Modeling Scenarios in Scenario-Based Multithreading for Real-Time Object-Oriented Modeling[*]

Saehwa Kim
School of EE and CS
Seoul National University
Seoul 151-742, Korea
ksaehwa@redwood.snu.ac.kr

Michael Buettner
School of EE and CS
Seoul National University
Seoul 151-742, Korea
buettner@redwood.snu.ac.kr

Mark Hermeling
IBM Software Group
IBM Singapore Pte Ltd
Singapore 486072, Singapore.
hermelin@sg.ibm.com

Seongsoo Hong
School of EE and CS
Seoul National University
Seoul 151-742, Korea
sshong@redwood.snu.ac.kr

Abstract – *The paper presents our scenario modeling framework in scenario-based multithreading for object-oriented real-time modeling. Our modeling toolset allows scenario modeling by transforming a given UML 2.0 model to a scenario model. Our toolset provides for extended notion of scenarios that supports (1) concatenated scenarios, (2) mutually non-concurrent scenarios, (3) ports or structured classes with multiple cardinality, (4) message buffering, and (5) dynamic structures. Scenario models are intermediate models acting as bridges to gradually lead to the desired implementation. Consequently, our scenario modeling framework not only helps designers to more easily understand the model but also enable the identification of a feasible task set in a systematic way.*

Keywords: Real-time object-oriented modeling, UML 2.0, design methodology, embedded software, real-time systems, model transformation, scenario-based modeling.

1 Introduction

Embedded systems become extremely complex and sophisticate due to the widen application domain as well as the increased demand for safety, reliability, and performance requirements. As a result, it becomes inevitable for embedded system designers to rely on systematic software development methods and tools for system design, synthesis, and tuning at various stages of system development.

Object-oriented modeling tools for embedded systems allow developers to take advantage of not only efficient tool-based development but also the benefits of object-oriented technology such as encapsulation, polymorphism, and inheritance. However, current modeling tools for object-oriented modeling, such as IBM Rational RoseRT [4], ARTiSAN Real-Time Studio [1], I-Logix Rhapsody [5], and IAR visualSTATE [3], lack in providing predictable and verifiable timing behavior and the automatically generated code is not always acceptable. For real-time embedded systems it is of the utmost importance to generate executables that can guarantee timing requirements with limited resources. Currently, designers must map design-level objects to implementation-level tasks in an ad-hoc manner. Because task derivation has a significant effect on real-time schedulability, tuning the system with this approach is often extremely tedious and time-consuming.

In our previous work [7][8][9], we have proposed a systematic, schedulability-aware method of mapping object-oriented real-time models to multithreaded implementations. This is based on the notion of scenarios. A scenario is a sequence of actions that is triggered by an external input event, possibly leading to an output event [7]. In [8], we presented a multithreaded implementation architecture based on mapping scenarios to threads. This is contrary to the architecture found in current modeling tools that map a group of objects to a thread. In [9], we presented a complete tool set implementation of the scenario-based multithreading architecture for UML models as well as experimental results that validate this implementation. Our implementation exploits an established UML modeling tool, RoseRT, by designing a scenario-based run-time system that maintains backwards compatibility with the RoseRT run-time system.

In this paper, we present our scenario modeling framework in our scenario-based toolset for object-oriented real-time modeling. Enabled by our scenario-based multithreading, our modeling toolset allows scenario modeling by transforming a given UML 2.0 model to a scenario model. This intermediate scenario model not only helps designers to more easily understand the model but also enable the identification of a feasible task set in a systematic way, acting as bridges to gradually lead to the desired implementation. Designers can also refine the intermediate scenario model. Specifically, each scenario can be associated with timing constraints such as period and deadline.

[*] The work reported in this paper is supported in part by MOST (Ministry of Science and Technology) and MOCIE (Ministry of Commerce, Industry and Energy) under the SystemIC 2010.

Motivated by our case study of real-world models such as PBX systems [6], we have extended the notion of scenarios found in our original method. Specifically, we extend the notion of scenarios so that it can support (1) concatenated scenarios, (2) mutually non-concurrent scenarios, (3) ports or structured classes with multiple cardinality, (4) message buffering, and (5) dynamic structures. We extended our tool to support such scenario modeling functionality. Scenario modeling enables the systematic identification of a feasible task set since scenario models act as bridges to gradually lead to the desired implementation. Our tool also allows designers to browse visualized scenario models.

The notion of using scenario models as intermediate models for better output generation was inspired by model transformation technique. There also has been research activities focused on model transformation in the UML framework that provide various model transformation techniques where transformations are specified in UML [2][10]. These techniques can be integrated with our approach to derive intermediate models of scenarios and logical/physical threads.

The remainder of the paper is organized as follows. Section 2 summarizes UML 2.0 that we chose as our real-time object-oriented modeling language. Section 3 presents an overview of our scenario-based multithreading, comparing it with traditional structured-class-based multithreading. Section 4 explains our scenario modeling environments with extended notion of scenarios. Section 5 presents how our tool supports the visualization of scenario models. The final section concludes the paper.

2 Overview of UML 2.0

UML 2.0 [13] is a general purpose modeling language developed by the OMG [11], and contains corrections and new content based on user feedback on the UML 1.x modeling language. One of the important additions in UML 2.0 is the concept of structured classes. This concept makes it possible to define the run-time structure of a class as the composition of multiple structured classes connected together. It has been developed to properly represent complex, event-driven, potentially distributed real-time and embedded systems. The additions to UML 2.0 are inspired by ROOM [12]; another object-oriented modeling technique for real-time systems.

The basic element of model construction in UML 2.0 is a structured class. A structured class represents an object within the system that communicates with other structured classes exclusively through interfaces called ports. Structured classes connected together define the run-time structure and communication channels of an application. A finite state machine, represented by a state diagram, represents the behavior of a structured class. Receiving messages via ports causes the state machine to make transitions, executing the logic contained in the structured class.

For our toolset, we exploited IBM Rational Software Rose RealTime (RoseRT), which is a modeling tool that allows users to design object-oriented real-time systems using UML 2.0 and generate complete executables directly from these designs.

3 Scenario-based multithreading of UML 2.0 models

In structured-class-based multithreading the entity which can be manipulated is a message. It is possible to map the incoming messages of a structured class to a certain thread, and possible to map a single message to a thread or assign it a priority. But in most cases the designer does not conceptualize in terms of individual messages, but in terms of message chains. It is more natural that an entire message chain would be mapped to a thread, or timing metrics would be considered from the start of a chain to the end.

Also, it is not possible in structured-class-based multithreading for a message coming into a structured class to be processed on different threads in different situations. This imposes great limitations on the designer. Our scenario based multithreading allows the user to define priority and thread mapping for a complete message chain instead of individual messages. Structured classes will execute on different threads at different times depending on which scenario message sequence it is participating in at the moment. This not only is more akin to the way a designer would conceptualize a problem, but it also allows much greater flexibility in model design.

Moreover, structured-class-based multithreading may degrade the performance of real-time systems by extending blocking time unnecessarily. We have shown performance evaluation results for this in [6][9].

4 Scenario modeling

We have extended the notion of scenarios to support (1) concatenated scenarios, (2) mutually non-concurrent scenarios, (3) ports or structured classes with multiple cardinality, (4) message buffering, and (5) dynamic structure. In this section, we explain how we have extended our tool to support such scenario modeling functionality.

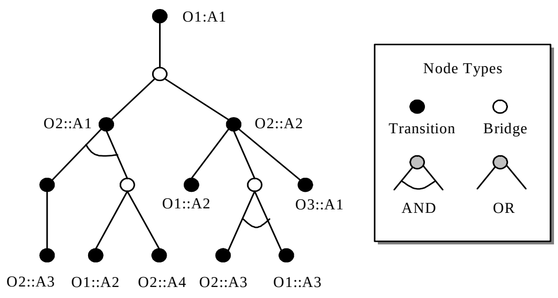


Figure 1. An example AND-OR transition tree

4.1 Concatenated scenarios

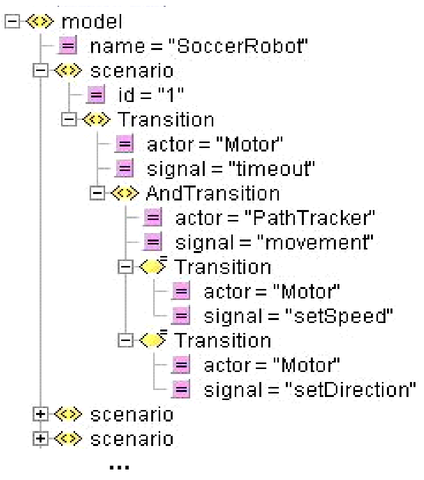
In some cases, the basic concept of a scenario that begins with an external message and continues until the end of the message chain may not be flexible enough to meet the needs of designers. There may be situations where the designer's concept of what should be a scenario extends beyond the end of a message chain.

For example, in a soccer robot system, designers may wish to model as a scenario the execution chain initiated by a timeout event in a motor structured class, and flowing through the transitions associated with adjusting the speed and direction of the motor, and processing an acknowledge message sent back from the motor on a hardware communication port. Our tool would identify two scenarios that make up this chain; one beginning at the timeout in the motor acknowledge and continuing until the new speed and direction information is sent to the motor structured class, and one beginning when the motor structured class receives the acknowledge message.

In order to model the entire chain as a single scenario, there must be a mechanism for concatenating these two message chains. Our tool allows designers to indicate any number of scenarios which should be concatenated, and designers can then manipulate the entire chain as one scenario. With this, timing constraints are assigned to the entire concatenated chain, and the component scenarios are assumed to be mutually non-concurrent.

4.2 Mutually non-concurrent scenarios

A set of mutually non-concurrent scenarios is a group of scenarios which will never execute concurrently. When designers designates a set of scenarios as mutually non-concurrent, this indicates that all the scenarios in the set may be mapped to the same thread and no member of the set need preempt any other member of the set. By thus grouping scenarios our model transformer is able to limit the number of threads while still providing the necessary level of concurrency, and so reduces context switch overhead and static memory requirements.



```
⊟ model
  name = "SoccerRobot"
  ⊟ scenario
    id = "1"
    ⊟ Transition
      actor = "Motor"
      signal = "timeout"
      ⊟ AndTransition
        actor = "PathTracker"
        signal = "movement"
        ⊟ Transition
          actor = "Motor"
          signal = "setSpeed"
        ⊟ Transition
          actor = "Motor"
          signal = "setDirection"
  ⊞ scenario
  ⊞ scenario
    ...
```

Figure 2. An example XML document for scenario visualization

4.3 Multiple cardinality support

Because structured classes and ports may be replicated in UML 2.0, it is necessary to differentiate between the multiple instances of a replicated structured class or port when identifying scenario initiation signals. Currently in RoseRT it is not possible to map messages from a replicated port to different threads depending on the replication index.

Our tool considers each replication index as separate, which enables the mapping to different threads of scenarios that are initiated by the same signal sent from different replication indices. With this, it is possible to have concurrency and a priority hierarchy between scenarios that are started from different replication indices of the same port or structured class.

4.4 Scenarios from buffered messages

Most modeling tools allow for the deferring of messages to a later time and then, when some condition is met, those messages are recalled. Example conditions that trigger the recall of messages are receiving some special message that flushes deferred messages, such as a timeout-event, or the deference of a desired number of messages. When such a condition occurs, the deferred messages are recalled and the actual event processing is started. In such a case, designers may not want to model the event flows initiated by each individual message as separate scenarios. Instead, they may wish to model as a scenario only the flow that is initiated by the recall. To support this, our tool detects the recall function as a scenario starting point. This allows designers to model a group of external messages that are recalled together as one scenario.

4.5 Dynamic structure support

Dynamic structures offer little complication for our toolset. A scenario is defined by its starting point and the body of the scenario is determined by following the message chain until it reaches a point where no further message is sent.

Our toolset considers all possible branches in the message chain, so a scenario consists of all possible execution paths. At run time the message chain of a scenario may end at any number of points depending on conditional statements, as well as on what structured class instances have been incarnated or imported.

5 Visualizing scenarios

For the visualization of scenario models, our toolset supports AND-OR transition tree like Figure 1. In Figure 1, Ox:Ay represents transition y of structured class x. A node denotes either a transition or a conjunction or disjunction of messages, and an edge denotes message flow. Transition nodes are classified into AND-Transition and OR-Transition. An AND-Transition must send out all of its outgoing messages in the left-to-right order. An OR-Transition sends only one of its outgoing messages depending on the condition within the transition. When a transition has nested conjunctions or disjunctions among its outgoing messages, bridge nodes are used. They are classified into AND-Bridge and OR-Bridge nodes.

Our toolset generates XML documents for AND-OR transition trees. For example, Figure 2 shows such an XML document displayed by an XML viewer.

6 Conclusions

We have presented our scenario modeling framework in scenario-based multithreading for object-oriented real-time modeling. Enabled by our scenario-based multithreading, our modeling toolset allows scenario modeling by transforming a given UML 2.0 model to a scenario model. Motivated by our case study of real-world models, we have extended the notion of scenarios found in our original method. Specifically, we extend the notion of scenarios so that it can support (1) concatenated scenarios, (2) mutually non-concurrent scenarios, (3) ports or structured classes with multiple cardinality, (4) message buffering, and (5) dynamic structures.

We extended our tool to support such scenario modeling functionality. Scenario modeling enables the systematic identification of a feasible task set since scenario models act as bridges to gradually lead to the desired implementation. Our tool also allows designers to browse visualized scenario models. Consequently, our scenario modeling framework not only helps designers to more easily understand the model but also enable the identification of a feasible task set in a systematic way.

In the future, we will continue our research based on other real-world applications including support for distributed systems. We are also considering the potential application of quality of service concepts or models to our research.

References

[1] ARTiSAN Software Tools Incorporation. Real-Time Studio, http://www.artisansw.com

[2] W. Ho, J. Jézéquel, A. Guennec, and F. Pennaneac'h, “UMLAUT: an extendible UML transformation framework,” Proc. of Automated Software Engineering (ASE'99), 1999.

[3] IAR Systems Incorporation, visualSTATE, www.iar.com

[4] IBM Rational Software Corporation. Rational Rose RealTime User Guide: Revision 2001.03.00, 2000.

[5] I-Logix Incorporation. Rhapsody tools. http://www.ilogix.com

[6] S. Kim, M. Buettner, M. Hermeling, and S. Hong, “Experimental assessment of scenario-based multithreading for real-time object-oriented models: a case study with PBX systems,” Proc. of International Conference on Embedded and Ubiquitous Computing (EUC), 2004.

[7] S. Kim, S. Cho, and S. Hong, “Schedulability-aware mapping of real-time object-oriented models to multithreaded implementations,” Proc. of International Conference on Real-Time Computing Systems and Applications, 2000.

[8] S. Kim, S. Hong, and N. Chang, “Scenario-based implementation architecture for real-time object-oriented models, Proc. of IEEE International Workshop on Object-oriented Real-time Dependable Systems,” 2002.

[9] J. Masse, S. Kim, and S. Hong, “Tool set implementation for scenario-based multithreading of UML-RT models and experimental validation,” Proc. of IEEE Real-Time/Embedded Technology and Applications Symposium, 2003.

[10] D. Milicev, “Automatic model transformations using extended UML object diagrams in modeling environments,” IEEE Transaction on Software Engineering, Vol. 28, No. 4, 2002.

[11] Object Management Group (OMG). http://www.omg.org.

[12] B. Selic, G. Gullekson, and P. T. Ward, “Real-time object-oriented modeling,” John Wesley and Sons, 1994.

[13] Unified Modeling Language (UML). http://www.uml.org.