01 Jan 1997

# High-Order Object Model Based Software Analysis

Xiaoqing Frank Liu
*Missouri University of Science and Technology*, fliu@mst.edu

Hungwen Lin



## Recommended Citation

X. F. Liu and H. Lin, "High-Order Object Model Based Software Analysis," *Proceedings of the 21st Annual International Computer Software and Applications Conference, 1997*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1997.
The definitive version is available at https://doi.org/10.1109/CMPSAC.1997.624800

# High-order Object Model Based Software Analysis

Xiaoqing Frank Liu and Hungwen Lin

Department of Computer Science
University of Missouri
Rolla, Missouri 65409

## Abstract

*The integration of object-oriented modeling and structured analysis (SA) for developing a well-structured object-oriented software systems is a challenge for requirements analysts and software designers. Some of the existing object-oriented modeling techniques adopt approaches that are very different from SA, and others have clumsily stayed with SA after a few modifications. Our High-order Object Modeling Technique (HOOMT), however, attempts to strike a mean between both extremes while at the same time provides an effective modeling method. HOOMT consists of two models, the High-order Object Model and the Object Information Flow Model. By using familiar divide-and-conquer concept and functional decomposition, our approach also ensures a less stressful migration of SA analysts to object-oriented platforms. This paper introduces HOOMT, its concepts, and notations.*

## 1 Introduction

### 1.1 Background

One of interesting and important issues in object-oriented analysis (OOA) and design is how to identify structure of an object model and tightly integrate it with structured analysis technique. Objects are the basic conceptual units in the object-oriented (OO) analysis. But simple objects and their relationships can be organized and encapsulated into a high-order object which may be complex.

Many of today's object-oriented analyses adopt bottom-up approach by identifying objects first and overall system structures later. As one can see, if OOA is to proceed in a straight bottom-up fashion, OO software engineers will face the formidable task of uncovering structures within a network of objects linked by complex communications and relationships. On the other hand, SA does not usually suffer from such weakness since the system structure is carefully developed along the way as the processes and data flows are analyzed and decomposed in a top-down fashion.

To arrive at a more structured system in a systematic manner, OOA can borrow the idea of top-down approach from SA. Not only will the resulting system be structured by integrating SA and OOA, the migration to object-orientation will also become smoother for software engineers who are used to SA. Several studies and papers have concluded that integration of the two methods are possible and desirable. In his paper, Ward shows that "there is no fundamental opposition between real-time structured analysis/structured design and object-oriented design" [9]. Others have found from their experiences or surveys that structured analysis and object-oriented analysis have high degrees of compatibilities [6, 8].

### 1.2 Review of Relevant Object-oriented Modeling Techniques

Several approaches such as Coad/Yourdon's OOA, Hierarchical Object-oriented Design (HOOD), Object-Oriented Software Development Method (OOSD), and a Semantically Rich Method of Object-oriented Analysis (SOMA) have been developed to identify structure of an object model. However, these approaches do not support the structured analysis of functional requirements represented by methods of objects.

OOA introduced by Coad/Yourdon represents the backbone of today's OOA. It has complete descriptions of objects, attributes, and methods. In addition, this OOA defines the concept of 'subject.' A subject is "a mechanism for guiding a reader through a large, complex model"[1]. In other words, a subject helps grouping 'related' object classes together by creating an artificial boundary. This is Coad/Yourdon's way of managing visual complexity of the system. However, a subject is not an object, it does not encapsulate objects and their relationships. Also, Coad/Yourdon's object-oriented analysis does not incorporate SA concept or notations.

OOSD [2] and HOOD as described by Graham in his book [4] allow you to decompose the top-level ob-



0730-3157/97 $10.00 © 1997 IEEE

ject into low-level objects and describe information flows between objects. However, they do not have mechanisms for analysis of functional requirements represented by methods of objects. In addition, these methods do not support inheritance.

SOMA [4], developed by Graham, adopts many notations and ideas from previous models especially Coad/Yourdon's OOA. Its use of the 'layer' concept is very important. In this method, the layer provides another more concrete and higher abstraction level that does what Coad/Yourdon's subject does but more. Layer can be thought of as an object.

In his paper, Graham compares SOMA's layer to Coad/Yourdon's subject, "identifying subject areas is a top-down process aimed at breaking the problem up into manageable chunks. This is useful but the 'subjects' are not objects and have no formal status or semantics within the model"[3]. The layer exists at the top of a composition structure and each of its methods must be implemented by or linked to its component objects' methods. Being just at the top of a composition structure, a layer is essentially an artificial "object wrapper" that introduces no new methods of its own.

On the other hand, Rumbaugh's Object Modeling Technique (OMT) uses models and diagrams that greatly resemble those found in structured analysis. However, OMT does not have high-level object and its module only serves as "a logical construct for grouping classes, associations or view of a situation." Further, "the boundaries of a module are somewhat arbitrary and subject to judgment." Like subjects, modules help partition and organize a complex system, but modules are not objects [7].

### 1.3 Our Approach

Our object-oriented analysis, formally called High-order Object Modeling Technique (HOOMT), consists of two models, the High-Order Object Model (HOOM) and the Object Information Flow Model (OIFM). HOOM has a Context Object Diagram (COD), a set of High-order Object Diagrams (HOOD), and a set of Primitive Object Diagrams (POD). On the other hand, OIFM comprises a set of High-order Information Flow Diagrams (HOIFD) and a set of Low-order Information Flow Diagrams (LOIFD).

Object Information Flow Model and High-order Object Model work hand in hand to obtain object diagrams and object information flow diagrams for an interested system. Because both models are hierarchical and structural in nature, the resulting structure of the system is more organized and manageable than the traditional OOA. HOOMT approach gives analysts a systematic way for decomposing complex systems top-down for analysis.

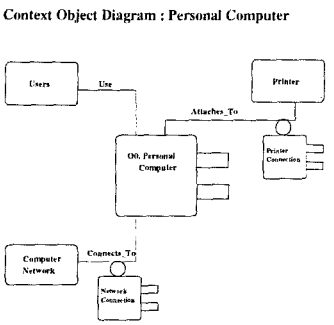


Figure 1: **Context Object Diagram for "Personal Computer"**

The next section will detail the semantics and notations used for each model. We will also discuss the methodology for constructing both models.

## 2 HOOMT

### 2.1 High-order Object Model (HOOM)

High-order Object Model is built hierarchically by first deriving a Context Object Diagram showing the interested system as a high-order object interacting with its surroundings. Subsequent analysis involves decomposing a high-order object to simpler component objects. The process of decomposition continues until only primitive objects remain. Before further discussion, we now define a few concepts used in this model.

A high-order object (HOO) is an object that can be decomposed to low-order component objects. A component object can either be another high-order object or a primitive object (PO). If a component object is primitive, then it means this object is simple and requires no further decomposition in the analyst's mind. In addition to encapsulating methods and attributes, high-order objects in HOOM encapsulate other objects and the relationships among them.

To better illustrate HOOM, a simple example of a personal computer is shown in Figure 1. As a high-order object, "Personal Computer" interacts with external objects "Users," "Printer," and "Computer Network." Notice that "Personal Computer" is symbolized by a rounded rectangle with two attaching rectangular boxes that represent attributes and methods. On the other hand, external objects are symbolized by simple rounded rectangles. The lines running between objects are static relationships between these objects. Some of these static relationships such as the "Network Connection" can be complex and therefore

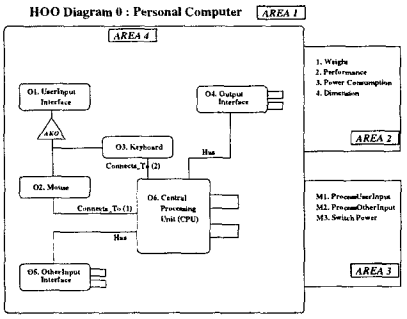


Figure 2: **HOO Diagram for "Personal Computer"**

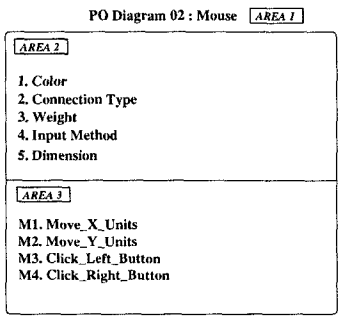


Figure 3: **Primitive Object Diagram for "Mouse"**

considered by the analyst as another high-order object.

"Personal Computer" is subsequently decomposed to six component objects in the High-order Object Diagram of Figure 2. A High-order Object Diagram has four major areas as indicated in Figure 2. In Area 1, the title "HOO Diagram 0 : Personal Computer" indicates that this diagram is a High-order Object Diagram for the high-order object "Personal Computer" with the object label '00.' The second and the third areas list the attributes and methods. Finally, Area 4 displays component objects of "Personal Computer" and their relationships. Note that the object "OtherInput Interface" strictly includes only input devices that are not part of the user interface, for example, a temperature data-acquisition card.

In addition to encapsulating component objects, "Personal Computer" encapsulates relationships among component objects. For example, in "HOO Diagram 0" the specialization relationship between "UserInput Interface," "Mouse," and "Keyboard" objects is represented by an "AKO" (A-Kind-Of Relationship) symbol.

Primitive objects are represented by simple rectangles with labels written in them. In "HOO Diagram 0," "Mouse" and "Keyboard" are primitive objects. Descriptions of primitive objects are detailed in Primitive Object Diagrams. Figure 3 shows the POD for "Mouse." Unlike a high-order object, a primitive object has no component objects. Its attributes and methods are listed in the Area 2 and Area 3 respectively. At this level, three component objects in Figure 2 are still at high-order, and decomposition of each will be required for analysis.

Methods of a high-order object may be implemented by one or more methods from its component objects. For example, the method "Switch Power" of "Personal Computer" (See Figure 2) may be implemented by a series of activations of similar "Switch Power" methods from component objects "Output Interface" and "Other Input Interface."

At the end of High-order Object Modeling, we have hierarchical layers of High-order Object Diagrams and Primitive Object Diagrams for the system of interest. The eventual designers of the system can be guided through the analysis of the system via this hierarchy of the object diagrams. More discussions about HOOM, such as object inheritance and cardinality constraint, can be found in the paper [5].

## 2.2 Object Information Flow Model

Object Information Flow Model details the transformation of object information flows through information processes. A High-order Information Flow Diagram (HOIFD) is first derived from a High-order Object Diagram. Processes in HOIFD are then decomposed to low-order information processes in Low-order Information Flow Diagrams. The purpose of this model is to help analysts identify and better understand methods from objects during the High-order Object Modeling.

OIFM provides a functional view of the system in terms of information source, sink, and process. An object acts as an information source or sink in this model. An information source provides information flow that flows into and triggers a process. An information sink accepts the output flow from one or more processes. In this way, OIFM shows how information is transformed from a source object to a sink object.

To illustrate OIFM, we again resort to the use of "Personal Computer" as our example. Objects are represented by rectangular boxes and the processes by circles. The HOIFD for "Personal Computer" is shown in Figure 4. Since "Personal Computer" has the object label '0,' its corresponding HOIFD is given the diagram label "HOIFD 0." Objects "Users" and "Personal Computer" are information sources that initiate the process "Switch Power." At the other end of

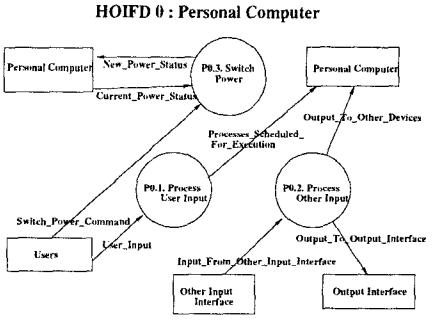


Figure 4: **High-order Information Flow Diagram for "Personal Computer"**

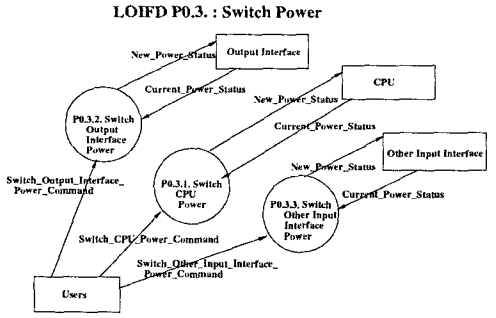


Figure 5: **Low-order Information Flow Diagram for "Switch Power" Process**

"Switch Power," object "Personal Computer" serves as the information sink.

Each process in HOIFD corresponds to a method in the high-order object HOIFD. At this point, we wish to better understand how each of these high-order processes behave, so we further decompose each process to sub-processes in Low-order Information Flow Diagram (LOIFD). LOIFD in Figure 5 shows the decomposition of the high-order process "Switch Power." The low-order process "Switch CPU Power" has the process label "P.0.3.1" which means it is decomposed from the high-order process "P.0.3."

The notations and techniques used in constructing an Object Information Flow Diagram are consistent with those used in deriving Data Flow Diagrams. This is not a coincidence, HOOMT attempts to integrate OOA with SA.

## 3 Conclusion

In this paper, we have presented the High-order Object Modeling Technique, which provides a structured approach for analysis and design of both the object and functional models. In essence HOOMT tightly integrates OOA with structured analysis. It uses the all-too-familiar concept of divide-and-conquer and decomposes high-order objects in HOOM and processes in OIFM to their component objects and subprocesses respectively. The top-down development of the HOOM and OIFM provides a systematic approach to the realization of a more structured system with highly cohesive clusters.

Thus far in this research, we have developed OIFM and HOOM for OOA. The future direction of the research will be geared towards developing a dynamic model using State Transition Diagrams and finding a set of appropriate metrics for evaluating the effectiveness of HOOMT.

## Acknowledgments

We would like to thank Dr. Lijun Dong for her comments on the draft of this paper.

## References

[1] Coad, P. and E. Yourdon, *Object-Oriented Analysis*, 2nd ed., Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[2] Colbert, E. "The Object-Oriented Software Development Method: A Practical Approach to Object-Oriented Development", *TRI-Ada 89 Proceedings*, October 1989, pp. 400-415.

[3] Graham, I. "Migration Using SOMA: A semantically rich method of object-oriented analysis, Journal of Object-Oriented Programming", Vol. 5, No. 9, February 1993, pp. 31-42.

[4] Graham, I. *Object-Oriented Methods*, 2nd ed., Addison-Wesley Publishing Company Inc., New York, 1994.

[5] Liu, Xiaoqing Frank. "HOOM: A High-order Object-oriented Model", to be submitted for publication.

[6] Loy, P.H. "A Comparison of Object-Oriented and Structured Development Methods", *ACM SIGSOFT Software Engineering Notes*, Vol. 15, No. 1, January 1990, pp.44-48.

[7] Rumbaugh, J *et al. Object-Oriented Modeling and Design, Prentice Hall*, Englewood Cliffs, NJ, 1991.

[8] Vazquez, F. "Using Object Oriented Structured Development To Implement A Hybrid System", *Software Engineering Notes of ACM SIGSOFT*, Vol. 18, No. 4, October 1993, pp. 44-53.

[9] Ward, P.T. "How to Integrate Object Orientation with Structured Analysis and Design", *IEEE Software*, March 1989, pp.74-82.