A Comparison of Real-time Object-Oriented Modeling Methods ROOM and OCTOPUS

Guilan Dai and Baowen Xu

Department of Computer Science & Engineering
Southeast University
Nanjing 210096, P. R. China
{adalab, bwxu} @seu.edu.cn

Abstract

ROOM and OCTOPUS are two real-time object-oriented modeling methods used commonly. The paper analyses and compares ROOM and OCTOPUS based on details where philosophies of both methods are in sharper contrast, such as model representation, class and object and inheritance, concurrency, and state charts.

Key Words

Formal method, informal method, semiformal method, modeling, modeling languages, modeling methodology, OCTOPUS, object-orientation, object-oriented modeling method, real-time systems, ROOM.

1 Introduction

Software methodology plays an important in the development and maintenance of large and complex real-time systems. An important category of software methodologies is based on building models. A modeling method needs the support of modeling language and tool. Modeling language is the basis of a method. It defines what can be modeled and how it is specified and decided on the performance of modeling language [1,2,3,4]. Though a modeling method does not correspond to only a modeling language, a method combining with the specific language can lead to better effect. Therefore, the modeling methods discussed in this paper are consistent with their underlying languages.

Object-oriented modeling technology helps in the development of systems in a way that naturally maps to the inherent nature of the systems being built, providing benefits such as reusability, extendibility and robustness[1,2,4]. The two commonly used real-time object-oriented modeling methods are ROOM (Real-time Object-oriented Modeling) and OCTOPUS. They attempt to introduce object-oriented technology into real-time systems and aim at coping with the characteristic problems of real-time domains, such as concurrency, synchrony, communication, interrupt, hardware port and end-to-end response times. Both technologies are clearly superior to their competitors and predecessors (such as OMT, Fusion, OOSE)[2] in terms of ability of expression, maturity and their resulting software is robust, flexible, and reliable.

We analyze and compare ROOM and OCTOPUS so that we can borrow from their good ideas. At the same time, this can not only provide theoretical basis for the further development of modeling language, but also guide developers to select appropriate modeling methods.

Rather than comprehensive, this paper focuses on details where the philosophies of both methods are in sharpest contrast, for example, model representation, concurrency, state-chart, class and object and inheritance. Other elements of modeling methods such as the linkage between high-level abstract model and implementation, implementation language and so forth are not discussed in details here, since they have little difference in the two methods.

2 Model Representation

The OMT and the Fusion methods are fundamental for the development of OCTOPUS, which inherits the separation of structural, functional and dynamic aspects from OMT and combines it with the basic separation between the analysis phase and the design phase borrowed from Fusion. OCTOPUS uses the object model, functional model and dynamic model to describe systems and subsystems. The three models

complement each other. The relationship between the models is expressed by using the same names of the same components that may appear in different models, and by associating the components of the functional and dynamic models with the classes of the object model. Each model uses an appropriate set of notation [2].

ROOM represents its language model in graphical form too. The basic element of ROOM is actor. In general, a ROOM actor denotes an active object that has a clearly defined purpose. In this context, the term active means that an actor has its own execution thread and can, therefore, operate concurrently with other active objects in its domain. An actor consists of port, behavior, variable, function and SAP (service access points). Each actor has one or many ports through which it can communicate with other actors by exchanging messages. At the same time, each actor can optionally have a behavior component. This component can initiate activities by sending messages as well as response to external message. Leaf actors only include behavior [1].

If we study the two language models in details, we can conclude that:

- *OCTOPUS represents three models in three different sets of notations. This leads easily to semantics discontinuity and scope discontinuity. ROOM uses a single integrated set of modeling concepts to build a given model, and eliminates semantics discontinuity and scope discontinuity of the development process, and increases reliability of systems.*
- *OCTOPUS uses three different models to describe the system. This helps to early detect errors or inconsistency. ROOM is supplemented with treating requirement models and design models as programs written in a very high-level modeling language. In a full-fledged, executable model, all the elements of the language, whether those used to represent high-level schematic details as well as those used to represent low-level details, or those used to represent different kinds of details such as structure versus behavior, are compilable, executable. In addition, ROOM represents the system in an integrated graphical form so that it can clearly convey semantics. Therefore, the models of ROOM are highly observable.*
- *OCTOPUS explicitly separates the analysis model and the design model. This easily causes phase discontinuity. ROOM uses a single set of modeling concepts throughout requirement definition, design and implementation. It is possible that large portions of a design model are also portions of the associated requirement model. This can not only eliminate phase discontinuity, but also simplify the modeling process.*
- *The language model of OCTOPUS is informal. Since natural languages are inherently ambiguous, they prevent from communicating between developers and users. However, ROOM is formal, which can not only override ambiguousness, but also make it possible to automatically generate software.*

3 Object, Class and Inheritance

Object-oriented technology has become the buzzword of the decade. Behind the obvious enthusiasm of the software community, there lies the fact that it simplifies the design and construction of software system. The most basic way it simplifies design and construction is by reuse.

OCTOPUS views a subsystem as composition of objects that are specified only through class. The properties of a super-type are inherited by its subtypes. Inheritance allows subclass to reuse the interface and even the implementation of the super-class, and therefore the objects of a subtype also belong to the super-type.

In contrast, ROOM defines object as logical machine called actor. An actor may be implemented as software component, as digital hardware, analog hardware, or manual procedure. In order to increase the expressiveness, ROOM provides a number of advanced modeling features such as layer, multiple containment and replication, in addition to supporting conventional object paradigm.

ROOM defines all objects by classes, even the highest-level objects are defined by actor classes. A run-time system is an instance of an actor class. This is different from OCTOPUS in that the system is defined as object but it itself is not a class. ROOM has three kinds of classes. These are actor class, protocol class and data class. Each kind of class specifies different objects and has different hierarchical inheritance mechanisms. In ROOM inheritance is seen as abstract mechanisms which help to cope with complexity. The high-level structure and high-level behavior of actor class may be inherited. This increases granularity of reuse. Both actor class and protocol class have more

flexible inheritance mechanisms through which all attributes inherited from the parent class are but eliminated and overridden. Therefore, in ROOM, the subclasses can be used in the places where the super-class may not be used.

ROOM is superior to OCTOPUS in that ROOM defines system as class but OCTOPUS defines system as object. The main reasons are:

- *Replacing a group of objects with a simple class, we need not describe every object. This can not only save effort but also save memory, and increase maintainability of the systems.*
- *By specifying system as class, we may define system variables in different environment and allow introducing smaller system into larger system. This increases granularity of reuse and the maintainability of the systems.*

In addition, ROOM has more flexible inheritance mechanisms by which may increase extendibility, flexibility, reliability of the systems.

4 State Charts

Reaction of the system on an event is affected by the state in which the system may be. This is another feature of real-time reactive systems. State-machine technology is the most direct and common ways of modeling behavior. The use of state charts reduces 'state explosion'[5].

OCTOPUS uses Statecharts to describe state transitions of the system. Statecharts extends Harel's state-charts with three mechanisms, i.e., hierarchy, concurrency, and communication. However, The over-simplified and over-idealized design considerations embodied in Statecharts are problematic for modeling practical real-time software:

- *The communication model of Statecharts assumes a fully reliable and instantaneous broadcast;*
- *Statechart execution model assumes that transitions from one state to the next state are instantaneous.*

These two design considerations are key to some of the most powerful concepts in Statecharts, such as the decomposition of a state into "and" substates since the assumption that a system always must be in some state. Though similar concepts increase expressiveness of the model, specifications that are modeled using these concepts may be difficult to implement. For example, in some cases, distributed systems are particularly sensitive to the defects in communications and the defects may decrease reliability of modeled systems [1, 2].

ROOM uses ROOMchart to describe state transitions of a system. ROOM extends and modifies Statecharts to make full use of the essential features of object-oriented paradigm, to eliminate phase discontinuity between requirement model and design model, and to make it possible to automatically generate efficient software implementation from high-level behavior specification. This probably results in some loss because of the elimination of 'and' states. Fortunately, this loss can be supplemented, to a degree, with combining some of the other ROOMchart features.

We'll illustrate this method using a simplified example of a digital Beep Pager (BP). A Beep Pager provides four functions. These are a timekeeping function measuring the time progress by a series of clock ticks, a call function (an alarm that is triggered when a call signal reaches), a time-display function displaying the current time in either 24-Hmode or 12-Hmode), and a PC-note recording phone call numbers.

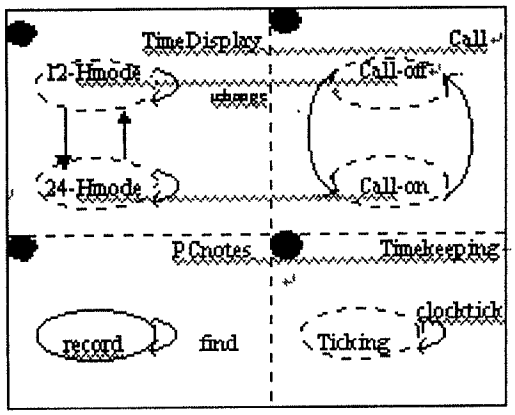

Figure 1 Statechart of a Beep Pager

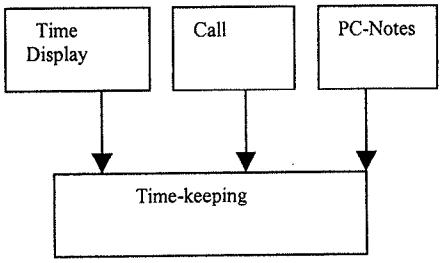

Figure 2 ROOMchart of a Beep Pager

In OCTOPUS Statechart formalism, the state of this system may be modeled by a composite state consisting of the time-displaying state, the calling state,

the phone note state and the time-keeping state, as shown in Figure 1. Each of these four high-level states also contains a child state machine that describes the behavior of the particular function.

In contrast, in ROOMchart model, each of the orthogonal states of the Beep Pager can be modeled as a distinct object. This is possible because the mechanisms required to implement the time-displaying function are distinct from the mechanisms required to implement the call function as well as the mechanisms required to implement the phone call number. We should use concurrent objects to model these three functions (since the orthogonal states are concurrent), even though the functions share some common function (such as the time-keeping function). As shown in Figure 2, each of the orthogonal states may be represented using an actor. For sharing function, layer mechanism is effective. The state machines within the individual actors would be similar to the corresponding ones in Figure 1.

The principal difference between ROOMchart and Statechart models is the communication forms. In ROOMchart model, all communication is explicit. This requires more effort from the modeler, since the communication protocols and message formats must be formally defined. On the other hand, by making the interaction between concurrent entities explicit, unnecessary couplings between orthogonal states may be avoided. Such a coupling can occur when an action associated with one transition unexpectedly generates an event that triggers a transition in another orthogonal state. The likelihood of an unnecessary coupling is increased by the fact that, in Statechart, an event is not necessarily generated as an explicit communication, but could be the side effect of some other operation.

From the above we can see that, in some sense, ROOMchart is the modifications and extensions to Statecharts.

5 Concurrency

Concurrency is inherent in almost real-time systems. Concurrency is a powerful mechanism in modeling real-time systems. Meanwhile, it creates new problems regarding the consistency of data. As the level of concurrency increases, synchronization problems increase as well. The basic difficulty of applying object-oriented methods to the development of real-time systems is how to combine the concept of concurrency with object.

OCTOPUS uses the implicit concurrency model in the analysis phase. In the design phase, the concurrency model is gradually made more explicit. OCTOPUS maps directly objects to processes in operating system. In order to simplify synchronization design between processes, OCTOPUS group a conceptual union of objects together with synchronous interactions between them into an object group. In addition, shared objects are reduced as few as possible in the design process of object group. Each object group may be mapped to a single process, called the object group process. Either a call for a member function of the object is executed as a part of a given process, or a direct member data access is performed. The part of an event thread that is covered by an object group specifies the hierarchy of member function calls the execution thread of the object group process has to follow.

ROOM uses the implicit concurrency model, and its basic scheduling unit is actor. ROOM assumes the underlying environment can provide concurrency mechanisms and the kernel can support concurrent unit or thread. The most straightforward way to implement an actor is to match the basic scheduling unit with a thread of the host environment. An alternative is to encapsulate the entire complexity of ROOM actors comprising the design into a single scheduling unit and then provide an internal customized multithread environment within that unit. This kernel-within-a-kernel approach usually requires more work, but it provides an opportunity for much greater throughput. We can come into conclusions that:

- *In ROOM, an actor is mapped to a thread. This increases throughput and granularity of concurrency, since proprietary resources that belong to a thread are few, and states that describe a thread are less than a process. Memory resources needed in creating threads are far away less than creating processes. Therefore, the overhead of switching and communicating between threads is less than that between processes.*
- *In OCTOPUS, we must manually map object to process, especially for the production of object group. In ROOM, however, these work can be automatically done by ObjecTime tool sets. This may not only save effort but also eliminate factitious errors so that it can increase reliability.*

6 Conclusion

Both OCTOPUS and ROOM are commonly used object-oriented real-time modeling methods. They primarily intend for soft real-time systems. ROOM is more suitable for distributed systems, whereas OCTOPUS is more suitable for embedded systems. However, OCTOPUS can be applied in distributed systems based on a distributed operating system supporting location-transparent message passing between the objects.

The paper analyses and compares ROOM and OCTOPUS based on details where philosophies of both languages are in sharper contrast, such as model representation, class (including object and inheritance), concurrency, and state chart. It may be concluded that, in some sense, ROOM is superior to OCTOPUS, particularly, in extensibility, reliability, and granularity of concurrency and reuse.

OCTOPUS manually controls event scheduling by defining event significance table, in this aspect ROOM is deficient.

Acknowledgments

The authors wish to thank Zhenqiang Chen, Yuan Liu and Shengzhi Li for their valuable comments.

References

[1] Selic, B. and Gullekon, G., Real-Time Object-Oriented Modeling, Katherine Schowdter, John Wiley & Sons, Inc., 1994.

[2] Maher, A. and Jula, K., Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion, Prentice Hall, Inc., 1996.

[3] Claus, L. and Thomas, L., Formal Development of Reactive Systems, Springer-Verlag, 1995.

[4] Yourdon, E. and Argila, C., Case Studies in Object Oriented Analysis & Design, Prentice Hall, Inc., 1996.

[5] Harel, D., "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Program July 8, 1987.