object-oriented systems, however object-oriented development methods themselves are defined by procedural, struc-

† 종신회원 : 명지대학교 컴퓨터공학부 교수
논문접수 : 2001년 4월 16일, 심사완료 : 2001년 7월 23일

tural, and functional models. In those methods, activities and artifacts are not properly modularized and encapsulated : Shared artifacts can be accessed randomly and coupled tightly by their related activities. These make hard to reuse activities and artifacts of the conventional methods in different system developments. Life-cycle reuse is a harder issue. Procedural sequencing of tightly coupled activities makes it almost impossible to find reuse patterns of the development lifecycle [13].

In this paper, we present OMOS (Object-oriented software development Method for Object-oriented software System), an object-oriented modeling of object-oriented software development method. Object-Oriented modeling of object-oriented software development methods encapsulates activities and artifacts as classes. Life-cycle modeling is defined as interactions among the objects. By properly modularizing activities and artifacts and generically defining life-cycle process, we can achieve reusability for many different problem domains.

This paper is organized as follows : In section 2, we classify conventional software development methods based on 1) paradigms applied to target software, and 2) paradigms used for defining software development methods. Section 3 presents OMOS, an object-oriented design of object-oriented software development methods. In section 4, we illustrate OMOS in detail with an example. We conclude in Section 5 with summaries and future research directions.

2. Object-Oriented Software Development Method

Most traditional software development methods develop systems based on procedural, functional and structural paradigms, such as structured analysis and design technique [4], structured analysis and structured design (SASD) [5-6]. Structural models distinguish functions from data, where functions have behavior that transforms information contained in data. In an object-oriented paradigm, a world is viewed as a set of objects interacting with each other and an object encapsulates data with its related functions. Object-oriented software development method is a development method for object-oriented systems. Object-oriented models provide better modularity, reusability, maintainability, and extensibility than structural models [7-8].

Based on applied paradigms to target software system and method itself, we can categorize four different kinds of software development methods as shown in <table 1>. (1) object-oriented software development method for object-oriented software systems (OMOS), (2) object-oriented software development method for structural software system (OMSS), (3) structural software development method for object-oriented software system (SMOS), and (4) structural software development method for structural software system (SMSS).

〈Table 1〉 Modeling Paradigm vs. Method Modeling Paradigm

Target Software System Modeling Paradigm / MethodModeling Paradigm	Object-Oriented	Structural /Procedural /Functional
Object-Oriented	Object-oriented Method for Object-oriented software System (OMOS)	Object-oriented Method for Structural software System (OMSS)
Structural /Procedural /Functional	Structural Method for Object-oriented software System(SMOS)	Structural Method for Structural software System (SMSS)

Structural software development method(SMSS) supports a structural way of development for structural modeling of software systems. Elements of the development methods as well as the target systems are modeled based on a structural paradigm. Activities and artifacts are compared to functions and data respectively. A Lifecycle-process is modeled as control flow or functional mapping among the activities modeled as functions. Conventional development methods such as Method/I and IEM (Information Engineering Method) are examples of developing systems based on functional, procedural, and structural models [9-10].

Software development methods for object-oriented systems (SMOS) can be also considered as structural if the development methods are not modeled as an object-oriented way. In current object-oriented methods, target systems are modeled in an object-oriented way, but not the methods themselves. Activities (functions or operations) and related artifacts (data or attributes) are separately modeled. Shared artifacts can be accessed randomly and coupled tightly by their related activities. Also, life-cycle processes are sequenced procedurally. Procedural sequencing of tightly coupled activities makes it almost impossible to find reuse patterns of the development lifecycle. Current object-oriented methods, thus, cannot gain benefits of object-oriented modeling, such as, better modularity, reusability, maintainability, and extensibility. Most of object-oriented methods such as OMT(Object Modeling Technique) [11], OOA/DA (Object-Oriented Analysis and Design with Application)

[12], OOSE (Object-Oriented Software Engineering)[13], Navigator[14], RUP(Rational Unified Process) [15-16] can be classified as this category.

Object-oriented modeling of object-oriented software development method (OMOS) encapsulates activities and artifacts as classes. Life-cycle modeling is defined as interactions among the objects. <Table 2> shows the relationships between SMOS and OMOS.

<Table 2> SMOS vs. OMOS

Structural Method for Object-oriented software System (SMOS)	Object-oriented Method for Object-oriented software System (OMOS)
Artifact	Model Object
Activity	
Lifecycle process	Interactions among objects

Process tailoring is the activity of creating a process description by elaborating, adapting, and/or completing the details of process elements or other incomplete specifications of a process. Specific business needs for a project will usually be addressed during process tailoring. [20] Advantages of OMOS in the process tailoring can be summarized as follows :

(1) Reuse of both objects and processes for software development : OMOS provides encapsulation of software development process. An object-oriented software development method is defined in terms of objects (activities and artifacts) and interactions among the objects (process model). Operations of an object are defined in a single syntactic unit and hidden from the users of the object. The only operations visible are those provided in the object interface. The process tailoring in OMOS can be easily performed in the object-oriented fashion : we examine objects and the interfaces of the objects to configure where to perform process tailoring.

(2) Flexible application of development processes : Extensibility and portability are other difficult things to attain in software development methods. There are no distinctions between specifications and implementation of activities in the structured software development methods. Object-oriented models provide clear distinction between these two parts. In OMOS, an abstract software development method can be defined. Based on that, elaborating, adapting, and/or completing the details of process elements or other incomplete specifications of a process can be done by properly setting object attributes, and overriding object operations. Therefore, various versions of a development method can be easily specialized depending on a problem domain.

(3) Self-documentation of development experience and easy maintenance : Objects are easier to be managed independently than function and data pair. In OMOS, all elements are objects : related process artifacts and activities are documented and stored in a single object unit for better comprehensibility. Iterative and evolutionary lifecycle models can be modeled by maintaining objects and their interactions.

In the next section, we present OMOS, an object-oriented method of object-oriented software systems.

3. OMOS : Object-oriented software development Method for Object-oriented software System

OMOS is an object-oriented method for object-oriented system. OMOS is defined in terms of *model-objects* and *life-cycle events*. *Model-objects* encapsulate artifacts and their related activities. Life cycle processes can be modeled as *life-cycle events* that represent interactions among *model-objects*.

OMOS develops model-objects and life-cycle events through two phases : *object modeling*, and *interaction modeling*. In the subsequent sections, we will explain each of the two phases.

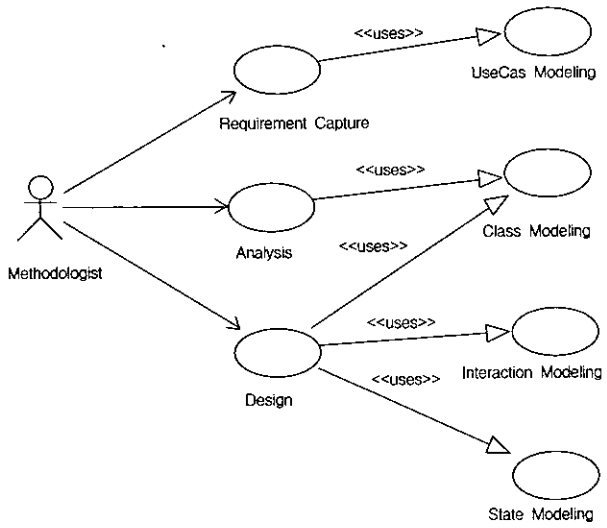
3.1 Object-Modeling of OMOS

Object-modeling phase of OMOS identifies classes and their relationships that constitute software development methods. All artifacts of the software development methods, for example, deliverables, are encapsulated as classes, while related tasks and activities are defined as operations of the class. Classes are organized into the class view according to their relationships : association, generalization, and various kinds of dependency, such as realization and usage.

OMOS develops classes and their relationships with the following evolutionary path : 1) finding classes, 2) defining class attributes and class relationships, and 3) defining class operations.

A use-case driven approach is performed to find out classes that constitute the object-oriented software development methods [17-19]. In OMOS, an actor is responsible to define schedules, activities and processes of the Object-Oriented software development, especially, in the phases of

Requirement Capture, Design and Analysis. (Figure 1) shows the use-case diagram of OMOS object modeling. Five packages are developed to pack the classes : specification package (SpecificationModel), use-case modeling package (UseCaseModel), interaction model package (InteractionModel), class model package (ClassModel), and state model package (StateModel). SpecificationModel packs classes that are used to define requirements specifications, common terms and glossary. UseCaseModel packs use-case classes, actors, and relationships. ClassModel, InteractionModel, and StateModel include classes that constitute class diagrams, sequence diagrams, and state diagrams, respectively.



(Figure 1) Use-Case Modeling of Object-Oriented Development Method

OMOS defines class attributes and relationships among the classes by analyzing static structures of the classes. For example, an Actor class of UseCaseModel idealizes a user of a system, i.e. OMOS methodologist. Actor attributes characterize a OMOS methodologist, such as ActorName and ActorDescription. Deliverables are identified and properly encapsulated as classes. For example, requirement specifications, glossary and terms are deliverables of the Requirement Capture phase, thus encapsulated as classes. Classes are organized according to their relationships. For example, RequirementSpec are related to Actor, UseCase class with the association relationship, since well-defined requirements specifications are guidelines for actors, use cases and classes of the modeled system. Refer to Section 4 for more detailed explanations on class relationships.

Class operations are defined by software development activities of RUP(Rational Unified Process) [15-16]. RUP is composed of requirements capture, analysis & design, implementation and test phase. Each phase is further refined by activities. OMOS models each task and activity of RUP with class operations. <Table 3> shows part of RUP tasks and the corresponding OMOS class operations.

Refer to Section 4 for detailed explanations on class operations.

〈Table 3〉 OMOS class operations in Requirement Capture Phase (selection)

RUP Activity	RUP Task	OMOS class	OMOS operation
Capture a common vocabulary	Find common terms	Requirement Spec	Find CommonTerms()
		Term	Describe Terms()
		Glossary	Add Terms()
	Evaluate your results	Glossary	Evaluate Glossary()
Find Actors and Use Cases	Find actors	Requirement Spec	FindActor()
	Find use cases	Requirement Spec	Find Use Case()
	Describe how actors and use cases interact	Requirement Spec	Describe Acor And Use Case Interaction()
	Evaluate your results	UseCase	Evaluate Use Case()
		Actor	Evaluate Actor()

3.2 Life-Cycle Modeling of OMOS

Lifecycle modeling of OMOS defines the order and manner in which the activities of the software lifecycle are executed. The lifecycle modeling has two components : 1) *content*, and 2) *time*. The content component identifies tasks and activities that comprise the software lifecycle. The time component defines the time-related components, such as, prototypes, architectural baselines, and the software release [1-3]. Lifecycle modeling of OMOS is a process of capturing time-dependent behaviors of OMOS objects. In the lifecycle modeling of OMOS, we use RUP to identify tasks and activities that comprise the software lifecycle. The time component of OMOS is defined by the sequences of activities among the OMOS objects, the flow of control across the objects, and the interactions between the objects.

OMOS defines lifecycles of UseCaseModel, ClassModel, InteractionModel, and StateModel package identified in the Object-Modeling.

Use Case Modeling is a process to define the lifecycle of activities related to use case diagramming. The lifecycle model of UseCaseModel is defined by the interactions among the UseCaseModel classes. All the RUP activities for the use-case diagramming in the requirements capture phase are described by the sequence of message exchanges among

the UseCaseModel classes. *Class Modeling* is the process that defines time-series interactions among ClassModel classes. The contents of the class modeling are defined by the activities in RUP, especially class diagramming activities in the analysis and design phase. *Interaction Modeling* defines the interactions among InteractionModel classes. The contents of the interaction modeling are defined by the activities in RUP, especially interaction diagramming activities in the analysis and design phase. *State modeling* is the process that defines the interactions among the State-Diagram classes. The contents of the state modeling are defined by the activities in RUP, especially state diagramming activities in the analysis and design phase. <Table 4> summarizes OMOS lifecycle modeling discussed so far.

〈Table 4〉 OMOS lifecycle modeling

RUP Phase	OMOS Lifecycle Modeling	RUP Activities	OMOS Messages
Require-ments	Use Case Modeling	Use-case diagramming activities	UseCaseModel Messages
Analysis/ Design	Class Modeling	Class diagramming activities	ClassModel Messages
	Interaction Modeling	Interaction diagramming activities	InteractionModel Messages
	State Modeling	State diagramming activities	StateModel Messages

4. Example

In this section, we present detailed descriptions on OMOS with an example of UseCaseModel. As we discussed in Section 3.1, UseCaseModel is a package that contains use-cases and actors for modeling object-oriented software development. Four classes are identified in UseCaseModel and correspond to the Requirement phase of RUP : Actor, Use-Case, Association, and Generalization. <Table 5> shows classes and their attributes that characterize UseCaseModel.

〈Table 5〉 UseCaseModel : Classes and Attributes (selection)

Class	Description	Attributes
Actor	Idealize users responsible to interact with systems and external users.	ActorName
		ActorDescription
UseCase	Describe how actors use use-cases	UseCaseName
		UseCaseDescription
		FlowOfEvent
		SpecialRequirement
		UseCaseType
Associa-tion	Describe how actors are associated with use-cases	AssociationName
Generali-zation	Describe generalization relationships between actors and use-cases	RelationType

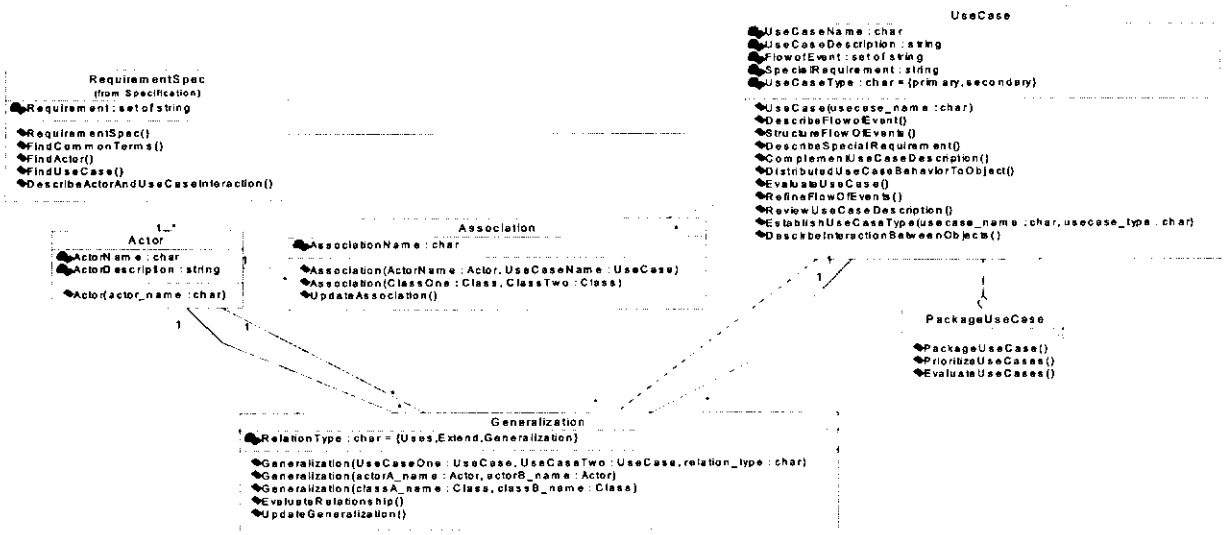
Class operations are defined based upon the RUP tasks and activities related to use-cases and actors. <Table 6> summarizes RUP activities and tasks that are encapsulated

〈Table 6〉 RUP activities and tasks for OMOS UseCaseModel package

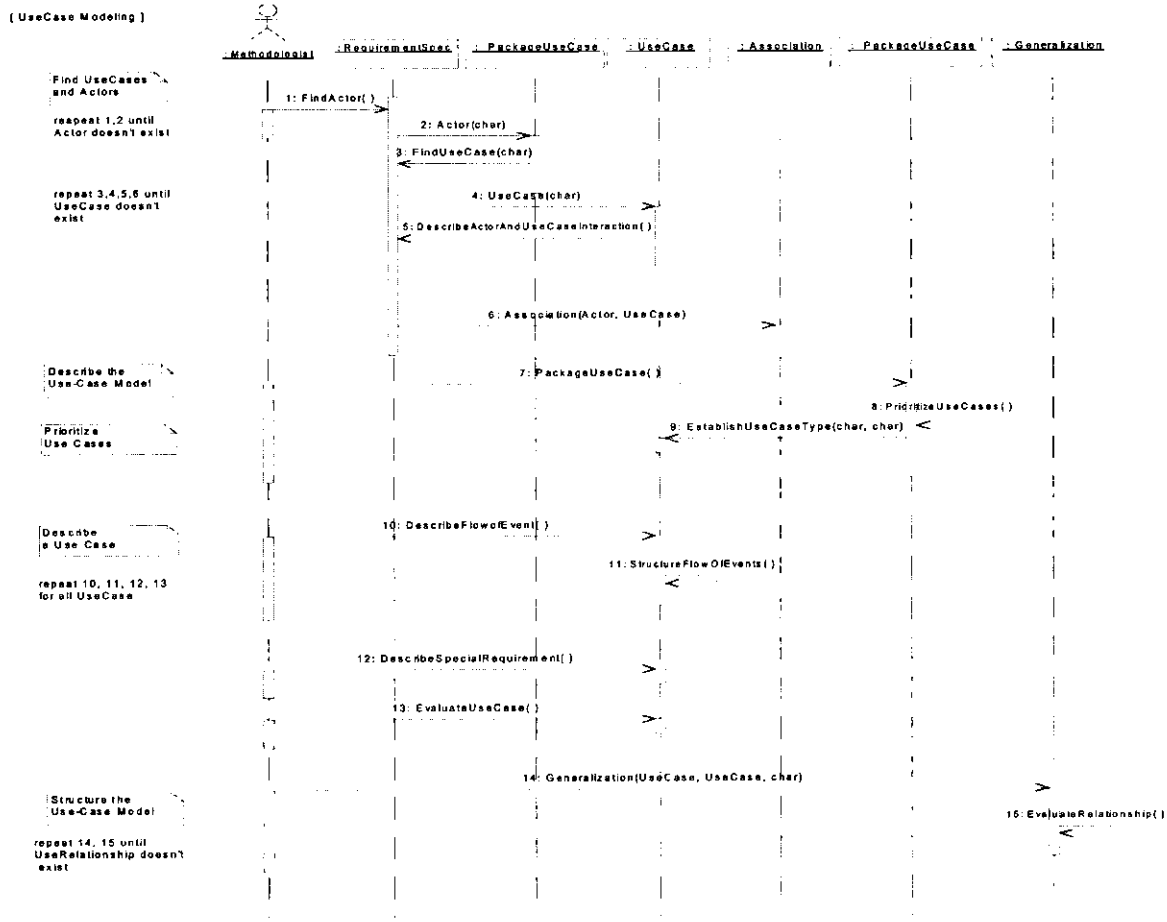
RUP Activity	RUP Task	OMOS Classes	OMOS Messages
Find actors and use cases	Find actors	RequirementSpec	FindActor()
		Actor	Actor()
	Find use cases	RequirementSpec	FindUseCase()
		UseCase	UseCase()
	Describe how actors and use cases interact	RequirementSpec	DescribeActorAndUseCaseInteraction()
	Package use cases and actors	Association	Association()
Prioritize Use Cases	Prioritize use cases and scenarios	UseCase	PrioritizeUseCases()
	Evaluate your results	UseCase	EvaluateUseCase()
Detail a Use Case	Detail the flow of events of the use case	UseCase	StructureFlowOfEvent()
	Structure the flow of events of the use case	UseCase	StructureFlowOfEvent()
	Illustrate relationships with actors and other use cases	Association	Association()
	Describe the special requirements of the use case	UseCase	DescribeSpecialRequirement()
	Describe communication protocols	UseCase	ComplementUseCaseDescription(), DistributeUseCaseBehaviorToObject()
	Evaluate your results	UseCase	EvaluateUseCase()
Structure the Use-Case Model	Establish include-relationships between use cases	Generalization, UseCase	Generalization()
	Establish extends-relationships between use cases	Generalization, UseCase	Generalization()
	Establish generalizations between use cases	Generalization, UseCase	Generalization()
	Establish generalizations between Actors	Generalization, Actor	Generalization()
	Evaluate your results	Generalization	EvaluateRelationship()

into UseCaseModel operations. For example, a task of finding actors is encapsulated in RequirementSpec as a class operation of FindActor().

Static structures of the classes are modeled as relationships among classes as in a class diagram of (Figure 2). (Figure 2) shows static structure of object-oriented software development methods related to use case modeling. The main constituents are classes and their relationships.



(Figure 2) Use Case Model : Class Diagram



(Figure 3) UseCaseModel : Sequence Diagram

List of attributes and operations are shown in the separate compartments : deliverables and other artifacts that characterize each class are encapsulated as classes and their attributes, while tasks and activities of classes are encapsulated as class operations. (Figure 2) is static because it does not describe the time dependent behavior of the system, which is described in (Figure 3).

(Figure 3) shows the flows of messages across many objects of UseCaseModel, therefore provides dynamic view of object-oriented software development methods related to use case modeling. Sequences of message exchanges among UseCaseModel classes are described in the form of sequence diagram.

5. Conclusions

In this paper, we have presented OMOS, an object-oriented method for object-oriented software system. OMOS is defined in terms of *model-objects* and *life-cycle events*. *Model-objects* encapsulate artifacts and their related activities. Life cycle processes can be modeled as *life-cycle events* that represent interactions among *model-objects*. Advantages of OMOS are 1) reuse of both objects and processes for software development, 2) flexible application of development processes, and 3) documentation of development experiences and easy maintenance.

Users can design their own software development methods based on OMOS elements. For example, existing class attributes and operations of OMOS can be modified and deleted to define user's own development method; new attributes and operations can be added for new activities, and artifacts. Message sequences among objects can be modified according to new transformations that users use to develop and maintain the software and the associated products. We have applied OMOS to two Korean software companies. The followings were considered in order to reflect the different characteristics that their projects showed : short-term vs. medium-term, task-oriented vs. deliverable-oriented, limited solution (single function) vs. integrated and customized solution, and single location vs. multi-location. We started from the abstract classes of OMOS, and then complete the details of the classes by properly setting objects attributes and overriding objects operations until the companies satisfied the resulting development method. We found that the most developers could understand the development process easily, since the tailored software development methods were visible and self-explanatory through UML diagrams.

In this paper, we showed object-oriented modeling of the development process in OMOS. We would like to continue our efforts to other process models, such as, project management process, and quality management process, and configuration management process. Also, we plan to apply OMOS to various application domains, from small projects to very large, complex, multi-team projects delivering millions of lines of code. We believe practices from various applications would provide valuable feedback to improve OMOS.

References

[1] Watts S. Humphrey, *Introduction to the Personal Sogwrae Process*, SEI series in Software Engineering, Addison Wesley, 1997.

[2] ISO, "ISO/IES 12207 International Standards Information Technology Software Life Cycle Processes," Aug., 1995.

[3] C. Mazza, J. Faircolough, et. al., *Software Engineering Standards*, Prentice Hall, 1994.

[4] Ross D. T. , *Applications and Extensions of SADT*, IEEE Computer, April, 1985.

[5] Yourdon E., *Modern Structured Analysis*, Yourdon Press/ Prentice Hall, 1989.

[6] Yourdon E. *Structured Walkthroughs*, 4th edition, Englewood Cliffs, Prentice-Hall./Yourdon Press, 1989.

[7] Timothy Budd, *An Introduction to Object-Oriented Programming*, Addison-Wesley, 1991.

[8] Brad J. Cox, *Object Oriented Programming : An Evolutionary Approach*, Addison-Weseley, Redaing, MA, 1986.

[9] Arthur Anderson & Co series, "METHOD/1 Overview School," 1988.

[10] Martin, J., *Information Engineering (3 Volumes)*, Prentice-Hall, 1989.

[11] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.

[12] Grady Booch, *Object-Oriented Design with Applications*, Redwood City : Benjamin/Cummings, 1991.

[13] Ivar Jacobson, *Object-Oriented Software Engineering*, Addison-Wesley, 1995.

[14] Ernst & Young Navigator Systems Series, "Accelerated System Development-Infrastructure Development," 1995.

[15] Ivar Jacobson, Grady Booch, and Jim Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[16] Rational Unified Process Process Manual 5.0, "Project Management," 1999.

[17] Grady Booch, Jim Rumbaugh, and Ivar Jacobson, *Unified Modeling Language Users Guide*, Addison-Wesley, 1997.

[18] Martin Fowler, *UML Distilled : Applying the Standard Object Modeling Language*, Addison-Wesley, 1997.

[19] Jim Rumbaugh, Ivar Jacobson, and Grady Booch, *Unified Modeling Language Manual*, Addison-Wesley, 1997.

[20] Mark P. Ginsberg, and Lauren H. Quinn, *Process Tailoring and The Software Capability Maturity Model*, CMU/SEI-94-TR-024, Technical Report, Carnegie Mellon University, 1995.

최 성 운

e-mail : choisw@mju.ac.kr

1985년 한국외국어대학교(상학학사)

1988년 미국 오레곤주립대학교(공학석사)

1992년 미국 오레곤주립대학교(공학박사)

1993~현재 명지대학교 컴퓨터공학부 부교수

OMG KSIG 회장

한국정보 컨설팅(KIC) 기술자문

관심분야 : 컴포넌트 프레임워크, 객체지향 소프트웨어공학