Bulletin

The Rose Family Grows

Analysts: Jaga Shahani and Steve Garone

IDC Opinion

How will object-oriented analysis, modeling, design, and construction (OOAMDC) tools address the needs of developers of embedded and real-time software systems?

With the race for a real standard for object-oriented analysis and design notation won by UML, user and vendor efforts alike have now turned to how to use UML in a variety of development venues. One of these is the development and deployment of real-time systems. UML is well-suited for this use, although a vendor-driven approach to extending UML to support real-time modeling constructs was needed.

Rational Software Corp. and ObjecTime Ltd. have pooled their experience to create a real-time profile for UML that has resulted in a version of Rational's highly successful Rose product specifically targeted toward real-time users: Rose RealTime. Rose 98i, a new version of its standard Rose product, provides greater integration with other important and related products and, more importantly, adds support for UML's business process modeling capabilities.

Together, these product versions move Rational into an even more competitive position in a market that, from the standpoint of product revenue, it has led for four years.

A View of Real-Time Systems

A fundamental shift in perception is occurring, with personal computing systems moving from being considered computational devices to being considered real-time communication and entertainment controllers.

Real-time systems have been around for a long time. One typically associates real-time systems with telecommunications, aerospace, defense, automotive, and industrial control applications. However, with the emergence of multimedia computing and communications, real-time systems are becoming more pervasive. A fundamental shift in perception is occurring, with personal computing systems moving from being considered computational devices to being considered real-time communication and entertainment controllers.

Whether in handheld devices or fast network switches, the need for real-time systems exists. With the explosion in the Internet and Web domains, real-time systems are in even greater demand to ensure fast, predictable service. In several instances, the systems being developed are complex, networked, and distributed.

Real-time systems are usually characterized as having the ability to:

- Respond reliably and predictably to events
- Be fast and deterministic in their responses
- Perform concurrent processing of multiple tasks
- Be embedded in a device or system with limitations on memory, physical properties, processor performance, etc.

While a number of real-time software development methods have been introduced, the design and development of complex real-time systems have been high-risk adventures. Methods have either been ill suited to the process or not used effectively to model and design large, complex real-time systems.

Object-oriented modeling methodologies have made considerable strides over the last five years, culminating in the adoption of Rational Software's Unified Modeling Language (UML) by the OMG as a standard.

Object-oriented modeling methodologies have made considerable strides over the last five years, culminating in the adoption of Rational Software's Unified Modeling Language (UML) by the Object Management Group (OMG) as a standard. UML is finding applicability in data-driven application development and acceptance in IT organizations.

However, object-oriented methods and notations for complex real-time systems have taken a back seat. While some methods have addressed the analysis and design of complex real-time systems, they have not found wide adoption. Notable among these is the Shlaer-Mellor method, which uses three basic models to express a system: the information model that identifies relationships between objects, a state model that formalizes dynamic behavior, and a process model that formalizes the processing required. Communication between objects is displayed via an object communication model, which is similar to the architecture diagram in UML. Real-time constraints are expressed through a thread of control chart, which allows annotation of each state with its action time.

The Shlaer-Mellor method presents a rigorous and extremely well-defined formalism for the analysis, design, and implementation of all kinds of systems. However, the changes in the development paradigm it poses to organizations and the high degree of "up front" work it requires have made it practical for only very large, complex applications.

There have been many proprietary and competing methods and tools, none of which achieved widespread adoption. UML incorporates the best practices found in all of these methods and provides a foundation for rapid market acceptance and adoption.

While industry experts have been touting the value of the object-oriented approach for real-time system development for a decade, development teams have chosen to forgo reuse and dynamic binding in favor of performance and time to market.

While industry experts have been touting the value of the object-oriented approach for real-time system development for a decade, development teams have chosen to forgo reuse and dynamic binding in favor of performance and time to market. A significant contributor to this attitude has been the lack of a robust method, notation, and tool that can handle the timing constraints that real-time systems have. Also, the nondeterminism associated with certain object-oriented properties and the suitability of the methods to model real-time constraints and the run-time have been contributing factors to slow adoption.

UML is well suited to adopt real-time modeling constructs, and Rose RealTime now brings the object-oriented paradigm squarely into this domain.

UML is well suited to adopt real-time modeling constructs, and Rose RealTime now brings the object-oriented paradigm squarely into this domain. Rational Software and ObjecTime have collaborated to define a comprehensive approach for the application of UML to the development of complex real-time systems. In the same way that UML incorporates popular and effective software practices, Rational and ObjecTime have collaborated to incorporate the types of practices found in the real-time domain into an engineering tool optimized for the real-time developer. The ROOM (Real Time Object Oriented Modeling) method developed by Selic and others has been in use for some years to model complex real-time systems. The partnership led by Rumbaugh and Selic focused on taking the design patterns and concepts from ROOM into UML.

UML: The Real-Time Highlights

UML was developed with built-in mechanisms (stereotypes) to support domain-specific modeling. constructs. Stereotypes provide a way of classifying model elements through the addition of UML metaclasses with new attributes and semantics.

Central concepts in ROOM were captured through specializations of UML's stereotype, tagged values, and constraints mechanisms.

Fundamentally, no new modeling concepts were required to tailor UML for the real-time domain. Central concepts in ROOM were captured through specializations of UML's stereotype, tagged values, and constraint mechanisms. UML provides two basic and complementary diagrams to express the logical structure of systems:

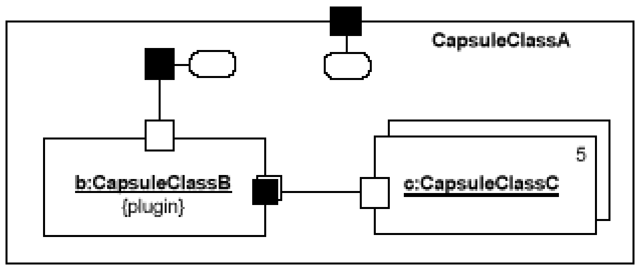
- The **class diagram** is a static view of the system and shows the relationship among entities (classes) that make up the system. It is a generalized view of all instances and contexts of the defined classes.
- The **collaboration diagram** captures relationships that exist in a specific context.

In UML for the real-time domain, the specification of a complex real-time system can be expressed through a combination of class and collaboration diagrams.

In UML for the real-time domain, the specification of a complex real-time system can be expressed through a combination of class and collaboration diagrams. Three new constructs were defined to address modeling structure: capsules, ports and connectors.

Capsules are the fundamental modeling element of real-time systems. Figure 1 depicts an example use of capsules. Capsules are a pattern for providing concurrency in UML notation. A capsule is a specialization of a class, having the same properties as a class (attributes and operations) and possibly participating in relationships.

Figure 1
Capsule Collaboration



Source: Rational Software Corp. and ObjecTime Ltd., 1999

However, a capsule also has certain special properties that distinguish it from a class. A capsule interacts with other objects through signal-based boundary objects called ports. These ports are the only mechanism they have for communicating with other objects in the system. The structure of a capsule is completely private, and no outside object can access its attributes. Messages are the sole method of communication between capsules, and these are received and sent through ports. The behavior of a capsule is defined and controlled by an optional state machine only. Capsules that do not have state machines are containers for subcapsules. A state machine is the only element that can access the protected parts of a capsule. Because capsules can only communicate through ports, their implementation can be kept completely private. This encapsulation makes them highly reusable.

The internal organization of a capsule can be formally specified as a set of networked subcapsules — referred to as capsule collaboration — and the subcapsules play capsule roles. Capsule roles represent a specification of the type of capsules that can occupy a position in a collaboration. Capsule roles are owned by the container capsule and cannot exist independently. A network of collaborating capsule roles is joined by connectors.

Ports are boundary objects that act as interfaces. They are not the same as UML interfaces. Ports include both structure and behavior, and are objects that send and receive messages for capsules. They are owned by the capsule and are created and destroyed along with the capsule instance. Ports are associated with protocol roles that specify the messages that are sent and received.

Capsule collaboration diagrams are key to the expression of the architecture of a real-time system.

Capsule collaboration diagrams are key to the expression of the architecture of a real-time system. A capsule collaboration displays the communication between capsules and the composite structure of its capsule roles, ports, and connectors. Roles in a capsule collaboration are restricted to capsule roles; association roles are not allowed.

Since capsules communicate with each other through ports, a capsule collaboration shows the capsules' ports. These ports can be shown as public ports (accessible by other capsules) and protected ports (private to the structure of the capsule).

Rationals Rose RealTime Environment

Rose RealTime is a software development environment that allows the development of real-time applications using UML.

Rose RealTime is a software development environment that allows the development of real-time applications using UML. It encompasses all phases of application development, from system requirements, analysis, and design through to code generation, debugging, testing, and deployment. In many ways, Rose RealTime is very similar to standard Rose. Key differences are 1) the ability to generate complete applications directly from the UML design, targeted to real-time operating systems, and 2) UML model debuggers that allow a developer to control and observe an executing application at the UML model level (e.g., set a break point on a state machine).

Structure Diagrams

Capsule structure is defined using the Capsule Structure Editor. This includes how instances of a capsule class are composed of other capsule class instances and protocol class instances (ports). The Structure Editor consists of three parts:

- The structure diagram area
- The structure browser
- The structure toolbox

Structure elements, such as ports and capsule roles, may be dragged onto the structure diagram area from any browser (usually from the model browser). The structure browser can be used to navigate to and open editors and specification dialogs on contained structure elements.

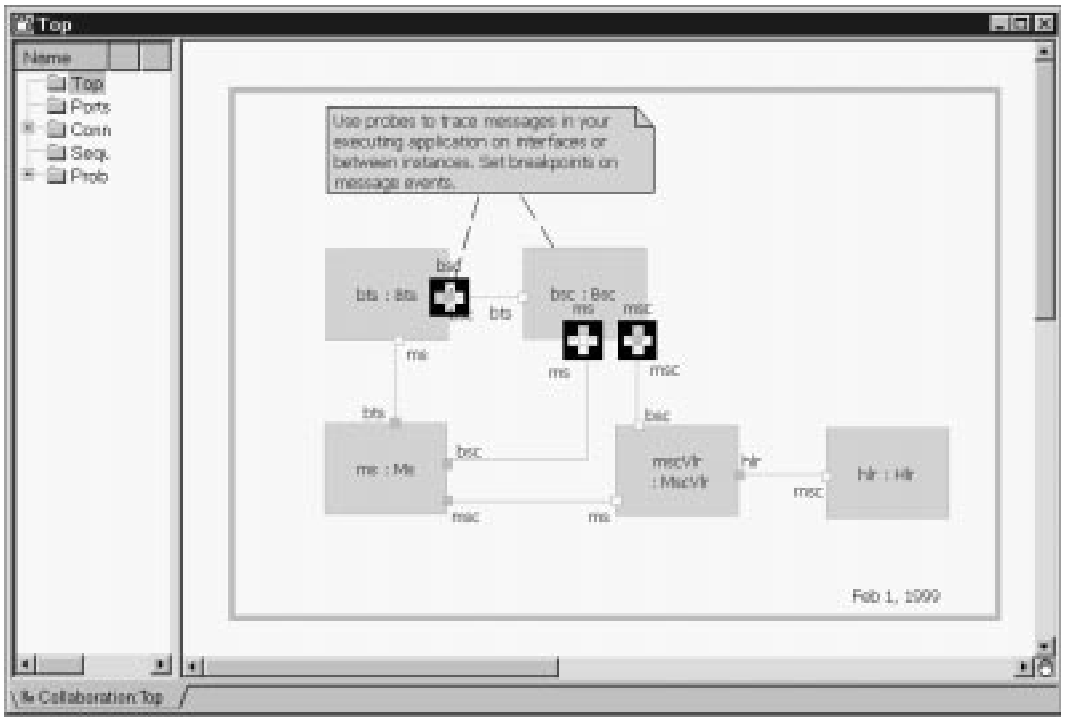
Collaboration Diagrams

To show association or collaboration among capsule roles, Rose RealTime provides the Collaboration Diagram Editor.

Figure 2 shows an example of a collaboration diagram. To show association or collaboration among capsule roles, Rose RealTime provides the Collaboration Diagram Editor. The Collaboration Diagram Editor consists of two parts:

- The diagram area
- The tool box

Figure 2
Sample Collaboration Diagram from Rose RealTime


Use probes to trace messages in your executing application on interfaces or between instances. Set breakpoints on message events.

Source: Rational Software Corp., 1999

Elements such as classifier roles, capsule roles, and association roles are added using the tool box.

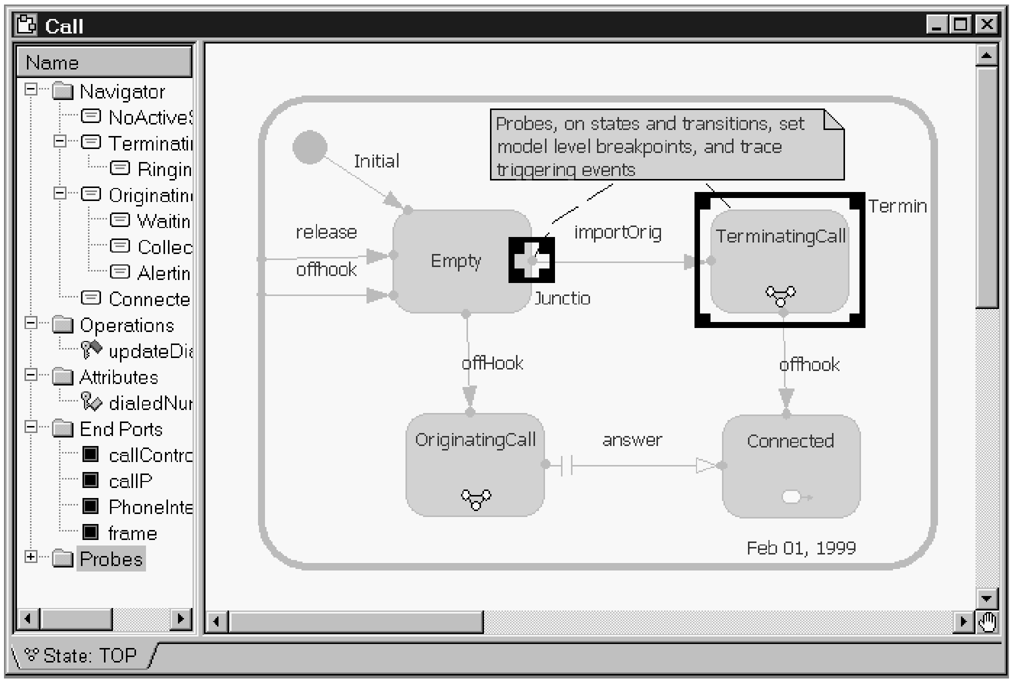
State Diagrams

The State Diagram Editor is used to define the finite state machine for a class. The utility of the class diagram depends on the type of class it is specifying:

- **For capsule classes,** the state diagram will result in a complete code implementation generated for the class. The state diagram defines the majority of a capsule class implementation.
- **For protocol classes,** the state diagram specifies the expected operation of any capsules that contain one of the protocol's roles. The protocol state diagram defines the allowable sequence of message inputs and outputs with respect to the protocol roles. There is no code generated for the protocol class behavior.
- **For data classes,** the state diagram captures the abstract behavior for the class. This does not result in any code being generated for the data class.

Figure 3 depicts an example of a state diagram. The State Diagram Editor consists of three parts: the diagram area, the navigator area, and the toolbox. Behavior elements, such as states and transitions, are added using the toolbox. The State Diagram Editor window has tabs on the bottom to allow quick navigation to any nested states and to the Capsule Structure Editor.

Figure 3
Sample State Diagram from Rose RealTime



Source: Rational Software Corp., 1999

The State Diagram Editor allows the creation or editing of states, state transitions, choice points, initial states, junction points, and end states (except for capsules).

The State Diagram Editor allows the creation or editing of states, state transitions, choice points, initial states, junction points, and end states (except for capsules). The state machine can be nested, allowing the creation of hierarchical state machines. Hierarchical state machines maintain a state history. When a transition terminates on a hierarchical state, the history mechanism may be triggered to determine which substate becomes the active state.

Building Executable Models

Transforming design models into code has always been a difficult task. It is at this point in the development cycle that departure from the original requirements usually happens. Rose RealTime is able to generate 100% of the code directly from the UML model diagrams. Rose RealTime generates source code, which is compiled and linked into machine language using an external compiler and linker.

In IDC's opinion, the model compiler that generates complete applications from the UML model is a major contribution to enhancing developer productivity.

In IDC's opinion, the model compiler that generates complete applications from the UML model is a major contribution to enhancing developer productivity. It allows users to stay focused on the problem domain and let the model compiler translate their design into code. Designers can quickly generate executable models to observe behavior, which in turn will greatly advance the iterative processes of requirements specification, modeling, and implementation.

Running and Debugging Executable Models

Once components have been built successfully, Rose RealTime provides an execution environment that can be used to run, control, and observe models running on a processor. It is possible to run and control multiple component instances from within Rose RealTime.

In Rose RealTime, a user can observe the state transitions in the state machine of an application while it is executing on the target.

In Rose RealTime, a user can observe the state transitions in the state machine of an application while it is executing on the target. In addition, message events can be traced on the states and transitions within state machines, on the interfaces of objects, and on message interactions between collaborating objects. Model execution can be stopped using model-level breakpoints, triggered by any of these traced message events.

This type of high-level debugging is relatively new to most developers. Without automatic application generation, developers must manually translate their designs into the implementation language. This manual step causes a disconnect between the design and the implementation. With Rose RealTime, this disconnect is eliminated, so developers can debug using their choice of model- or source-code-level debuggers — using the appropriate level of abstraction for the problem.

Real-Time Operating Systems and Language Tools Supported

Rose RealTime runs on Solaris, HP-UX, and Windows NT platforms. Rose RealTime supports both host and embedded target execution. In the embedded real-time market, it supports all of the leading operating systems, including Wind River's VxWorks, Integrated Systems' pSOS+, Mentor Graphics' VRTX, and Enea's OSE. Support for additional target operating systems is available from ObjecTime Limited.

In addition to support for model-based debuggers, Rose RealTime provides integrated support for source language debugging tools.

In addition to support for model-based debuggers, Rose RealTime provides integrated support for source language debugging tools. These include products from Wind River, Green Hills, Mentor Graphics, and GNU.

Integration with Configuration Management

Model elements can now be versioned and stored in a repository for use and sharing among members of a large team.

Rose RealTime provides integrated configuration management (CM) support for Rational's ClearCase and Microsoft's Visual SourceSafe, SCCS, and RCS. An application interface allows integration with other CM systems. This integration brings a level of large-team support. Model elements can now be versioned and stored in a repository for use and sharing among members of a large team. A saved Rose RealTime model file contains all diagrams, classes, and packages. A team can place the model elements in the CM system, permitting a finer level of granularity and allowing individual diagrams, classes, and packages to be stored, versioned, and checked in as individual files in the CM repository. This allows different developers to check out files to work on in parallel on different parts of a complex model.

Rose RealTime does not currently support automatic differencing and merging of files. Changes to versions of a file must be made manually. However, a Visual Differencing tool is included. This tool is a graphical presentation of the differences between versions of a model or file.

The results of comparison are shown in a Difference Browser. There are three kinds of differences that are reported: additions, deletions, and modifications. Since models are hierarchical in nature (packages contain classes, classes contain relationships and operations, operations contain parameters, etc.), this is represented in the reported differences. A change to a model element lower in the hierarchy would report modifications to all enclosing model elements in the hierarchy.

Integration with CM systems brings large-team development support to Rose RealTime.

Integration with CM systems brings large-team development support to Rose RealTime. It takes Rose RealTime from being a single-user environment to a team-based environment. Complex real-time systems such as a telephone switch or a command-and-control system are designed and built by large software development teams working in parallel. Tracking and controlling the design model as it evolves make the use of version control and configuration management essential.

Rational Rose 98i: The Rest of the Story

Over the past year, Rational has made important enhancements to its standard Rose 98 product.

Over the past year, Rational has made important enhancements to its standard Rose 98 product. These range from integration with IDEs to the addition of large-team support and business process modeling. Rose 98i adds the following:

- Activity diagrams
- The Model Integrator tool, which is integrated with Rational ClearCase

- IDE-specific round-trip engineering for Visual C++ 6.0
- Enhanced Java and CORBA integration
- A Web publisher

Of these activity diagrams, the Model Integrator and the Visual C++ integration are worthy of special attention.

Activity Diagrams

Activity diagrams use UML to model business processes and work flows. In addition, activity diagrams can be used to describe flow or a sequence of events in a system. An example of an activity diagram is shown in Figure 4.

Activity diagrams are flowcharts that describe a set of activities and transitions within a system. An activity diagram is a special case of a state machine where most of the states are activities, and transitions may take place on the completion of an activity rather than by an external trigger or event. The activity diagram has a basic set of notations that include states, transitions, decisions, and synchronization bars.

Decisions have a set of defined guard conditions that control the transitions or a set of alternative transitions once an activity has been completed. Synchronization bars are used to show parallel process flows and concurrent threads in a system.

For large-team support, the Model Integrator brings features that contribute greatly to transforming Rose from a single-user toolset to a large-team environment.

For large-team support, the Model Integrator brings features that contribute greatly to transforming Rose from a single-user toolset to a large-team environment.

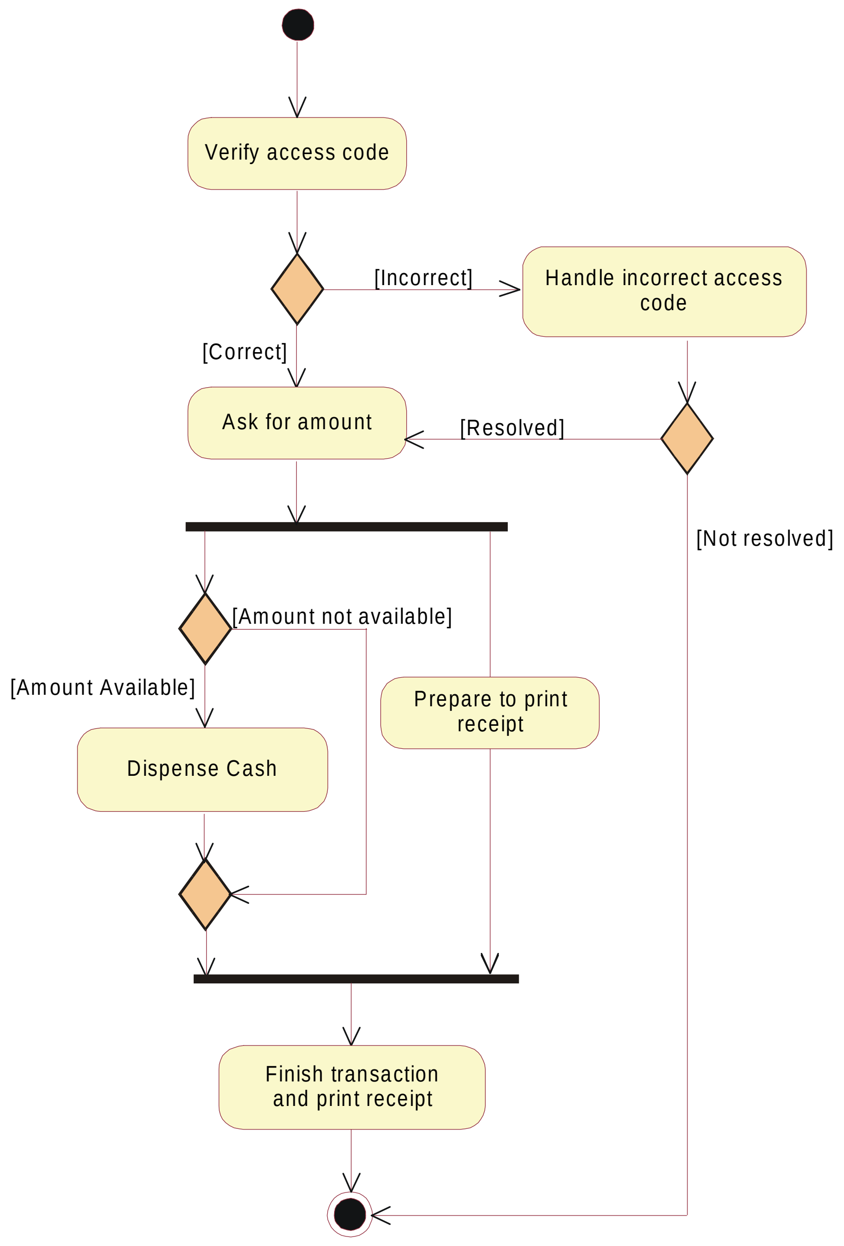
The Model Integrator

The Model Integrator is a visual merge-and-compare tool. It allows large design teams to work concurrently and in parallel on different or the same part of an object model.

The Model Integrator allows true parallel and concurrent model development. The Model Integrator is a visual merge-and-compare tool. It allows large design teams to work concurrently and in parallel on different or the same part of an object model. It supports the comparison and merging of model elements from up to seven different contributor files. These can be multiple versions of the same model, unrelated models, or a combination. Comparison and merging of model files can be with or without associated subunits.

In many ways, the Model Integrator works much like an n-way *diff* tool, executing automatic merges wherever possible and notifying users when there are conflicts. With the Model Integrator serving as a ClearCase Type Manager, nonconflicting changes are automatically merged, and conflicting changes are identified and presented visually for resolution.

Figure 4
A Simplified Activity Diagram



Source: Rational Software Corp., 1999

Visual C++ Integration

Rose 98i/Visual C++ employs an "update in place" model to locate and update existing code.

Rose 98i/Visual C++ employs an "update in place" model to locate and update existing code. Code generation does not require "protected regions" to place code. Only those items and declarations represented by semantics in the model are updated; all other code is left undisturbed.

Rich semantics enable "legacy" applications to be leveraged by creating models from existing code. Existing components can be created as models. Renaming of class and member declarations across code generation and reverse-engineering cycles is supported. Rose 98i supports the generation and reverse engineering of MFC messages as well as command-and-notification message handlers.

Visual C++ integration makes liberal use of wizards to improve ease of use and eliminate reliance on complex code-generation properties and rules.

One such wizard is the Model Assistant Tool, which replaces the standard Rose specification dialog for Visual C++ assigned classes. The Model Assistant Tool provides an enhanced user interface for viewing and editing Visual C++ and MFC semantics.

IDC is of the opinion that these key enhancements in Rose 98i make it the tightest available integration with Visual C++.

Packaging and Pricing

Rose 98i is available for Windows 95, 98, and NT and comes in three packages.

Rose 98i is available for Windows 95, 98, and NT and comes in three packages:

- The **Enterprise Edition** is the full-blown set, complete with modeling, full language support, and Web publishing of models and documentation.
- The **Professional Edition** supports a choice of Visual Basic, C++, or Java and does not include Web publishing.
- The **Modeler Edition** supports modeling only.

All editions are integrated with version control and configuration management tools. Enterprise, Professional, and Modeler are priced at $3,955, $2,388, and $1,612, respectively, for single-user licenses with one year of support. Rose 98i for Unix is available on Sun Solaris, HP-UX, IBM AIX, SGI IRIX, and Digital Unix; the product is priced at $3,955 for single-user licenses with one year of support.

Rose RealTime is available in two packages:

- The **Modeler Edition** provides support for C++ executables on the host system and is priced at $6,800 (node-locked) for a single user with one year of support.
- The **Developer Edition** provides support for C++ executables for host systems and target real-time operating systems, such as Wind River Tornado/VxWorks, ISI pSOS, Microtec VRTX, and Enea OSE. It is priced at a $9,600 (node-locked) for a single user with one year of support.

Summary and Conclusions

With Rose RealTime, developers now have a language and tools to create and describe architecture models of real-time systems and ensure that the design models are complete and executable.

Rose RealTime's strengths are in modeling, code generation and the visualization of models during execution. IDC's opinion is that these key features coupled with the full complement of industry-standard UML support makes Rose RealTime a major contender as the de facto st andard for real-time embedded system development. In real-time embedded application development, as teams get closer to code, they tend to stray further and further away from the original requirement. With Rose RealTime, developers now have a language and tools to create and describe architecture models of real-time systems and ensure that the design models are complete and executable. Since the code is generated directly from the design, the UML design and the implementation are always synchronized, thus eliminating manual design translation errors.

With the Model Integrator enhancements, Rose 98i establishes itself in the mainstream of large-team IT development. IDC believes that Rose's position has been made even stronger, with the result that the use of this environment will become much more prolific.

Document #: 20089
Publication Date: August 1999
Published Under Services: Development Life-Cycle Management