Pakistan Journal of Applied Sciences 2(9): 922-924, 2002

Object Oriented Modeling of Multimedia Applications

Pantano Rokou Franca Maria
Department of Cultural Technology and Communication
University of Aegean, Greece

Abstract: The field of multimedia software engineering is still unexplored and in a quite virgin state. Multimedia engineering is an emerging area combining software engineering, multimedia computing, visual languages and visualization. An analysis of how visual modeling of structure and dynamic behavior of a multimedia application differs from modeling conventional software yields the need of specific modeling procedures and languages. Therefore, multimedia application process has been truncated to implement-and-test methods. There are no specific multimedia development processes nor graphical notations adapted to them. In order to fill this gap we introduce a hybrid of OOM (Object Oriented Modeling)adapted to the specific needs of multimedia applications. Here, will be presented an innovative approach of multimedia software engineering from two different, yet complementary, perspectives: 1. The application of multimedia computing and visualization languages to the practice of software engineering and 2. The application of software engineering principles properly tailored to the development of multimedia applications and systems.

Key Words: Multimedia Software Engineering, Object Oriented Modeling, Unified Modeling Language (UML), UML Diagrams

Introduction

Multimedia applications are interactive software systems in which objects of diverse discrete and continuous media types are combined and presented together. Multimedia application is defined an application that integrates two or more media types and shows time and/ or action related dynamic behavior (Selic *et al.*,1994). An analysis of how visual modeling of structure and dynamic behavior of a multimedia application differs from modeling conventional software yields that aspects of the graphical user interface and time-dynamic behavior ought to be integral parts of a coherent multimedia application model. In this sense, we extend the model-view-controller paradigm towards multimedia.

Materials and Methods

Framework: In the beginning of the 90s, the great diversity of proposed object-oriented design notations and methods caused a lot of problems in the software development field. In particular, due to a lacking standardized approach, object-oriented models were hardly understandable by designers, who were not an expert of the used, specific notation. Therefore, existing object-oriented models were hardly reusable in projects where another notation has been chosen, and thus one of the main advantages of an object-oriented approach were extremely cut down. This created a strong motivation, particularly in industry, to standardize on a single object-oriented notation. In response, the Object Management Group (OMG) defined the Unified Modeling Language (UML), which it adopted in 1997 as its standard notation for object-oriented analysis and design.

While the UML was clearly successful in unifying the different graphical notations, it has been argued that it was less successful in providing a shared definition of the pragmatics and semantics behind the underlying concepts. In particular, the following issues have been noted:

- lack of clear guidelines on which aspects of a system are to be modeled by which diagram types,
- lack of heuristic, pragmatic guidelines on how these diagram types are to be actually used for the various aspects,
- lack of precise rules on how to transform a UML description into fragments of programming language code (e.g., Java) or GUI builders, and, last but not least,
- lack of a precise meaning (semantics) of UML-based descriptions.

The Benefits of UML:

1. Your software system is professionally designed and documented before it is coded. You will know exactly what you are getting, in advance.
2. Since system design comes first, reusable code is easily spotted and coded with the highest efficiency. You will have lower development costs.
3. Logic 'holes' can be spotted in the design drawings. Your software will behave as you expect it to. There are fewer surprises.
4. The overall system design will dictate the way the software is developed. The right decisions are made before you are married to poorly written code. Again, your overall costs will be less.
5. UML lets us see the big picture. We can develop more memory and processor efficient systems.
6. When we come back to make modifications to your system, it is much easier to work on a system that has UML documentation. Much less 'relearning' takes place. Your system maintenance costs will be lower.
7. If you should find the need to work with another

developer, the UML diagrams will allow them to get up to speed quickly in your custom system. Think of it as a schematic to a radio. How could a tech fix it without it?

8. If we need to communicate with outside contractorsor even your own programmers, it is much more efficient.

Types of UML Diagrams: Each UML diagram is designed to let developers and customers view a software system fromadifferent perspective and in varying degrees of abstraction. UML diagrams commonlycreated in visual modeling tools such as GDPro include (UML, 1997):

- Use Case Diagram displays the relationship among actors and use cases.
- Class Diagram models class structure and contents using design elements such as classes, packages and objects. It also displays relationships such as containment, inheritance, associations and others.
- State Diagram displays the sequences of states that an object of an interaction goes through during its life in response to received stimuli, together with its responses and actions.
- Sequence Diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension (time) and horizontal dimension (different objects).
- Collaboration Diagram displays an interaction organized around the objects and their links to one another. Numbers are used to show the sequence of messages.
- Activity Diagram displays a special state diagram where most of the states are action states and most of the transitions are triggered by completion of the actions in the source states. This diagram focuses on flows driven by internal processing.
- Component Diagram displays the high level packaged structure of the code itself. Dependencies among components are shown, including source code components, binary code components, and executable components. Some components exist at compile time, at link time, at run times well as at more than one time.
- Deployment Diagram displays the configuration of run-time processing elements and the software components, processes, and objects that live on them. Software component instances represent run-time manifestations of code units.

Modeling Structure: The structure of a system identifies the entities that are to be modeled and the relationships between them (e.g., communication relationships, containment relationships). UML provides two fundamental complementary diagram types for capturing the logical structure of systems: class diagrams and collaboration diagrams.

Class diagrams capture universal relationships among classes-those relationships that exist among instances of the classes in all contexts. Collaboration diagrams capture relationships that exist only within a particular context-a pattern of usage for a particular purpose that is not inherent in the class itself. Collaboration diagrams therefore include a distinction between the usage of different instances of the same class, a distinction captured in the concept of role. In the modeling approach described here, there is a strong emphasis on using UML collaboration diagrams to -explicitly represent the interconnections between architectural entities. Typically, the complete specification of the structure of a complex real-time system is obtained through a combination of class and collaboration diagrams. Specifically, we define three principal constructs for modeling structure: Capsules correspond to the ROOM (RealTime Object Oriented Modelling) concept of actors. They are complex, physical, possibly distributed architectural objects that interact with their surroundings through one or more signal-based 1 boundary objects 2 called ports. A port is a physical part of the implementation of a capsule that mediates the interaction of the capsule with the outside world-it is an object that implements a specific interface. Each port of a capsule plays a particular role in a collaboration that the capsule has with other objects. To capture the complex semantics of these interactions, ports are associated with a protocol that defines the valid flow of information (signals) between connected ports of capsules. In a sense, a protocol captures the contractual obligations that exist between capsules(ROOM,1994).

There are unique challenges faced in real-time software development. Every real-time software developer recognizes that the requirement for latency, throughput, reliability, and availability are far more stringent than for general purpose, or business software. For real-time system developers, understanding the impact of design decisions and effectively communicating functionality can be a daunting task. An overriding concern is the architecture of the software. This refers to the essential structural and behavioral framework on which all other aspects of the system depend.

To facilitate the design of good architectures, it is extremely useful to capture the proven architectural design patterns of the domain as first-class modeling constructs. UML for Real-Time combines UML, role modeling and ROOM concepts to deliver a complete solution for modeling complex real-time systems. UML, role modeling and ROOM are briefly described below.

UML is a general-purpose modeling language for specifying, visualizing, constructing and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. UML has a strong set of general purpose modeling language concepts applicable across domains. Role modeling captures the structural communication patterns between software components. In UML 1.1, collaboration diagrams, which form the basis of structural design patterns, became first class modeling entities(UML,1997). ObjecTime Limited (ObjecTime) was a member of the UML 1.1 definition team and contributed the role modeling capabilities of ROOM to the UML standard.

ROOM is a visual modeling language with formal semantics, developed by ObjecTime. It is optimized for specifying, visualizing, documenting, and automating the construction of complex, event-driven, and potentially distributed real-time systems. It incorporates the role modeling concepts discussed in this document that enable the capture of architectural design patterns UML for Real-Time is a complete real-time modeling standard, co-developed by ObjecTime an Rational Corporation, that combines UML 1.1 modeling concepts, and special modeling constructs and formalisms originally implemented in ObjecTime Developer and defined in the ROOM language(Selic and Rumbaugh). ObjecTime Developer is a software automation tool that provides model execution capabilities, and automatically generates complete code for complex real-time applications from these modeling constructs. UML for Real-Time supports all of the automation capabilities that are available in ObjecTime Developer today. Modeling Perspectives as software systems become increasingly more complex, software architecture, and techniques for capturing it, become increasingly more important. In addition the architecture of these systems

References

Selic, B., G. Gullekson, and P.Ward, 1994 "Real-Time 0Object-Oriented Modeling," John Wiley & Sons, New York, NY.

"UML Semantics," version 1.1 (1 September 1997), The Object Management Group, doc. no. ad/97-08-04.

"UML Notation Guide," version 1.1 (1 September 1997), The Object Management Group, doc. no.ad/97-08-05.

"UML (1 September 1997) Extension for Objectory Process for Software Engineering" version 1.1 , The Object Management Group, doc. no. ad/97-08-06.

Real-time architectural modeling whitepaper by B. Selic and J. Rumbaugh: <http://www.objectime.com>

OMG's UML 1.1 standard <http://www.rational.com>

ROOM, John Wiley and Sons,1994. NY.

Pantano Rokou F. 2002 Interactive Multimedia: Technologies, design and implementation, Ed. Kritiki, Athens.

Franca Garzotto, Paolo Paolini and Daniel Schwabe, (Jan. 1993) HDM---a model-based approach to hypertext application design <http://cq-pan.cqu.edu.au/Reading/papers/p1-garzotto.pdf>, ACM Transactions on Information Systems Vol. 11, No. 1, Pages 1-26.

Daniel Schwabe, Gustavo Rossi Simone and D. J. Barbosa, Systematic hypermedia application design with OOHDM <http://cq-pan.cqu.edu.au/Reading/papers/p116-schwabe.pdf> in: Proceedings of the the seventh ACM conference on Hypertext '96, pages 116-128.

✓ Jim Conallen,October, 1999 Modelling Web Application Architecture with UML <http://cq-pan.cqu.edu.au/Reading/papers/p63-conallen.pdf>, Communications of the ACM, 42(10), 63.