

Developing and debugging Real-Time Software with ObjecTime Developer

ObjecTime Developer is a software development tool that implements UML for Real-Time[1] constructs and enables software developers to generate complete applications[2] directly from their visual design models. ObjecTime Developer also supports the formal design semantics of MSCs so that they are useful for expressing the intended behavior of a system and for automatically verifying system behavior at run-time. Race condition analysis of the Message Sequence Chart can pinpoint design flaws or alternate execution scenarios. Built in model level debugging including state-machine animation and breakpoints, and integration with third party source debuggers, compilers, configuration management and operating systems provides a complete high performance real-time software development environment.

INTRODUCTION

The need to reduce time to market and increase functionality and quality impacts every software developer. For real-time software developers there are additional unique challenges. Every real-time developer recognizes that the requirement for latency, throughput, reliability, and availability are far more stringent than for general-purpose software. Understanding the impact of design decisions and effectively communicating functionality can be a daunting task. An overriding concern is the architecture of the software[4] and whether it will scale to meet the needs of the application and the organization. Architecture is the essential structural and behavioral framework on which all other aspects of the system depend and modern software automation tools like ObjecTime Developer® have been designed for the development of complex real-time software architectures.

Real-time software developers use ObjecTime Developer to design, implement, debug and test their applications. Their designs are specified using the UML for Real-Time[1] graphical capsule structure, finite state machine (FSM), and message sequence chart (MSC) notations. These notations are ideally suited to specifying the designs of event driven, concurrent, real-time software systems, while their rigorous formal semantics allow them to be compiled just like a programming language [2]. Much of the detailed code in many complex real-time applications is concerned with the structure and decision logic of the system and is directly generated from the graphical design specifications.

In this article we will discuss the capabilities of modern real-time software development tools and ObjecTime Developer. Using the example of a switched Cellular Phone network we demonstrate advanced compiler and debugger automation based on Unified Modeling Language (UML) and UML for Real-Time modeling constructs. We discuss how Active Objects [3] in the form of UML capsule, collaboration and finite state machine (FSM) representations have proven themselves to be ideal for developing real-time systems.

Finally we discuss some general capabilities required of modern software engineering tools.

REAL-TIME DEVELOPMENT TOOL

In traditional software development the entire application is hand written using a 3GL language like C. The design may be captured using informal graphical notations, however once coding begins it quickly becomes out of sync with the implementation. ObjecTime Developer is a software development tool that automatically generates complete C and C++ real-time applications directly from graphical design models. The code is produced directly from the design, so the design and the code are always in sync. Other real-time development tools are proprietary, don't have proven code generation capabilities or are not tailored to deal with the specific concerns of difficult real-time problems. ObjecTime Developer is tailored to the needs of the real-time domain with UML for Real-Time modeling constructs and proven code generation.

The shift from pseudo code and assembly language to higher order compilable languages brought with it dramatic productivity and quality improvements to software development. The shift from architectural pseudo code (e.g., informal graphical design notations) and 3GL languages to compilable graphical design notations (e.g., UML for Real-Time) has brought with it equally dramatic improvements. By eliminating most of the hand coding of traditional 3GL development, the seemingly impossible goals of increasing functionality, reducing time to market, and improving quality are realized.

Programming languages require editors, compilers and debuggers. To these ObjecTime Developer adds UML for Real-Time graphical design editors, a model compiler that generates C or C++ applications directly from the design models, and graphical symbolic debuggers. High level graphical design models are debugged using the same graphical formalisms that the designers used to specify their designs Figure 1 shows Capsule structure, FSM and MSC debugger windows. Detailed code can also be edited and debugged using traditional tools.

METHODOLOGY

REDUCING TIME TO MARKET

ObjecTime Developer was conceived by a group of real-time software engineers faced with the challenge of accelerating time to market for their complex real-time software systems, while increasing quality, maintainability and evolvability of software developed within their large development teams. Faced with the difficulty of applying inadequate, general-purpose development tools to the construction of complex, fault tolerant real-time applications they set out to find a next generation solution. Their response was ObjecTime Developer, a software development toolset designed

exclusively for real-time software development. Hundreds of man-years of development effort have optimized the toolset for the demands of large-scale, distributed applications.

REAL-TIME APPLICATIONS

ObjecTime Developer is being used to develop a wide range of real-time applications ranging in size from 10K to 1 Million lines of code (ncsloc) such as:

- OC-192 Transmission systems
- GSM Wireless systems

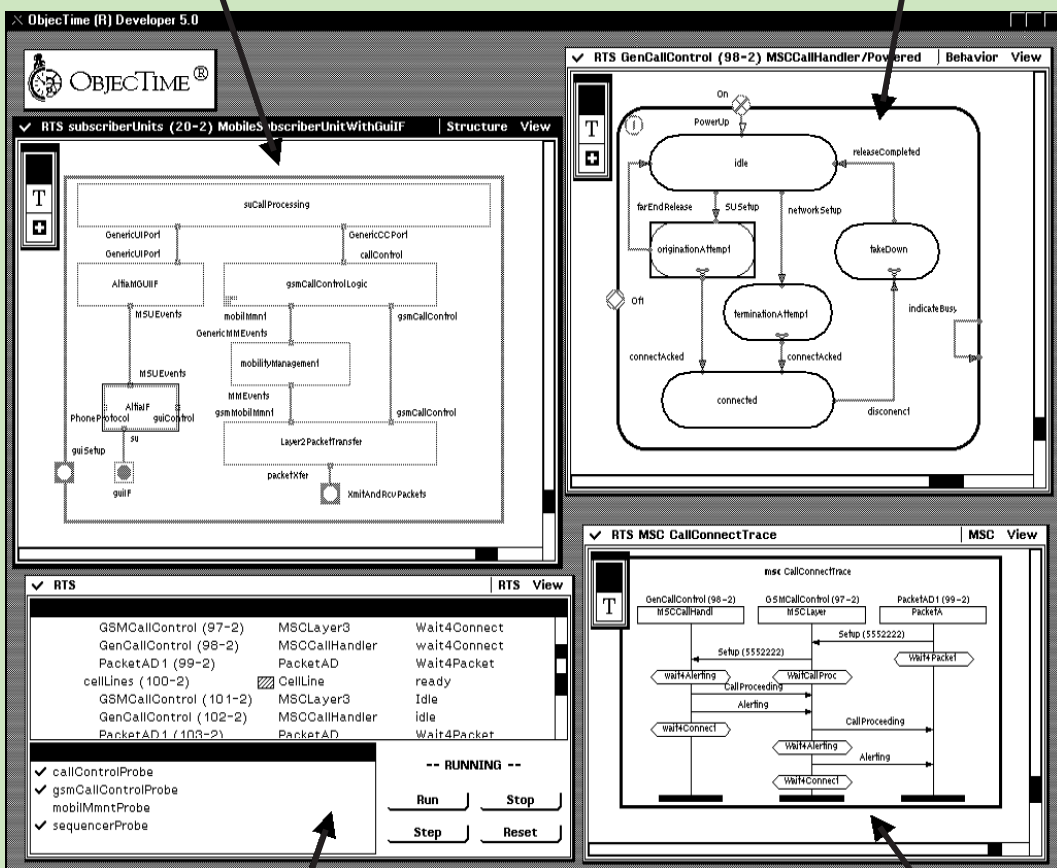
OBJECTIME DEVELOPER RUN-TIME VIEWS

Structure Monitor

Shows the capsule communication and containment relationships. Port probes are used to set model break points and trace messages. Messages between two or more capsules can also be traced. Message traces can be dynamically represented as MSC.

Behavior Monitor

Animated FSM shows how the capsule responds to messages. Transition and state probes set model break points and trace points. Data watch points show values of capsule attributes



Run-Time System (RTS) Control Panel

The RTS Control Panel is the primary interface for controlling an executing application. It lists all capsules, probes and has execution control buttons.

Message Sequence Charts (MSC)

A MSC traces the collaboration between selected capsule instances, the messages they exchange and their resulting changes in state. This MSC was dynamically captured from an executing model.

Figure 1. Run-Time Model Debugger Views

AD OBJECTIVE

METHODOLOGY

- Internet telephony system (Voice over IP)
- digital cross connect systems
- ault management/monitoring system
- optical wave division multiplexor
- missile control systems
- command & control interface design

Some of the companies using ObjecTime Developer are:

- Nortel Networks
- Lucent Technologies
- Motorola
- Lockheed-Martin
- Ericsson
- Boeing
- Kodak

COMMUNICATING THE DESIGN

Graphical models of a system's structure (the components and how they interact) and behavior (the system's response to events) offer a high-level, easy-to-understand means of communicating requirements and design concepts to team members, management and customers. This capability is particularly helpful when new members join the team—they quickly become familiar with the design, without needing to review reams of code in search of the big picture. Designers can view and execute the design from a number of complementary perspectives, including high-level structure, FSM behavior, MSC, and detailed code.

ACTIVE OBJECTS

Through the remainder of this article we will be using the design of a wireless phone network in our examples. A wireless phone network contains a number of elements that collectively deliver switching and call management functionality. The design was developed using ObjecTime Developer, which uses Active Objects, based on UML for Real-Time Capsules, as its primary design constructs. Capsules are complex, potentially concurrent, and possibly distributed active architectural components. They interact with their surroundings through one or more signal-based boundary objects called ports. Ports implement protocols that define a set of incoming and outgoing messages. Figure 2 - ChannelMgr Structure shows the design of a simple capsule with three ports.

UML collaboration diagrams are used to describe the structural decomposition of a UML for Real-Time

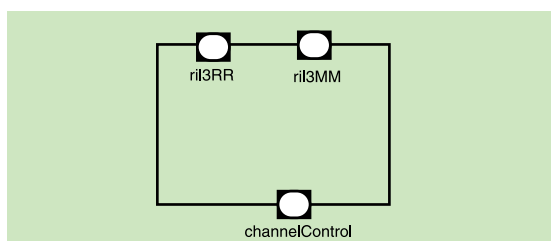


Figure 2. ChannelMgr Structure

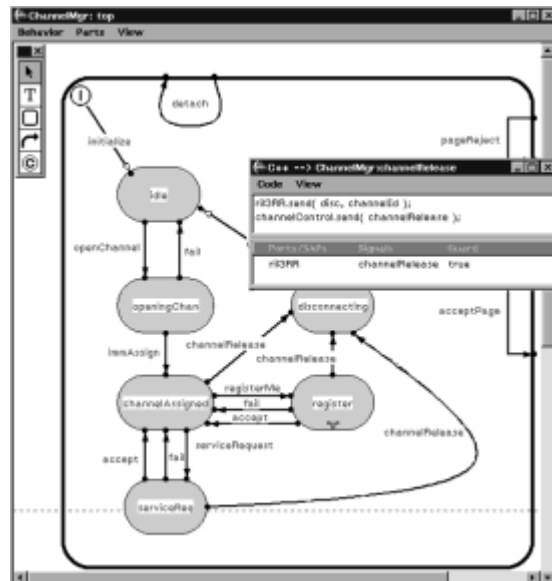


Figure 3. ChannelMgr Behavior

Capsule class. A UML collaboration defines a pattern of interacting objects, or set of object roles, that work together to provide some cooperative behavior. The functionality of simple capsules is realized directly by a state machine associated with the capsule. More complex capsules combine the state machine with an internal network of collaborating sub-capsules joined by connectors. The structure monitor in Figure 1, as well as Figure 5 show capsule collaboration diagrams. Sub-capsules are capsules in their own right, and can themselves be decomposed into sub-capsules. This type of decomposition can be carried to whatever depth is necessary, allowing modeling of arbitrarily complex structures with a basic set of structural modeling constructs. The state machine (which is optional for composite capsules), the sub-capsules, and their connections network represent parts of the implementation of the capsule, and are hidden from external observers.

PORTS, CONNECTORS AND PROTOCOLS

Figure 4 - Shows the Ril3RR protocol specification. This protocol specification is used on the structure of the ChannelMgr capsule (Figure 2) to define its ril3RR port. Figure 5 - Shows the top-level structure of our wireless communication system. The ports of capsules are connected to each using connector, which specify the communication paths between capsules. Ports provide a mechanism for a capsule to export multiple different interfaces; each tailored to a specific role. They also provide a mechanism to explicitly connect an exported interface of one capsule directly to the interface of another capsule. By forcing capsules to

In Signal	Data Class	Out Signal	Data Class
immediateAssignment	Null	channelRequest	Null
immediateAssignmentReject	Null	pagingResponse	ChannelId
pagingRequest	Route	channelRelease	Null
channelRelease	Null	disc	Null
us	Null	pagingReject	Route

Figure 4. Ril3RR Protocol specification

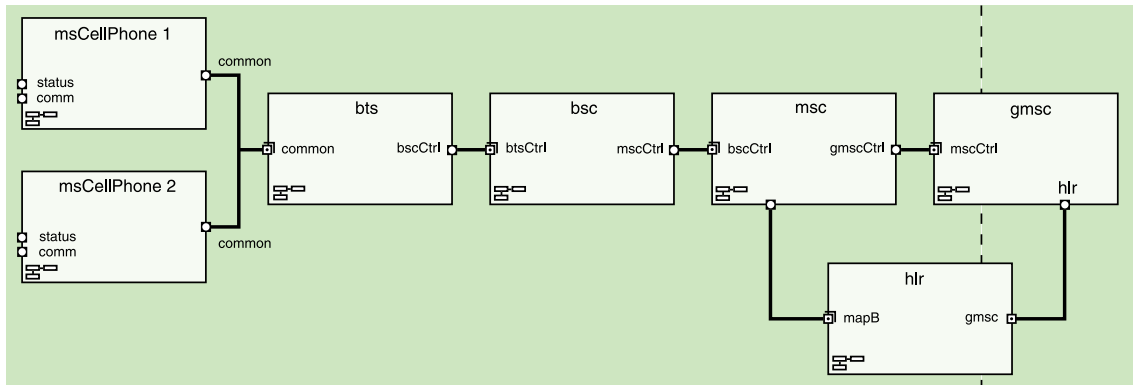


Figure 5. GSM Top Structure

communicate solely through ports, it is possible to fully de-couple their internal implementations from any direct dependencies on the capsules with which they communicate.

HIERARCHICAL FINITE STATE MACHINES (FSM)

A UML hierarchical FSM can be used to define the behavior of a capsule. Messages received on a capsule's ports trigger the transitions within the FSM. Detailed C or C++ action code associated with a transition is executed whenever the transition is triggered. Figure 3 - ChannelMgr Behavior shows the top level FSM for the ChannelMgr capsule of shown in Figure 2. In the transition specification dialogue overlaid on the FSM we can see that the channelRelease transition is triggered by receipt of channelRelease message on ril3RRR port. The simple C++ action code associated with the transition results in messages being sent to the capsule's ports. Each message send consists of a signal field, and an optional data.

PREEMPTION AND BLOCKING BEHAVIOR

All actions in real-time systems are triggered by the occurrence of externally generated events. One very important externally generated event in real-time systems is the passage of time (i.e. a clock tick). Other external events come from hardware, including user interface devices, process sensors, and communication links to other systems[5]. Software must be either blocked waiting, or periodically checking hardware for events. Many events that real-time applications must react to are application level events not directly generated by the hardware, for example the detection of an overload condition that forces a system reconfiguration. Large real-time applications often have many hundreds or even thousands of concurrent threads of execution that must collaborate with each other, and react to asynchronous events in a timely manner, for example in a telecommunication switch there are many simultaneous calls all in different states of progress.

The normal condition of a real-time system (from a CPU's perspective) is idle waiting for an event. Similarly, the normal condition of a capsule state machine is blocked waiting for a message event. Capsules are ideal for expressing the design of real-time software. By adding a message queue and dispatch loop to an

operating system task, capsule execution can be serialized providing a very a lightweight design construct that allows many capsules (i.e.,1000's) to execute on the same operating system task.

With serialized messages, only one message event is processed per task, per capsule at a time. A designer can use message priorities to let high priority events get handled before lower priority queued events. The time it takes for the detailed code in a transition to execute is normally quite small compared to the time allowed to respond to the event, so capsules can usually coexist on the same task without conflict. Sometimes message priorities cannot resolve timeliness conflicts, for example if a transition calls a blocking operating system service or is compute intensive. A capsule can be placed in a higher priority operating system task to let the operating system preemption mechanism meet response time needs. Tasks have stacks and use heavy weight context switching, while capsules use a very light weight scheduling mechanism and have no stack. The designer of a real-time capsule-based system can use message priorities, Active Objects, and operating system tasks to fine-tune the responsiveness of their system. Capsules make concurrency a first class development construct without the heavy weight context switch and stack overheads of operating system tasks.

Scenario Based Development

A use case defines a set of use case instances, where each instance is a scenario that describes sequence of actions a system performs. Scenarios can be captured as UML sequence diagrams, also known as MSCs. An MSC is a diagram that describes a pattern of chronological ordered message interactions among objects. The sequence diagram in Figure 6 shows the messages exchanged within the wireless communication system of Figure 5 when a cell phone successfully registers itself in the network.

The message ordering in an MSC, and in a real system, could be ambiguous. Ambiguous ordering could indicate that there are alternate execution scenarios, a problem in the design, or a situation where message ordering doesn't matter. Race condition analysis finds ambiguous message sequences -- identifying race conditions by just looking at the code or MSCs is nearly impossible. ObjecTime Developer includes advanced race condition analysis capabilities obtained under license from Lucent Technologies. In

AD SYSTRAN

METHODOLOGY

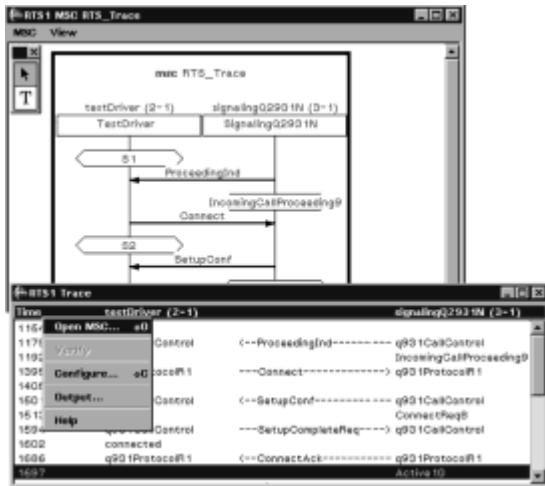


Figure 10. Run-Time message tracing and MSC Generation

1. MSCs associated with a model can be used to automatically verify run-time behavior.
2. Examine the available MSCs or define new ones by tracing or using the built in MSC editor.
3. Select the MSC(s) to verify
4. Select Run to automatically generate a complete executable test harness including the application the components.
5. Compare the generated MSC with the specification MSC. Mismatched signals will be identified using a causal differencing engine and graphically highlighted in side-by-side specification and trace MSCs.

TOOL CHAIN INTEGRATION

A good real-time software automation environment will support many target platforms and real-time operating systems. It will be integrated with compilers, source debuggers, and requirements management tools and will provide integrated model configuration management (CM). ObjecTime Developer supports most popular development platforms (Unix, NT), and a large number of real-time targets including compilers and debuggers. Many modeling and application generation tools have proprietary CM capabilities, or can only place whole models or packages under configuration control. ObjecTime Developer provides CM at the class level providing the needed granularity to keep developer interference to a minimum. It comes with advanced class level check-in/check-out, version control, and class merging capabilities designed for managing parallel development of components across multiple projects and teams. ObjecTime Developer comes ready to run with third party configuration management systems like ClearCase, RCS, SCCS and PVCS.

CONCLUSION

ObjecTime Developer is a software automation tool designed to meet the development needs of real-time software development teams. It lets software developers build applications using UML for Real-Time graphical design models, and then automatically generates

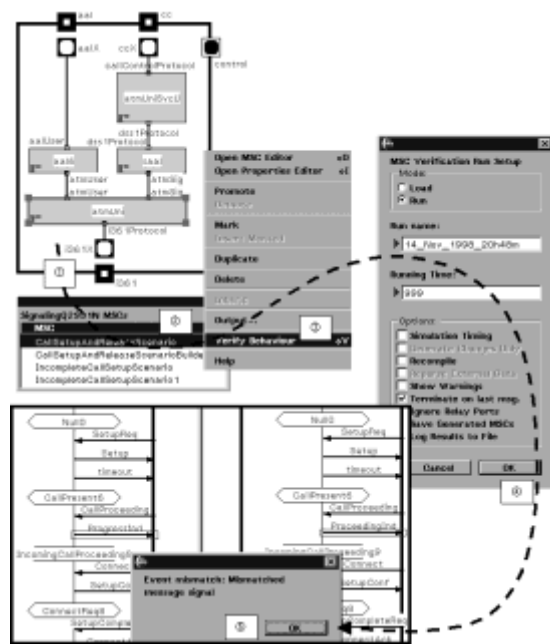


Figure 11. Automatic scenario verification

complete production quality C and C++ applications for a variety of real-time operating systems directly from their graphical designs. Application generation of fully or partially complete designs, plus animated visual and symbolic debuggers, encourage early and continuous design refinement and validation. ■

REFERENCES

- [1] Selic, Bran; Rumbaugh, Jim, Using UML for Modeling Complex Real-Time Systems, <http://www.objecttime.com/otl/technical/umlrt.pdf>, March 11, 1998
- [2] Lyons, Andrew; UML for Real-Time Overview, http://www.objecttime.com/otl/technical/umlrt_overview.pdf, April 1998
- [3] Booch, Grady; Rumbaugh, J., Jacobson, I.; The Unified Modeling Language: User Guide, Addison-Wesley, 1999
- [4] Kruchten, Philippe; Rational Unified Process: An Introduction, Addison-Wesley, 1999.
- [5] Designing for Concurrency, <http://www.objecttime.com/otl/technical/concurrency.pdf>, 1998
- [6] Selic, B.; Gullekson, G.; and Ward, P.; Real-Time Object-Oriented Modeling. John Wiley & Sons, 1994.

Andrew Lyons has over 17 years of experience in real-time development, ranging from writing kernels for small hard real-time systems to software architectures for large distributed life critical systems. Throughout his development experience, he has applied methods and tools to pragmatic development. Andrew is a senior application specialist at ObjecTime Limited with significant consultancy and training experience. He is part of a collaborative effort with Rational Software on the definition of UML for Real-Time and the Rational Unified Process. He can be reached at andy@objecttime.com.