Rational Rose RealTime

A Guide for Evaluation and Review

Table of Contents

Introduction

This guide is intended for reviewers and evaluators of Rational Rose RealTime. It is designed to provide an overview of the product and company, background on key issues such as the Unified Modeling Language (UML) and real-time systems, and to introduce key features and new enhancements to the product. You can use this guide as a resource to facilitate your evaluation of Rational Rose RealTime.

This guide is designed to supplement your reviewer's kit, which contains a complete working copy of the program including full documentation. Special technical support is available as well as additional information resources for those who would like to explore application modeling in more detail [see Section 11, Information Resources].

This guide is organized into the following sections:

1) Introduction - the section you are reading now
2) Application Modeling Background – brief overview of UML modeling
3) Real-time Modeling Background – brief overview of the support for real-time systems, and executable models
4) Rational Rose RealTime – overview and discussion of new features and functionality
5) Market for Rational Rose RealTime – the audience for whom Rational Rose RealTime is designed
6) Evaluation Criteria – checklists for a review or evaluation
7) Packaging and Pricing – Rational Rose RealTime packaging options and license variants
8) System Requirements – minimum system requirements needed to run Rational Rose RealTime
9) Training and Support – description of training and support services
10) About Rational Software Corporation – overview of Rational Software Corporation
11) Information Resources – list of additional informational resources and Web links
12) Reviewer Contacts – list of phone and fax numbers, addresses, and e-mail addresses

Appendix A –Evaluation Criteria

Application Modeling Background

Visual Modeling

Visual modeling provides a way to understand complex problems and communicate that understanding to others. The resulting models capture the essentials of complex problems by eliminating nonessential detail that only serves to confuse the issue. This process of capturing essentials while filtering out the nonessential detail is referred to as abstraction. Abstraction allows people to focus on the big picture—how the components of the problem relate and interact—without getting bogged down in the specific details of any component or technical implementation.

Visual modeling enables software developers to analyze complex software problems at a highly abstracted level using a set of well-defined graphical icons. Software developers use visual modeling to create different diagrammatic views of the system they are building and gradually add detail to the models, which enables the models to evolve into actual software implementations.

Visual modeling and abstraction are critical in the development of modern, distributed, component-based applications. There simply is too much happening in even seemingly simple applications for developers to jump right in and begin programming. The days when one person could immediately grasp a software solution in its entirety and knock out the implementation in a burst of inspiration are long gone (if those days ever truly existed). At the very least, developers need to communicate what they are doing so others can maintain and enhance the application after they themselves have moved on to something else.

Importance of Architecture

Architecture is the set of significant decisions about the organization of a software system. The right UML models and diagrams will illuminate and offer insight into serious software development problems. The wrong models and diagrams, while they may provide a façade or meet documentation requirements, will mislead and cause teams to focus on irrelevant issues.

Because architecture is important it is usually one of the fundamental reason for creating and maintaining models in UML. Therefore, tools that support UML should have the ability to facilitate architecture-first modeling and enforce conceptual control. An example is to be able to identify when low-level design details violate architectural policies or to have support for easily navigating where key classes are used in a model. Likewise, the ability to diagnose which classes are needed to implement high-level requirements, such as use cases, can provide valuable feedback when assessing architectural impacts derived from project changes to scope, cost, or time.

Unified Modeling Language

There have been many approaches to visual modeling, such as OMT, Booch, and OOSE. Each offers recognized strengths and drawbacks, and each has its adherents. While they all support the concept of visual modeling and abstraction, each uses a different method and set of notation, the graphical icons used to identify components, relationships, and interactions. The confusion resulting from the competing visual modeling approaches led to a period of religious modeling methodology wars in which adherents promoted their favorite approach. The obvious solution was a standard approach that incorporates the best of the individual approaches.

The Unified Modeling Language (UML) emerged as the solution to end the modeling methodology wars. UML addresses data modeling (entity/relationship diagrams), business modeling (workflow), object modeling, and component modeling. It can be effectively used to visualize, specify, construct, and document the elements of a software application. Used with all processes throughout the project lifecycle (requirements capture, analysis and design, implementation, testing), UML provides a standard language that may be understood by everyone involved with the project. UML was officially introduced in June 1996. Version 1.0 was released in January 1997. In November 1997, the Object Management Group (OMG) approved Version 1.1 as the standard language for analysis and design. Currently, the UML is at Version 1.5 and is owned by the OMG (http://www.omg.org).

Version 2.0 of the UML will be adopted some time in 2004, and a key advance will be the introduction of structuring concepts. These are based on the structure modeling of Rational Rose RealTime. The standard is not fixed until it is formally adopted, but Rose RealTime users will be the first to make full use of the underlying ideas.

Real-time Modeling Background

Support for Real-time Systems

As stated above, visual models are important tools for developing complex systems because they help manage complexity and increase comprehension. They can be used to represent the system at an abstract level, hiding unnecessary details. They enable communication about the composition and operation of a system. With Rational Rose RealTime, the model also serves as the basis for the implementation. *All of the implementation details required to build an executable are contained within the model.*

The UML uses visual notations to describe various views of an object model. Classes are the fundamental building blocks of this object model. The abstract structure, behavior and configuration of the software are described through diagrams. The implementation details of each class can be specified through UML diagrams and various class properties.

Rational Rose RealTime emphasizes some features of UML that are particularly applicable to real-time system development.

Active Objects

Rational Rose RealTime uses **an industry standard, proven, and robust design pattern centered on the concept of active objects.** Active objects are objects that have an encapsulation shell that encapsulates their data, their run-to-completion behavior, and most importantly their own thread of execution thus preventing thread interaction problems. Active objects are a crucial design aid for many types of systems, particularly event-driven, reactive systems with some degree of concurrency, parallelism and/or distribution.

Concurrency

Most real-time systems must be capable of performing many simultaneous activities. External events are unpredictable (user input, error conditions, hardware malfunctions, interrupts, etc..), and the software must be able to handle these various inputs at any time.

Capsules and ports

Rational Rose RealTime provides built-in lightweight concurrent classes designed around the active object design pattern, known as *capsule*s, the UML 2 specification has defined these classes as structured classes. Like any UML class, capsules can be described by class diagrams.

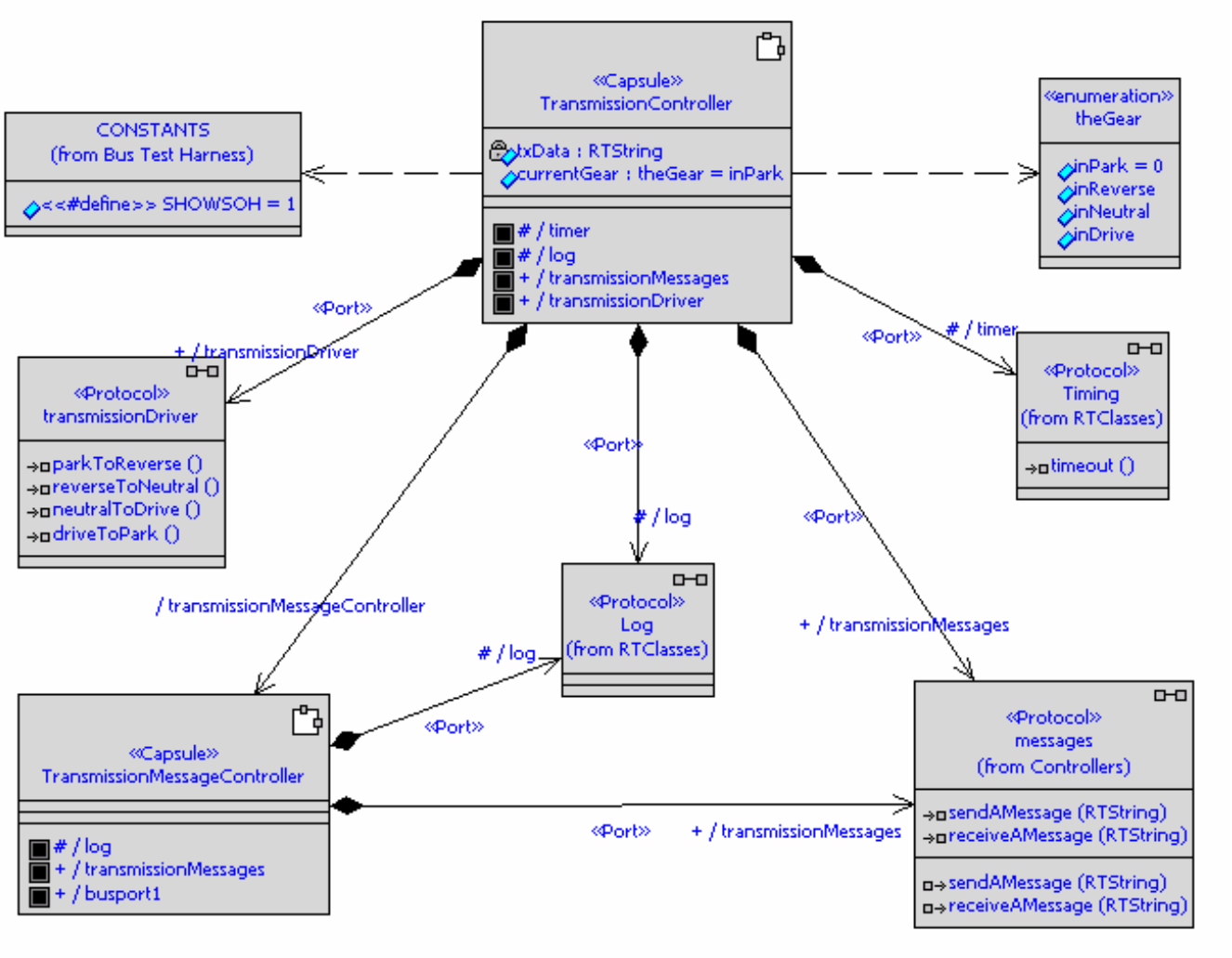

Figure 1 – Capsule and protocols example

The capsule structuring concepts are the basis of UML 2.0 “classes with structure”. In addition to their concurrency properties, capsules provide a high degree of encapsulation – insomuch as all attributes and operations are logically protected. Capsules communicate through special message-based interfaces called Ports. A capsule sends and receives messages through its ports. These messages are explicitly defined in Protocols. The ports are in turn connected to other capsules, enabling the transmission of messages among capsules. The advantage of the message-based interfaces is that a capsule has no knowledge of its environment outside of these interfaces, making it much more flexible, reusable, and resilient over regular passive objects.

Structure diagrams

A diagram is used to specify the capsule’s interface and its internal composition. The diagram is called a structure diagram, and is equivalent to the UML 2.0

"internal structure diagram". The semantics around the structure diagram allow Rose RealTime to generate detailed code to implement the communication and aggregation relationships among capsules.

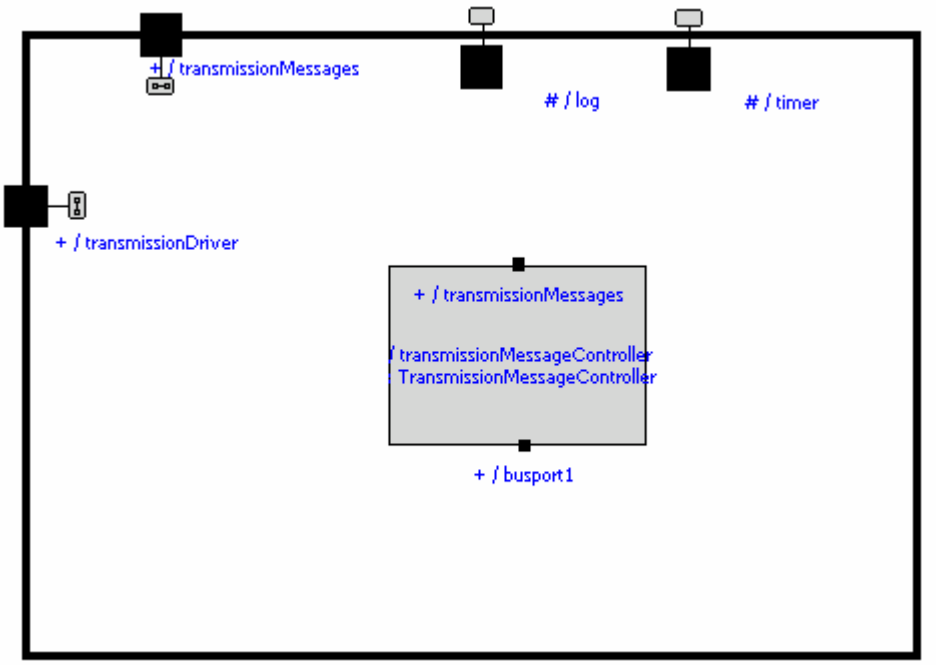

Figure 2 – Structure diagram example for one capsule

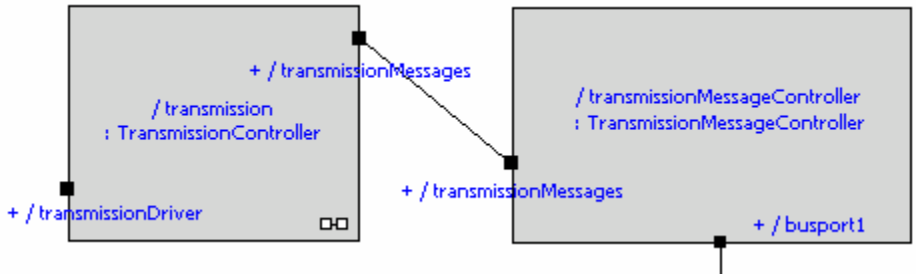

Figure 3 – Structure diagram example between capsules

State Diagrams

The behavior of a capsule is graphically captured with the use of a hierarchical state machine or statechart. These state machines support run-to-completion semantics – critical to the reliable operation of active objects. The capsules input stimulus are defined by the union set of all input signals for all end ports found on the capsules structures.

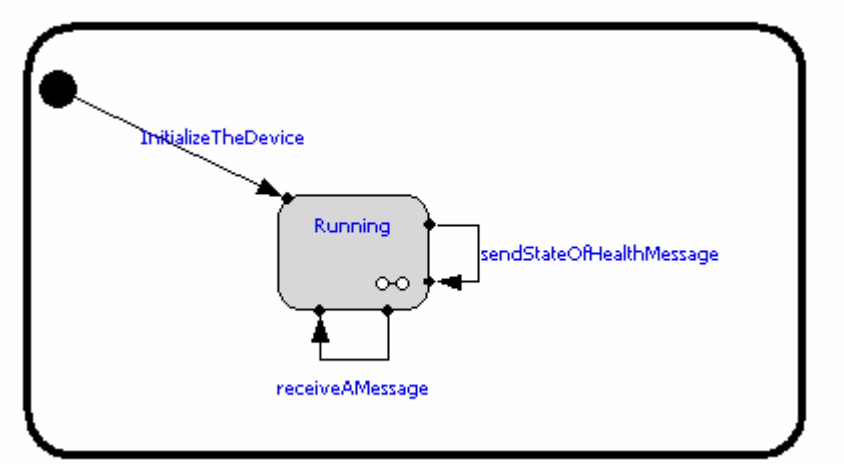

Figure 4 – Top-level state diagram example

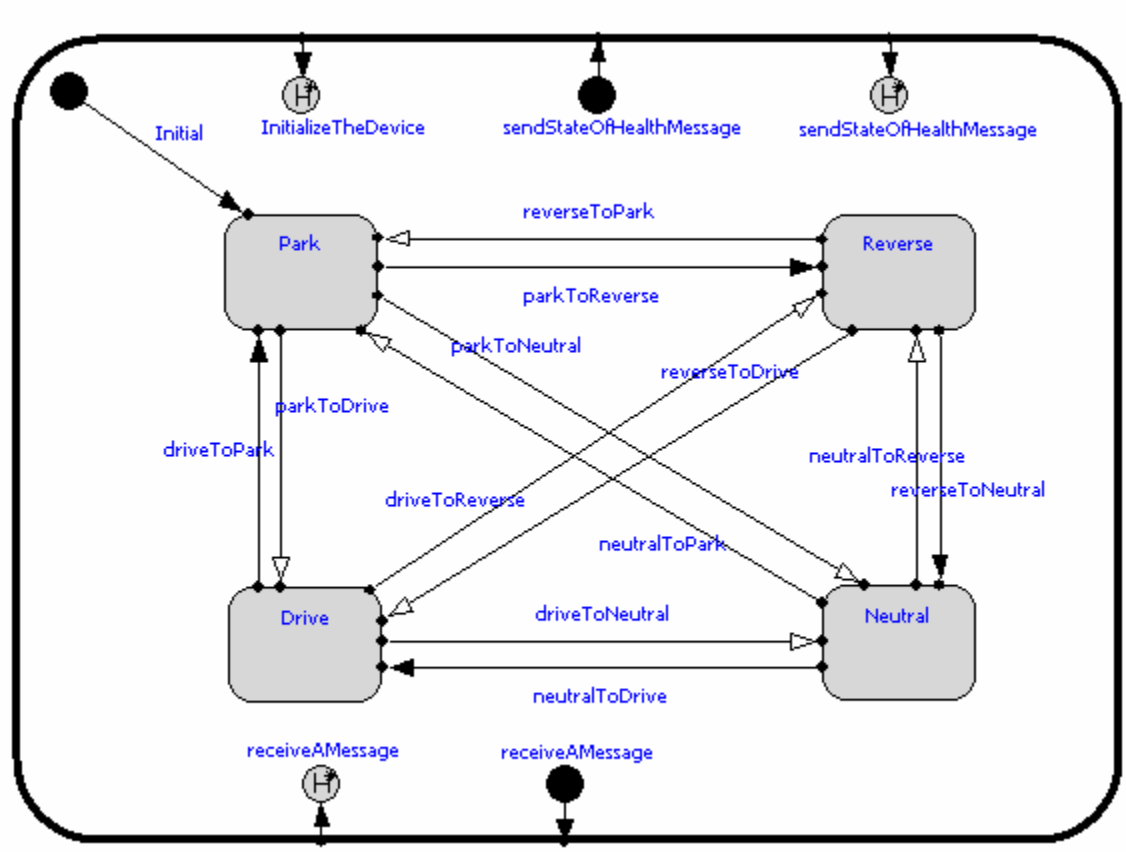

Figure 5 – Decomposition of Running state from Figure 4

State Machine Behavior for Passive Objects

Not all applications need the full power of encapsulated active objects, and not all objects in an application need to be capsules. Rose RealTime fully supports modeling of normal or "passive" objects, and code generation from them. However, even these passive objects (and thus their classes) may have behavior that is best represented by a state machine, and Rose RealTime can generate code from state machines on non-capsule objects. Since these objects do not have full encapsulation (including ports and their own conceptual threads of execution) they do not require any run-time support. Therefore, applications built entirely from non-capsule classes (with or without state machines) can have very small memory footprints and need no RTOS.

Model-Driven Development

The UML model is semantically rich that, like a traditional programming language, it contains all of the information required for complete generation of your application. At all stages of the development, Rational Rose RealTime can generate, compile and run a complete C, C++ or Java implementation. The ability to execute models (even partially defined) has a revolutionary impact on the software development process. The results are higher quality software, and shorter and more predictable delivery cycles. Executing models is the surest way to find problems and issues that white boarding and document reviews don't find. Even high-level architectural models can be executed.

In addition, model execution can be used to better understand the problem, to detect errors and problems in requirements and architecture specifications, to explore alternative designs quickly, and to test design models continuously during the development process.

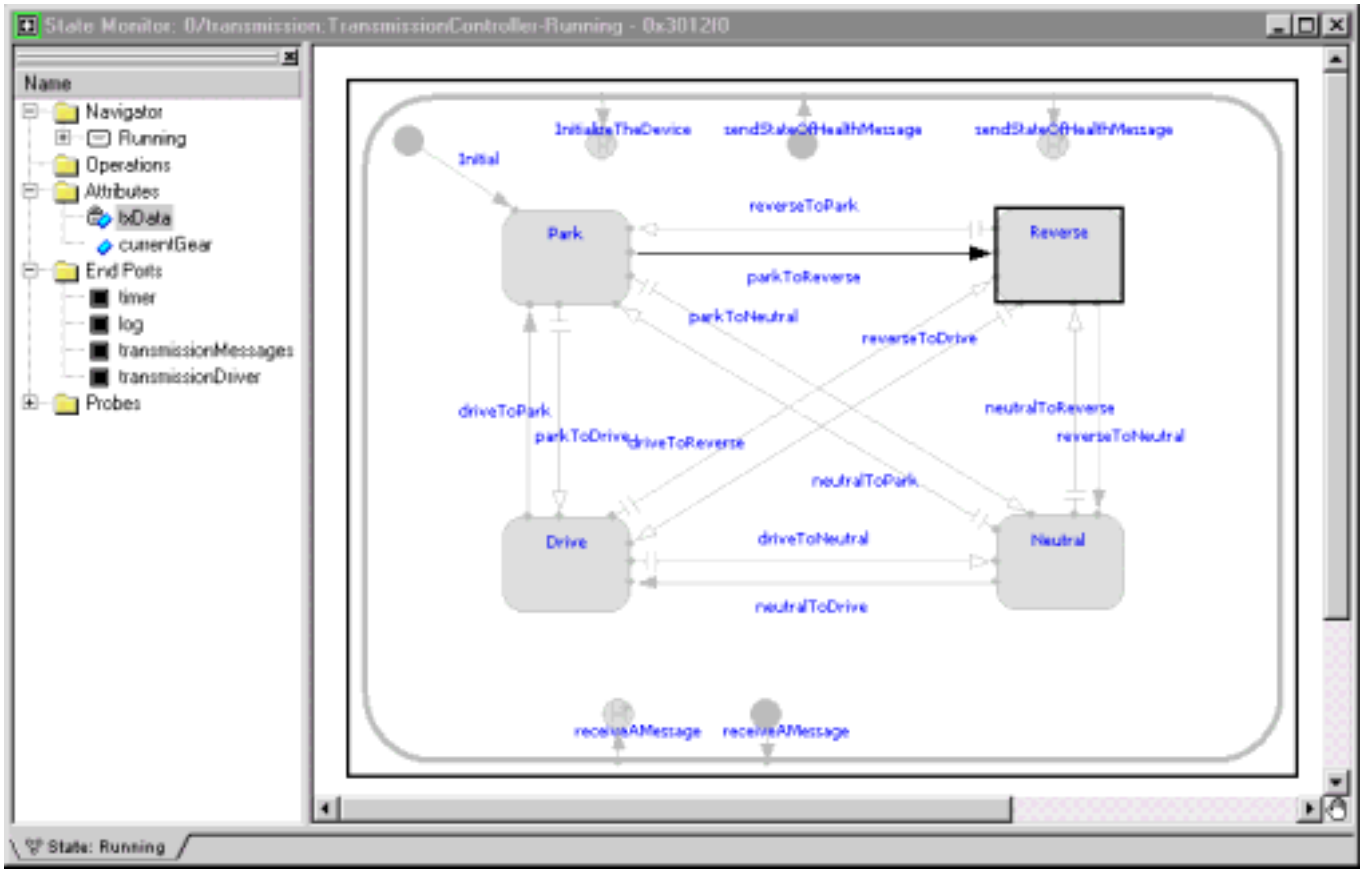

Figure 6 – Example of an executing model

Process note: To make the best use of Rational Rose RealTime, Rational recommends that end users should use an iterative and incremental process like the Rational Unified Process (RUP). The aim is to get the model running as early, and often as possible. Making small, incremental changes and running your model each day will bring much better results than making widespread changes and then working for weeks to get the model running again.

Rational Rose RealTime

Rational Rose RealTime is a complete UML development environment that supports the challenges of real-time embedded software development. Rose RealTime can be used through all phases of the software development lifecycle, from initial requirements analysis through design, implementation, test and final deployment. It provides a single interface for model-based development that integrates with other tools required during the different phases of development. For example, Developers work directly through Rose RealTime to generate and compile the code that implements the model.

Using Rose RealTime, Developers work at a higher level of abstraction, letting the tool take care of the routine details. This is a natural and logical evolution in computer languages. Just as third generation language tools provided greater productivity than assembly language coding, visual development tools provide significant productivity gains over current third generation languages.

Rose RealTime includes features for:

- Creating UML models using the elements and diagrams defined in the UML
- Generating complete C, C++, and Java applications for those models with UNIX, Windows, and various RTOS targets.
- Executing, Observing, and Debugging models on host or target
- Validate host and target applications with automated test generation
- Highly scalable multi-user team environment

Each of these major topics is covered in detail in the Rose RealTime Online Help. See Help, which can be found through Help > Help Topics. Rose RealTime's online help now contains animated demonstrations of many common activities. See Help -> Contents -> How Do I? This opens the watch and learn document which has links to various animated demonstrations.

The Market for Rational Rose RealTime

Rational Rose RealTime is designed for the serious developer intending to build complex real-time, embedded or technical systems. These include teams who develop systems for telecommunications, data communications, defense and aerospace, automotive, medical, and industrial control applications. Rose RealTime is available on both the Windows NT and UNIX platforms and supports a wide range of hosts and targets out of the box, such as Windows NT, Solaris, HP-UX, WindRiver Tornado II, Green Hills Integrity, ENEA OSE, VRTX, and QNX. In addition the Target Deployment Port technology allows it to be easily adapted to virtually any 8- to 64-bit processor, RTOS (or no operating system at all) and C, C++ or Java compiler. This can be done by the customer, Rational or a third-party partner. Rational has not yet encountered a customer environment that cannot be supported.

Criteria for Evaluation

Appendix A contains checklists provided as a starting point for evaluating or reviewing a comprehensive visual real-time development environment.

Pricing and Packaging

Rational Rose RealTime is currently available in two packages:

1) Rational Rose Technical Developer
2) Rational Suite for Technical Developers

Rational Rose RealTime provides:

- UML design, execution, and visualization
- Complete C, C++, and Java code generation
- Application deployment to hosted environment
- Application deployment to host and target
- Target services library source code
- Rational Connexis – for model visualization of distributed applications
- Rational QualityArchitect - RealTime – providing automatic validation test generation for both target and host applications

Rational Suite for Technical Developers provides:

- Rational Rose Technical Developer
- Rational Rose Developer for UNIX
- Rational PurifyPlus
- Rational Team Unifying Platform

Rational currently offers two types of licenses, node-locked and floating. These two key types of licenses are designed to meet specific customer and product needs.

Node-locked license keys are licensed to a specific host's disk id. The software can be used only on that host.

Floating license keys are controlled by a license server. When Rational products are run on client machines, the product requests use of a floating license key from the license server. The license server grants the client use of the license key if a key is available for the product being used. The license server considers the license key in use until the user exits the client application. The server will continue to grant license keys to client requests until the pool of available license keys has been exhausted.
Any user with administrator privileges can install the license software from the Rational product CD. Floating license keys are licensed to the server machine only; they are not licensed to the clients running Rational products.

The Rational License Manager can run on either UNIX, Linux, or NT and can serve both UNIX, Linux, and/or NT licenses.

System Requirements

Rational Rose RealTime is designed for simple insertion into your software development environment, processes, and workflows. Rose RealTime includes seamless integration with other Rational products and support for a variety of commercial real-time operating systems. Rose RealTime system requirements are listed below:

Platform requirements - Windows NT

The minimum supported configuration for running Rose RealTime on Windows NT is:

- Windows NT 4.0, build 1381, with service pack 6a
- Minimum Pentium 150 MHz; we recommend 500 MHz or faster CPU
- Minimum 128 MB of RAM; we recommend 256 MB of RAM
- Minimum 552 MB of disk space for the Rose RealTime installation
- Minimum display 1024 X 768; we recommend 1280 X 1024 or better
- Postscript printer for printing
- Browser requirement — Internet Explorer 5.01 or 5.5 or NetscapeNavigator 4.7 or 6.0. We recommend Internet Explorer 5.01 or 5.5.

Platform requirements - Windows 2000

The minimum supported configuration for running Rose RealTime on Windows 2000 is:

- Windows 2000 Professional, Build 2128, Release Candidate 2 (RC2)
- Same disk, RAM, and browser requirements as Windows NT, above.

Platform requirements - Windows XP

The minimum supported configuration for running Rose RealTime on Windows 2000 is:

- Windows XP Professional, Build number 2600.
- Same disk, RAM, and browser requirements as Windows NT, above.

Platform requirements - Linux

The minimum supported configuration for running Rose RealTime on Linux is:

- RedHat 7.3, 8

Platform requirements - Unix

The minimum supported configuration for running Rose RealTime on Unix is:

- Solaris 2.6; Solaris 7, 8, 9 ; HPUX 10.20
  For Solaris operation,the minimum workstation is an UltraSparc 10 with 500 MB of RAM: We recommend an UltraSparc 60 with 600 MB of RAM.We recommend the Solaris 2.8 operating system.
- Please see the Rational Rose RealTime website for a list of the **required Unix patches** applicable to your operating system.
- For HPUX operation, we support installation of the HP 700 series architecture.

The minimum is 500 MB of RAM; we recommend 600 MB of RAM.
Minimum 370 MB of disk space for the Rose RealTime installation

- Postscript printer for printing
- Browser requirement Netscape Navigator 4.51 minimum. At the time of release Internet
- Explorer 5 for UNIX had not been tested.

Compilers

You must have a C or C++ compiler installed on your system if you wish to make use of the code generation and execution capabilities of Rose RealTime C or C++. Different compilers are required for host workstation and for embedded system targets. The list of supported compilers and targets is provided in the *Getting Started Guide*.

Java Development Kit

You must have the Sun JDK 1.3 or later installed on your system if you wish to make use of the code generation and execution capabilities of Rose RealTime Java.

Real-Time Operating Systems

If you are planning to deploy your model on a real-time operating system, your operating system, hardware and tool lineup may be one of the out-of-the-box or “reference” lineups listed in the *Getting Started Guide*. See the *Target Guide* for instructions on configuring the Services Library and compiling for new target platforms or speak to your Rational Software representative. Both reference and non-reference configurations are fully supported by Rational. Issues encountered on non-reference platforms will be evaluated by Rational Support using reference platforms.

After your installation has been completed, we recommend that you get started by reviewing the online tutorial. See the tours and tutorials reference, which can be found through Help > Help Topics.

Training and Support

Rational provides a full range of support:

- Annual maintenance and support contracts
- Online technical support (http://www.ibm.com/software/rational/support/)
- Telephone and email support.

IBM Rational telephone support staff are available to answer customer questions from 6:00 A.M. to 6:00 P.M. Pacific time, Monday through Friday.
[Note: a technical support contact is available for reviewers]

Rational University speeds learning by combining a proven software development process and practice with formal education and training. In addition to expert product and professional education programs, Rational University courses provide a conceptual framework that will change the student's approach to software development. More information on Rational University.

For Rational Rose RealTime specific training courses please click here.

IBM Rational also offers an on-site 2-day Evaluator's Workshop to assist evaluators and reviewers in the use of Rational Rose RealTime. In addition, there is a Quickstart package that includes licences, training and mentoring. Your IBM Rational sales representative can provide you with details.

Information Resources

IBM Rational information - http://www.ibm.com/software/rational/

Rational Rose Technical Developer product information is available at:
http://www.ibm.com/software/awdtools/developer/technical/

Rational Rose RealTime patch releases, updates, and example models can be found at:
http://www.ibm.com/software/awdtools/developer/technical/support/

Additional information on the Unified Modeling Language (UML):
http://www.ibm.com/software/rational/uml/

3

Glossary and Acronyms

API – Application Program Interface

IDC – International Data Corporation

ISV – Independent Software Vendor

OMT – Object Modeling Technique

OOSE – Object-Oriented Software Engineering

Product suites: Multi-platform line of integrated product suites that unify software teams

Requirements management: Finding, documenting, organizing, and tracking changing software and system requirements

RCS – Revision Control System

RTOS – Real-Time Operating System

RUP – Rational Unified Process

SCC – Source Code Control

SCCS – Source Code Control System

Software quality and test automation – Providing integrated programming and testing tools to simplify the creation of components and to replace expensive, tedious, and error-prone manual testing, resulting in higher-quality applications in less time with lower risk.

Software configuration management (SCM) – Controlling the day-to-day team development of software, as it is created, modified, built, and delivered.

Software process automation – Providing guidance to software managers and developers about how to create software that is a competitive business asset .

UML – Unified Modeling Language

Visual modeling - Creating a graphical blueprint of a software application, its components, their interfaces, and their relationships, which make the system easier to understand and manipulate.

Appendix A – Evaluation Criteria

Installation and setup	Rose RealTime	Other
Complete, easy-to-follow installation instructions	X	
Single licensing mechanism for all Rational products	X	
Floating and node-locked licenses available	X	
OS and system requirements clearly stated	X	
Various installation media available (CD default)	X	
Uninstall option	X	

Documentation and Tutorials	Rose RealTime	Other
Installation Guide	X	
Release Notes	X	
Getting Started Guide	X	
Language-specific Programmer's Guide(s)	X	
Language-specific Target Guide(s)	X	
User's Guide	X	
Latest manuals available on-line	X	
Complete on-line product tour and tutorials	X	
Sample models available	X	

Help and Support	Rose RealTime	Other
On-line help in HTML format	X	
All on-line help searchable	X	
Complex help searches available (wildcard, and, or, multiple keyword)	X	
On-line help printable	X	
Help for help	X	
Toll-free support during business hours	X	
E-mail support during business hours	X	
24x7 support available	X[1]	
Customer email user's group and contribution area	X	

[1] There is an additional cost for 24x7 support

Tool Integrations and API	Rose RealTime	Other
Requirements		
Rational RequisitePro	X	
RTM	X[1]	
DOORS	X[2]	
Configuration Management Version Control Tools		
Rational ClearCase	X	
Microsoft Visual SourceSafe (NT only)	X	
RCS (UNIX only)	X	
SCCS (UNIX only)	X	
Documentation		
Rational SoDA (Software Documentation Automation)	X	
Memory Leak Detection, Path Coverage, Bottleneck Analysis		
Rational Purify	X	
Rational Quantify	X	
Rational PureCoverage	X	
Rational Test RealTime	X	
Software Development Process		
Rational Unified Process	X	
Change Request Management and Defect Tracking		
Rational ClearQuest	X	
Development Environments		
Microsoft Visual C++ 5.0 or greater	X	
WindRiver Tornado cross development environment	X	
Green Hills MULTI	X	
Mentor Graphics Spectra compatible	X	
HP-UX	X	
Sun Solaris	X	
GNU	X	
OSE Diab/SDS tools compatible	X	
Development Tool Suites		
Rational Suite for Technical Developers	X	
API		
100% open API accessible via scripting	X	

[1]Available from Integrated Chipware

[2]Available from METEX

Target Environments, Language support	Rose RealTime	Other
Windows NT/2000/XP	X	
Solaris 2.6; 7, 8, 9	X	
Linux RedHat 7.3, 8		
HP-UX 10.2	X	
WindRiver VxWorks 5.4/5.5	X	
Green Hills Integrity	X	
ENEA OSE	X	
QNX	X	
Windows CE	X	
Your embedded platform	X	
ANSI C code support	X	
C++ code support	X	
Java code support	X	
J2ME support	X	
J2SE support	X	
Target porting wizard	X	
Host code generation and execution	X	
Target code generation and execution	X	
Full code generation with services libraries for large applications	X	
Full code generation with no service libraries for minimal-footprint applications	X	
Distribution license included	X	
Source code for Service Libraries	X	

Modeling	Rose RealTime	Other
Supports UML Modeling	X	
Use Case Diagrams	X	
Sequence Diagrams	X	
Structural Collaboration Diagrams	X	
Class Diagrams	X	
State Diagrams	X	
Structure Diagrams	X	
Component Diagrams	X	
Deployment Diagrams (multiple)	X	
UML Real-time profile support	X	
Capsules	X	
Capsule roles	X	
Ports	X	
Protocols	X	
Connectors	X	
User-defined diagram auto population, to *n* levels	X	
User-defined diagram filtering, hiding	X	

Auto diagram element layout	X	
User-defined model navigability	X	
Active Object design pattern support	X	
User-definable color for all modeling elements	X	
Supports the model-view-controller paradigm	X	

Real-time Support	**Rose RealTime**	**Other**
Hierarchical state machines for active classes	X	
Hierarchical state machines for non-active classes	X	
High-level architectural abstractions (i.e. precise interface definitions allow for loosely coupled architectures)	X	
Hierarchically decomposed active objects for managing complex architectures	X	
Optional and plug-in roles for dynamic sessions	X	
Distributed system modeling support through ports	X	
Capability to prototype “human in the loop” and “hardware in the loop” through ports	X	
Capability to create “unwired” ports enabling layered architectures	X	
Application-level concurrency management through encapsulated thread of control	X	
Encapsulated thread of control enables early and continuous unit testing	X	
Programmable timers available	X	

Model Execution	**Rose RealTime**	**Other**
Active class state machine model debugging with state change animation	X	
Auto generation of sequence diagrams from model executions	X	
Supports build outside toolset	X	
Build Avoidance for incremental, fast builds	X	
Save execution settings between runs (watch variables, probes, etc..)	X	
Command line interface	X	
User-definable breakpoints	X	
User-definable traces	X	
User-definable watch window	X	
User-definable probes (event insertion)	X	
Debugging step mode supported	X	

Tool Usage	**Rose RealTime**	**Other**
Intuitive, customizable GUI	X	
Model Explorer-like browser	X	

Parallel development support	X	
Distributed development support	X	
Multi-user support	X	
Automatic model and code synchronization	X	
Graphical visual differencing and merge utility	X	
Version control to the granularity of the class level	X	
Support for RoseLink Partner add-ins	X	
Rose Extensibility Interface (REI) for scripting	X	
VBScript environment, including debugging	X	
Basic textual report generation	X	
WYSIWYG diagram printing	X	
HTML web model publishing	X	
Show usage report	X	
Show access violations report	X	
Show code occurrence report	X	
Show references report	X	
Rose98i and Rose2000 model importing	X	
ObjecTime Developer model importing	X	
Integrated documentation and URL attachment via drag and drop (docs, spreadsheets, pictures, etc..)	X	
User-defined model navigability	X	
User-selectable code editors	X	
User-customizable log	X	
Year 2000 compliance	X	

Company	**Rose RealTime**	**Other**
Industry leader	X	
Full range of development tools	X	
Worldwide local sales and service support	X	
Training available (onsite and open enrollment)	X	
Industry and thought leadership (Grady Booch, Jim Rumbaugh, Ivar Jacobson, Bran Selic, Phillipe Kruchten, Walker Royce, Terry Quatrani etc..)	X	