# Managing Complexity in Object-Oriented Analysis

So-Ran Ine, Cheong Youn, Uddin Mirza Misbah, Kwon-Il Lee,
Seung-Hoon Cha, Bo-Gyun Byoun, and Doo-Hwan Bae

## CONTENTS

## ABSTRACT

The current approaches in Object-Oriented Analysis have limitations on modeling complex real world systems because they require priori knowledge about objects and their interactions before applying them. This may be practical in small systems and systems with clear domain knowledge, but not in large real world systems with unclear domain knowledge. Our approach uses a stepwise refinement technique in a top-down manner to the Object-Oriented Analysis stage with the application of use cases. This approach is especially good for new areas where we do not know all the information in advance. We present the approach with an example of its application to the B-ISDN service modeling and distributed systems.

## I. INTRODUCTION

The current approaches to Object-Oriented Analysis (OOA) are severely restricted in their application to modeling complex real world 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 or easy 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 obtain in advance.

Here we present a solution to this problem which takes the fact into account that modeling complex systems can not assume a prior knowledge of the functionality and the internal detail of the systems. This approach combines two simple but very powerful ideas in the modeling process of OOA. The first one is the use of *use cases*. This was proposed and applied by Jacobson in OOSE [1]. In addition, it has been used in OMT [2] and also incorporated in the initial draft of the Unified Method [8]. However, our application of this idea 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 in software testing techniques, introduced by Myers [5]. The merits of this concept have been well publicized in terms of 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 [6] 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 consider one object (the system) and its users. Thus we give priority to the object interactions over its decomposition. In other methods, (for example, Structured Analysis) the functional decomposition of the object itself may have priority over the object interactions (the dataflows to/from other objects). Some of the dataflow, for example, may represent complex dataflows which can be decomposed into individual dataflows at the next (lower) levels after the function has been decomposed.

In our case, the interactions between the object (initially the system) and users are identified first and only then the object is decomposed. This can be done by looking at each use case. The primary focus here is on capturing the user requirements. Obviously they can be found out more easily from the use cases. Since use cases represent 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. Therefore, our approach hides the complexity of the system from the developer at each level.

At the next level, the system is considered as a white-box. As a first step, objects are found from the use cases. Then internal objects (those objects which do not appear in any use case) are found from the problem domain. Once objects have been found, we find their interactions. There are two types of interactions in the system: the interactions between objects and system users, and interactions among the objects themselves.

If an object identified here is complex, we can apply the same concept of "black-box and then white-box" to it. This is done by treating the complex object as the system and other objects which interact with it as its users (actors). Then the interactions become use cases for those actors. So we use those use cases to find "sub-objects" and then find interactions among those sub-objects. Hence the idea of "black-box and white-box" is applied recursively at each level.

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. Obviously this process of decomposition of objects depends on the complexity of the system involved. For complex systems several levels may be needed, whereas for small simple systems only one or two levels may be enough.

Applying of *use cases*, in this way, to discover black-box objects and then using of white-box technique has not been the focus of attention of other Object-Oriented development methods (such as, the OMT, OOADA, OOSE, and the Unified Method). In a limited sense, the concept of a composite object (i.e., an object composed of other objects) is close to a black-box. However, this concept is considered less important. As a result, few guidelines (or process) are provided in the above Object-Oriented development methods.

In this paper, we provide a detailed process of applying the black-box and white-box concept along with their deliverables. We also provide guidelines and heuristics for each step of the process. In addition, we also demonstrate the application of this process by modeling a video conferencing system from the B-ISDN service modeling domain.

## II. BACKGROUND

Different object-oriented analysis and design methods have been proposed during the past several years with varying degrees of success. Some of the more popular approaches are Rumbaugh's OMT [7], Booch's OOADA [8], Jacobson's OOSE [1], and Shlaer-Mellor Method [9] [10]. A new object-oriented approach called UML (Unified Modeling Language)[3][4] has recently been announced as the result of unifying Booch's OOADA, Rumbaugh's OMT, and Jacobson's Objectory methods.

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 [2] and OOADA [8]. It has even been incorporated into the Unified Method (combined method formed from OMT and OOADA methods) [3].

However, Jacobson applies the *use case* concept throughout the Object-Oriented life cycle. The Unified Method [3] also incorporates similar concept to the determination of objects and their associations as well as interface properties of the system. We initially consider *use cases* to find objects and their interactions and extend this idea to other levels of the Object-Oriented Analysis.

The *use cases* have also been used in several other methods. For example, the MOSES method by Henderson-Sellers [11] provides activities, guidelines, and deliverables for all aspects of an OO project. It has notations 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 applies them for different subsystem responsibilities.

A 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 are hidden from other objects. It is not yet 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 can be considered bottom-up methods in their application but our approach uses top-down decomposition of objects.

We choose to show this relationship explicitly by using decomposition of large objects, treated initially as a 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) [12]. 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 of OOAD recommend the use of Object Flow diagrams for this purpose.

Our approach differs from the above methods in three ways. First and foremost it explicitly uses the concept of black-box and white-box. It gives priority to objects over their decomposition. Decomposition is only carried out after the interactions of the object with other objects have been identified. Secondly, we not only focus on objects and events, we also use the interactions between the objects. Thirdly, we use *use cases* with the black-box concept to find the above objects and interactions. Finally, ours is a recursive approach.

In our approach, the system is initially treated as a black-box. Its inner details are not considered at that level. Only its interactions (*use cases*) with the users are considered (Fig. 1). After that is done, the system is treated as a white-box and will be decomposed into more objects and their related interactions at the next level. Hence at each level black-box objects are considered and then at the successive levels they are considered as white-boxes and decomposed into more black-boxes. Thus, a complex system may have several levels of decomposition whereas a simple one will have only a few (possible one) levels of decomposition.

## III. OUR APPROACH

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

Our approach consists of two main stages: a black-box stage and a white-box stage. The whole process consists of applying these two stages recursively to the required level of simplicity.

### 1. The Black Box Stage

In this stage the whole system is considered as a black box object. We ignore the inner details and the working of the system. The following steps are applied during this stage.

*First) Identify Actors*

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.

*Second) Identify the Use Cases*

Once actors have been identified, we can find the *use cases* for them. An actor may have more than one *use cases.* Each *use case* represents a complete sequence of transactions between the actor and the system. A *use case* may have more than one participating actors. However, there will be only one actor who is responsible for initiating the *use case.* For example, a customer of a bank may invoke a *use case* to request a loan from the bank. The bank's loan officer will also participate in this *use case.* Although he will not be the principle actor for this *use case.*

*Third) Identify interactions*

We describe the interactions that take place between the system and the actor(s) of each *use case* in an informal language. The interactions from the actors and their responses from the system are listed in time order. They are numbered to show their order of occurrence. Although an informal style could be used for the description of the *use cases*, this is not recommended due to the lack of clarity. A structured language description provides a better view of what is happening in what sequence between the user and the system.

*Fourth) Identify events for individual scenarios*

In this step, we identify events from each *use case.* Events are symbolized by interactions and their responses between users and the system. Each interaction between them is taken as an event. We can develop individual event flow diagrams for each of the scenarios. This is not difficult to do since each use case represents a scenario. So its description can be used to develop the relevant event flow diagram.

## 2. The White Box Stage

In this stage the system or an object is considered as a set of (sub)objects. We examine the details of the system by looking at the component objects (or sub-objects) and their interactions among themselves. At this stage we are looking at the inner functionality and the structure of an object to fulfill the user requirements.

*First) Identify Objects from the Use Cases*

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 [7] and OOADA [8] in that the complex objects are rejected in these approaches while we consider them as valid objects.

*Second) Identify Objects*

Using the black-box stage information (*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.

*Third) Identify interactions*

Using the above information domain, we now identify interactions. There are two types of interactions within a system: those that are between the objects of the system and its actors, and those that are between the objects themselves. While the former are easier to identify because they are part of the *use cases*, the latter type of interactions have to be found from the problem statement or the problem domain knowledge.

## 3. Recursion

In our approach, the above two stages are applied recursively. In the first instance, the system is treated as a black-box. Its actors and use cases are identified. From the use cases, system's interactions with its actors are identified. Then the same system is treated as a white-box and decomposed into several objects and their interactions are identified.

This procedure is repeated for any object which is complex by treating it as a system and then applying the above stages. In this case, the objects which interact with the object in consideration become its users (actors) and their interactions become their use cases with the object. These uses cases will, however, be smaller in detail and the number of interactions. Thus the black-box stage, as applied, consists of identifying the actors of the objects, their use cases, and the events between the object and its actors. Then we treat the same object as a white-box to be decomposed into further sub-objects (using the use cases and problem domain) and their interactions. This process is recursively repeated until sufficiently simple objects are found. This makes our 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.

The above steps, black box, white box, and recursion are applied in sequence. However within steps described in sections 2.1 and 2.2, 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.

## IV. AN APPLICATION OF THE APPROACH

In this section, we demonstrate the effectiveness of our 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 [13]. A standard *video conference* [14] is a specific multimedia, 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 our 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. among single individuals or groups of individuals at two or more locations.
>
> This *video conference* service will provide conference management functions such as setting up a conference, identifying participants on the *video conference*, connecting participants to a conference, disconnecting conference participants, terminating the *video conference*, floor grant, identifying the speaker, and controlling a speaker's microphone. These functions will only be available to the conference chairman.
>
> In addition, participants are provided with terminal handling functions or audio/video and signaling functions such as floor request, fax transfer, still picture transfer, and text transfer.
>
> When the chairman requests the opening of a *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 an invitation request message to selected participants and wait for their reply.
>
> When the system delivers an 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:

i) Join the conference: In this case the participant selects the acceptance button. The system will send an acceptance message to the chairman and display 'wait a minute' message to the participant .

ii) Does not join the meeting: In this case the participant selects the 'refuse' button. The system will send a refusal message to the chairman and go to the previous state.

iii) Does not make no reply: This is handled by the system using a time-out. If the participant does not reply for a given period of time, the system will send a 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:

i) opened by the chairman, the system will display opening message to the chairman and the participants and open the conference.

ii) canceled, then the system will display 'the conference got canceled' to the chairman and the participants and the conference will terminate.

When an 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 be closed.

## 1. Applying the Black-Box Approach

*First) Identify Actors*

From the above problem statement we find two actors:
chairman
participant

*Second) Identify 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

*Third) Describe the Use Cases*

Now we develop *use case* description for each of the *use case* identified above. The description for each *use case* can be developed by following the interactions and their responses betweenthe 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 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.

*Fourth) Identify Events*

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 Fig. 1.

## 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.

*First) Identify Objects from the Use Cases*

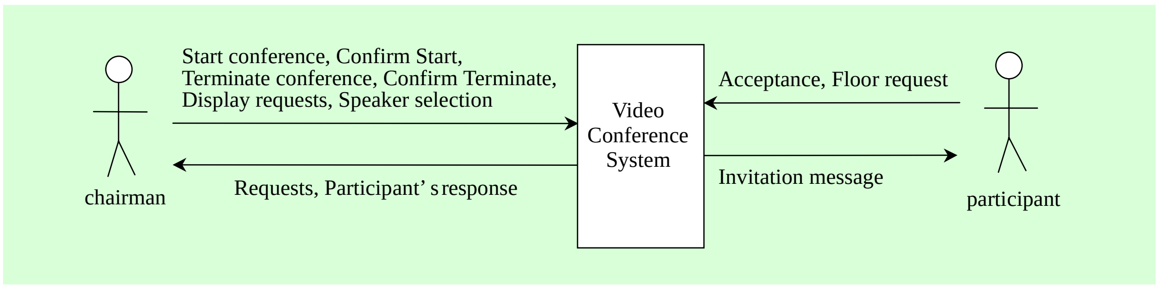
Start conference, Confirm Start,
Terminate conference, Confirm Terminate,
Display requests, Speaker selection
Requests, Participant' s response
chairman
Video Conference System
Acceptance, Floor request
Invitation message
participant

Fig. 1. System with events (Black-Box approach).

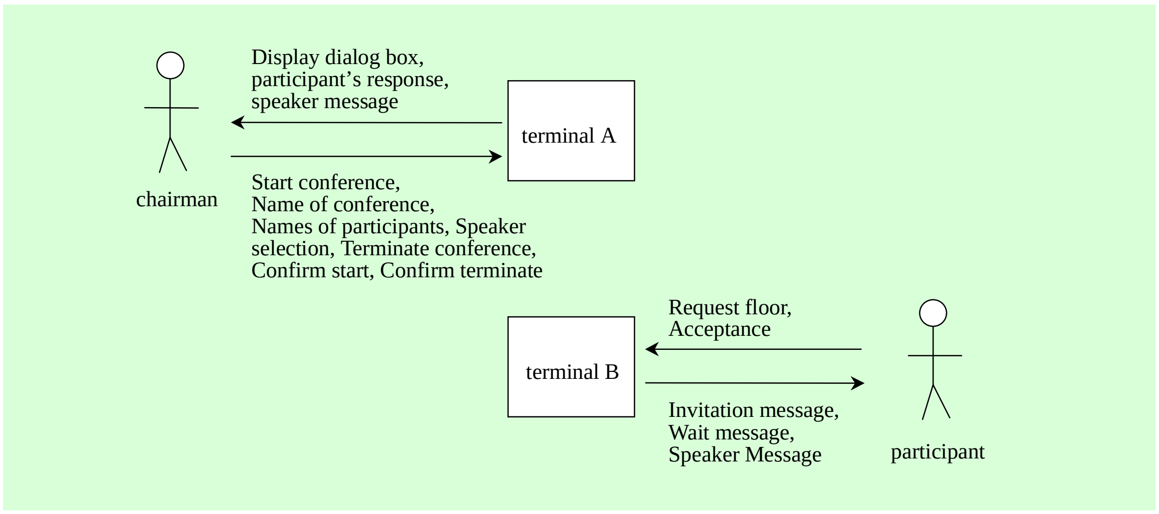

Fig. 2. Interactions between actors and system objects.

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.

*Second) Identify Internal Objects*

From the problem and solution space, we find the following additional classes of objects. These are internal objects since they do not participate in any use case. So we get:

*call control, multipoint control*

*Third) Identify interactions*

Interactions may be with the external elements or internal to the system. For this step we need to consider both of them. Firstly, we find interactions between system objects and the external elements (actors). Figure 2 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*).

Secondly, 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 [4]. Hence, we can represent the interactions among system objects (found from the problem and the solution space) as shown in Fig. 3. Other functionality can be shown similarly. Also note that similar to terminals A and B, there are two call control objects A and B.

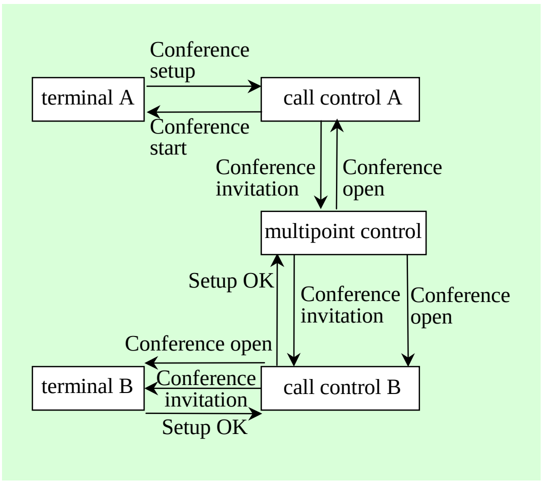

Fig. 3. Interactions among system objects.

## 3. Recursion

In this step, 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 our approach.

Considering the *multipoint control*, we proceed as follows:

### A. Black-Box Approach

*First) Actors*

Looking at the interactions with *multipoint control* we find the following actors:

- CCA
- CCB

*Second) Use Cases*

- Conference Invitation

*Third) Use Cases Description*

Now we describe the use case *Conference Invitation*:

1. Call Control A (CCA) sends Conference Invitation request to Multipoint Control Unit (MCU).
2. MCU asks the CCA for Conference name.
3. CCA gives Conference name.
4. MCU asks participant's name.
5. CCA gives participant's name.
6. MCU sends conference invitation message to CCB.
7. CCB sends acceptance.

*Fourth) Events from the use cases*

We identify the following events from the use case *Conference Invitation.*

- Conference Invitation
- Request conference name
- Conference name

- Request participant's name
- Participant's name
- Conference Invitation message
- Invitation acceptance

From these we develop the EFD(Event Flow Diagram) for objects interacting with the users (CCA and CCB). The EFD is shown in Fig. 4.

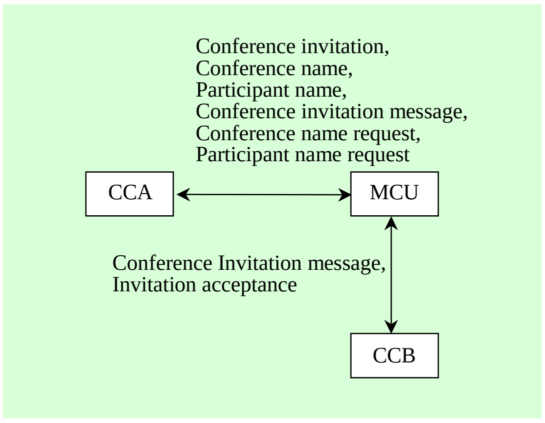

Fig. 4. EFD for events between system and its users.

## B. White-Box Approach

*First) Objects from the Use Cases*

Now we identify Objects from the Use Case *Conference Invitation*. In this case we find that there are no new objects that can be found from the above use case.

*Second) Internal Objects*

In this step we identify objects from problem domain. The objects identified are:

Network Interface Unit (NIU)
Control Processor Unit (CPU)
Audio Processor Unit (APU)
Video Processor Unit (VPU)
Data Processor Unit (DPU)

*Third) Interactions*

First we identify interactions between objects and users and then interactions between objects themselves. Since this is a white-box step, we will be looking at the problem domain to find out the interactions. In this case, it is a video conference subsystem which is concerned with handling conference control functions identified above as well as the transmission of actual video, audio, and data signals. From this point of view, the conference control functions are also requested in the form of control signals. Hence all these video, audio, data, and control are treated as signals. Thus Fig. 5 shows the interactions between objects and its users.

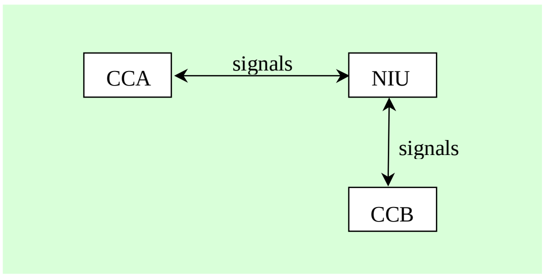

Fig. 5. The interactions between objects and users.

Then we find interactions between the objects only. The above signals are received by the NIU which distributes the signals coming from outside to the respective processing unit (audio, video, data, or control). The CPU provides the overall control of the conference by interpreting the conference control functions and issuing appropriate instructions to the NIU, Audio, Video, and Data processing units. For example, it provides audio information about the conference to the APU and video information to the VDU. The NIU is responsible for routing the incoming signals to their relevant processing units.

Keeping this information in mind, we get Fig. 6.

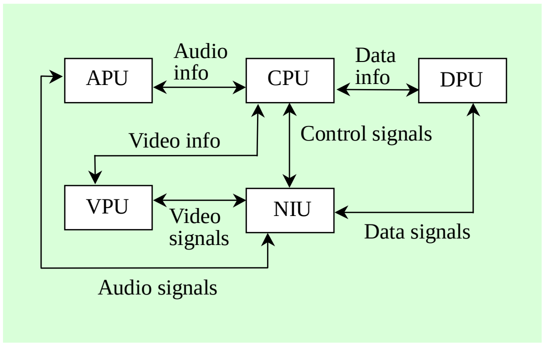

Fig. 6. Interactions between the objects only.

## V. ADVANTAGES AND DISADVANTAGES

The current approaches to Object-Oriented development have limitations in their application to systems which are complex in nature or in which developers do not have prior knowledge about the objects in the system. Our approach tries to solve this problem by not assuming prior knowledge about the application domain. Therefore, it is easier to apply under such a circumstances.

In our approach, the development of the system is based on expressing system functionality in terms of what is visible to users. Use cases are an important tool for expressing such functionality of a system. Therefore, this approach applies the use case concept to explore the underlying functionality of the system. Since the use cases are easier to identify, they facilitate the development of the system.

Also, since the development is based on use case concept and the system's functionality is expressed in terms of use cases, users can better understand the proposed system. They can approve, reject or recommend changes to it. Thus the application of use cases helps in system validation.

Our approach uses incremental development. Analysts start the development of a system using its user visible aspects such as its users and their use cases and interactions. Then as the development continues, the picture of the system becomes clearer and clearer to the developers. They can then also include the domain knowledge and complete the system development in an incremental and recursive way, all the while building on the knowledge gained from previous stages and creating more objects and their interactions.

Finally, the development is based on decomposition of objects on the basis of their complexity. An object's complexity is determined relatively by looking at its interactions with other objects (a black-box view). A simple object will have very few interactions with other objects and vice versa. Complex objects are further decomposed into sub objects and their interactions (a white-box view). This process simplifies the development and hides the complexity of a system. It is a top-down approach and results in a better structure and picture of the interaction between system objects.

A limitation of our approach is that it covers only the analysis part and not the design part of the development process. However, we believe that a good analysis technique provides a strong foundation for the design phase. Our approach in that respect provides a way forward even in applications where other Object-Oriented development techniques may not be able to proceed further because of lack of domain knowledge.

Our approach has a user influenced representation for functionality of objects. We represent system functionality in terms of use cases which is a good way of showing such functionality. At the highest level a use case represents the system functionality but as it is decomposed into smaller use cases it represents functionality of the corresponding objects. Since use cases are easier to find out than hidden system functionality we believe the application of use cases to be a powerful tool for system development.

## VI. A COMPARISON WITH COMMON OBJECT-ORIENTED 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 OOADA by Booch, OMT by Rumbaugh, The Unified Modeling Language, and OOSE by Ivar Jacobson.

### 1. 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, an example in the development of the analysis model from the requirements model and then the design model from the analysis model and so on. In the testing stage use cases are also 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 and 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 box concept. Secondly, we do not 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.

## 2. OOADA

Booch based this technique on his earlier Object-Oriented Design (OOD). The OOD was basically a design technique but OOADA extends that to cover the analysis and implementation stages. The OOADA technique suggests the use of logical and physical models considering both their static as well as dynamic properties. OOADA consists of many diagrams and notations. It supports a macro and a micro process.

The macro process drives the development process throughout all of its stages: analysis, design, implementation and testing. During this process requirements are established, a model of the desired behavior is developed, and the system is developed and maintained. The macro process contains the micro process. The micro process consists of identification of classes and objects, their semantics, relationships, and implementation.

OOADA uses use cases in the use case analysis to establish core requirements during the conceptualization stage. However, it is not use case driven. Instead its iterative and incremental development process consisting of macro and micro processes.

The OOADA does support two constructs for the development of larger projects. One of these is the class category concept. It is used to partition the logical model. It is basically an aggregation containing classes and/or other class categories. However, it only represents a name space and does not support the representation of normal association types, used between different classes, between different categories. The only associations that can be shown between categories are of type 'using'.

The other partitioning concept is that of subsystem. However, it is a design concept and is used to group together the program code. Hence it is a partitioning tool for the physical model rather than the logical one.

In contrast, our approach uses the semantics of the white and black boxes throughout the analysis stage. The identification of objects and their interactions at each stage are based on the application of white and black box concept. On the other side, OOADA uses no black or white box concept.

Another important difference is that our approach is top-down while the OOADA is a purely bottom up approach. First, objects are found and then categories and subsystems are formed as a way of showing dependencies and not for simplifying the analysis process which is the case in our approach.

## 3. 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.

## 4. MOSES

The MOSES method by Henderson-Sellers [11] provides activities, guidelines, and deliverables for all aspects of an OO project. It has notations for different phases of the development process. A major benefit of the MOSES method is the availability of a complete Object-Oriented based lifecycle known as the 'fountain' model. The approach claims that the fountain lifecycle is O-O-O that is Object-Oriented analysis, Object-Oriented Design, and Object-Oriented implementation.

One of the activities in the MOSES is the Scenario Development (essentially the *use case* concept). The purpose is to describe an interaction with a system from which, events and interactions can be found. MOSES uses the term O/Cs for Class & Objects. It is a class with object instances. The method uses scenarios for identifying operations and applies them for different subsystem responsibilities.

To manage the complexity of large projects, MOSES provides the notation of subsystems. However, these subsystems are not conceptual entities in the same way as classes. They act as a grouping mechanism.

The MOSES method is a complete methodology for software development. It uses scenarios for identifying Classes and Services. These activities are part of the development of the event model in MOSES and does not make it use case driven. In addition MOSES has the same obstacles in its application to complex system as the OMT and OOADA methods.

## 5. The Unified Modeling Language

The Unified Modeling Language [3] effort started as a way of unification of popular Object-Oriented development methods like OMT, OOADA, and OOSE. Not surprisingly it includes symbols and notations with semantics to represent the concepts found in these three methods plus additional constructs from other Object-Oriented methodologies. However, the unification effort had a shift in the emphasis from the method's process to its notation and semantics only. This forced a change of name from The Unified Method to The Unified Modeling language.

The Unified Modeling language incorporates similar concepts to the determination of objects and their associations as well as interface properties of the system. The notation supports application of the use case concepts and OOSE object types depending on their functionality. However, use cases are only used to support the development of sequence diagrams (the UML name for event trace diagrams). There is no process specified for applying the use case concept to exposing objects and their interactions.

The Unified Modeling language supports notation for expressing complex applications. These are in terms of the grouping constructs called packages. Earlier Categories were used for logical groupings while subsystems were used to group the code model. However, the concept of a package is much wider than a composite object. It can be used to express groups of classes or use cases as well as processor groups.

Like OOADA, the UML semantics suggest the grouping of classes or use cases into packages purely to manage them better and show the dependencies among themselves. The concept of package is not used to identify classes or associations as we have done in our approach, to facilitate analysis or to overcome lack of prior knowledge about the problem domain. The UML's use of packages is also bottom up like OOADA because the first classes and use cases have to be identified and then grouped together into packages. On the contrary, our approach is top down as described earlier and hence facilitates the analysis stage by providing more information about objects and their interactions.

In the Unified Modeling language there is no process specified with the application of the notation although its authors assume a use case driven, architecture centric, iterative, and incremental approach. On the other hand our approach is use case driven and a step wise recursive approach and uses the notion of complex objects to handle the complexity of large projects early in the analysis stage.

## VII. 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 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. This has been used in several other methods 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 our 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 [15] and functional decomposition as used in the Object-Oriented Analysis and Design method by Martin and Odell [12]. 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 consideration.

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 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.

Our approach was applied to an example system (a *video conference*). The application of our approach demonstrated the practicality and usefulness of the approach for complex systems where no prior knowledge about objects exists. Although ours is not a complete methodology – it covers only 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 methodologies to complex systems such as distributed systems and telecommunications. Without a good analysis stage, the design and its ensuing implementation stages cannot guarantee the success of system development effort.

## REFERENCES

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

[2] J. Rumbaugh, "OMT: The Development Process," *Journal of Object-Oriented Programming*, Vol. 8, No. 2, May 1995, pp. 8-16.

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

[4] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language for Object-Oriented Development,* Documentation Set, Version 0.9, Rational Software Corporation, July 2 1996.

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

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

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

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

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

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

[11] B. Henderson-Sellers, and J.M. Edwards, "Book-Two of Object-Oriented Knowledge: The Working Object," Prentice-Hall, Sydney, 1994.

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

[13] ITU-T I.150, *B-ISDN Functional Features,* Geneva, 1990.

[14] ITU-T F.732, *Broadband Video Conference Services*, Geneva, 1989.

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

**So-Ran Ine** received the B.S., M.S., and Ph.D. degrees in computer science from Hongik University in 1978, 1982, and 1991 respectively. Since she joined ETRI in January 1978, she participated in the various projects, related to telecommunication network, computer network, computer system software such as distributed system software, client-server based application development tool, software engineering, and CORBA security service. She is currently a Principal Member of Engineering Staff of ETRI and head of Software Engineering Section. She is a member of Korea Information Science Society, Korea Information Processing Society, and Korea Institute of Information Security & Cryptology. Her research interests include protocol engineering, software engineering, system software security technology, and the middleware technology. Her contact point is srine@computer.etri.re.kr.

**Cheong Youn** received the B.S. degree in physics from Seoul National University, Seoul, Korea, in 1979, the M.A. degree in computer science from Sangamon State University, Springfield, IL, in 1983, and the Ph.D. degree in computer science from Northwestern University, Evanston, IL, in 1988. He was a full-time lecturer at Wayne State College from 1983 to 1985 and Northwestern University from 1985 to 1987. He was a Member of Technical Staffs at Bell Communications Research from 1988 to 1993. Currently, he is an Associate Professor of Computer Science Department, Chungnam National University, Taejon, Korea. His current research interests include software engineering, CALS, object-oriented modeling and design.

**Uddin Mirza Misbah** received the Ph.D. degree in Computer Science from UMIST, UK in 1990. He was on a Korea Science and Engineering Foundation post-doctoral research fellowship at Chungnam National University from 1995 to 1996. Currently, he is an Assistant Professor of Computer Science Department. at the GIK Institute of Engineering Science and Technology, Topi, Pakistan.

**Kwon-Il Lee** received the B.S. degree in computer science from Chungnam National University in 1988. Currently she is a senior researcher in Distributed Computing Section, ETRI, Taejon, Korea. She is a member of Korea Information Science Society, Korea Information Processing Society, and Korea Institute of Information Security & Cryptology. Her research interests include the distribued computing, software engineering, and system software security technology

**Seung-Hoon Cha** received the B.S. degree in 1995 and the M.A. degree in 1997, both in computer science from Chungnam National University, Taejon, Korea in 1997. Currently, he is a researcher at Agency for Defense Development, Taejon, Korea. His current research interests include object-oriented modeling, distributed computing and multimedia system.

**Bo-Gyun Byoun** received the B.S. degree in 1996, and the M.A. degree in 1998, both in computer science from Chungnam National University, Taejon, Korea. Her current research interests include software engineering and distributed computing environment.

**Doo-Hwan Bae** received the B.S. degree from Seoul National University, Seoul, Korea, a M.S in Computer Science from University of Wisconsin-Milwaukee and a Ph. D in computer & information Science from University of Florida. Currently, he is an Assistant Professor of Computer Science Department, Korea Advanced Institute of Science and Technology. His research interests include object-oriented technology, distributed & parallel software engineering, software development methodology and process model.