Object Based Requirements Modeling for Process Continuity

A.I. Anton (anton@cc.gatech.edu)
T.A. Gale (tomg@cc.gatech.edu)
W.M. McCracken (mike@cc.gatech.edu)
J.J. Shilling (shilling@cc.gatech.edu)

Center for Information Management Research
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280

Abstract

The re-engineering of information systems and business processes that occurs during enterprise analysis demands a robust, rigorous, and expressive technique for modeling elicited system requirements. The requirements models must also be easily understood by end users. This paper presents a hybrid object based modeling technique which supports progressive formalization of requirements models with minimal transformations. The technique strives to impose consistency and coherence on the modeling process yielding a higher level of process continuity, enhancing end-user comprehension, and facilitating communication between analysts and end users.

1 Introduction

As new paradigms emerge, it is important for organizations to re-examine and perhaps even re-design their enterprise analysis process in order to ensure that the organization's current needs are being satisfied in a relevant manner. Enterprise analysis attempts to rebuild businesses around information. The ultimate goal of re-engineering is to identify new methods of re-designing information systems and organizational structures. The focus of the research presented in this paper is the requirements portion of the enterprise analysis process. Specifically, our endeavor encompasses the creation of a representation that aids the user comprehension element of the process. This paper presents a representation and modeling technique for the information gathered as a result of requirements elicitation.

Traditionally, the requirements elicitation and modeling process uses techniques and representations that are difficult for a user to understand and often forces transformations to occur when communication takes place between the user and analyst. In other cases, representations are informal (to promote communications, such as natural language), and suffer the inherent problems of inconciseness, ambiguity and incompleteness. The need to formalize requirements to maximize correctness has been well documented in the literature [7] [8] [15], and represents a significant improvement in specifying systems (this has been shown both experimentally as well as in actual application of these techniques). The problem of communicating with users who are not mathematically inclined or computer literate still exists. If a specification is written using VDM [15], for example, and the user is not well versed in the representation, then a translation has to occur to communicate the analyst's understanding of the problem to the user. This translation is typically from a formal representation to an informal natural language representation.

Coad proposes that object based modeling techniques enhance the requirements elicitation and modeling processes by providing a common vocabulary familiar to both users and analysts [6]. An object based requirements modeling technique can be accessible to non-technical / computer naive users, and yet is rigorous enough to allow for translation into a full object oriented design. An object based model jointly developed by both the analyst and a group of end users, using a simple notation with different levels of complexity and abstraction, may provide a model that is easily understood and readily accepted by both parties. Object based modeling techniques may allow

0-8186-1060-3425/93 $03.00

dynamic simulation of the requirements model during the requirements elicitation process to shorten the evaluation/feedback/modification cycle.

The initial phases of our research in requirements modeling techniques included the development of a rigorous, highly expressive object based modeling technique which is described in this paper. This representation is the final form into which the requirements model will be translated. The object based model will be reduced and simplified, maintaining the rigor, but hopefully ensuring at all times that it is easily understood by non-technical end users.

2 Requirements for an Adequate Modeling Technique

Many requirements modeling techniques yield poor end user comprehension due to either misinterpretations of the elicited requirements by the analyst or because the modeling representation fails to convey the appropriate meaning. These factors further complicate the already difficult communication process which takes place between analyst and end user. In order to address these and other problems, we believe that an adequate requirements representation technique should exhibit the following characteristics:

- Facilitate communication between analysts and end users
- Increase user comprehension
- Make it easy for analysts to modify the knowledge content without making structural changes to the model
- Facilitate the reuse of artifacts from the software development process (software, designs, requirements, etc.)
- Employ a single evolvable uniform notation (with minimal transformations)

These characteristics serve as a basis for our proposed modeling technique.

Since end users may have a difficult time understanding the representation of the requirements as stated by analysts, an adequate representation should minimize this lack of understandability in order to increase user comprehension. The model must be understandable by end users and customers. It should also serve as an efficient communication vehicle between both parties. Iterative refinement of the model should also be supported at all phases.

Modifications to the knowledge content should be a simple process for the analyst and these modifications should not necessarily result in changes to the actual structure of the model. In addition, a requirements technique should employ a single notation that is usable throughout the entire requirements and design process. The notation should be evolvable from informal to more formal notations as progression toward implementation ensues.

In addition to the above characteristics, we believe that an adequate modeling technique should also support extensive reuse (see section 6.5). It is our opinion that the use of the object based paradigm throughout the software development process enables the reuse of artifacts other than software including requirements and design.

3 Why Functional Techniques Are Inadequate

Traditional requirements analysis techniques, for the most part, have been functional in nature (i.e. SADT [17], SREM [1], etc.). The requirements analysis process is composed of two main tasks: elicitation and modeling [14]. Requirements are typically elicited from end users via interviews or questionnaires of the user group community. It is then the responsibility of the analyst to synthesize the gathered information and create a model or representation that can be validated by end users [16]. As stated in Section 2, the model or representation developed by the analyst is often not easily understood by the end users. In addition, as progression from an informal towards a more formal representation occurs, the representation changes not only in level of formality and level of detail, but in the actual appearance (i.e. notation, diagrams, structure, etc.) and possibly in its intuitive appeal to the customer. As formality increases the level of comprehension of the model for the end users often decreases.

Structured Analysis (SA) techniques emphasize functional decomposition [16]. The main focus is in the actual functions provided by the system to the users [18]. The system is organized around these functions or their corresponding procedures. According to Coad [5], the object oriented (OO) approach provides a better fit to end users' mental models. He proposes that people organize complex concepts by breaking down the organizational thought process into aggregation, classification and differentiation. He claims that OO methods fit this model. This suggests that functional techniques may not be as intuitive to end users since they may view the world in terms of objects rather than functions (see section 4.1). In functional

decomposition, modifications have more potential for far reaching repercussions and side effects than in OO systems. Clearly defined system boundaries are inherent in SA designs making extensions more difficult than in OO techniques. "An object oriented technique is more resilient to change and more extensible" [19].

Functional decomposition in SA techniques is arbitrary and is dependent on those persons involved in decomposing the problem into processes and subprocesses. However, in OO techniques, the decomposition is based on the objects in the particular problem domain. Analysts working on different systems in a particular domain tend to identify similar objects. This increases reusability of components from one project to the next [19].

The structural changes that often times occur throughout the progression of a requirements model from an informal representation to a more formal representation may contribute to the difficulties in communication between the analyst and the user. The transition from requirements to design may be difficult for end users to follow due to structural changes. Functional modeling techniques do not provide what we consider to be the necessary evolutionary formalization. This may make user verification and validation more difficult than they need be.

4 Why Object Based Techniques Show Promise

One type of modeling technique which has received significant attention in recent years is object based modeling [3] [5] [18] [20]. This section describes why we believe object based techniques may be capable of satisfying the requirements for modeling techniques presented in Section 2.

4.1 User Comprehension

Object based modeling techniques may provide a higher level of end user comprehension than that achieved via more traditional modeling techniques. The primary reason for this is that real world entities are modeled as objects which the user may already be familiar with. This may improve several aspects of user comprehension including:

- Mental models - When the object being modeled maps directly to an entity which the user is familiar with, it provides an initial frame of reference (mental model) that allows the user to quickly comprehend the entity.
- Visualization - Each object being modeled provides a convenient subcomponent of the system which can be visualized in isolation. This can aid the user in understanding what the overall system of interacting objects is trying to accomplish by allowing users to visualize and comprehend each object separately rather than being forced to comprehend the entire system as a monolith.

4.2 Reusable Artifacts

In object based modeling techniques, the system is decomposed into encapsulated objects which may be modeled in isolation, as a component of a subsystem, or as a framework of objects. Since each object or framework of objects is a stand alone entity or subsystem with all necessary capabilities and data structures, these object models may be reused in a fashion similar to their software counterparts. The requirements model and formalized design for these objects may be utilized in other systems which require classes of objects with similar capabilities.

Although this paper focuses primarily on the requirements analysis and design phases of the software development lifecycle, our observations about the reusability of requirements, design, and implementation artifacts are applicable to all phases of the software lifecycle.

4.3 Minimizing Transformations

Transformations that are required throughout the transition from the object based requirements model to the final implementation are reduced due to the direct correlation between object based modeling techniques and commonly used object oriented design and programming techniques. The initial use of an object based model allows the transformations required to transition from analysis to design and implementation to be viewed simply as a formalization of the model rather than one that involves radical structural changes.

This seamless process of migrating the object model from analysis into design and implementation has several advantages:

- Loss of information is minimized - The amount of information lost in the transformation process is proportional to how radical the transformation is. The more radical the transformation, the higher the probability that some initial requirements information will be lost. It appears logical that by reducing structural transformations, loss of information is also reduced.

- Customer validation is simplified - The amount of effort required from a customer to comprehend and validate the design of the system is reduced because the final design and implementation mimic the object model created during the initial requirements phase.
- Process fluidity is enhanced - The development lifecycle becomes a much more coherent process with the model 'flowing' through the analysis, design, and implementation phases, being enhanced, formalized, and refined throughout its progression, but with its overall structure remaining intact.

4.4 Evolving Formalism

The initially object based model resulting from the requirements elicitation and modeling process is iteratively refined into a more formal representation, suitable for migration into the design phase. The actual objects which have been identified still exist, but the details of what these objects do and how it is accomplished becomes progressively more formalized.

The initial object based requirements model may identify several classes of required objects and may contain a natural language description of the capabilities and attributes of those objects. The initial model of the dynamic behavior of an object may include a list of the set of possible states the object may enter as well as a natural language description of what causes an object to transition from one state to another.

As the model is further refined, these natural language descriptions are replaced by more formal notations which describe the model in sufficient detail so as to clearly indicate the correct performance expected of each object.

The iterative formalization process described above produces an object model of sufficient rigor such that it may be used with minimal modification as the overall design of the system.

5 A Hybrid Object Based Technique

5.1 Modifying Current Object Based Modeling Techniques

While there are a plethora of object based modeling techniques currently in use, some modification of these techniques is needed in order to meet our goals. The proposed technique is a combination of several features of existing object based techniques, with some additional features intended specifically to better meet our criteria (see Section 2).

5.2 Three Views Within the Model

The object model consists of three different views of the requirements to be specified. This three viewpoint approach was adopted from the Object Modeling Technique (OMT) [18] [19].

The three views included in the model are:

- Static model - provides a representation of all static aspects of the objects in a system. In particular, all capabilities (methods) and data structures (attributes) associated with each individual object class and all relationships between objects.
- Dynamic model - contains a state/transition based model of the interaction between objects in a system.
- Functional model - provides all of the functional detail regarding the capabilities of each object class presented in the static model and used in the dynamic model.

The static and dynamic model are presented in diagrammatic form, while the functional model is in textual format. The following sections detail each of the views listed above. As an example, the modeling technique will be used to model the workings of a simple traffic light system.

5.2.1 Static Model

OMT provides an expressive notation for diagramming the methods, attributes, and relationships of object classes [18] [19]. Each object class is contained within a rectangle with round corners which is divided into three sections:

- Class name - Descriptive name for the object class.
- Methods - List of all methods (capabilities) of the object class. As the model is formalized, the method list will specify data types for all arguments and return values.
- Attributes - List of all data attributes of the object class. As the model migrates from informal to formal through iterative refinement such details as data types and initial values are added.

The OMT static model also diagrams all relationships between object classes. The technique has robust and expressive notations for indicating attributes of relations such as cardinality and inheritance.

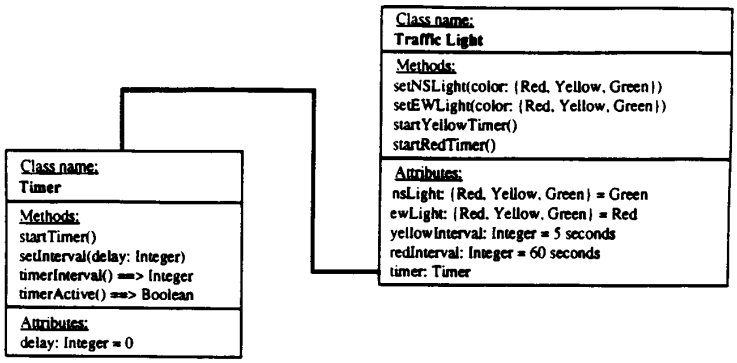

Figure 1: Static model for traffic light

Figure 1 shows a static model example for a traffic light system. This example depicts two classes, the traffic light class and the timer class. The timer class will be used to control the intervals at which the lights change color.

This example depicts the finalized formal version of the static model. This model should contain sufficient detail about the static information associated with the system to transition the model into the system design phase. For brevity, some information has been omitted from this model, but normally the final version of the static model should contain the following information:

- Data types for all attributes
- Initial values for all attributes
- Formal parameter descriptions for all methods
- Data types of return values for all methods

For clarity, titles have been added to each of the three sections of the static object model so that each section's contents may be easily identified. In OMT the use of titles is omitted. We believe that titles increase comprehension by providing important cues to users who may only occasionally view these models.

The initial version of the static model formed during requirements elicitation would use a natural language notation for the description of attributes and methods. The transition into this more formal syntax would occur as the requirements analysis phase progressed. Because the overall structure of the object classes remains constant throughout this transition, user comprehension is enhanced.

Note that the timer class is an excellent candidate for requirements reuse. Many event driven or reactive systems require the use of timers to cause some activity on a regular basis. If the object model requirements repository contained a timer object, it could quickly be integrated into other requirements models.

5.2.2 Modifications to the OMT Static Modeling Notation

The following list summarizes the minor modifications which we have made to the original OMT static modeling technique:

- Titles were added to each section of the object class to make the contents of each section of each object unambiguously clear.
- Objects which this object has relations to are listed explicitly as attributes of the object. The motivation for this modification was to provide as much information as possible during the times when the object might be viewed in isolation. This might be the case in some situations of browsing through libraries of reusable requirements.
- Methods are listed before attributes within the static model to allow the object classes to be displayed with the attributes suppressed. The suppression of attribute information is used to view the external (visible) interface of an object while encapsulating internal attribute information.

5.2.3 Dynamic model

The dynamic model of a system describes all of the run-time interactions between object classes. These interactions specify how work is accomplished in the OO system. The exercise of detailing all object interaction within the system has the side effect of exposing what services each class of object must support. The

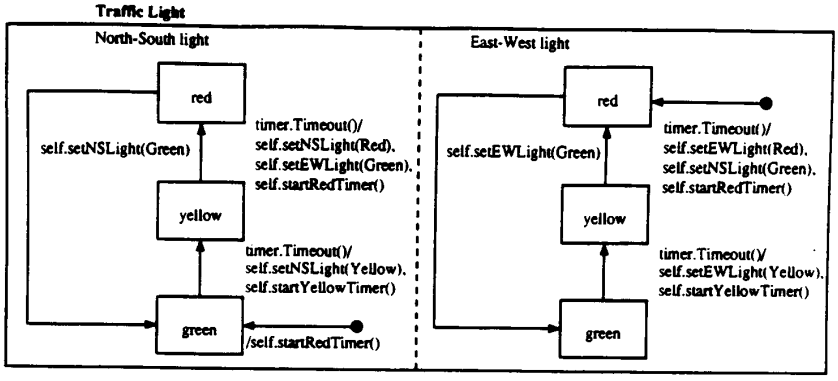

Figure 2: Dynamic model for the traffic light

dynamic modeling process produces all information about a class required for the static model. These services required from a class become the methods which this class must support. In detailing these methods, internal attributes required to maintain state information required by these methods may also be exposed.

The representation which we have adopted is an object based extension to Harel's state charts [10]. Because of the ability for state charts to represent parallel states and their inter-STD communication capability, state charts lend themselves nicely to the modeling of object based systems.

In the dynamic model, each object class will be modeled as one state chart. The interactions between object classes will be modeled using the inter-STD communication notations of state charts. Since each object instance will maintain its own state independent of other objects of the same class, each instance of an object class will have its own copy of the state chart for the object class.

The Object Charts [3] and OMT [18] techniques have also based their dynamic models on state charts. The primary difference between these two notations and ours lies in the identification of those events within the system which are allowed to cause state transitions, and what actions can occur when an event occurs. These events and actions are specified on the transitions between states, and are separated by a slash. The event within the object system which caused the state transition appears on the left hand side (LHS) of the slash. The actions to be taken when the event occurs are specified on the right hand side (RHS) of the slash. This is similar to notations used in traditional state transition diagrams [8]. See sections 5.2.4 and 5.2.5 for a detailed description of the semantics of our events and actions.

We refer the reader to [18] for a general discussion of the use of state charts for modeling the dynamic aspects of OO systems. We will point out the differences between their notations and ours by detailing what is allowed on the LHS and RHS of state transitions.

Figure 2 shows a state chart for the traffic light object class. The dashed line between the two STDs indicate that object instances of this class have two parallel states, which in this case are the colors of the north/south and east/west lights. The transition arrow which has no source state (arrow emitting from the dot) indicates the start state of each of the parallel states. In this case the start states indicate that when an instance of the traffic light class is created, the color of the north/south light will be green, and the color of the east/west light will be red.

Note several aspects of this diagram:

- On the state transitions, the information on the left hand side of the slash indicates the event within the system which causes this object to transition between the two states at the source and destination of the transition arc. If there is no slash embedded in the transition information, the text specifies the event which causes the transition, and there are no actions to be taken on the transition.
- The information on the right hand side of the slash on the state transitions indicates any actions which should be taken as a result of the event which caused the transition to occur.
- This diagram, like the static model shown in section 5.2.1 uses formal notations for the events and actions on the state transitions. This model reflects a formalized version of the traffic light model. In the requirements analysis process, this

model would have initially been created with natural language specifications for events and actions. It would be progressively formalized and refined throughout the requirements phase, and would eventually evolve into the formalized notations depicted above. This notation is sufficiently rigorous to create an OO design for the traffic light. It is important to note, however, that the overall structure and layout of the dynamic model will not change as the state transition information is formalized. The states generally remain the same. This provides continuity during the formalization of the requirements and enhances user comprehension.

The following sections detail exactly what can appear on the left and right sides of the slash on a state transition. These constraints and the formal notations used to model them are the primary modifications which we have made to the Object Charts [3] and OMT [18] techniques.

5.2.4 Events Which Can Cause State Transitions

The contents on the left hand side of a state transition (preceding the slash) indicates what significant event occurrence in the system causes this state transition. The expressions which are allowed on the left hand and right hand side of the transitions provide the power and expressiveness in the dynamic modeling technique. Some limitations are required, however, in order to allow for smooth transition into an OO design and implementation.

In our dynamic modeling technique, the following expressions may occur on the left hand side of the slash in a state transition:

- External event - an event caused by some agent outside the system. This may be from a device or from some other external stimuli.
- Internal event - an event generated by some object instance which this object has as an attribute. For example, the traffic light object has a timer as an attribute. The traffic light can therefore detect timeout events generated by this timer. Requiring that objects may only detect events generated by objects to which they have direct relations (through explicit attributes) enhances the comprehensibility of the system because the reader will never have to search to find what object generated this event. Events provide the capability for objects to cause some action to occur within the system without requiring that the object generating the event know explicitly who the receiver objects will be. This enhances the reusability of object classes by reducing inter-class coupling.
- Local method invocation - a message being sent to this object to invoke a method may also cause a state transition.
- Guard condition becoming true - guards are boolean conditions which, when satisfied, will cause a state transition. See [10] for a discussion of the use of guards in state charts.

Both events and method invocations used on state transitions may have parameters. These parameters may be used in actions taken as a result of the transition or within guard statements.

5.2.5 Actions Taken on a State Transition

The right hand side of the slash on a state transition specifies any actions which are taken as a result of the transition. Allowable actions include:

- Send a message to itself
- Send a message to another object
- Cause an external or internal event

5.2.6 Actions Upon Entering a State

States may have a continuous or single-shot execution of an action which takes place upon entering a state. This can be signified by a DO: action in the state. This notation is taken directly from OMT [18] [19].

5.2.7 Syntactical conventions

The following examples illustrate the syntactical conventions used for the above event and action specifications in the dynamic model. Events and object names begin with an upper-case letter, while methods begin with a lower-case character.

Events causing a transition:

```
timer.Timeout()
```

Method invocations causing a transition, or as actions on transitions:

```
self.setEWLight(Yellow)
timer.timerActive()
```

Broadcast events used as actions on transitions:

```
Broadcast.Timeout()
```

The last example uses the keyword Broadcast to indicate that this object is causing an internal event with unknown recipients.

Class: Traffic Light

Service: startRedTimer()

Attributes read:	redInterval
Attributes modified:	None
Objects read:	None
Objects modified:	timer
Pre-conditions:	
	timer.timerActive() = False
	redInterval <> 0
Post-conditions:	
	timer.timerInterval() = redInterval
	timer.timerActive() = True
Algorithm:	
	timer.setTimerInterval(redInterval)
	timer.startTimer()

Figure 3: Functional model for startRedTimer method

5.2.8 Functional Model

In our view, the functional view of the object model is simply an annotation to the dynamic model. The dynamic model provides all necessary information about the run-time interaction of objects, but may not provide the level of detail needed in order to actually create an OO design for the system. These details are specified in the functional model:

- Error conditions in the system
- Algorithms for methods used in the dynamic model

The error conditions are specified in the form of pre-conditions which must be true in order for the method to be executed and post-conditions which must be true after the method's execution.

There should be a functional model specified for every method in every object class within the system. Note that the functional model for a method does not show any contextual information about how and when a method will be used. This information is contained in the dynamic model. This lack of contextual information is the main reason why we view the functional model as adding annotative information to the dynamic model rather than being a stand-alone model.

An example functional model for the startRedTimer() method in the traffic light class is shown in Figure 3.

6 Analysis of the Technique

This section provides an analysis of the proposed object based modeling technique.

6.1 Integration of different views

It is important to model elicited requirements from different perspectives, perhaps even based on the domain and problem space, in order to provide multiple views of the problem. Each of the views in our technique (static, dynamic, and functional) is directly related to each other and each provides additional complimentary information to the next. For example, the information provided by the static model may be used to provide a class hierarchy and class layout. The information provided by the dynamic and functional models may be jointly used to determine the contents of different object methods.

In order for the OMT [19] to meet our criteria, it is felt that a tighter coupling between the dynamic and functional models is needed. The functional and dynamic models in OMT provide orthogonal views of the same information. They appear to be redundant with no clear integration or mapping between the two. The dynamic model provides all necessary data and control flow information. We feel the functional model should then be used to further augment the dynamic model.

6.2 User Comprehension

In any form, the representation of requirements should be easy to understand. It may be the case that a function-based approach is better than an object based approach in some domains. However, we believe that end users can, for the most part, interact more easily with object based models because of the association of real world objects with the users' mental models. Rumbaugh addresses the issue of objects viewed as real-world entities [18] [19].

The concept of an object being in a certain initial state and then changing state is intuitive. With minimal training, users can adopt the notion of modeling objects in states. Statechart notations provide specifications of processes, objects, and their interactions that are both unambiguous and concise [8] [10]. Davis states that there is a tendency to resort to "extremely formal notations" in order to ensure less ambiguity, verifiability, completeness, and consistency [8]. He advocates the use of formal notations but points out the need for a 'translation' tool to automatically translate the formal specification into an easily understood version. Our technique may eliminate the need for this translation by making the model easier to understand. It is in fact possible that the object model may actually be created by the end users themselves with the aid of an analyst.

6.3 Minimizing transformations

Traditionally in requirements elicitation there has been no direct correlation between the analysts' synthesis of the stated requirements and the user's view of the model or representation [16]. We have identified the need to provide a mapping between the user model and the model needed for the actual implementation. Continuity is essential during the entire process so that as users see successive refinements to the model, they will continue to recognize and understand it.

The overall format of each of the models remains structurally the same but the formality is increased throughout the requirements process. This provides for the development of a model with multiple views which eventually becomes sufficiently formal to base a design and implementation on. In order to better understand the concept of minimizing transformations, see the discussion of overlays in section 6.4. It is our opinion that the lack of structural transformations accords consistency and coherence to the development process and simplifies the formalization process.

6.4 Evolving Formalism

When requirements are first elicited from users, the resulting representation is usually natural language based. The object based approach presented in this paper allows for a model that may evolve easily from a natural language representation to a more formal representation. In order to attain 'evolving formalism', varying levels of abstraction need to be established. The levels of abstraction and formalism may be dictated by the desires of the development team and the customer. Eventually, this model may be formalized into a VDM like notation [15] [11] if that level of formalism is desired.

We can view the formalization process as simply adding more information and/or detail to the initial model (or structure) via the use of overlays. Each successive overlay of the initial structure offers more detail and formalism which enable the users and analysts to view the system at the appropriate level of formalism respectively.

One issue that needs to be addressed is the preservation of information as the degree of formalism progresses. It is vital to allow for the translation (or evolution) of an abstract view to a more detailed view without the introduction of errors (i.e. correctness preservation). As explained in the previous section, the addition of detail in each successive refinement of the model (in particular the dynamic model) is accomplished via the use of overlays. Although structural consistency and coherence are maintained, the correctness of the information preserved is an issue which must be resolved. The use of overlays is a mechanism for preserving layers of abstraction (and formalisms), serving as a communication vehicle between analysts and end users. However, there is no panacea. This method is as suspect as any other translation mechanism. However, the key to this approach is that the representation is consistent throughout the entire process. The representation is not changed during the development of the specification. Many methods and techniques depend on initial natural language requirements descriptions prior to development of a formal specification (for example, using VDM) as expressed by the contractual model [4].

6.5 Reusable Artifacts

It is difficult to demonstrate that organizing a system around 'objects' is better for users than organizing a system around 'functions'. Although the issue of understandability may be difficult to verify, the issue of maintainability is readily accepted and understood.

Developing and maintaining software systems is expensive. Studies estimate that maintenance costs range from 50 to 85 percent of the total cost of a software product [1] [2]. Fjeldstad and Hamlen estimate that 47% of the time devoted to maintenance activities are for actual enhancement and correction tasks while 62% is dedicated to comprehension activities [13]. These figures suggest that maintenance costs may be significantly reduced if efforts are devoted to increasing the level of comprehension of the system specifications. An object based modeling representation provides modularity by way of information hiding and encapsulation [12]. This modularity supports software reuse, not only for purposes of maintenance but also, for purposes of designing and building similar systems that can employ (or reuse) components of another systems software requirements library. We refer the reader to [12] for further discussion of how object oriented languages promote reusable software.

The three models in the proposed object based technique represent a stand-alone description of an object so that it may be included in a reusable requirements library and design system. Groups of interacting object class models might be incorporated into a framework of objects which cooperate to perform some high level task. Frameworks support reuse by providing 'abstract designs' which may be used by object systems with similar requirements or for "families of related problems" [12].

7 Conclusions

The concept of objects being a more natural and effective method of representing requirements has been discussed in the literature [6]. The method and representation described in this paper was motivated by our reviewing contemporary approaches to enterprise analysis, not by our need to develop yet another object modeling technique. We began this project with the goal of understanding enterprise analysis in the context of using collaboration technologies as a method of elicitation. We quickly came to the conclusion that the representations discussed in the literature had focused on traditional representations and were forcing analysts to translate user needs into abstractions that may be unfamiliar to users.

The goals of this work have not changed. We intend to develop a collaborative requirements elicitation method that allows users to represent their business processes and to interact directly with a set of tools that support representing those processes. We also intend to experiment with this technique to ensure that it is in fact a more viable technique than using traditional representations.

8 Future Research Directions

8.1 Enterprise Analysis

Previous studies in collaborative requirements analysis for business information systems have focused on natural language representation of the elicited requirements. Often times, the translation of the elicited requirements into a more formal representation leads to information loss resulting in a representation which is inconsistent with the content and structure of the originally elicited requirements. One area of research interest is the development of a collaborative technique that would mitigate these problems. We anticipate that the object based representation presented in this paper would be a useful starting point in collaborative requirements development. In particular, we are interested in the effects of eliciting requirements from groups of users by way of objects as opposed to functions. Minimizing information loss as progression from an informal to formal representation ensues has not received significant attention in previous studies. We feel that this coupled with the ability to elicit object based requirements from groups of users would perhaps simplify the elicitation and modeling process. Studies could be conducted to determine whether eliciting requirements in this fashion is actually feasible and whether the loss of information could be minimized in order to increase correctness and user comprehension of the final model.

8.2 Automated Tool Support

The technique outlined in this paper lends itself to automated tool support. We are currently investigating the development of several classes of tools to support the overall requirements modeling process:

- Modeling tools - would allow users and analysts to create the static, dynamic, and functional models described within this paper. These tools would also support the progression from informal to more formalized notations.
- Simulation tools - would allow users to trace the flow of control within the system as well as the interaction between objects.
- Code generation tools - would allow automated code generation from the formalized models.
- Reusability tools - would support the reuse of requirements, designs, and other artifacts of the object system which is being modeled.

References

[1] M. Alford, "SREM at the Age of Eight: The Distributed Computing Design System," *IEEE Computer*, Vol. 18(4), April 1985.

[2] B. W. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

[3] D. Coleman, F. Hayes, and S. Bear, "Introducing Objectcharts or how to use Statecharts in Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 18(1), pp 9-18, January 1992.

[4] B. Cohen, "Justification of Formal Methods for System Specification," *IEE Software and Microsystems*, August 5, 1982.

[5] P. Coad and E. Yourdon, *Object Oriented Analysis*, Yourdon Press, 1990 (First edition).

[6] P. Coad and E. Yourdon, *Object Oriented Analysis*, Yourdon Press, 1991.

[7] A. M. Davis, "The Analysis and Specification of Systems and Software Requirements," in *System and Software Requirements Engineering*, Eds. Richard H. Thayer and Merline Dorfman, IEEE Computer Society Press Tutorial, 1990.

[8] A. M. Davis, *Software Requirements: Analysis and Specification*, Prentice-Hall, 1992.

[9] R. Daniels, A. Dennis, G. Hayes, J. Nunamaker Jr. and J. Valacich, "Enterprise Analyzer: Electronic support for group requirements elicitation," University of Arizona, 1991.

[10] D. Harel, "On Visual Formalisms," *Communications of the ACM*, Vol. 31(5), pp. 514-530, May 1988.

[11] F. Hayes and D. Coleman, "Coherent Models for Object Oriented Analysis," in *Proceedings of OOPSLA '91*, pp. 171-183, 1991.

[12] R.E. Johnson and B. Foote, "Designing Reusable Classes," *JOOP*, Vol. 1(2), pp. 22-35, June-July 1988.

[13] R.K. Fjelstad and W.T. Hamlen, "Application Program Maintenance Study: Report to Our Respondents," in *Proceedings GUIDE 48*, Philadelphia, PA, 1979 (*Tutorial on Software Maintenance.* G. Parikh and N. Zvegintozov, editors, IEEE Computer Society, April 1983, IEEE Order No. EM453).

[14] J.C.S.P. Leite and P.A. Freeman, "Requirements Validation Through Viewpoint Resolution," *IEEE Transactions on Software Engineering*, Vol. SE-17(12), December 1991.

[15] B. Meyer, "On Formalism in Specifications," *IEEE Software*, pp. 6-26, January 1985.

[16] D.T. Ross, "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Transactions on Software Engineering*, Vol. SE-3(1), January 1977.

[17] D.T. Ross, "Applications and Extensions of SADT," *IEEE Computer*, Vol. 18(4), April 1985.

[18] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object Oriented Modeling and Design,* Prentice Hall, 1991.

[19] J. Rumbaugh, "The evolution of bugs and systems," *JOOP*, pp. 48-52, November-December 1991.

[20] S. Shlaer and S.J. Mellor, *Object Lifecycles Modeling the World in States,* Yourdon Press Computing Series, Prentice-Hall, 1992.

A Modeling Example

This appendix presents a simple information systems example using the modeling technique described in this paper. The object class presented is a production order which might be used within a manufacturing enterprise. The production order maintains information about the current state of an order of parts to be manufactured.

The production order is normally used in the following way:

- The order object is created and opened when an order is received from a customer.
- The order is then entered into the manufacturing enterprise's production scheduling process.
- Once the order has been scheduled, it may enter the actual production process.
- Eventually, the order is completed, then closed out.

Note that this class, which would normally be used in the context of many other classes representing the various artifacts of the manufacturing enterprise, may be viewed in isolation with its full semantics in evidence. This is largely due to the fact that this notation presents each other object class which the production order class has a relationship to as an explicit attribute of the object class rather than relying on a graphical notation (such as arcs between the boxes representing different classes) to indicate all relationships. Since each class can be viewed in isolation, it makes the notation more amenable to requirements reuse and reusability tool support.

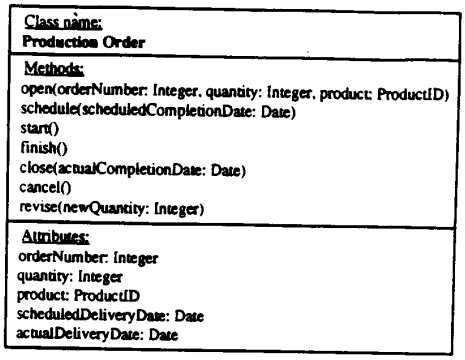

Figure 4: Static model for Production Order class

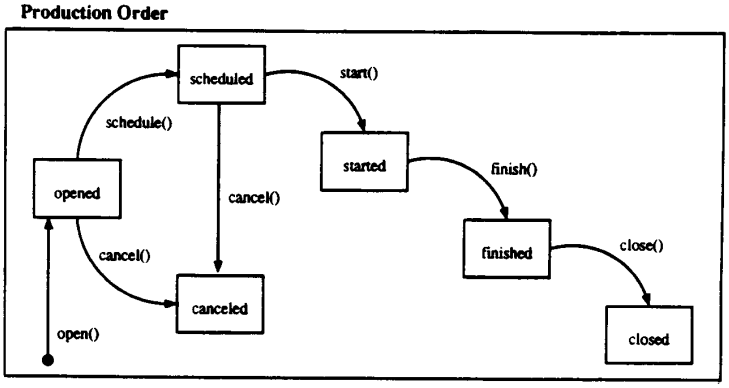

Figure 5: Dynamic model for Production Order class

Class: Production Order

Service: open(orderNumber: Integer, quantity: Integer, product: ProductID)

Attributes read:	None
Attributes modified:	orderNumber, quantity, product
Objects read:	None
Objects modified:	self
Pre-conditions:	None
Post-conditions:	
	self.orderNumber = orderNumber
	self.product = product
	self.quantity = quantity
Algorithm:	
	self.orderNumber = orderNumber
	self.product = product
	self.quantity = quantity

Figure 6: Functional model for open method