단계별 객체지향 분석기법
(SOOA : A Stepwise Object-Oriented Analysis Method)

윤 청[†] 인소란[††] 이권일[†††] 차승훈[††††] 배두환[†††††] 변보균[††††††] Mirza Misbah[†††††††]
(Cheong Youn) (So-Ran Inn) (Kwon-Il Lee) (Seung-Hoon Cha) (Doo-Hwan Bae) (Bo-Gyun Byoun)

요 약 현재 사용되는 객체지향 분석 방법들은 분석 기법을 적용하기 전에 시스템을 구성하는 객체들과 이들 간의 상호작용에 대한 사전 지식을 요구하며, 실제 복잡한 시스템을 모델링 하는데 한계점을 가지고 있다. 이러한 접근 방법들은 규모가 작거나 응용 분야에 대하여 잘 알려진 시스템에는 적용하기 쉬우나 실제로 규모가 큰 새로운 시스템에 적용하기에는 한계가 있다. 본 논문에서는 객체지향 분석에 적용할 수 있는 하향식 접근 방법에 기초한 단계적 모델링 방법을 제안한다. 이 접근 방법은 우리가 시스템에 대한 모든 정보를 미리 파악할 수 없는 새로운 분야를 분석하는데 더욱 유용하게 사용할 수 있다. 본 논문에서는 이 접근 방법을 미래의 통신 서비스인 B-ISDN 서비스 모델링에 적용했고 긍정적인 결과를 얻을 수 있음을 보였다.

Abstract Current approaches in Object-Oriented Analysis have limitations on modeling complex real systems because they require the prior knowledge about objects and their interactions before applying the analysis techniques. This is possible in small or well known systems but in large real systems this may not be feasible. We suggest a stepwise refinement based top-down engineering approach to Object-Oriented Analysis. This approach is especially good for new areas where we do not know all the information in advance. We applied this approach to the B-ISDN service modeling and distributed systems and found encouraging and promising results.

1. Introduction

The current approaches to Object-Oriented Analysis are severely restricted in their application to modeling complex real systems because they require the prior knowledge about objects and their interactions before applying the Object-Oriented techniques. These can be easily applied to small systems because the functionality of the system is mostly well understood already or easier to find out. But in complex systems such as in new areas of information and communication technology, information about objects and their interactions is difficult to have in advance. In this case a different Object-Oriented analysis approach needs to be used which can make the modeling of such systems easier for the analyst and developer.

Here we present an approach which takes the fact into account that modeling complex systems can not assume a prior knowledge of the functionalities and the internal detail of the systems. This approach combines two simple but very powerful ideas in to the modeling process of OOA. The first one is the use of *use cases* to the OOA. This is not a new idea and has been proposed and applied by Jacobson in OOSE [1]. In addition, it has been used in OMT [7] and also incorporated in the initial draft of the Unified Method [8]. However, our application of this idea

† 종신회원 : 충남대학교 컴퓨터과학과 교수
†† 종신회원 : 한국전자통신연구원 S/W 공학연구실장
††† 정 회 원 : 한국전자통신연구원 선임연구원
†††† 비 회 원 : 국방과학연구소 연구원
††††† 종신회원 : 한국과학기술원 정보 및 통신공학과 교수
†††††† 비 회 원 : 충남대학교 컴퓨터과학과
††††††† 비 회 원 : Professor in the faculty of Computer Science at the GIK Institute of Engineering Science and Technology, Pakistan
논문접수 : 1996년 10월 28일
심사완료 : 1997년 8월 27일

to the OOA is to determine the initial objects and their interactions with the users of the system. The *use cases* can be used to find events between objects and the users and their responses.

The second concept used is the application of a black-box and white-box concept to the OOA modeling. This concept has been used primarily software testing techniques, introduced by Myers [13]. The merits of this concept have been well publicized modularity and abstraction as tools to simplify the process of analysis and design. However it has not been applied as such in the Object-Oriented Analysis methodologies because it was considered close to the functional techniques such as Structured Analysis [2] rather than the Object-Oriented approaches.

In our approach, initially the whole system is considered as a black-box. So the analyst and developer only considers one object that is the system and its users. At this stage, the interactions between the system and users are identified. This can be done by looking at each *use case* and also from the problem domain such as a problem statement. The primary focus here is on capturing the user requirements. Obviously they can be more easily found out from the *use cases*. Since *use cases* only consider the interactions between the users and the system, the level of abstraction at this point in time is the highest possible in the analysis approach. This approach at each level hides the complexity of the system from the user.
At the next level, the system is considered as a white-box. Objects found from the *use cases* and the problem domain are studied. Their interactions and associations (between the objects within the system) are also found and modeled. At this level, information from the previous step are used. Objects and their interactions are found and their interactions are described.
If an object identified here is complex, we can apply the same concept to the object. This recursion continues to as many levels as are feasible for a particular system. Taking each object in turn, its complexity is judged and if possible it is decomposed into further (sub)objects and their interactions. This is the black-box and white-box concept in recursion. Obviously this process of decomposition of objects depends on the complexity of the system involved. So several levels may be needed for a very complex system whereas for small simple systems only one or two levels may be enough.

Application of *use cases*, in this way, to discover black-box objects and then the use of white-box technique has as such not been the focus of attention of other Object-Oriented development methods (for example, the OMT, OOADA, OOSE, and the Unified Method). In a limited sense, the concept of a composite object (an object composed of other objects) is close to a black-box. However, this concept is considered less important and rarely used.

2. Background

The Object-Oriented techniques for the Analysis and Design processes have been proposed and applied in academic as well as industrial environments over past many years with varying degrees of success. Some of the more popular approaches are Rumbaugh's OMT [3], Booch's OOADA [4], Jacobson's OOSE [1], and Shlaer-Mellor Method [5] [6].

The concept of *use case* was proposed by Jacobson in his OOSE [1]. Since *use cases* are very helpful in determining user requirements, the concept has gained widespread acceptance in the Object-Oriented community. It has found usage in many Object-Oriented Analysis and Design methods like OMT in its second version [7] and OOADA [4]. It has even been incorporated in the Unified Method (combined method formed from OMT and OOADA methods) [8].

However, Jacobson applies the *use case* concept throughout the Object-Oriented life cycle. The Unified Method [8] also incorporates similar concept to the determination of objects and their associations as well as interface properties of the system. We only initially consider *use cases* to find objects and their interactions but do not extend this idea to other levels of the Object-Oriented Analysis.

The *use cases* have also been in some other methods. For example, the MOSES method by Henderson-Sellers [12] provides activities, guidelines, and deliverables for all aspects of an OO project. It has different graphics and text models for different phases of the development process. One of those activities is the Scenario Development (essentially the *use case* concept). The purpose is to describe an interaction with a system from which O/Cs (MOSES term for Class & Objects; A class with object instances), events and interactions can be found. MOSES uses scenarios for identifying operations and develops them for different subsystem responsibilities.

The concepts of black-box and white-box have been used in software testing. It was originally introduced by Myers in [13]. The black-box is treated as one single entity and only its externally visible functionality along with its interactions with other objects in the environment are considered. Nothing inside the black-box is visible. This is an abstraction concept and reduces the complexity of a system which is being considered as the black-box. This concept has, therefore, been the focus of attention in many testing techniques. These are generally called black-box testing techniques as they ignore the inner details (such as the code and its control flow etc.) and focus only on the externally visible functionality of the system.

In many respects, an object represents a black-box. Its inner details like the process logic and data is hidden from other objects. However, it is not partitioned into sub-objects. Instead, such a relationship between a complex object and its sub-objects is characterized by a composition relationships. Hence generally Object-Oriented Analysis and Design methods are bottom-up methods in their application.

We show this relationship explicitly by using decomposition of large objects, treated initially as black-box, into sub-objects. Somewhat similar idea has been used in the approach by Martin and Bell in Object-Oriented Analysis and Design (OOAD) [9]. As stated earlier, this is somewhat related to the composition concept also found in other Object-Oriented techniques discussed above (where it is one of several types of relationships). However the OOAD focuses on the events and objects of the system. The approach can be either applied in a top-down or a bottom-up manner. The authors recommend the use of Object Flow diagrams for this purpose.

Our approach differs from the above methods in several ways. First, and foremost it explicitly uses the concept of black-box and white-box. Secondly, we don t only focus on objects and events, we use the interactions as well as associations between the objects. Thirdly, we use *use cases* with the black-box concept to find the above things (objects, interactions and associations). Finally, ours is a recursive approach.

3. The Stepwise OOA (SOOA) Approach

In summary the SOOA approach can be described as follows:

Initially the system is treated as a black-box. Its inner details are not considered at that level. Only its interactions and associations with the users are considered. After that is done, the system is treated as a white-box and will be decomposed into more objects and their related interactions and associations in the next level. Hence at each level a black-box objects are considered and then at the successive levels they are considered as white-boxes and decomposed into a more black-boxes. Thus, a complex system may have several levels of decomposition whereas a simple one will have only a few (possible one) level of decomposition.

In this section, we describe the SOOA approach in detail. The steps serve as guidelines to the approach. In the next section we demonstrate the application of the SOOA approach by taking an example system.

The SOOA approach contains two main stages:

first is the black-box stage and the second is the white-box stage. The whole process consists of applying these two stages in recursion to the required level of simplicity.

3.1 Consider the system as a black box

In this stage the whole system is considered as a black box. There is only one object and that is the system. We ignore the inner details and the working of the system. For example in figure 1, the system is seen as just one black-box by a user who has no idea of what is inside the box. But, he/she can use the system using the two levers on top to get some predetermined functionality. However, unless the user knows what functionality can be provided by the system and how to operates(the sequence of uses of the levers), the system cannot be used with predictable results.

In this step, first of all we identify the actors that use the system. This includes the users in their different roles as they use the system for different purposes and goals. A user may play different roles with the system for doing different transactions with the system. Each role is identified. Then we identify the *use cases* for each role. We describe the interactions that take place between the system and the actor of the *use case*. Considering figure 2, we have a system with predefined functionality, known to the user in terms of the *use cases*. Hence the user can follow different *use cases* with the system. Each *use case* is a sequence of interactions between the user and the system (achieved by pressing the levers up or down in certain sequence). In that sense, figure 2 shows a system as a 'usable black-box'

Once the description of each *use case* has been developed, we identify objects from each *use case*. *Use cases* contain nouns many of which qualify as objects within the system. They are considered as potential objects. Then each potential object is scrutinized to determine if it is indeed an object. Here we differ from OMT [3] and OOADA [4] in that the complex objects are rejected in these approaches while we consider them as valid objects.

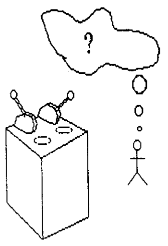


Fig. 1 A black box representing the sytem

Next, we identify events from each *use case*. Events are symbolized by interactions and their responses between users and the system. Each interaction between these is taken as an event and its appropriate response is generated by the system. So we list these events.

3.2 Consider the system as a white box

In this stage the system is considered as a set of objects. We examine the details of the system by looking at the component objects (or sub-objects) and their interactions among themselves. There is only indirect consideration for the user at this stage because we are looking at the inner functionality and structure to fulfill the user requirements (which are really to support the *use cases*).

Using information of the black-box stage (*use cases*, events, objects, and interactions) as well as the knowledge from the problem domain (including a problem statement if it exists), we identify objects. Although many of the objects can be identified from the *use cases*, there are some objects which cannot be identified from the *use cases*. This is due to the fact that *use cases* directly represent only the dynamic aspects (interactions) and data of a system seen by the user. So only the objects participating in the interactions which are part of the *use case* can be identified. However, a system may well have objects which do not participate in any *use case* and are part of the structure of the system (hence they display static properties of the system). They can be identified by looking at the static as well as dynamic aspects of the system.

Using the above information domain, we now identify interactions between the objects of the system and the actors and also the interactions between the objects themselves including any associations. While the former are easier to identify because they are part of the *use cases*, the latter type of interactions and associations have to be found from the problem statement or the problem domain knowledge.

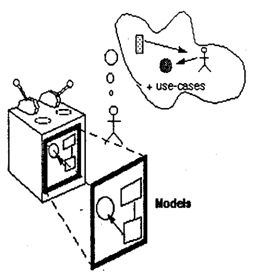


Fig. 2 A blacek-box with use cases

3.3 Apply The Above Stages Recursively

The above two stages are applied recursively. In this first instance, the system is treated as a black-box and the above mentioned steps applied. Then the same system is treated as a white-box and decomposed into several objects and their interactions and associations at the next lower level.

Then the same stages are repeated at the next level. That is by treating each object first as a black-box and identifying its interactions and associations with other objects and then treating the same object as a white-box to decompose into further sub-objects and their related interactions and associations to the next level of the hierarchy. This process is recursively repeated until sufficiently simple objects are found. This makes the SOOA approach recursive.

The determining factor for the recursion is the complexity of the objects involved. The complexity of an object can be determined by looking at the interactions between the particular object and other objects, and the data that is passed between them during those interactions.

These three steps are applied in sequence. However within steps described in sections 3.1 and 3.2, the sub steps can be applied in parallel. For example the identification of objects, events, and *use cases* can be done in parallel. Some objects may be identified with their *use cases* and events and then more of these can be added later.

At this stage, One of the practical issues to be the possibility of the occurrence of redundant objects in different subcomponents. Such overlapped objects can be filtered by comparing their behaviors. Since objects having different names may indicate the same object in the real world, the object's behaviors must be compared carefully. In order to facilitate this filtering process, we may use objects/classes dictionary in which objects/classes name and level number of refinement are recorded.

4. An Application of the Approach

In this section, we demonstrate the effectiveness of the SOOA approach. The example system is from the telecommunication field. It was chosen because it represents a broad range of complex systems for which no prior knowledge can be assumed while applying the Object-Oriented Analysis and Design methodologies.

The system we have chosen is one of a broad range of services provided by a B-ISDN network [14]. It is called *Video conference service* [15]. A standard *Video conference* is a specific multi-media, multiparty service that provides two or more geographically separate users with the capacity for exchanging different types of information such as audio, moving images, data etc. Here we will present a brief statement of the problem (considering a much simplified video conference) and then discuss the application of the SOOA approach.

A Video conference provides the necessary arrangements for real-time conferencing in which both voice and moving picture video information can be exchanged together with optional non-moving visual information, signaling information such as identification of speaker, etc. among single individuals or groups of individuals at two or more locations.

This Video conference service will provide conference management functions such as conference setup, identify participants on the video conference, connect participants to a conference. disconnect conference participants, terminate the video conference, floor grant, identification of speaker, and control of speaker s microphone. These functions will only be available to the conference chairman.

In addition, participants are provided terminal handling functions of audio and video and functions of signals such as floor request. fax transfer, still picture transfer, text transfer, etc.

When the chairman will request video conference to be opened, at that time the terminal will validate the qualification of the chairman. If he/she is qualified, then initial screen will be displayed. Other participants should be ready to receive the invitation message.

If the chairman selects the menu called *open video conference*, a dialog box will be displayed. At this time, the chairman can enter the conference name and select the participants from the list or enter new participants.

After the chairman selects the conference and the participants, he/she presses the necessary button to confirm the action. The system will then send invitation request message to the selected participants and wait for their reply.

When the system sends invitation request message to the expected participant, his/her terminal will display conference name and the name of the chairman. The participant can choose one of three options:

1) join the conference: In this case the participant selects the acceptance button, the system will send acceptance message to the chairman and display wait a minute message to the participant.
2) not join the meeting: In this case the participant selects the refuse button, the system will send the refusal message to the chairman and go to the previous state.
3) make no reply: This is judged by the system by using a timer. If the participant does not reply for a given period of time, the system will send the refusal message to the chairman and go to the previous state.

As replies arrive, the dialog box will show the response of the participants. Then the chairman has the right to open cancel the conference. If the conference is:

1) opened by the chairman, the system will display opening message to the chairman and the participants and open the conference.
2) canceled, then the system will display the conference got canceled to the chairman and the participants and the conference will terminate.

When opening message comes from chairman the system will be in opening state . This is done by displaying a dialog box showing the information of the Video conference. If a participant presses the confirm button, the dialog box will be displayed with the above conference information. If the cancellation message came from the chairman then it will be notified to all participants and the system will close.

4.1 Applying the Black-Box Approach

4.1.1 Identifying Actors

From the above problem statement we find two actors:

chairman
participant

4.1.2 Identifying the use cases

Now for each actor, we find the relevant *use cases*. This is done by looking at the problem statement. So the following *use cases* are found for the *Video conference*.

For the **chairman** the *use cases* are:

conference setup
identify participants
connect participants to the conference
disconnect participants
terminate the video conference
floor grant

control speaker s microphone

For the **participant** the *use cases* are:

floor request
transfer fax
transfer still picture
transfer text

4.1.3 Developing the use cases

Now we develop *use case* description for each of the *use case* identified above. Due to lack of space here, we will only show description for a couple of *use cases*.

The description for each *use case* can be developed by following the interactions and their responses between the system and the actor during that *use case*. For example, for the *use case* **conference set up** we get the following description:

1. The conference chairman selects *start conference* from the menu.
2. The system displays a dialog box.
3. The conference chairman enters name of the conference and the list of the participants.
4. The conference chairman presses the *confirm* button to send the data.
5. The system displays invitation message to the participant.
6. The participant chooses acceptance.
7. The system returns the response of the participant.
8. The conference chairman presses the *enter* button to start the conference.

Similarly the *use case* **conference termination** would be described as:

1. The conference chairman presses the *terminate conference* button.
2. The system displays a dialog box.
3. The conference chairman presses the *confirm* button to send the data.
4. The system proceeds to terminate the conference.

Also, we will describe two more *use cases*. First *use case* is **request floor** for the conference participant. Its description is as follows:

1. The participant selects *request floor*.
2. The system sends request to the chairman along with participant s information.
3. The system displays message request being processed to participant.

The second *use case* is **grant floor**. This *use case* is used to choose the next speaker of the conference from among those participants who requested to speak.

1. The system displays the list of requests from participants.
2. The chairman selects next speaker.
3. The system switches on the speaker s microphone and displays the corresponding message on the participant s and chairman s terminals.

4.1.4 Identifying Object Classes from the use cases

Now we look at the *use cases* and identify objects and classes. For this we look at nouns in the *use cases* and remove unlikely classes and objects (like action nouns, attribute nouns, vague nouns, etc.). Hence we get the *terminal* as the only good class. The class "terminal" is recorded in object dictionary.

4.1.5 Identifying events from the use cases

Events are stimuli to which a system or user must respond. Hence we now identify events from the *use cases*. The events identified are shown in figure 3.

4.2 Applying the White-Box Approach

In this step we find objects from problem space. Then we find interactions between the actors and the objects. Finally, we find interactions amongst the objects themselves.

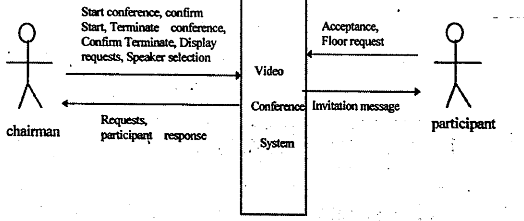

Start conference, confirm Start, Terminate conference, Confirm Terminate, Display requests, Speaker selection
Requests, participant response
Video Conference System
Acceptance, Floor request
Invitation message
chairman
participant

Fig. 3 System with events (black-Box approach)

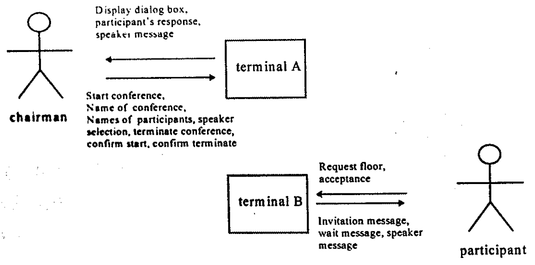
4.2.1 Find Objects from the problem/solution space

From the problem and solution space, we find the following additional classes of objects, and added in objects/classes dictionary:

call control, multipoint control

4.2.1 Find Interactions between Objects and Actors

Considering the system as a set of objects, we find interactions between system objects and the external elements (actors). Figure 4 shows the interactions between the system objects and the actors. Note that each actor interacts with his/her own terminal (so we distinguish between them as *terminal A* and *terminal B*).



Display dialog box, participant's response, speaker message
terminal A
chairman
Start conference, Name of conference, Names of participants, speaker selection, terminate conference, confirm start, confirm terminate
Request floor, acceptance
terminal B
Invitation message, wait message, speaker message
participant

Fig. 4 Interactions between actors and system objects.

4.2.3 Find Interactions among Objects

Now we find interactions among objects within the system. Again for this purpose, we need to consult the problem and the solution space. Here for example, we can consider the ITU-T structure for the description of services [16]. Hence, we can represent the interactions among system objects (found from the problem and the solution space) as shown in figure 5. Please note that due to lack of space here, we are showing only a subset of such functionality (the conference set up). Other functionality can be shown similarly. Also note that similar to *terminal* A and B, there are two *call control* objects A and B.

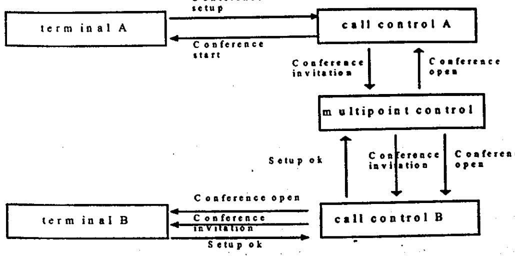


Fig. 5 Interactions among system objects

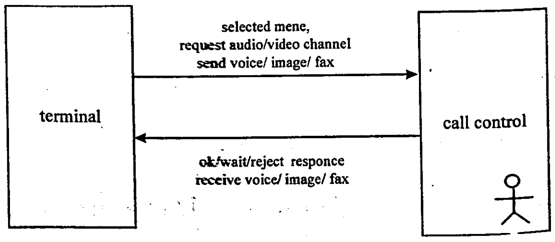
4.3 Apply the Above Steps Recursively

Here we repeat the above steps in sequence. First, we consider each object as a black-box and look at its interactions with other objects. Then, we decompose that object into further sub-objects. The guide to decomposition is the complexity of the interactions to the object. For example, in our case the *terminal* and *multipoint control* objects can be decomposed further using SOOA.

Let's consider the terminal. Actor of the terminal is "call control." Use cases of this actor are relatively simple compared with the use cases of the higher level, and can be found easily from the higher-level interaction diagrams as follows :

Conference setup
Conference start
Conference open
Conference Invitation
Setup OK

From the above use cases, identified objects are menu, buttons, audio, video, recorder and fax. Muen is an object to start conference, to accept participation, and to record the contents of conference, The "audio" and "video" objects are to speak/listen the discussion issues and to see the participants or related materials. The objects, "recorder" and "fax" are straightforward. These objects are added in objects/class dictionary. Identified events are repre-



terminal
selected mene, request audio/video channel send voice/ image/ fax
ok/wait/reject responce receive voice/ image/ fax
call control

Fig. 6 Identified events from "terminal" use cases

sented in figure 6, and interactions among objects are in figure 7.

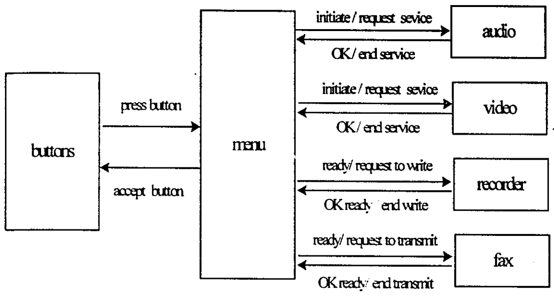


Fig. 7 Interaction among subsystem objects

Through these recursive refinement, we analysis and design the entire system. Additionally, during these steps, it is possible to check the completeness and consistency of our diagrams using objects/ classes dictionary, cross reference inspection between diagrams of each level.

We explain the application of our SOOA approach to the system of a general video conference. This example is adequate to explain our approach. In case of a more complex system, the handling of complexity is achieved by stepwise decomposing of the system to simple subsystems.

Thus we have demonstrated the ease of use of our approach with an application. This is because our approach does not assume prior knowledge about the application domain. On the contrary current popular Object-Oriented approaches have limitations in their application to complex systems because they assume such prior knowledge.

For example, the OOADA [4] technique, by Booch, has trouble dealing with the development of larger projects. Although it supports two constructs for this purpose: namely, the class category and the subsystem. The class category construct is for the partitioning of the logical model. It only represents a name space and does not support the representation of normal associations between different classes, among different categories. The subsystem construct is for the partitioning of the physical model rather than the logical one.

Similarly OMT [3,7], by Rumbaugh, supports complex objects called composites. It is an extended form of aggregation and is viewed at a higher level than its parts. Composites may have associations between themselves representing associations between classes belonging to these composites. Such composites don't have any semantics but are used to organize the understanding of the model. This concept is different from ours in that we use complex objects as black-box first and then as white-box to decompose the system objects. In addition we also include the interactions between the objects in our model, which increases the ease of understanding of the system.

5. A Comparison With Common OO Approaches

In this section we present a comparison of our approach with some of the commonly used and studied Object-Oriented development approaches. We take the examples of OMT by Rumbaugh, and OOSE by Ivar Jacobson.

OOSE

The concept of *use case* was proposed by Jacobson in his OOSE [1]. Since *use cases* are very helpful in determining user requirements, the concept has gained widespread acceptance in the Object-Oriented community. However, Jacobson applies the *use case* concept throughout the Object-Oriented life cycle, for example to develop the analysis model from the requirements model and then the design model from the analysis model and so on. In the testing stage also use cases are used to test the system thus produced.

To handle large projects, OOSE recommends the use of 'subsystem' concept. This concept is used for grouping objects within the system. However, the object types are different in OOSE from the object types in our approach. Among the many criteria for placing objects in subsystems, OOSE recommends using functional coupling. Objects within a subsystem should have strong functional coupling among them than other objects.

OOSE recommends looking at an object's

environment to find out if it is strongly functionally related to another object. This includes looking at it's communications with actors, the effect of change in an object on the other objects, and the operations they perform on one another.

Our approach is different from the OOSE approach in many respects. First, we have a black box and white concept. Secondly, we dont have the same types of objects as used by OOSE. As a result the complex objects in our approach have different interactions among themselves as well as internally to the ones found in an OOSE application.

OMT

Rumbaugh developed the OMT [2] method as a way to organize software as a collection of discrete objects which also incorporates both data structure and behavior. The essence in OMT is the identification and organization of application domain objects.

The OMT supports an iterative process of development. Objects, and associations are added and clarified in iterations. The process consists of three stages: the development of :

- ▶ an object model
- ▶ a dynamic model
- ▶ a functional model

These three models represent the three views of a system: information, behavior, and function. These are developed and refined in the analysis, design, and the implementation stages.

Rumbaugh later evolved the OMT to add more features and representations. The more important extensions included the use case concept and their relation to scenario development in the dynamic model. The iterative process of OMT was also modified and Rumbaugh suggested using a use case driven iterative approach in the development of the three models.

The OMT in its latest version includes representation for complex objects, known as 'composite objects'. The composite object is an extended form of aggregation. The composite is viewed at a higher level of abstraction than its parts. A composite may contain classes and associations. Composites may have associations at the composite level to represent associations between classes from different composites.

Composites, however, do not have any semantics involve but serve to organize the understanding of the model. This concept is similar to our approach. Our complex objects have somewhat similar concept. However, we use complex objects as black-box first and then as white-box to decompose the system objects. Our approach also includes the interactions between objects. We also provide guidelines for decomposing complex objects.

6. Conclusion

Most of the currently available Object-Oriented Analysis and Design methods assume detailed prior knowledge about system before starting the analysis phase. This knowledge is then used to find objects and their interactions and associations in the system. While this approach may work in simple systems, it is very difficult to apply in the analysis and design of large systems. This is also the case in systems being developed for the newly emerging technologies like information and communication.

This paper presented an approach that can be successfully applied in such systems. It combines two powerful concepts of *use case* and black-box. The *use case* concepts helps in identifying objects and their interactions as well as associations. This has been used in several other methods also discussed earlier. However the treatment of *use case* concept here is different from its main proposer, Ivar Jacobson in OOSE [1].We did not classify objects as done by Jacobson and all objects are equally treated.

The other important concept we apply in SOOA approach is the black-box concept. The application of this concept simplifies the complexity of the system and provides a very powerful structuring mechanism for the analysis and design of complex systems. This is somewhat similar to the concept of ensembles in the Fusion method [10] and functional decomposition as used in the Object-Oriented Analysis and Design method by Martin and Odell [9]. However, we have focused on the recursive application of black-box and white-box concepts at different layers of the system resulting in a stepwise analysis of the system under consi-deration.

The recursion is applied at each level. At the top most level, the system is considered as a black-box and its interactions are identified with the users of the system. Then the system is considered as a white-box and it is decomposed into a set of objects into the next lower level. Then each of those objects is considered as a black-box and the interactions and associations between those objects are identified. For this the domain knowledge can be used in addition to the *use cases* already developed in the first step. This recursion (or stepwise refinement) continues until the desired level of simplicity in objects is reached. The SOOA approach was applied to an example system (a video conference). The application of the SOOA approach demonstrated the practicality and usefulness of the approach for complex systems where no prior knowledge about objects exists. Although, the SOOA is not a complete methodology -- it covers the analysis part of the Object-Oriented development process it provides a useful approach towards solving a major problem in the application of Object-Oriented development metho-dologies to complex systems such as distributed systems and telecommunications. Without a good analysis stage, the design and its ensuing imple-mentation stages cannot guarantee success of a system development effort.

References

[1] Jacobson, I., Christerson, M., Jonsson, P., Over-gaard, G., *Object-Oriented Software Engineering*, Addison Wesley, 1992.

[2] Tom Demarco, *Structured Analysis and System Specification*, Yourdon Press, 1979.

[3] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenson, W., *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[4] Booch, G., *Object-Oriented Analysis and Design with Applications*, The Benjamin/Cummings Pub-lishing Company Inc., Redwood City, California, 1994.

[5] Shlaer, S., Mellor, S.J., *Object-Oriented Systems Analysis: Modeling the World in Data*, Yourdon Press, Englewood Cliffs, New Jersey, 1988.

[6] Shlaer, S., Mellor, S.J., *Object Lifecycles: Modeling the World in States,* Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[7] Rumbaugh, J., OMT: The development process, In Journal of Object-Oriented Programming, Vol. 8, No. 2, May 1995, pp. 8-16.

[8] Booch, G., and Rumbaugh, J., *Unified Method for Object-Oriented Development*, Documentation Set, Version 0.8, Rational Software Corporation, 1995.

[9] Martin, J., *Principles of Object-Oriented Analysis and Design*, Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[10] Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., Jeremaes, P., *Object-Oriented Development: The Fusion Method*, Prentice-Hall, Englewood Cliffs, New Jersey, 1994.

[11] ITU-T, "Q.71: ISDN 64 kbit/s Circuit Switched Bearer Services", 1988.

[12] Henderson-Sellers, B., Edwards, J.M., *BOOKTWO of Object-Oriented Knowledge: The Working Object*, Prentice-Hall, Sydney, 1994.

[13] Myers, G., *The Art of Software Testing*, Wiley, 1979.

[14] ITU-T, "I.150: B-ISDN functional features", Geneva, 1990.

[15] ITU-T, "F.732: Broadband Videoconference Services", Geneva, 1989.

[16] James, C. and Armstrong, Jr., "six GUI builders face off", *SunWorld*, December1992.

[17] Lee, J.S, Lee, S.,B, and Chi, D.H., "A Modeling Tool for X-Window Application Software Development", ETRI Journal, Vol 15, No. 2, October 1993, pp. 75-84.

윤 청

1979 서울대학교 물리학과 졸업(학사). 1983 Sagamon State University Computer Science 졸업(석사). 1988 Northwestern University Computer Science 졸업(박사). 1983~1985 Wayne State College 전임강사. 1985~1987 Nothwestern University 전임강사. 1988~1993 Bell Communication Research 선임연구원(MTS). 1993~현재 충남대학교 컴퓨터과학과 부교수. 관심분야는 소프트웨어 공학, CALS, 객체지향 모델링 및 설계

인 소 란

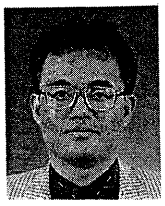
1978년 홍익대학교 전자계산학과 졸업. 1982년 홍익대학교 이공대학원 전자계산학과(석사) 데이타베이스 전공. 1987년 정보초리 기술사(전자계산기 조직 응용분야) 취득. 1991년 홍익대학교 이공대학원 전자계산학과(박사). 소프트웨어공학 전공. 1978년 ~ 현재 한국 전자통신 연구소 근무중 컴퓨터 연구단 S/W 공학연구실장. 관심분야는 프로토콜 공학, 컴퓨터 통신, 분산시스템, 소프트웨어, 클라이언트-서버기술, 분산 객체 기술.

이 권 일

1988 충남대학교 계산통계학과 졸업(이학사). 1996 전자계산기 조직응용 기술사 1988~현재 한국전자통신연구원 컴퓨터 연구단 소프트웨어 공학 연구실 선임연구원. 관심분야는 분산시스템, 소프트웨어 공학, 컴퓨터 보안 등.

차 승 훈

1995 충남대학교 전산학과 졸업(이학사) 1997 충남대학교 대학원 전산학과 졸업(이학석사). 1997~현재 국방과학연구소 연구원. 관심분야는 객체지향 모델링, 분산 컴퓨팅, 멀티미디어 시스템



배 두 환

1980년 서울대학교 조선공학과 학사. 1987년 위스콘신-밀워키대학 전산학 석사. 1992년 플로리다 대학 전산학 박사. 1992년 ~ 1994년 플로리다 대학 전산학과 조교수. 1995년 ~ 1996년 한국과학기술원 정보 및 통신공학과 조교수. 1996년 ~ 현재 한국과학기술원 전산학과 조교수.

변 보 균

1996 충남대학교 컴퓨터과학과 졸업(이학사). 1996~현재 충남대학교 컴퓨터과학과 석사과정 재학중. 관심분야는 소프트웨어 공학, 분산객체기술

Mirza Misbah

1997 UMIST, UK in Computer Science(학사). 1990 UMIST, UK in Computer Science(박사). 1995~1996 KOSEF post doctoral research fellowship at Chungnam National University. 1996~현재 조교수 in the faculty of Computer Science at the GIK Institute of Engineering Science and Technology, Topi, Pakistan