

Magazine

> **Back Issues**

[Premiere Issue](#)
[Winter 1999](#)
[Spring 1999](#)
Summer 1999
[Search Archived Articles](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

Contact Us



[About Rational Rose](#)



[About the UML](#)

Magazine

Summer Issue, June 1999



Cover Story

The Importance of Teams

Grady Booch

Features

Stupid Rose Tricks

Douglas Earl

Integration Focus: Rose and the Rational Unified Process

Per Kroll

Rose RealTime or Zen and the Art of Telling Knock-Knock Jokes

Matthew Drahzal

Overcoming Father Time: Parallel Development through Component Engineering

Christian Buckley and Darren Pulsipher

Inside Rose: Building JavaBeans™ with Rose J

Jeffery Hammond

The Perfect Couple: MSVM + VB

Janine Rood and Shoshanna Buszianowski

Departments

Publisher's Note

Adam Frankl

Letters to the Editor

Terry Quatrani

Amigo Page

Grady Booch
The Importance of Teams

Rose Around the World

Nasser Kettani

Extreme Rose

Tom Schultz
Rose, Modeling, and UML Extensibility

Magnus Opus

Magnus Christerson & Davyd Norris
Landscaping with Rose

Extending Rose

John Hsia
Your Guide to Rational Rose Add-Ins

Veronica's Web

Rose 101

Naveena Bereny
The Web Publisher

RoseLink Review

Chandrika
Shankarnarayan
MIMB from Meta Integration Technology

Advertiser Index

Copyright © 1998 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

[Magazine](#)

> [Back Issues](#)

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

[Summer 1999](#)

[Table of Contents](#)

[Scripts & Models](#)

[Product Reviews](#)

[Book Reviews](#)

[About Rose Architect](#)

[Contact Us](#)



[About Rational Rose](#)



[About the UML](#)



The Importance of Teams

[Sidebar:](#) Rational's solution to team development...Rational Suite

by **Grady Booch**

Grady Booch is Chief Scientist at Rational Software Corporation.

As Gerald Weinberg pointed out over 25 years ago in his seminal work, *The Psychology of Computer Programming*, "computer programming is a human activity."¹ When you are up at 2 AM trying to stamp out an elusive bug, programming does indeed look like an *isolated* human activity. However, there's a big difference between cutting code and shipping products: deploying quality software is a team sport that requires a group of people with a variety of skills working together toward a common goal.

Associated with his work on the Capability Maturity Model (CMM) for development organizations, Watts Humphrey observed that "the history of software development is one of increasing scale. Initially, a few individuals could hand craft small programs; the work soon grew beyond them. Teams of one or two dozen professionals were then used, but success was mixed. While many organizations have solved these small-system problems, the scale of our work continues to grow. Today, large projects require the coordinated work of many teams. Complexity is outpacing our ability to solve problems intuitively as they appear."²

There are perhaps two major forces that drive this complexity. First, there is technology push, wherein the ever-declining cost of hardware and the growing availability of high-speed networks makes it possible to develop automated solutions that, even a year ago, would not have been economically feasible. Second, there is the societal pull, wherein individuals and organizations have come to rely upon such automation, and in so doing have developed an insatiable demand for systems that are better, faster, and cheaper. Combine that with a world-wide shortage of skilled software developers, and the net result is that the typical development team is being asked to do more with less: more features and more quality with less resources and less time.

Weinberg goes on to note that "for the best programming at the least cost, give the best possible programmers you can find sufficient time so you need the smallest number of them."³ That is certainly sound advice: all things being equal, it's better to have a small team than a large one, and to be relatively unconstrained by schedule. But, as Fred Brooks points out, "the dilemma is a cruel one. For efficiency and conceptual integrity, one prefers a few good minds in doing design and construction. Yet for large systems one wants to find a way to bring considerable manpower to bear, so that the product can make a timely appearance."⁴

Most problems are sufficiently complex that they simply require a lot of hard work and sustained labor, more than can be carried out by a single developer working in isolation.⁵ However, you cannot just expect a project to succeed by staffing it with superstars and arming them with powerful tools. Walker Royce points out that, although hiring good people is important, it's far more important to build a good team.⁶ He goes on to explain that balance and coverage are essential characteristics of such a team, and describes this by analogy. "A football team has a need for diverse skills, very much like a software development team. There has rarely been a great football team that didn't have great coverage: offense, defense, and special teams, coaching and personnel, first stringers and reserve players, passing and running. Great teams need coverage across key positions with strong individual players. But a team loaded with superstars, all striving to set individual records and competing to be the team leader, can be embarrassed by a balanced team of solid players with a few leaders focused on the team result of winning the game."⁷

In the context of software development, "winning the game" means developing and deploying quality software in a predictable and sustainable fashion. As Barry Boehm demonstrates in COCOMO (a model for software cost estimation), the capability of the team has the greatest impact upon productivity.⁸ Tom DeMarco and Tim Lister go on to note that, within a team, an organization's most productive people will tend to outperform its least productive by a factor of 10:1.⁹

Historically, most advances in software development languages and tools have focused on improving the productivity of the individual developer. This is not to say that such advances are unimportant; I'd rather have a fast compiler than a slow one. However, given the importance of teams to modern software development, such advances in individual productivity have diminishing returns relative to winning the game. As such, it makes sense to turn our attention to the software development team, and ways to prove its productivity.

Organizing the Software Development Team

Drawing upon his experience inside Microsoft, Steve McConnell notes that "it takes more than just a group of people who happen to work together to constitute a team. In their book *The Wisdom of Teams*, Katzenbach and Smith define a team as 'a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable.'"¹⁰ He goes on to enumerate the characteristics of a hyperproductive team:¹¹

- a shared, elevating vision or goal
- a sense of team identity
- a results-driven structure
- competent team members
- a commitment to the team
- mutual trust
- interdependence among team members
- effective communication
- a sense of autonomy
- a sense of empowerment
- small team size
- a high level of enjoyment

There are many different ways to organize such a team: the business team, the chief-programmer team, the skunkworks team, the feature team, the search-and-rescue team are among those that McConnell identifies.¹² Although no one organization is optimal for every team or problem domain, the surgical team (as Brooks calls it¹³), also known as the chief programmer team, is perhaps the most common and most effective.

In this organization, the central player in the development organization is the architect, who is responsible for the "conceptual integrity of all aspects of the product perceivable by the user."¹⁴ Don't confuse the role of the architect and the project manager, however, as they encompass very different activities.¹⁵ A team's architect is responsible for designing the system, whereas the project manager is responsible for designing the team and making it possible for them to do their job.

Within a team, you should choose an architect who possesses a vision large enough to inspire the project to great things, the wisdom born from experience that knows what to worry about and what to ignore, the pragmatism necessary to make hard engineering decisions, and the tenacity to see this vision through to closure.¹⁶ On small projects, one architect is sufficient. On larger projects, architects are a recursive feature; different subsystems

will require their own architect, with the overall conceptual integrity of the system being maintained by a single architect or, more commonly, a small team of architects.

Surrounding these architects at each level in the system are application developers, people who love to code and who are able to turn abstractions into reality.

Thus, the center of gravity of the development team should be formed by a tight grouping of the organization's project manager, its architect, and its application developers. Jim Coplien, in his study of hyperproductive organizations, has found that many such teams exhibit this common pattern or organization.¹⁷

These three skill groups are necessary but, as it turns out, insufficient. The successful deployment of any complex system requires a variety of other skills: toolsmith, quality assurance, system integration, build and release, and analyst are some of the identifiable roles you'll need to fill on most projects. Capers Jones notes that "the software industry reached the point of needing specialists as long ago as twenty years.... For some large enterprises, there may be more than one hundred different occupational groups within an overall software organization."¹⁸ In large projects, you'll have specific individuals for each such role; in smaller teams, each individual will end up playing multiple, simultaneous roles.

Improving Team Productivity

Staffing a project with the right people who have the right skills is important, but that alone does not explain the differences in productivity one sees among such teams. In this context, DeMarco and Lister speak of a "jelled team" which they define as "a group of people so strongly knit that the whole is greater than the sum of the parts. The productivity of such a team is greater than that of the same people working inunjelled form.... Once a team begins to jell, the probability of success goes up dramatically."¹⁹

Essential to the formation of jelled teams is this precondition: a project must honor and respect the role of every one of its developers. This means that each project must recognize that its developers are not interchangeable parts, and that each brings to the table unique skills and idiosyncrasies that must be matched to the needs at hand and calibrated within the organization's development culture. This is one of the five basic principles of software staffing that Boehm describes: "fit the tasks to the skills and motivation of the people available."²⁰

DeMarco and Lister suggest a simple formula for creating a jelled

team:²¹

- get the right people
- make them happy so that they don't want to leave
- turn them loose

They go on to note ways to develop an organizational culture that encourages jelled teams to develop and flourish:²²

- make a cult of quality
- provide lots of satisfying closure
- build a sense of eliteness
- allow and encourage heterogeneity
- preserve and project successful teams
- provide strategic but not tactical direction

Teams and Tools

When projects were simple and teams were small (often involving teams of one), an organization could get by with only the simplest of tools: an editor, a compiler, and a linker would be quite sufficient for many problems. Add a debugger, and you were really rocking.

Amazingly, lots of organizations still get by with only this minimalist set of tools. However, in the context of contemporary software development, that's akin to banging rocks together to light a fire: you can do it, but it's not particularly efficient. Rather, as the complexity of your project grows, you must consider other, more targeted tools, such as tools for requirements capture, visual modeling, configuration management and version control, performance analysis, and testing.

Personal integrated development environments are important in making the individual programmer productive. Indeed, there's been a long history of maturation of such tools, and their advances have been essential in enabling the creation of larger and more complex systems. However, there are limits to the cool features you can provide to the individual developer, and it would appear that we as an industry are getting close to those limits. Indeed, there is only so much you can do to help an individual developer bang out code faster.²³

A recent trend in the industry has been the integration of such tools, tools that individually address point problems but that collectively cover the full spectrum of lifecycle activities. That's certainly a predictable advance, but it's not necessarily the most important one. A greater improvement in the overall productivity

will only come from tools that empower the team as a whole. You must integrate your team, not just your tools.

Ed Yourdon points out that "the only way such a tool could be a silver bullet is if it allows or forces the developers to change their processes."²⁴ In other words, while it's important to supply your development team with good tools, you'll only see a state change in your team's productivity if you apply tools that encourage and enforce a sound development process.

The Next Generation of Team Tools

Yourdon's comments suggest three trends that will likely drive future software development tools.

First, there is the need to unify a team's tools around a common process and a common set of artifacts. Development teams create and modify artifacts such as requirements, plans, models, code, components, tests, and so on. They employ tools to create and modify those artifacts. Insofar as those tools are clumsy to use or get in the way of manipulating those artifacts, they detract from the primary focus of the development team, namely, deploying quality systems in a predictable and sustainable fashion. That means that, across the set of tools used by the development team, they must share and preserve a uniform understanding of the semantics of these artifacts and the activities that manipulate them.

Second, there is the need to optimize a team's tools to the different skill sets that exist within the team, since many of a project's artifacts will be created and manipulated by different stakeholders. For example, a requirement may be created by an end user, elaborated upon by the project's architect, and referenced by the project's quality assurance personnel. Each one of these stakeholders has a different view into the artifacts of the project. As such, it would be suboptimal to provide a "one size fits all" development environment. Rather, it's far better to provide a tool set that permits different, simultaneous views into a project's artifacts, optimized to the needs of each individual stakeholder.

Third, there is the need to simplify the delivery of these tools, especially for development teams that are distributed in time or in space. Practically, this means we'll likely see such toolsets become Web-centric, since the Web is the quintessential common vehicle for providing access to and visualization of information. Already, we find projects that use the Web as a repository for all of their project's artifacts. Open communication is a key enabler in forming a jelled team, and making all such artifacts visible to the project facilitates that kind of communication.

Growing a Team

All that being said, how do you grow a team? Even if you've loaded up your team with all the latest and greatest tools and techniques, what must you do next to turn a disjoint set of individuals into a jelled team whose productivity is greater than the sum of its parts?

There are three pragmatic techniques that I've seen work.

First, identify clear roles and responsibilities suitable to your development culture and necessary for your particular domain, and then match individuals with the right skills to those roles.²⁵ For example, perhaps the most important role you need to identify is that of architect: an architect is the person or persons responsible for establishing the significant design decisions for the system and for creating and validating a suitable architectural style. An architect may not be your fastest or most clever programmer, but he or she must certainly be your most wise.

Second, consider the essential artifacts of your project, and organize your team around the set of activities that produce and manipulate those artifacts. Clearly, the most important artifact of the software project is the running system itself, but that alone is insufficient: the team must create a scaffold of other artifacts around that system in order to build it in an efficient and predictable fashion. This means selecting what other work products you need — such as software architecture documents, test plans, releases, and so on — and then establishing a plan for growing and evaluating those artifacts. For example, in an iterative style of development, it's important to drive each iteration according to essential use cases (which specify the desired behavior of the system and additionally serve as test cases) and according to project risk (which changes with each iteration, and may be manifested as technical, economic, or business risk). By focusing on artifacts such as these in a controlled and measured way, you create an environment that encourages your team to drive their work to closure and to focus on the most important things they can do at each moment to mitigate the risks of the project.

Third, leverage tools that let each individual manipulate these artifacts in a manner appropriate to their specific role and consistent with other roles, and in a manner that reduces the interference between individuals. This is what the emerging generation of team tools is all about: providing an environment that encourages individual skills and enables those individuals to work productively and cooperatively.

Wrapping Up

These are indeed interesting times. The challenges of software development are certainly not going to go away, for we as an industry are continually being driven to do more with less. Methods and processes help; so do languages, frameworks, and point tools. However, software development is ultimately a human endeavor, and as such it's ultimately the efforts of the software development team that enable us to deliver quality systems in a predictable and sustainable fashion. Tools that unify, optimize, and simplify the work of that team represent the next state change in helping create jelled teams.

Bibliography

Boehm, B. 1981. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall.

Booch, G. 1996. *Object Solutions: Managing the Object-Oriented Project*. Menlo Park, CA: Addison-Wesley.

Brooks, F. 1995. *The Mythical Man-Month: Essays on Software Engineering, anniversary edition*. Reading, MA: Addison-Wesley.

DeMarco, T. and Lister, T. 1987. *Peopleware: Productive Projects and Teams*. New York, NY: Dorset House.

Gilb, T. 1988. *Principles of Software Engineering Management*. Wokingham, England: Addison-Wesley.

Glass, R. 1998. *Software Runaways: Lessons Learned from Massive Software Project Failures*. Upper Saddle River, NJ: Prentice Hall.

Humphrey, W. 1989. *Managing the Software Process*. Reading, MA: Addison-Wesley.

Jones, C. 1996. *Patterns of Software Systems Failure and Success*. London, England: Thomson Computer Press.

McCarthy, J. 1995. *Dynamics of Software Development*. Redmond, WA: Microsoft Press.

McConnell, S. 1996. *Rapid Development: Taming Wild Software Schedules*. Redmond, WA: Microsoft Press.

Royce, W. 1998. *Software Project Management*. Reading, MA: Addison-Wesley.

Shneiderman, B. 1980. *Software Psychology: Human Factors in Computer and Information Systems*. Cambridge, MA: Winthrop Publishers.

Weinberg, G. 1971. *The Psychology of Computer Programming*. New York, NY: Van Nostrand Reinhold.

Yourdon, E. 1997. *Death March*. Upper Saddle River, NJ: Prentice Hall.

References

1. Weinberg, p. 3
2. Humphrey, p. vii.
3. Weinberg, p. 69
4. Brooks, p. 31
5. Booch, p. 190
6. Royce, p. 43
7. Royce, p. 43
8. Boehm, p. 642
9. DeMarco, p. 45
10. McConnell, p. 275
11. McConnell, pp.278-279
12. McConnell, pp. 304-313
13. Brooks, p. 29
14. Brooks, p. 256
15. Booch, p. 200
16. Booch, p. 197
17. Booch, p. 207

18. Jones, P. 215

19. DeMarco, p. 123

20. Boehm, p. 669

21. DeMarco, p. 93

22. DeMarco, p. 151

23. As an aside, I've often said that the best way to accelerate software development is simply by writing less software. That's why object-oriented techniques and component-based development are useful: the former encourages you to write less code via inheritance and abstractions that form a balanced set of responsibilities, and the latter encourages you to reuse and adapt existing components and frameworks rather than writing your own from scratch.

24. Yourdon, p. 183

25. For example, the Rational Unified Process specifies a common set of roles, which it calls "workers".

Rational's solution to team development ... Rational Suite

Rational Software Corporation is addressing the topic of team development through Rational Suite, a complete software lifecycle development solution designed for both Microsoft Windows and UNIX platforms. Through this solution, organizations can easily adopt a single set of best practices for all of their project teams; a philosophy that is crucial for any organization committed to producing the highest quality software possible in today's changing, business-critical software development climate.

One of the best features about Rational Suite is its versatility, as it adds value whether customers use all or some of the suites. Developers already using some products within the Rational Suite will find that the deployment of the entire suite empowers the entire project team, from analyst to tester. Rational Suite also provides an ideal way for developers new to Rational to give their organization an integrated suite of proven, market-leading products that lay the groundwork for automating the entire development lifecycle. Everything from the initial acquisition and deployment process, through to training and support is made easier with the convenience and comfort of working with one world-leading organization.

Rational is the first company to deliver an integrated solution for the entire cross-functional software team. According to Rational VP Eric Shurr, "We are very proud of the philosophy of team development that is at the heart of Rational Suite. Through this complete lifecycle development solution, you can unify analysts, developers, and testers by breaking down the barriers that normally exist between cross-functional teams. You can optimize each practitioner's performance by delivering the right market-leading tools to each member of the team. Plus you can simplify the solution - and lower your total cost of ownership."

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

[Magazine](#)

> [Back Issues](#)

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

[Scripts & Models](#)

[Product Reviews](#)

[Book Reviews](#)

[About Rose Architect](#)

[Contact Us](#)



[About Rational Rose](#)



[About the UML](#)



Feature



Stupid Rose Tricks

by Douglas Earl

Doug Earl has been a Rose developer long enough to remember when Rose fit onto a single floppy disk. He works out of Rational's Milwaukee office. He has two children, ages 7 and 4.

Over the years, Rose has grown to include many features, some of which are not readily apparent to new users. This article points out some of the tricks, shortcuts and lesser-known features of the core portion of Rose (e.g. the diagram editor and specification dialogs, but not code generators or reverse engineering tools). If you are already a Rose user, you probably already know many of these, but most readers will probably find at least one or two new things here.

Editing tricks

Probably the most basic gesture in Rose is placing an item on a diagram. The most common way to do this is to click on the appropriate icon in the toolbox, then click on the diagram surface at the desired location. But what if you want to place five of the same kind of item? By default, you have to reselect the toolbox icon for each item placed. But there are two ways to speed up the process. The first way is to use the shift key while placing the item. As long as the shift key is held down, you can place another of the same kind of item in the diagram. When the shift key is released, you will be put back in select mode. The other way is to use the "lock mode" of the toolbox. When the toolbox is in lock mode, after placing an icon you remain in placement mode until the arrow tool is selected. How do you enter lock mode? There are two ways: right mouse click anywhere on the toolbox and select "Lock Selection" to toggle the mode, or customize your toolbox to include the lock icon, which also acts as a toggle. By the way, you can customize your toolbox by right clicking anywhere on the toolbox and selecting "Customize...". From the resulting dialog, you can remove little used icons or add them back in.

O.K. let's say you're speeding along and you've used the shift key to place five new classes in your diagram. By default, Rose gives

each class a name, something clever like "NewClass1". Now you want to give each class a meaningful name. To change the name, simply click on the class in the diagram and start typing the new name. Since Rose realizes that the class hasn't been named yet, it automatically puts you in edit mode, erases the "NewClass1" name and replaces it with what you type.

Now, add a few attributes and operations to your classes. Each attribute and operation will be assigned a default visibility (public, for example) which is denoted by a little icon to the left of the name. If you want to change the visibility, there is no need to bring up the appropriate specification sheet; rather you can do it directly from the diagram. Do this by selecting the class, then left click on the visibility icon to bring up a menu of choices.

After you've named your classes and added attributes and operations, perhaps you realize that you really wanted one of the classes to be a parameterized class. No need to delete it and recreate it, rather just select it and use the edit/change into menu item to change it. Or, you can use one of the more esoteric features of Rose that I would guess even the most seasoned Rose users don't know about: select the parameterized class icon on the toolbox, press the Alt key and click on the class you want to change. Presto-chango! It's now a parameterized class. This feature also applies to relationships, e.g. if you want to change an association into an inheritance relation. Just select the desired ending icon from the toolbox and click on the diagram element you want to change while holding the Alt key.

Speaking of the Alt key, Rose 98i has another somewhat obscure, but helpful use of it. On sequence diagrams, press the Alt key when you want to move a message *and* its originating focus of control (the long thin box that the message is drawn from) up or down. If the Alt key is not used, then only the message is moved and the focus of control stays fixed. Note that this only works if there are no incoming messages to the focus of control.

Navigation to Referenced Items

Many elements in Rose reference other elements in the model, for example an object on an interaction diagram may reference the class of which it is an instance. You can use the "browse referenced item" feature of Rose to navigate to the thing being referenced. For example, create a class, create an object on a collaboration or sequence diagram, make the object an instance of the class from the object's specification dialog. Now that the reference is set up, you can select the object in the diagram, and select the "browse/referenced item" menu item (or ctrl-R) which will take you to the diagram where the class appears. Browse referenced item

also works for a message on a sequence or collaboration diagram to navigate to the referenced operation's specification dialog (note you must select the message name, not the arrow). To set up the reference to the operation, select an operation from the message's specification dialog or right click on the message arrow and select an operation from the popup list. By the way, this list is a multi-select list on collaboration diagrams - select another operation from the list to add another message to the same message arrow. Select an operation which already has a check mark to remove it.

Finally, you can navigate from a class which is imported from some other package (indicated by "(from packageName)" shown below the class name) to the diagram that the class is defined in. For example, in a package, MyPackage, one of your classes inherits from an MFC class, say CString. Since CString resides in a different package, it will show the owning package beneath its name, e.g. "(from MFC)". If you want to see CString in the package where it is defined in (perhaps to learn more detail about CString such as what class it inherits from or what classes it uses) select Ctrl-R to take you to the diagram in the MFC package that it appears in. If there is more than one diagram that displays CString, the first one that Rose finds will be used.

Drag and drop

In the object reference example above, there is an easier way to create an instance of a given class on an interaction diagram (i.e. a sequence or collaboration diagram). And that is to use drag and drop from the browser. Drag the desired class from the browser and drop it in the desired interaction diagram. If you drop it on an open area, a new unnamed instance of the class will be created (denoted by ": ClassName" in the object name). If you drop it on an existing object, that object will become an instance of that class.

Drag and drop works for many other sources and targets. For example:

- **From specification to specification.** Drag attributes or operations to move them from one class to another. Use Ctrl-Drag to make a copy instead of performing a move.
- **Between specification and browser.** As above, drag or Ctrl-Drag to move or copy attributes and operations.
- **Within a specification.** Drag attributes and operations and drop them in the same list to reorder them. (The new order is reflected on classes in the diagram.)
- **Within the browser.** Again, move or copy attributes and operations. In addition, move diagrams from one package to another, relocate elements (e.g. a class) from one package

to another, and assign classes to components.

- **From diagram in the browser to a note.** This creates a link to the diagram. Double-clicking on the note will bring up that diagram.
- **Text from external applications to a diagram.** Drag selected text from an external application, such as Microsoft Word, into any diagram to create a note containing the selected text.
- **Between inheritance relations.** Drag the arrow end of an inheritance relation (generalization) and drop it anywhere on another inheritance to create a "tree" or "comb" style layout.

Use Copy/Paste to Override Base Class Methods

Besides using Ctrl-drag to make copies of attributes and operations as mentioned above, you can use copy/paste. This is especially useful in the operations tab of the class specification in order to override base class operations. For example, if class MySub inherits from MyBase and you want to override virtual methods defined in MyBase:

1. Bring up the operations tab on the class specification for MySub.
2. Make sure "show inherited" is checked.
3. Select the operations from MyBase that you want to override.
4. Right click to bring up the context menu - do this twice, first to copy and then again to paste.

By the way, one of the new features in the 98i release of Rose is that you can resize the specification dialogs. You may not have noticed since the frame of the dialog doesn't look any different.

Copy/Paste/Delete in diagrams

When it comes to copy, paste and delete in Rose diagrams, it's important to understand the concept of model vs. view. Rose allows you to have multiple views of an underlying model element. For example, you can have a class appear many times on multiple diagrams (or even the same diagram). Each class will be a view of the same underlying model item. To see this, try placing two classes on the same diagram and giving them the same name. You can verify that they are two views of the same underlying model by changing the name of one of the classes in the diagram (or change it in the browser) - both classes in the diagram will update to show the same name.

Because of this separation of the view and the model, Rose needs to provide two ways to delete an element: one to delete something from the diagram only (a shallow delete), and another to delete the diagram element and the underlying model element as well (a deep delete). Continuing with the example above, select one of the duplicate classes and hit the Del key (this performs a shallow delete). This removes that class from the diagram only, but it still exists in the model (as you can see by the fact that the class is still listed in the browser). Now undo the delete and try it again, but this time deep delete it by selecting "Delete from Model" (Ctrl-D) on the edit menu. Notice that both classes are removed from the diagram since the underlying model element has been removed. Long time Rose users will be happy to find that the ability to shallow delete now also applies to relationships in 98i.

How about copy and paste in terms of model and view?

Interestingly enough, Rose supports both deep and shallow pastes, depending on the situation. Normally, when you copy something in a diagram, and then paste it, Rose simply creates another view of the thing that was copied instead of creating a duplicate model element of that thing. But the copy that Rose performs is deep, i.e. if you copy a class, all its attributes, operations, etc. are copied as well. When you paste, Rose checks to see if a model element with the same name already exists in your model. If so, it hooks up the copied view to the existing element and throws away the copy of the model. If not, it does a "deep paste" causing a new model element to be created. There are a couple of ways you can take advantage of this:

1. Duplicate a model element by copying it, renaming the original, and then pasting. Since the original has been renamed, Rose has nothing to hook up the pasted view to, so it creates a new element in the underlying model.

2. Copy an element from one model to another by copying the item(s), using file:open to open another model (or file:new to start a new one), and pasting. Assuming there is no model element with the same name in the second model, the pasted element will be added to that model.

That's it for now. Hopefully you found something in this article that you didn't already know, and if so, hopefully you can put it to use.

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

Magazine

> **Back Issues**

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

Contact Us



[About Rational Rose](#)



[About the UML](#)



Rational Product Integration Focus: Rose and the Rational Unified Process

Electronic Coach in a Box

by Per Kroll

Per Kroll has spent the last 7 years assisting companies in implementing best practices, as consultant, mentor and trainer. Per is currently marketing manager, responsible for marketing and product direction of the Rational Unified Process. He can be reached at pk@rational.com.

The best tools in the world are useless if you don't know how to use them effectively. Take the UML for instance. Do you know when in your project to use which UML diagram? Or Rose. What is the best way for you to structure your Rose models? Helping you to fully realize the benefits of these and other tools is the goal of the Rational Unified Process. This e-coach is a searchable knowledge base that gives you the guidelines, templates, and tool mentors you need to get the most out of the industry leading tools and the UML.

The UML has provided us with notation and semantics that allow us to visually communicate requirements, architectures, and designs. But it's just a language; it doesn't tell us how to develop software. So in parallel with unifying the different modeling languages on the market, Rational has been driving the unification of process as well. We have unified different disciplines such as requirements management, visual modeling, configuration management, functional testing, performance testing, and business modeling into one consistent full lifecycle process (**Figure 1**). The product, Rational Unified Process, is the result of this work, and is designed to more effectively harness the power of the UML and tools such as Rose. A part of this unification work has been to unify also the approaches of the Amigos, Grady Booch, Ivar Jacobson, and Jim Rumbaugh.

Using the UML

So how can the Rational Unified Process be used to help your development efforts? Well, suppose you're working on a project to design an e-commerce application. You've just been given a use case for which you are to supply the design. Where do you start? With analysis and design, of course, which means you take a look at the Activity Overview - Analysis & Design. ([Figure 2](#))

In this view, the activities that will be of most interest to you are use case analysis and use case design. If you click on the activity Use Case Analysis you find that the activity is divided into a number of steps. One of these steps is Find Classes from Use-Case Behavior. ([Figure 3](#)) In this step you will find extensive guidance in analyzing a use case and coming up with appropriate classes using some of the stereotypes defined in the UML process extension <<Boundary>>, <<Entity>> and <<Control>>.

Tool Mentors

Where does Rose fit into this picture? To answer that question, the RUP has a feature called *Tool Mentors*. Tool Mentors describe how to use a tool in context within the Unified Process. In this case, there is a Tool Mentor for Rose which describes exactly in what order you pull down menus, and what information to include in which dialog boxes. Tool Mentors ensure that each practitioner not only understands the thought process, but also understands how to carry out the necessary day-to-day activities to use the tool to its maximum capability. It's about process made practical.

Looking up a UML Concept

Suppose you are continuing to work on your model, and you reach a point where you are uncertain whether to use an association or an aggregation between two classes. The Rational Unified Process, your e-coach, can help.

In your e-coach, go to a graph showing all the UML concepts used in Analysis and Design and click on Aggregation. There are subtopics available at this point to explain when to use Associations and when to use Aggregation ([Figure 4](#)). In a similar fashion, you can get guidelines for how to use the most popular and useful UML concepts.

Structuring your Rose Model

One thing I have noticed when I go out on projects is the difficulty people sometimes have in developing a good structure for their Rose model. What package structure should you set up? What naming convention should you follow? To address this, the Rational

Unified Process ships with a Rose Template which uses a set of packages within the standard Rose "views" to organize the artifacts defined by the process and which can be managed using Rose. In addition, a set of naming and modeling conventions are suggested which will make the task of organizing the model easier. Rose model examples are included for additional clarification.

Context-Sensitive Process Guidance

Simplicity is essential when searching for crucial information. When working on a deadline, you don't want to have to waste time looking for the right answers. The Rational Unified Process understands this, and provides you with easy ways to find information you need to work more effectively. For example, often when you are working in Rose, you want to check out concepts, activities and steps that are relevant to a specific diagram. This is simplified through the concept of Extended Help. Extended Help is a menu added to Rose that will display relevant process topics based on the context from which it is launched. Extended Help can also include other topics, including topics added by your project team. [Figure 5](#) shows an example of the process as launched from the Use-Case Diagram.

Documentation Assistance

Rose provides excellent support for developing a software architecture, from which it is often necessary to extract information and present it in a document. Not sure how to do this? Ask the Rational Unified Process, which ships with a large number of document templates covering all aspects of the development lifecycle. In this instance, start with the search engine to see what information is available. By searching for *Architecture*, you get a list of topics, beginning with the artifact Software Architecture Document (SAD). You can now select either a Microsoft Word template, or a SoDA template. (SoDA, for those who are unfamiliar with the product, is a document automation tool from Rational Software that can integrate with various tools, such as Rose, and extract information.) In this case, the SoDA template will automatically select all information that is architecturally significant. By using this process, you can easily create a SAD which will always contain that subset of your Rose model which you would like to present. This information can then be easily shared with other members of your team, or with those members of your company that need to understand the overall architecture of your project.

Summary

The Rational Unified Process is integrated with most Rational tools.

It is a e-coach that enables users to get the most out of both the UML, and Rational Rose. Customers such as Ericsson Telecom, Capital One, and Shared Medical Systems have all made the Unified Process an integral part of how they develop software. To find out more about it, please visit <http://www.rational.com/products/rup/>, where you can order a free multimedia CD *Exploring the Rational Unified Process*.

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

Magazine

> **Back Issues**

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

Contact Us



[About Rational Rose](#)



[About the UML](#)



Rose Real Time — The New Standard in RealTime Modeling

Or...Zen and the Art of Telling Knock- Knock Jokes

by Matthew Drazhal

Matthew Drazhal is Director of Product Management for Rose RealTime. A non-apologetic New Yorker and a former radar software developer, Matthew has often been described as somebody with "a firm grasp of the obvious." Matthew still looks upon the cold-war as the "good-old days," and can be reached at matt@rational.com.

I have a confession to make: I really love spending time with Rose (please don't tell my wife...). Using Rose has shaped the way I approach and think about software development. When some people try to crack a software problem, they reach for a whiteboard, or their favorite code editor. I reach for Rose.

This brings me to a transition problem. Since I would rather use Rose than fight with a coding environment, I rarely finish the jobs I start; rather, I find a new problem to tackle using Rose. This was a repeating pattern — until now.

This spring, Rational announced a new member of the Rose family: Rational Rose RealTime. RoseRT is a specialized version of Rose, developed specifically for the RealTime embedded development community (and since I built radar system software in a past life, I am, at least, an emeritus member of that community).

Rose RealTime has broken new ground in UML modeling; the development of Rose models which can be fully developed, compiled, animated, tested, and debugged, *all without ever leaving Rose*. In case you missed my emphasis, I will rephrase that. When using RoseRT, your UML design model becomes your deployable application. We have implemented, in essence, a UML compiler, which takes your UML models and generates executable applications.

UML for RealTime Development

This seems too good to be true, until you sit back and consider how powerful the UML really is. (Grady, Jim and Ivar really know their stuff). Using stereotypes and design patterns, along with a healthy dose of State diagrams, we really can develop complete applications from RoseRT. The stereotypes and design patterns for RoseRT are based upon the joint research done by Rational and ObjecTime Ltd. (specifically Jim Rumbaugh and Bran Selic) in defining and specifying these constructs. For additional details on this work, you can refer to their joint [paper](#) available at the Rational Web site.

Stereotypes for RealTime Developers

We should pause to take a moment to be thankful for Dr. Jacobsen's explaining to me the concept of a stereotype. It is what enables the Rose/Visual Basic developer to model a form, or a C++ developer to model a collection class. Stereotypes are the mechanisms we use to take a general design language like UML, and create the kind of system designs we are accustomed to.

Similar to the above examples, RoseRT understands some new Rose Stereotypes which are quite useful to RealTime designers. The two principal new stereotypes are *capsule classes* and *protocols*. These stereotypes provide us with the ability to implement design patterns which provide a superb architecture for event-driven software development, and to build a reliable, reusable set of developed components.

Capsules are a stereotype of class, and are the fundamental modeling element of RoseRT. A capsule represents independent flows of control in a system. Because they are a stereotype of class, capsules have much of the same properties as classes; for example, they can have operations and attributes. Capsules may also participate in dependency, generalization, and association relationships. However, they also have several specialized properties that distinguish them from classes. See [Table 1](#).

Before we can talk about these differences, we must also understand that other stereotype of collaboration called "protocol". The set of messages exchanged between two objects conforms to some communication pattern called a protocol. It is basically a contractual agreement defining the valid types of messages that can be exchanged between the participants in the protocol. To demonstrate how powerful this protocol concept is, I once heard Ivar Jacobsen state that the world's telecommunication systems are the largest distributed computer ever built, and it is through the magic of contracts (a.k.a. protocols) that we can make a call

from New York to Stockholm.

So, now that we understand that a protocol specifies messages between capsule classes, we can talk about the design patterns we use for capsule classes.

When we think of a capsule, we should think of a class which receives messages, and responds to these messages via the behavior specified in the State Transition Diagrams which we define in Rose. The container in which these messages are defined is specified as a protocol, and these protocols are received over a port (which is, by the way, is a stereotype of a UML association).

In addition to this, capsule classes provide a very lightweight modeling element for breaking a problem down into multiple threads of control. Each capsule instance has its own logical thread of control, though it may share an actual processing thread (known as a "physical thread") with other instances.

Example

If you are not already asleep, I would like to go through a quick example to show you how all these new stereotypes work together to solve a real-world problem. Unfortunately, setting up a real world problem might take the remainder of this magazine, depriving you of *Magnus Opus* and Terry Quatrani's *Letters to the Editor* columns. Instead, I will use the concept of a "knock-knock joke" to demonstrate these concepts. For non-native English speakers, the explanation of the joke is as follows. See [Table 2](#).

Now, we can begin this with a sequence diagram which documents the interaction between our two people above. This sequence diagram ([Figure 1](#)) shows the complete interaction which documents the protocol between our two actors.

In this diagram, I have added the state of "laughing" to show that we can consider the state of each object while we are thinking about the interactions between these two objects. In reality, there are many more states in these objects than are visible in [Figure 1](#). In fact, in this model, the reception of any of the above messages causes the state of the message receiver to change.

As we further work on this model, we may decide that we have other scenarios we must work out, and we may need to deal with odd error conditions¹. Certainly, we must deal with "rainy-day" scenarios as well as the above "happy-day" scenario. Unexpected interrupts and out-of-sequence messages are often the hallmark of a real-time system, and must be designed for².

In order to keep things simple, we will work with the above scenario, begin to define our capsule classes, and also define each capsule's response to the message traffic above. In all cases but one, the appropriate response is to send the next part of the knock-knock protocol to the other participant. The post-punch line response is specified as send the response and laugh one second. (Or, in more technical terms, send the response, set a one second timer, and start laughing. When that timer expires, stop laughing.)

I have defined two capsule classes to play the roles of the two people above, the Comedian and the StraightMan. I have created two State Transition Diagrams to define and document the behavior I expect each to play in the interaction (**Figures 2 and 3**). In RoseRT, state transition diagrams are very important to understand because actual executable code is generated from these diagrams.

In **Figure 2**, the Comedian enters the Ready state by sending a message to the StraightMan indicating that the Comedian has a joke to tell. The trigger for the transition from the Ready state to the GotTellJoke state is the TellTheJoke message from the StraightMan back to the Comedian. Reception of that message will cause the Comedian to send a Knock-Knock message to the StraightMan. The trigger for the transition from the GotTellJoke State to the WhosThere State is the GotWhosThere message from the straight man back to the comedian. This continues to the end of the joke.

This may be a little hard to understand when viewing only one side of the behavior, so let's include the StraightMan's State Transition Diagram to display his part of the interaction (**Figure 3**).

The sequence of state transitions in **Figure 3** is the response to the Comedian's activity in **Figure 2**. In this diagram, the StraightMan starts in the Ready state. The receipt of the GotAJoke message causes the StraightMan to respond "Tell the joke" and enter the ReadyForJoke state. Subsequently, the next state change occurs when the KnockKnock message is received.

For the StraightMan, this continues until the Laughing state. At this point, the StraightMan sets a one second timer and starts laughing (the logic to do this is in the specification sheet for each state and transition). The trigger for the return to the Ready state is the expiration of the timer. Oh yes, the StraightMan will also stop laughing at that time, preventing him from laughing eternally, and being considered the InsaneMan. When the StraightMan enters the Ready state, he is all set to receive the next joke.

Class Diagram

It is now time to discuss the resulting class diagram that we created based upon the analysis of the system. This class diagram will show the static structure of the system, and will display the parts of the Knock-Knock system so that we have a good understanding of how the software is architected.

In [Figure 4](#), we have the RoseRT model for architectural layout of our system. In this analysis, we have a capsule called a "KnockKnockJoke". This capsule is comprised of two other capsules, Comedian and StraightMan, both of which we know way too much about already. The interesting addition to this is the addition of the protocol TheJoke, which encapsulates the message traffic between the Comedian and StraightMan. In fact, if you look at the diagram, the only way that our players interact with each other is through this TheJoke protocol.

Compiling, Linking, and Running

Finally, after all this is complete, we can build our application. We do this by creating a component which contains all the above classes, selecting the component in the browser, and right-mouse clicking compile. RoseRT then creates the make file based upon the platform (I am running on Windows NT), generates the C++ code for the application (based upon structure, state transition diagrams, and transition and state action logic), and initiates the C++ compiler to generate the code. In addition, any errors which you may have introduced are validated and presented for your correction.

[Figure 5](#) shows the results of this compilation. From the results, we can see that although we compiled successfully, we still have some work to do improving our design. RoseRT has indicated a possible logic error which we may be interested in, the EndOf-Joke state has no outgoing state, and the application will hang at that point once it reaches that state. In our robust, final system, we should carefully consider warnings such as this.

Now that I have compiled, I can test this application on my host (Windows NT, Solaris, or HP-UX) before deployment onto my target system (RoseRT supports many of the popular real-time operating systems). I can run my application from RoseRT, set breakpoints, animate my RoseRT model so that I can watch how the application is running, and even enter code debuggers for more low-level information. I can perfect my application on my development platform, before my target computer has even been delivered.

Finally, I can use my deployment diagram to drag and drop my components to the target platform. I can ask RoseRT to build for the target platform, and then I have access to all the animation and run-time monitoring features in RoseRT that I used while I was still running on my development host.

Summary

Earlier, I said that I wished I never had to leave Rose to develop my applications. I wanted to design, develop, debug, and deploy my apps all from within Rose. I wanted to think in terms of UML, not in terms of a programming language. In that regard, RoseRT is a product that fulfills all my wants and fantasies³ (and gives me a few capabilities I never dreamed of).

I have tried to give you a small sampling of the way Rational Rose RealTime lets you think about how to design and develop concurrent, event-driven applications. I am hopeful that you are just as excited by the possibilities as I am about the realities.

References

1. For a fun time, tell somebody "I have a great knock-knock joke, you start", and smile after that person says "knock knock" and you reply "Who's there?".
2. The Windows NT "Blue-Screen-of-Death" is usually an inappropriate response in most deployed, robust systems.
3. Once again...please don't tell my wife. :)

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

Magazine

>Back Issues

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

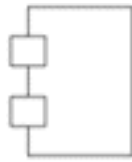
Contact Us



[About Rational Rose](#)



[About the UML](#)



Feature



Christian Buckley is co-founder of QOSES.



Darren Pulsipher is co-founder of QOSES.

Overcoming Father Time: Parallel Development through Component Engineering

by Christian Buckley & Darren Pulsipher, QOSES

There's never enough time to do it right, but there's always enough time to do it over." — Jack Bergman

Recent months have seen numerous opportunities to work on complex re-writes of legacy systems. Isn't everyone doing this right now? The whole Y2K problem has given many engineers the opportunity to "re-write" the code the way they wanted to write it the first time. (Well, that's what we'd like to think.) How is Y2K affecting your project schedules? Typically, there are three project management variables that can be used in software or system development efforts to regulate project fluctuations: time, resources, and money. With Y2K, however, time is no longer a variable. Last time we checked with Father Time, he would not let the year 2000 slip — no matter how hard we pushed. As for resources and money — they've always been scarce. So... when you are faced with a development schedule that seems impossible, what do you do? Even without the Y2K problem, time schedules have become increasingly tight. The proverbial "window of opportunity" has become smaller and smaller as time-to-market demands have grown. Competition has increased and qualified engineers have become increasingly

difficult to secure. So what can you — as the VP of Engineering — do? Hire more people to come up with a solution? Have your existing team work longer hours? Bring in an outside consulting team to "fix" your systems? Scrap your entire system and start over again? None of these are reasonable solutions. You need to solve your problems in a timely manner with your current resources and within your budget parameters. In a nutshell, you need to find ways to enable your engineering team to work in parallel.

Defining the Layers of the System

Take a look at how many companies — and your own company in particular — approach problems. Do different divisions or business units define a problem from their individual perspectives, without taking into account the "big picture"? Some companies have recognized this approach to problem solving as a weakness, and have taken steps to reverse segmented business practices. Many fall well short of what is truly needed, unleashing instead a series of well-intended policies, requirements meetings, and joint application design sessions that do not solve the problems. "Concurrent engineering" is about as overused and draws as much excitement from development teams as "cross-functional", "paradigm shifts", and "total quality management". Somewhere behind the well-intended jargon hides the formula these organizations need to move forward, but they just can't break the cycle. As a result, time is lost. Time *you* can't afford to lose. What most organizations lack is a process for communication and an understanding of how to develop software and systems in parallel. You cannot afford to hire more people, even if you could find them. A consulting group is probably not going to do much better, what with ramp-up time and a tight budget. And scrapping the system is not a viable solution. You need to work *smarter*.

There are several approaches to parallel development. One approach is to work on several releases at the same time. Popular tools like Rational's ClearCase can aid your software team in setting up parallel branches of development.

Some of our customers have tried and failed to develop multiple releases of a product at the same time. The largest problem with this simplistic approach is determining 1) where defects or enhancements are implemented, and 2) when it is propagated to the other releases being developed concurrently. Many companies have already learned that they cannot sustain the level of overhead required to merge code to and from their releases.

Another approach to parallel development is to define specific layers of the system. [Figure 1](#). Breaking the system into multiple *components* — to be built individually — can increase parallel development dramatically, but it too must be carefully planned and monitored.

Component(ize) the Architecture — Component(ize) Engineering

The key to parallel development is the identification of logical divisions in the software or system, dividing them into separate components that can be developed autonomously. Think of each component as a system unto itself, and follow the same development techniques you would use on the *complete* system. Begin by defining the component responsibilities and interfaces. Devise use cases and scenarios — derived from the complete system use cases and scenarios — to define the responsibilities and interface of the system.

Keep in mind that the complete system use case analysis must be completed *first*. The second step is to have the system architect define the component boundaries and the interfaces between components. Once the component boundaries and interfaces have been defined, the torch can be passed to others. Small teams of engineers can then take each component to work on as their "system". The interfaces should be the first to be implemented. The IDL (Interface Definition Language) is a great language for this step. Remember that any change to the interface will mean coordination between your group and other groups, and will require coordination through the system architecture group. (After all, they need *something* to do.)

Each team of engineers should develop their component as if it were a standalone product — one that is ultimately delivered to a customer. Okay, the customer just happens to be the group in the "cube farm" next door, but they should be treated as a customer all the same.

Since each component will go through several releases, coordination is critical. The first release of your component will be the interface. This will allow other components to communicate with your component right away. Subsequent releases, however, should be based on a system plan covering the complete system. It is a very logical process. Features required by the complete system should be supported by the separate components of the system. Think about it — it doesn't do any good to develop code that implements a feature that will not be needed until the next major release of the complete system. Your time is critical. If you focus on these low priority

features, you will be guilty of ignoring the features that need to be done for *this* release. Communication is always important — at all levels of the organization — but in this case, a good defect/ enhancement tracking and work task-tracking program will come in very handy.

Your initial focus should be directed toward components with the most dependencies and the highest risk. If a component has several high-level dependencies, it is typically your highest risk component and should be worked on first. However, this is not always the case. A component without the obvious high-level dependencies may use a new technology or language. Therefore, this component may be your highest-risk. While this is a special case and should be considered, the rule of thumb should be to start with the component with the highest dependencies.

Coordinating Component Releases

At the center of any parallel development effort is the coordination of component releases and the dependencies between the components. A strong project management team is essential to your success. Any number of issues or undefined tasks can add dimensions of complexity. If one component schedule slips, what will be the impact on other schedules? How can you minimize risk, avoiding failure of the entire project? The answer is an iterative development approach. If each component has a release cycle that is very short — let's say every two to six weeks — problems will pop up quickly. These problems can be addressed immediately, with resources shifted as needed. A short cycle leads to a tighter focus, improving the process of problem identification and of finding solutions.

[Figure 2.](#)

Another benefit to several short iterations is the adaptability to change. By building change into the process, your team becomes more adept at identifying problems and making adjustments. There is nothing worse than trying to pound a square peg into a round hole because that is what your fuzzy requirements call for. With short cycles, changes can occur quickly - in one iteration.

Points to Consider:

Circular Dependencies — Nothing kills productivity and increases complexity more than having two components dependent on each other. One of the biggest problems is determining what component to build first. We have had customers build the same code twice to make sure that the

circular dependency was handled properly, or so they thought. They seemed to have a problem re-building a particular release of their product. It is the same problem that biologists have been trying to solve for thousands of years: Which came first - the chicken or the egg? [Figure 3](#).

Overhead — If engineers and management are spending all of their time filling out paperwork and status reports, productivity will go down and frustration will increase. Use a good defect/enhancement-tracking tool that is accessible by those who need information about the components and the system as a whole. If you have the ability to look into the system for what you need, you will not have to interrupt someone else to ask.

Too Many Pieces — Be careful not to divide your system into too many components. The overhead of coordination of component development cycles can be costly in time and resources.

Finger Pointing — Watch out. One of the problems that typically arises is the "It's not my code causing the problem" syndrome. Finger pointing can become corrosive to the organization as a whole. Avoid setting up groups to compete against each other for reward, resources, and recognition.

Look at the Big Picture — There is an old Malaysian story, often told by environmental educators, entitled "The Elephant and the Six Blind Men". In the story, six blind men encircle an elephant trying to outdo one another in explaining what the elephant is. One of the men feels the side of the animal and declares "elephants are like walls". Another man grabs hold of a tusk and says, "elephants are sharp and dangerous". Yet another takes hold of an ear and claims, "they are like fans". "They are like soft fire hoses," says another as he handles the trunk. "No, they are like tree trunks," protests another as he holds onto a leg. "They are like whips," exclaims yet another with the tail in hand.

This story is told to illustrate the complexity of the elephant, and to suggest that to properly comprehend a subject we must back off a bit and at least see it from several sides. Each of the blind men walked away with a different understanding of the elephant. While each of their descriptions contained what they believed to be the truth — and from each perspective, it was the truth — none of them had a full understanding of the situation.

Do not fall into this trap. Remember that each component plays a role in the development of a larger system. Know how each

piece fits into the puzzle, and above all, communicate, communicate, communicate.

Getting Started

You now have a better idea of how to approach parallel development. It's all about working smarter, not harder. But this process calls for change - in the way your team operates, and in the way problems are analyzed. By building change into the process, however, your team will become more adept at identifying problems and making adjustments. Just remember these six steps:

- 1) **Identify** logical divisions in your software or system.
- 2) **Divide** your system into separate components that can be developed autonomously and assigned to small development teams.
- 3) **Define** the component boundaries and interfaces.
- 4) **Coordinate** among components.
- 5) **Reduce** risk by focusing on components with the most dependencies.

and *always...*

- 6) **Look** at the big picture.

About the Authors

Darren Pulsipher and Christian Buckley are both co-founders of QOSES, a software and process development company based in Tracy, CA. (<http://www.qoses.com>) The QOSES Web site offers the Internet's largest collection of free Rose scripts and models, as well as Florist, an add-on to the Rational Rose tool that helps in the development of UML models and the generation of effective documentation. Look for Darren and Christian in July when QOSES will be participating in the Rational Software User's Conference in Seattle, Washington.

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

[Magazine](#)

> [Back Issues](#)

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

[Scripts & Models](#)

[Product Reviews](#)

[Book Reviews](#)

[About Rose Architect](#)

[Contact Us](#)



[About Rational Rose](#)



[About the UML](#)



Building JavaBeans™ with Rose J

by Jeffrey Hammond

Jeffrey manages the CORBA and Java teams for Rational Rose. Before joining Rational, Jeffrey consulted on many object-oriented engagements for Andersen Consulting in the pharmaceutical and insurance industry. In addition to Java and CORBA, his interests include design patterns, XML, and an occasional game of golf. He can be reached at jhammond@rational.com

It's hard to believe that only three years ago, if you told a co-worker you were "visualizing some java" they would have assumed you were talking about the beverage, and your desire to make a run to the local Starbucks. As a programming language and development platform, Java™ has developed quickly and is now used widely in many industries. Success stories back up Java's benefits, and cement its place as a major development alternative for distributed, mission critical systems.

One of the reasons Java is so attractive is that it has a robust component model. By creating and using JavaBeans™ and Enterprise JavaBeans™, a developer can create a highly robust distributed system out of reusable building blocks. This article will focus on how to model and generate JavaBeans™ from Rational Rose by using the Rose J Add-In found in Rational Rose J Professional Edition and Rational Rose Enterprise Edition. While the example we will work through is relatively simple, the techniques can be easily be used to develop more complex beans. In order to work through this example yourself, you'll need the following software:

- Rational Rose 98i (Enterprise or Professional (J) versions)
- Java Development Kit 1.2

[Download the model.](#)

JavaBeans Distilled

One of the topics most frequently discussed in relation to the Rose J Add-In is its JavaBean support. This topic can be addressed by taking a brief look at exactly what JavaBeans™ are. If you could distill all the books and specifications about JavaBeans™ down to one sentence, I believe it would read something like this:

JavaBeans™ are Java classes that implement certain well-defined idioms in order to take advantage of services built into the core Java Development Kit.

Within this definition we have an answer to the question above. Since JavaBeans™ are Java classes written in a certain way, Rose J supports JavaBean development out of the box. The real question then is what are these idioms, and how does the Rose J Add-In help you to implement them? We can examine these idioms by working through a simple example of two JavaBeans™ communicating to each other.

ToggleSwitch and Light

Since JavaBeans™ are components, we'll build an example based on components in the real world. A toggle switch and light are components we are all familiar with. We'll work to define our participants, flows, and implementation in Rose, then generate and package the ToggleSwitch and Light classes as JavaBeans™. You can work along with the completed model, or start from scratch by setting Java as your default language (Tools:Model Properties>Edit — Notation Tab), and using the framework wizard to load the JDK 1.2 framework.

Participants and Flow

We'll use an activity diagram, LightBean model, to lay out the participants in this system, and the flow between them. A user enters the room, and flips the light switch. This turns the light on. The user spends some amount of time in the room, and then flips the switch to turn the light off before leaving the room. Our activity diagram establishes a swim lane for each participant. In addition, each participant is mapped to a class in the model. A *User* is an Actor, so we'll create one in the Use-Case View. We'll create JavaBeans™ for *ToggleSwitch* and *Light* in the Logical View. Since packages are an important part of the Java language, we'll create an *electrical* package in the Logical View, and place *ToggleSwitch* and *Light* in this package. [Figure 1](#) is an example of this activity diagram.

Turning a Participant into a JavaBean

In our example, turning *ToggleSwitch* and *Light* into JavaBeans™ involves the application of two idioms. These are:

- Creating JavaBean Properties
- Creating JavaBean Events

Creating JavaBean properties in Rose 98i is very simple. Create an attribute (e.g., *lightOn*) and double-click on the attribute in the Rose Browser. This opens the custom attribute dialog for the Rose J Add-In. At the bottom of the dialog, you have the option of making the attribute a JavaBean property. In this case we'll make it a simple property, and indicate we want it to be read and write. (See [Figure 2](#).) Press OK, and exit the dialog.

Generate code for *light* (Select *Light*, right click : Java: Generate) and browse it (Select *Light*, right click : Java: Browse). Notice the code that has been automatically generated for the *lightOn* attribute ([Figure 3](#)).

This idiom of generating "getters" and "setters" is how JavaBean properties are implemented. In addition to supporting simple property generation, the Rose J Add-In also allows you to create *bound* and *constrained* properties. Bound properties allow a JavaBean to signal other components in the system. For example, we could have a computer that could be informed whenever someone turned our light on and off. Constrained properties are like bound properties with an additional feature. They allow the external component to veto the change. In our example, a computer that was listening to the light being turned on could decide that it should not be turned on and throw an exception. Both of these property types implement additional idioms that can be automatically generated from the Rose J Add-In simply by identifying them as the desired type of property. However, in the interest of brevity, for now we'll stay with a simple property in our *Light* bean.

In order to create JavaBean events, you need to define event *sources* and *sinks*. Simply put, this means you must tell a class what events it can fire, and what events it can receive. In our case, we want the *ToggleSwitch* to fire when it is hit, and we want the *Light* to listen for the *ToggleSwitch* to be hit. While we could define our own Toggle event, it may be more appropriate to use an existing JDK 1.2 event. One of the most useful JavaBean events is the *ActionEvent* (`java.awt.event.ActionEvent`). In our case, we can use the *ActionEvent* for simple notification between *ToggleSwitch* and *Light*. We implement an *ActionEvent* source in *ToggleSwitch* by adding two listener management functions —

addActionListener(ActionEvent a) and *removeActionListener(ActionEvent a)*. This conforms to the JavaBean paradigm for listener management. If we were to use our own *Toggle* event, we would define *addToggleEvent(ToggleListener a)* and *removeToggleEvent(ToggleListener a)*. In each case the idiom relies on a *Listener* Interface. The Listener interface must inherit from the *java.util.EventListener* interface (or a sub interface). This gives us the idiom for the event sink in *Light*. By implementing the *ActionListener* interface, we give *Light* the power to receive an *ActionEvent* from *ToggleSwitch* (or any other Switch or component that fires an *ActionEvent*). ([Figure 4](#))

Generating the Classes

We'll take one or two additional steps before writing code. Since we want these classes to be visible components in our sample, we should derive them from an appropriate UI component. We'll use the standard command button *javax.swing.JButton*, for both classes (Note: because we are using the *JButton*, we can delete the Listener Management methods from *ToggleSwitch*. *JButton*'s parent, *Abstract Button*, already implements the same methods.) In addition, we'll define a wrapper class called *Tester* to run JavaBean in a small program. Make sure *Tester* has the following function defined:

```
public static void main (String[] args)
```

Finally, make sure you are generating a default constructor for each class by checking the Default Constructor checkbox in the generation option section of the Java specification (while not necessary, this will be one less item we have to write). ([Figure 5](#))

Before generating code, your model should look like the one shown in [Figure 6](#).

Generate *Light*, *Tester*, and *ToggleSwitch* by right clicking on them in the browser, selecting the Java sub-menu, and the Generate Code option. Once you've selected a base location on your Java CLASSPATH, you notice a component is created in the component view, and your Java source is generated.

Finishing the Code

Select *Light* and browse it (right click: Java: Browse Java Source). Notice that the class constructor is created, along with the required getter and setter for *lightOn*. In addition, the *actionPerformed* method was added because we are implementing the *ActionListener* interface, in which it is defined. To complete the

code add the following items:

To the constructor:

```
javax.swing.Icon lightPicture = new  
javax.swing.ImageIcon("light.gif");  
javax.swing.Icon lightPictureOff = new  
javax.swing.ImageIcon("lightoff.gif");  
this.setBackground(java.awt.Color.gray);  
this.setDisabledIcon(lightPictureOff);  
this.setIcon(lightPicture);  
this.setEnabled(false);
```

This initializes *Light* with on and off pictures to be presented to the user.

To the actionPerformed method:

```
if(getLightOn())  
{ setLightOn(false);}  
else  
{ setLightOn(true);}  
this.setEnabled(getLightOn());
```

This turns the light on and off by simply changing the current state to its opposite.

Select *ToggleSwitch* and browse it. Add the following line to the constructor:

```
Super("Push Me");
```

Finally, add the following testing and hookup code to the main function in the testing wrapper:

```
javax.swing.JFrame window;  
  
Light light = new Light();  
ToggleSwitch toggleswitch = new ToggleSwitch();  
toggleswitch.addActionListener(light);  
window = new javax.swing.JFrame("Light Switch Test");  
window.getContentPane().setLayout(new java.awt.FlowLayout());  
window.getContentPane().add(toggleswitch);  
window.getContentPane().add(light);  
window.setSize(400,400);  
window.setVisible(true);
```

Deployment and Testing

You can compile these classes right on the command line by going

to the electrical package and typing **javac *.java** . You can then run the test wrapper by typing **java electrical.Tester**. You should see the application shown in [Figure 7](#).

The ToggleSwitch is hooked up to the Light by adding it as an Action Listener (Tester main function, line 5). Once the event source and sink are hooked up, the business behavior is fulfilled.

Summary

While the example we worked through was simplistic, I hope this gives you a good feeling for the basic aspects of JavaBeans™, and what they look like in the UML. I encourage you to experiment by developing your own beans, and also by reverse-engineering them into Rose 98i. As JavaBeans™ become more available through in-house and commercial development, you can expect to see these patterns in many more Rational Rose models.

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

Magazine

> Back Issues

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

Contact Us



[About Rational Rose](#)



[About the UML](#)



Janine Rood is the newest member of the Rose marketing team at Rational Software. Her e-mail is jrOOD@rational.com.



Shoshanna Budzianowski is a Microsoft program manager for Visual Studio. You can reach Shoshanna at Shoshe@microsoft.com.

The Perfect Couple — MSVM and VB

by Janine Rood, Rational Software Corporation and Shoshanna Budzianowski, Microsoft Corporation

If you're a VB programmer who's curious about modeling but haven't yet gotten your feet wet, this article is for you! Did you know that you can take the UML out for a FREE 'test-drive' from the comfortable and familiar surroundings of Visual Studio? If you're using Visual Studio Enterprise Edition, you have in your possession Microsoft Visual Modeler, a 'learning edition' version of Rational Rose.

Microsoft Visual Modeler, packaged in Visual Studio Enterprise Edition, is joint effort of Microsoft Corporation and Rational Software. Microsoft contributed its expertise in rapid application development tools and its knowledge of easy-to-learn user interface design. Rational brought to the table its experience in enterprise-scale development tools, and its world-leading knowledge of modeling. Together they have produced an entry-level tool that brings visual modeling directly into the RAD client/server mainstream. Microsoft Visual Modeler offers you an easy-to-learn introductory subset of the industry standard UML for expressing software design decisions.

Rational Rose, which shares a common code base with Microsoft Visual Modeler, goes on to offer you the full UML notation that programming teams need to be successful throughout the full visual component development lifecycle on enterprise-scale projects.

Microsoft Visual Basic 6.0 enables you to rapidly create distributed applications consisting of reusable components. But in order to harness this power effectively, you need to think in terms of business objects and software architectures. That's exactly what modeling helps you do. Modeling makes component-based development scalable for enterprise and Internet applications.

Modeling can make your life better

If you are motivated enough to open a magazine called *Rose Architect*, it's pretty safe to assume that you're at least *interested* in modeling. If you are curious about what modeling can do for you, read on!

- **Models promote understanding** — Visual Modeling is a way of thinking about problems using a method organized around real-world ideas. Models are useful for understanding problems, communicating with application experts, modeling enterprises, preparing documentation, and designing programs and databases. Modeling promotes better understanding of requirements, cleaner designs, and more maintainable systems.
- **Models shorten development cycles** — Models are abstractions. They portray the essentials of a complex problem or structure, making it easier to manipulate. Abstraction is the fundamental human capability that permits us to deal with complexity. Engineers, artists, and craftsman have built models for thousands of years to try out designs before executing them. Building anything — a house, a car, or a software system — goes more quickly if you have a visual representation.
- **Models save money** — Staring at lines of source code or analyzing forms in Visual Basic does little to provide the programmer with a holistic view of an application. We build models of complex systems because we cannot comprehend such systems in their entirety. Crafting a model allows the designer to focus on the big picture of how components interact. This translates into shorter development times as well as significant cost savings.

In short, models help you as a developer to understand and manage complex programming problems, and models help you as a member of a development team to communicate more clearly and concisely, throughout the lifecycle of the development project.

As software grows increasingly complex, modeling tools become more essential to project success. Today, even relatively small

applications can consist of multiple systems of components. Learning how to manage this complexity is the first step in building scalable systems. With the trend toward three-tier enterprise application architecture in particular, modeling has become an even more critical tool for today's developer. A sound architecture is the foundation of a successful enterprise application development project, and modeling gives you the tools to ensure that you are designing the right architecture for your application. The right architecture will not only streamline the entire development process, but will also ensure that your application can easily be changed, or scaled, as requirements change or grow.

So what's all this fuss about the UML?

The UML is the industry standard methodology for modeling notation. Industry standards, it has been shown time and again, reduce complexity, increase leverage across the market, and generally make everyone's life easier. Evolved standards, such as Windows and the UML, have the added benefit of 'critical mass' leading to a broad base of industry acceptance. Simply put, the UML offers us a common language, or notation, for communicating semantics, syntax and notation throughout the analysis and design phases of a software development project.

What can I do with Microsoft Visual Modeler?

Microsoft Visual Modeler is designed specifically for the first-time modeler. It quickly allows you to become productive modeling three-tier architectures by minimizing your learning time.

With Microsoft Visual Modeler, you can start modeling static behaviors for your Visual Basic applications very quickly and easily. You can create and use model diagrams using UML, with explicit support for three-tiered architectures.

Microsoft Visual Modeler allows you to:

- **Identify and design business classes** — Using classes, you can represent the structural details of your system. You can create class diagrams showing the packages, classes, and relationships needed to realize the system requirements. The relationships supported are association, aggregation, dependency, and inheritance.
- **Map these classes to software components** — This capability enables you to create component diagrams which depict the dependencies among software components, including source code components, binary

components, and executable components. A source code component diagram shows the compile dependencies among files. A binary component diagram shows the allocation of classes to a run-time library. An executable component diagram can be used to show interfaces and calling dependencies.

- **Partition services across a three-tiered service model** — In order to effectively manage the classes needed to attack a problem using a three-tiered architectural approach, the design surface of a class diagram is divided into three sections representing the user interface, the business rules and the domain.
- **Design how components will be distributed across a network** — Deployment modeling allows you to show the configuration of run-time processing elements and the software processes that live on them.
- **Generate Visual Basic code frameworks directly from your model** — With the code generation feature you can create a Visual Basic code framework based on the classes and relationships defined in a class diagram. The code may be tailored by attaching property sets to modeling elements. Wizards are provided to assist in this process.
- **Reverse engineer your existing components and applications** — You can use reverse engineering to create a model based on existing Visual Basic code, ActiveX controls and classes. This capability also enables you to update your model with code-level information. Wizards are provided to assist in this process.
- **Synchronize your design and code implementation using round-trip engineering** — It is important to keep your model and your code synchronized. As you update your code, round trip engineering allows you to ensure these changes are also reflected in the original model.

Then what can I do with Rational Rose?

Microsoft Visual Modeler does not provide support for modeling dynamic behaviors, such as business requirements analysis, business scenario analysis with collaboration diagrams, and detailed component design. It does not offer robust support for team development or an extensible interface with the capacities for additional language support. For these kinds of capabilities, you'll need the power of Rational Rose.

After you've test-driven Microsoft Visual Modeler for a while and seen the benefits of modeling, you'll probably want to check out Rational Rose. It offers full support for modeling dynamic behaviors, team support facilities and an extensibility interface

with support for DDL generation for SQLServer and other databases.

Conveniently enough, Microsoft Visual Modeler and Rational Rose share a common code base, so designs started in Microsoft Visual Modeler can be directly loaded into Rational Rose with no intermediate steps and no wasted learning time. Try Microsoft Visual Modeler and take the UML out for a test-drive today!

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

Magazine

> Back Issues

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

Contact Us



[About Rational Rose](#)



[About the UML](#)



Publisher's Note



Welcome to the 4th issue of Rose Architect

by Adam Frankl

Adam Frankl is Publisher of Rose Architect. He can be reached at adamf@rosearchitect.com.

Just in the year since we started work on this magazine, the growth in the number of software developers using the UML to model their systems has been tremendous. We have learned about many new applications that developers have discovered for applying modeling techniques to solve real-world problems.

From my own experience, and from the reports of our readers, the most vexing problem facing software development teams is just getting the whole team to work together. As we face the reality of distributed development, difficult communication constraints, and fast-changing requirements, new techniques and tools are called for.

Rational's vision for solving this problem extends beyond Rose to deliver a complete Suite of products that together unify development teams, optimizing each individual's contribution while simplifying the entire lifecycle. Grady Booch explores some of the key issues in his major contribution in this issue.

Another important contribution to improving team development is the Rational Unified Process. Per Kröll, in his article *Desktop Coach in a Box* describes the RUP and how it helps developers put the UML to work. Component-based development has often been promoted as encouraging software reuse, but it has important ramifications for improving team development as well. Christian Buckley & Darren Pulsipher weigh in on the advantages of component based development for team coordination in their article *Overcoming Father Time: Parallel Development through Component Engineering*.

One of the most common requests from our readers is for more articles on Java. Jeff Hammond has written a key article on modeling JavaBeans and how to visualize and model them in Rose 98i. And in this issue's *Extreme Rose* column, Tom Schultz points out that there are differences in visualization and modeling technologies, and both are supported in Rose.

As our cover illustrates, models have been used to help coordinate teams in a variety of disciplines. After all...you can't win unless everyone on the team knows what the plays are. And, speaking of teams, please join the Rose Architect team at the Rational Software User Conference, July 25-29 in Seattle. Check out <http://www.rational.com/ruc/> for more information.

Enjoy!

Adam Frankl, Publisher

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

Magazine

>Back Issues

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

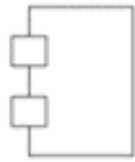
Contact Us



[About Rational Rose](#)



[About the UML](#)



Letters to the Editor



by Terry Quatrani

Terry Quatrani is the author of the best selling book Visual Modeling with Rational Rose and UML.

**Terry,
I received several copies of the Rose Architect magazine that includes my first article. You did a nice job of laying it out. Thank you for inviting me to write the article.**

Mike Blaha

Mike,

We enjoyed having you as a contributor. I would like to take this opportunity to invite all of our readers to submit articles. We would like this magazine to contain a whole range of articles written not only by Rational folks and our partners, but also by those of you in the Rose and UML communities who in many ways are the real experts. See [writers guidelines](#).

Happy writing,
TQ

Dear TQ,

I love your magazine. In Naveena Bereny's article on component modeling in the premier issue of Rose Architect she says that components can be reverse engineered by dragging DLLs, EXEs and tlbs into component diagrams. I have tried this with Rose 98/C++ Professional, and all that happens is that a file link is placed in the explorer pane under the Logical View folder. Nothing is actually reverse-engineered. How do I enable this functionality?

Lian Zerafa Managing Consultant Deloitte & Touche Consulting Group/DRT Systems

Lian,

I am glad that you like the magazine. Our goal is to have a magazine that helps all our Rose customers. I can answer your question for you. Reverse engineering of components is only available in Rose 98 Enterprise and Rose 98 VB/Professional. Unfortunately, it is not a feature of Rose 98/C++ Professional. This is also the case with Rose 98i.

Happy Modeling,
TQ

Terry,

You should think about a 'tip of the week' column that you could email out to interested readers on a weekly basis. This keeps them thinking about your magazine, and gives them some useful information.

Brian Courtney

Brian,

That is a great idea. However, a tip of the week may just kill me. Recently on our Web site we requested that our readers help us out by contributing tips. As they come in, they will be posted on the site. Readers, here is your chance for fame (and a shirt). [Send me](#) your tips and if we use one of them you will receive your very own Rose Architect shirt. To start things off we received a tip from Magnus Christerson who is our Senior Product Manager (sorry Magnus, you will not receive another shirt). Here is his tip:

Step 1) Right click on an item in the browser and do New > URL

Step 2) For the URL name, type in "mailto: tq@researchitect.com"

Step 3) Double click on the URL, and whoops, send TQ mail!

For more hints on using Rose, check out [Stupid Rose Tricks](#).

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.



Dassault Aviation deploys Object Technology and the UML for the business success

by Nasser Kettani

After 13 years working on several projects in different domain areas, Nasser Kettani joined Rational as senior consultant helping customers implement effective software development processes and technologies. He is the Marketing Manager of Rational Software France. He recently co-authored a book "De Merise a UML".

Dassault Aviation is a leading provider of military and civil aviation, and a leader in the development and integration of complex systems. Internationally located, with head offices in France, Dassault Aviation is widely known for the fourth generation Rafale multi-purpose combat aircraft and the Falcon Business Jets. Dassault's technological skills in the development of electronic and imaging have originated Dassault Systèmes, the developer of the widely known CAD/CAM system: CATIA. Founded 50 years ago, Dassault Aviation currently employees over 9,000 persons, with a revenue of 2,540M Euros (2,777 M\$).

Mathilde Thibault and Guillaume Tran Thanh head the Software Support Group, and have graciously agreed to share their experience of introducing Object technology and the UML into their company.

NK: *What kind of software development projects do you have?*

MT, GTT: Our MIS systems can basically be divided in three categories: ERP-based systems, TDMS-based systems (now developed using a new product from Dassault Systèmes: Virtual Product Model), and the last category which includes specific development projects that are developed using object technology. We run also other kinds of projects including Mission Critical Information Systems, Real-time Embedded Systems, Simulation Systems, and Web- based systems (Intranet and also Extranet). We use several programming languages including C, C++, Java, and Ada.

NK: *Does this mean that you use the UML for all these projects?*

MT, GTT: Definitely. UML is our corporate standard for all types of projects. We have been working with OOAD methods for more than three years. And we have finally decided that the UML will be our standard. We wanted to break the barriers that existed between the various application domains of our company. We have used the UML in several projects including manufacturing of aircraft and spare parts, and aircraft documentation. We also have projects with our partner, British Aerospace, on the Integrated Modular Avionics.

NK: *Can you explain to our readers what were your reasons for choosing the UML?*

MT, GTT: Well there are three major reasons. First, convergence. Our management wanted to create a methodology convergence amongst our projects and move away from our various existing methods. Second, international presence and partnership, because Dassault is an international company (including the United States) and has a network of partners (which include systems integrators and industrial partners like British Aerospace). And third, standardization. As an industrial company we cannot use proprietary methods.

NK: *What are the results that you expect from the choice of the UML?*

MT, GTT: We expect several enhancements. First, we enhance our reactivity and quality of software delivery. We create better communication amongst our teams using the same technology across the company. We have a use case-driven, architecture-centric development process that enhances the overall quality, robustness, and flexibility of our systems. Finally, since we are using object-oriented programming languages, we create a better traceability and accelerate our development process thanks to automatic code generation.

NK: *How do your teams, your management and your partners react to this process?*

MT, GTT: We definitely enhanced the communication with the business team that defines the overall business requirements; they were surprised to see our reactivity. Both our technical projects and IS projects have embraced Visual Modeling and the UML. It is also the case that we enhanced communication with our partners in the context of our mutual projects. Finally, there are several consulting companies that embraced this technology and that we

involve in our projects.

NK: *Did you train your teams?*

MT, GTT: Yes. Our process is to train the team that will work on a new project. Additionally, every project receives the support of our group and, when necessary, external consulting. The process is slightly different depending on the domains. We introduced Visual Modeling and the UML together with our code generator in order to gain immediate acceptance.

NK: *Why did you choose Rational?*

MT, GTT: Well there are many reasons. As an international company, we were looking for a company that supports our territory requirements with an international presence. Also, we wanted a tool that evolves over time with new functionality; so it is not just a tool that we chose but also the stability of the company that supports it. We were attracted by the partnerships that Rational Software showed, both strategic partners and third party partners and, of course, the Rational Rose multi-platform strategy (UNIX and Windows). Finally, we were looking for the worldwide leading tool, and Rational Rose is the leader.

NK: *How do Rational tools help you achieve your success?*

MT, GTT: Well Rational Rose is a component of the overall software engineering environment. We have developed our own code generators that allow us to generate up to 100% of the code of local applications, thanks to the Rose open API. We first developed our own documentation generator, but we finally moved to Rational SoDA, which has really improved over time, especially with the last release. Rose has proven to be open since we have integrated it with our own software configuration management system. There are few enhancements to Rose that we would like to see in the future, such as full support of the UML or a better model checking. [Activity diagram support and enhanced model checking capabilities are now part of the latest update of Rose.] But we are very happy that you finally introduced the licensing mechanism, especially floating licenses. That is critical for us, since it reduced our overall cost of ownership and it allows us to better plan our needs. Finally, we are looking forward to Rational Rose RealTime for our avionics and simulation systems.

Thank you Mathilde and Guillaume for taking time out of your busy schedule to share your experiences with our readers. I am sure they will appreciate your story.

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

Magazine

> **Back Issues**

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

Contact Us



[About Rational Rose](#)



[About the UML](#)



Rose, Modeling, and UML Extensibility

by Tom Schultz

Tom Schultz is Director of Target Suite Products at Rational Software. Previously, he was Director of Technology for the Rose Business Unit, responsible for the overall product direction of Rational Rose.

The past several years have been eventful for the industry and for Rose. We have seen the UML created, introduced and widely adopted to become the standard for the visualization of object and component based software development. The UML has also undergone revisions and has become a stable standard where revisions no longer contain fundamental changes, only a fine tuning of the specifications. We have also witnessed the maturation of Rational Rose as *the* tool for the UML, expanding to support the entire range of UML diagrams and notation. The question "where do we go from here" has to enter one's thoughts.

This issue of Extreme Rose is going to investigate the features of the UML that are going to be the most utilized in the near future - the aspects of UML extensibility and UML Model mapping. It will also cover how Rational Rose supports these constructs, so that you can take advantage of the new and exciting modeling already occurring today.

Visualization and Modeling

In order to get into a discussion of some of these concepts, it is important to establish and define two concepts and techniques that have helped define the directions of UML in Rose. These concepts are *Visualization* and *Modeling*. We tend to blend these two concepts - we even call Rose a Visual Modeling tool — but there are differences in these terms. *Visualization* pertains to the presentation and interaction of a user to a visual (graphical) display of pertinent information. *Modeling* is the application of these visualization techniques by the addition of semantics that pertain to a certain vertical area, proving a visual tool solution in that vertical area. With visualization, users are concerned with the presentation of the information and the ability to navigate through these data. Modeling is more concerned with the building of the visual representation model, and the semantics and transformation of the model into something that the user finds useful.

I'd like to think of the differences between visualization and modeling as analogous to the differences between pure science and applied science. In pure scientific research, discovery is the most important aspect and goal of the research; in applied science it is taking an idea from pure science and making it useful and pertinent in some discipline. Chemical Engineering takes compounds from pure chemistry and makes them into a business. In the same

way, modeling and visualization are linked as pure and applied science. Visualization is a pure science; modeling is an applied science. Visualization is the topic of its own article or two, looking at new visual techniques, traceability and navigation, use of color, browser and user-interaction paradigms, but here we will concentrate on modeling. Another useful analogy is that UML is the dictionary — the syntax of the language. We need to establish the grammar of the language — the application of UML models as a standard grammar to that defined language of the UML.

We are already seeing the effects of these efforts. There are movements to standardize how to use the UML for things like relational databases, the use of UML in the world of the Web and the support for Enterprise Java Beans. These efforts will help standardize modeling usage of the UML, which helps all who will use these artifacts or build tools for their use. In these mapping exercises, there is specific information and semantics for these artifacts that we wish to map to a UML representation. UML is an extremely robust language and applying the UML to any area involves mapping the artifacts to existing UML constructs. This is usually relatively straight forward, but sometimes involves the "extension" of UML to meet the needs of the mapping exercise. Luckily, the UML authors understood this would happen and placed natively in the UML tools for the "General Extension" of the UML to cover almost all issues. The specific features we will discuss are **stereotypes** and **tagged-values**.

UML Stereotypes and Tagged-Values

In the UML a *Stereotype* represents the sub-classification of a model element of the UML metamodel. The use of stereotypes allows you to provide additional distinctions in your model that are not explicitly supported by the UML metamodel. Stereotypes extend the basic modeling elements to create new elements. This provides the means to add information about modeling elements that may be specific to a project or process by mapping the additional semantics to a stereotype. All UML classifiers — everything from classes to relationships to components, etc., — can be "extended" by use of stereotypes.

Tagged-values are an extensibility mechanism of the UML permitting arbitrary information to be attached to models. Many kinds of elements have detailed properties that do not have a visual notation. In addition, users can define new element properties using the tagged value mechanism. A tagged value is a keyword-value pair that may be attached to any kind of model element (including diagram elements as well as semantic model elements). The keyword is called a tag. Each tag represents a particular kind of property applicable to one or many kinds of model elements.

Rose Support for Stereotypes and Tagged-Values

Let's take a look at how Rational Rose supports these UML extensibility constructs by creating a new stereotype of class and naming some tagged-value properties for this new element.

One way you can quickly create a new stereotype is by typing a new name in the Stereotype field of a model element's specification. The new stereotype will then be available in the Stereotype field for all model elements of that type (and that are assigned the same language), but only in the current model.

To create a new stereotype and make it available in **all** models in Rational Rose, you need to perform a few more steps.

If you desire, icons can be created for the stereotype to be used in diagrams, lists, and diagram toolbars. For each stereotype, four different icons may be supplied:

- A diagram icon (to customize the appearance of model elements with this stereotype in diagrams).
- A small and a large diagram toolbar icon (to be able to add a button for this stereotype to the diagram toolbar). The two different sizes correspond to the Use Large Buttons option on the Toolbars tab of the Options dialog box.
- A list view icon (to graphically display the stereotype for model elements in specification lists and in the browser).

Diagram icons have to be in Windows Metafile (.wmf) or Enhanced Metafile (.emf) format. Our example will only utilize the diagram icon. For more information on the use of the other icons, see the references at the end of this section.

Open the default stereotype configuration file, *DefaultStereotypes.ini*. The default stereotype configuration file defines a set of stereotypes, including the location of icon files as well as settings that affect the usage and display of the defined stereotypes. Typically, the extension of a stereotype configuration file is .ini, but any extension is allowed.

Each stereotype configuration file needs to be located in the Rational Rose installation folder, defined by the InstallDir registry value, and listed in the StereotypeCfgFiles registry section. It is recommended that you add all new stereotypes to the default stereotype configuration file.

In the default stereotype configuration file, add a line for the new stereotype in the section called [Stereotyped Items]. For our example, to add the class stereotype "Car" to an existing configuration file, add a corresponding line as follows:

```
[Stereotyped Items]
Class:Interface
Component:EXE
Component:DLL
Component:ActiveX
Component:Application
Component:Applet
Class:Car
```

Next you need to create a section for the new stereotype, named exactly as the line you added in the [Stereotype Items] section, for example:

```
[Class:Car]
Item=Class
Stereotype=Car
Metafile=&\MyStereotypeIcons\car.wmf
```

In our example, we have created a subdirectory called MyStereotypes in which we have placed the Windows Meta File that we have chosen for the diagram icon. The new folder is a sub-directory of the installed location for Rose, hence the use of "&" in the Metafile statement.

The new stereotype is now available in Rational Rose. [Figure 1](#) shows our new Car stereotype. For information on how to control the display of the new stereotype in diagrams and in the browser, see Controlling the Display of Stereotypes.

For detailed samples on user-defined stereotypes, please refer to your Rose documentation or <http://www.rational.com/products/rose/rosedocs.html>.

Tagged-values are supported in Rose, referred to as *properties*. I will not repeat a topic on Rose property creation, since this was covered recently in *Rose Architect*. Refer to your issues or check out the back issue area on www.rosearchitect.com for the articles by John Hsia on [Creating Custom Model Properties](#) (Winter 99) and Kevin Kelly's [User Defined Tags with Rose 98 Scripting](#) (Spring 99).

I can now go into the Model and using a script (Kevin's is available on the *Rose Architect* Web site) to add the property of Model to the Car stereotype.

There is one issue to cover with Tagged-values and their use in conjunction with stereotypes. The issue is that properties (Tagged-values) are associated in Rose with the model element, such as Class, Component, etc., not with the Element-Stereotype combination. Hence, if I extend Rose with "Car" specific properties for a Class in Rose, they will appear for *all* classes, not just ones stereotyped as "Car". There is discussion even in UML circles on using Tagged-values for just elements, or for element-stereotypes. Regardless, if you wish to have stereotype-specific Tagged-values, there is a way, and the extent of work to be done depends upon your usage of Rose. If you are writing Rose Scripts or using the REI to access Rose information, you only need to remember that when you retrieve information for an element, such as Class, you also need to read its stereotype, in order to know if the property is valid for that particular object. The pre-definition of the known properties for a stereotype is necessary in your script or code. You will need to have a filter function for property with its Rose Item (UML model element) and its stereotype as described below:

```
Function GetStereotypeProperty(theRoseItem As RoseItem, theSingleProp
As String, theStereotype As String) As String
    If theRoseItem.Stereotype = theStereotype then
        GetSingleProperty=theRoseItem.getpropertyvalue(toolname,thesingleProp)
    Endif
End Function
```

Summary

The use of the UML General Extension capabilities is a powerful concept in the mapping of particular problem domains, especially those that carry specific semantic information which we would like to portray with enough difference to note their uniqueness. Having this capability in Rose allows us to utilize these

extensions in providing real solutions in the application of UML modeling, meaning that we will see more of these UML mappings appearing in the future.

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

Magazine

> Back Issues

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

Contact Us



[About Rational Rose](#)



[About the UML](#)



Magnus Christerson, Senior Product Manager for Rational Rose, co-authored with Ivar Jacobson the Jolt Award winning book *Object-Oriented Software Engineering: a Use Case Driven Approach*.



Davyd A. Norris is a Software Engineering specialist with Rational Software.

Landscaping with Rose — Planning a System Architecture

by Magnus Christerson and Davyd Norris

Introduction

We often hear comments like, "If I show two objects talking to each other in a sequence diagram, why doesn't Rose automatically put in the relationship between their associated classes?" Other times it's, "If I show a relationship between two classes in different packages, Rose doesn't automatically add a dependency between the packages — fix this bug!"

While it might seem on the surface that this would be a great way to keep the design consistent, in practice this would be disastrous. Individuals would be able to create relationships from one object to any other and Rose would not only let them, but would adjust the model so that it generated "correct" implementation. The model would quickly degenerate into component custard and all hope of maintainability, reusability, and comprehensibility would be lost.

Instead, Rose is a design tool, and will help you keep architectural integrity in your design. While not getting in the way of your design, Rose will help you check your design periodically for inconsistencies that would violate these rules. In this article we will

examine ways in which this can be achieved using Rose.

Consequences of bad architecture

Probably the most obvious consequence of bad architecture is a system implementation that is hard to understand, with classes related to each other in a haphazard fashion. This makes it extremely difficult to maintain or fix the design and code later. Typically fixes in one place have impact on code in seemingly unrelated areas. Often these systems also show performance problems which are hard to explain and harder to improve. While at a code level these dependencies are very easily introduced, by accident or not, a model is much more explicit about such dependencies and clearly points them out. When a system is reverse engineered for the first time and reviewed by the development team, the team is often very surprised. The dependencies often surprise the team, and after some arguments that they are wrong, the team finally agrees that the model is correct and, unfortunately, accurately describes the system. The problems can seem to be a factor of the system as a whole and hard to pin down to a single component or area. A good team uses the new insight, and starts to change to improve the architectural integrity of the system. A good architecture will consist of concise packages of collaborating classes, arranged in successive layers of abstraction, with each package having a well-defined interface and limited dependencies across packages. This will localize any undesirable effects, much like the way fire doors are used to minimize the spread of fire through a building.

Systems exhibiting bad architecture also often show a predominance of circular dependencies — this is when you are able to follow the relationships from one class, package, or component around a path that leads right back to your starting point. What this means is that you will not be able to compile, build, test, or distribute any single part of the system without the others. This is a particularly nasty manifestation of bad design, which has a direct effect on build and release management as well as reuse. A good architecture, on the other hand, will have entities arranged in Directed Acyclic Graphs (DAGs), with the most generic nodes showing the smallest number of outgoing edges and the most application specific nodes showing very few incoming edges¹.

Perhaps the most telling sign of bad architecture in Rose is that the model will be almost impossible to work with in a multi-user environment. Rose's unit of reuse and multi-use is the UML package. Models can be split into *controlled unit* files along any package boundary and version controlled right alongside all your other project artifacts as if they were source code. If the architecture is right, each developer will only need to check out a small number of controlled units to work effectively. See the Spring

'99 issue of *Rose Architect* for the Magnus Opus article discussing an approach to accomplish this². Deadlocks are rare and are usually indicators of the need to refactor your architecture. This multi-use mechanism scales extremely well in practice. We have seen well architected systems being worked on by more than a hundred developers over many years, and badly designed systems that could not be used by more than three developers for any length of time.

Controlling the architecture of large teams

Bad architecture is completely avoidable — and Rose can help you avoid it. By following the three basic principles outlined below, you will have a far greater chance of catching architectural problems early on.

1. Take the time to establish your system architecture up front

Start your system architecture by defining the high level way in which the application works. One approach you can take is by looking at the different aspects of your system³:

Model the Logical structure - create the key packages used in your system and define the dependencies you want between them before you add any classes. As you add classes, interfaces, and relationships check that they obey the established package dependency rules, and either adjust the class or the package relationships as you go. Often this activity will require more than just simply changing relationships, and a more serious refactoring will be required where you move, merge, and split classes and packages to get the structure right.

Model the Physical structure — map classes to components and define the physical layout of the implementation files. This structure defines your build dependencies. Many people simply mirror the logical structure of the system, and Rose will do this by default if you do not create components in your model at all. This is a somewhat naïve approach, and often better designs can be achieved if you approach this activity from a reuse point of view¹.

Model the Dynamic structure — define the processes and threads in your system and show how they communicate with each other, as well as what logical classes they have control over. These dynamic elements can be modelled with both active classes and objects, or with active components, and serve to illustrate how the final system will work in a concurrent or distributed environment. This will have an impact on the package structure of your system as well as the interfaces of those packages.

Model the System structure — define the network topology and connectivity of your servers, CPUs and devices and the protocol used between them. Then map the processes from the dynamic structure to the deployment components. This will help you decide what classes and interfaces will need to be made into interface definitions in CORBA or COM, and guide your design of subsystems.

Finally, prove your architecture will work by modeling and implementing a few key end-to-end scenarios. Take them all the way through your system so that you can see any errors or design problems. Each time you evolve your system, take the time to go back and check that the key scenarios still function correctly from a design point of view.

2. Enforce and check the architecture

Depending upon the nature of your application, different types of architectural enforcement will be needed. In general, the bigger the system, the more enforcement you want. Rose provides a lot of built-in design checking as part of its core functionality, but almost anything else you might require can be implemented using RoseScript, and commercial RoseLink products exist today that check and report on design quality, such as MetricsONE from NumberSix Software. These design checks are like the warnings from your compiler. You can choose to ignore them, but you really shouldn't for larger systems.

At the scenario level, Rose provides the *Report/ Show Unresolved Objects* and *Report/Show Unresolved Messages* to detect objects that are not linked to a valid class, and object messages that are not linked to valid operations on the supplier class. This is useful to detect when developers may have changed key interfaces or operations. At the use case level, *Report/Show Participants in UC* will show every class, operation and component used on any of the interaction diagrams under the Use Case in the browser — this quickly shows you the sphere of impact when a Use Case changes. This is also available from RoseScript, so it is a simple matter to create scripts that create participating class diagrams automatically for you.

Perhaps the most architecturally important report Rose provides is the *Report/Show Access Violations* command, which will display all the classes that have illegal relationships. These exist where:

- The classes are in two packages that are not related via a dependency.
- A package dependency exists, but the direction of the relationship is inconsistent with the direction of the

relationships between the packages.

- A package dependency exists, but the relationship is to a private class within the supplier package.

Finally, there is the *Tools/Check Model* command which checks the entire model for unresolved references, including the ones above, and adds ones that might be caused by inconsistent versions of controlled units. These checks include:

- To the supplier of any kind of relationship between any Rose items
- From a view on a diagram to an item in the model
- From a logical package to its assigned component package and from a module to its assigned class
- From an object to its class
- From a message on an object diagram to an operation in a class
- From dynamic semantics in an operation to a scenario diagram

And again, you can simply add your own architectural checking by writing your own RoseScripts. For example, if your system contains many state machines, then in addition to the commands above you might also want to create a set of scripts that check for things like:

- Do all events acting on a state map to one of the operations on its class?
- Do all actions map to either internal class operations or, in the case where an event is sent, public operations on other classes?

3. Round-trip engineer the implementation

Many people tend to misuse the round-trip engineering capabilities of Rose, or just have a challenge getting it to work in their development process. Sometimes, they see it as a way of getting a little bit of implementation done for them after all their design effort, and complain when Rose does not generate more of the code for them. In truth, there are very few system architectures where you could generate all your code without having to spend inordinate amounts of time modeling, including adding code snippets in the model for full generation.

The real value in round-trip engineering lies in the ability to carry your system architecture into implementation, and then support it throughout its development. Why spend all that time creating your system in Rose, enforcing your architecture, only to break all the rules during a late mad all-nighter of hacks and kludges? The only way to have some sort of control over the architecture in code

without turning into a megalomaniac, pointy-haired boss is to create an implementation shell to code in — this is forward engineering. In fact, some of the largest projects where Rose is extensively used, where many hundreds of developers collaborate successfully on a system implementation, perform all changes from the model, and every change is introduced into the code as part of a code generation. They even argue that the need to do reverse engineering is a sign of a flawed design process.

If you want freedom to do architectural changes in code, you need to be able to quickly spot any design changes made during coding and assess the impact on the architecture, good or bad. The only way to do that successfully is by mapping the implementation back into the model — this is reverse engineering. And in this process you need to be able to update either the model or the code to incorporate these changes in a transparent fashion — this is round-trip engineering.

We don't want to kid you here — round trip engineering is not a trivial exercise and takes quite a bit of effort to get right. However, the teams that have implemented this additional level of architectural traceability find the benefits far outweigh the initial work, and once the round-trip process is in place it eases considerably and becomes a natural part of your weekly activities.

About the Authors

You can reach the authors at magnus@rational.com and dnorris@rational.com.

Footnotes

1. *Designing Object-oriented C++ Applications Using the Booch Method* — Robert Martin
2. *Modeling Large Systems*, Magnus Christerson, Nils Uden, Rose Architect, Spring 99.
3. *The 4+1 Views of Software Architecture* — Philippe Kruchten

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

Magazine

>Back Issues

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

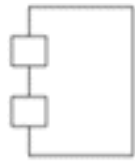
Contact Us



[About Rational Rose](#)



[About the UML](#)



Extending
Rose



Your Guide to Rational Rose Add-Ins

by John Hsia

John Sunda Hsia is the author of the training class "Extending Rational Rose". As a Technical Lead Engineer with the RationalLink partner program, he manages the training support of RoseLink, RequisiteLink Partner Program and ClearCaseLink partners in their effort to create add-ins and integrations. John has spent almost ten years working with visual modeling tools, object-oriented methodologies, structured methodologies and meta-CASE tools. John can be reached at jhsia@rational.com

Introduction

Rational Rose's COM server, built-in scripting language and add-in facility provides users and projects the ultimate flexibility in extending and customizing Rose to meet their specific needs. But do users actually have to write their own add-ins? With Rational Software's numerous commercial add-ins, approximately 100 RoseLink Partners integrating third party products, and tens of thousands of users writing their own scripts, chances are a desired add-in may already exist. This article is intended to serve as a guide to help Rose users successfully find an add-in rather than embarking on a Rose Add-In development project. [Download an examples-oriented approach here.](#)

Finding an Add-In

Where to begin? The natural starting point is Rational's Web site. At the time of publication, one could get a complete list of RoseLink Partners, not add-ins, through the Partner Locator facility (<http://www.rational.com/corpinfo/partners/index.jtmpl>). Like most users, I'm more interested in the actual add-ins. So to supplement that list, I've compiled a list of RoseLink Partner add-ins categorized by functionality and domains.

Please note that Rational has plans to update the Partner Locator facility with a search engine so queries based on keyword (e.g., Ada, Smalltalk) will be supported. The exact date of this upgrade is

unknown at this time.

Guide to the Guide

There simply isn't enough space or time to provide a complete description for each add-in. So each add-in only has a brief, if not terse, description. It's up to the reader to find more information. I've provided a specific URL wherever possible. In some cases, readers may have to do a little bit of searching in the partner's Web site. If information on an add-in can't be found within the partner, product, or press release area, a general search of the site for "Rational", "Rose" or "UML" may be needed.

Add-ins provided by Rational Software will also be referenced in this guide at the beginning of each category. Consult <http://www.rational.com> for more details.

Packaging of Add-Ins. Add-ins come in all shapes and sizes. Some add-ins are standalone extensions to Rose that make users more productive (e.g., metrics), some are options that integrate Rose with a third party product (e.g., Delphi), and some come bundled into that company's third party product (e.g., most object-orient database vendors). In addition, availability of add-ins may depend upon your operating system (i.e., Windows versus UNIX). There may also be dependencies on a particular version of Rose (e.g., 98 or 98i) and upon the existence of other add-ins (e.g., C++ or Java). And finally, the cost of an add-in will vary from free downloads to hundreds or thousands of dollars. Each RoseLink Partner, including Rational Software, determines their own pricing policies.

Black and White Categories - NOT. I wish that I could create a set of clear-cut categories with absolutely no overlap. But alas, this simply wasn't possible. So when you're looking for an add-in, don't forget to look for related categories. As an example, some users might consider VB an IDE while others consider it a Framework or Application Development Tool. The subsequent categories are:

- Business Process Modeling/Reengineering
- Databases and Data Modeling
- Document Generation and Reporting
- Frameworks and Application Development
- Integrated Development Environments (IDEs)
- Languages and Standards
- Meta Data Management
- Project and Process Management
- Requirements Management
- Rose Productivity and Others

Business Process Modeling/Reengineering

The add-ins that fall in this area include structured analysis, workflow, and process modeling add-ins. To some degree, Rose 98i might fall into this category with its support of the **UML activity diagrams**.

Rose ARIS Link from IDS Scheer

URL: <http://www.ids-scheer.com>

Rose Business Process Link from Ensemble Systems, Inc.

URL: <http://www.ensemble-systems.com>

Description: Web-publishing, storyboarding, workflow, and Activity-Based Costing. See the RoseLink Product Review in the premiere issue of Rose Architect.

Modeler Link - Rose from CASEwise

URL: http://www.casewise.com/product_links/rationallink.htm

Description: Bi-directional link to CASEwise's Corporate Modeler for business models and program designs.

Rose SilverrunLink from Argussoft

URL: <http://www.argussoft.ru>

Description: Bi-directional link to Structured Analysis model in SILVERRUN. See [article](#) by Oykhman and Novozhenov in Winter 1999 issue of Rose Architect.

Databases and Data Modeling

This topic is key to most applications, not just IT applications. In addition to the add-ins listed below, Rational Software provides add-ins for **Oracle8**, **DDL** and one called **Data Access**. Data Access comes to Rational from the acquisition of Vigor Technologies and addresses object-to-relational mappings.

Rose Caché Link from InterSystems Corp.

URL: <http://www.intersys.com>

Description: Round trip to Caché - a post-relational database management system.

ERwin/Rational Rose Translation Wizard from Logic Works, Inc. (now Platinum Technology)

URL: http://www.platinum.com/products/appdev/erwin_ps.htm

Description: Use Erwin to reverse-engineer relational databases to object models in Rose.

Rose GTSum Access Link from GTSum Corp

URL: <http://www.roseaccess.gtsum.com>

Description: Automatically derive a model from a Microsoft Access

database application. In beta.

RoseO2Link from Ardent Software, Inc.

URL: <http://www.ardentsoftware.com/object>

Description: Build powerful object applications using O2 - a complete object-oriented database management system.

Rose Objectivity Link from Objectivity, Inc.

URL: <http://www.objectivity.com>

Description: Round-trip development supported to Objectivity databases. Scheduled for 3Q99.

ObjectStore Blueprint from Object Design, Inc.

URL: <http://www.odi.com>

Description: Generate ObjectStore schema.

POET Rose Link 2.0 for POET 5.1 from Poet Software Corporation

URL: <http://www.poet.com/products/products.html>

Description: Visually model POET Object Databases and create POET Object Databases.

PowerDesigner Object/Data Bridge from Sybase

URL: <http://www.sybase.com/products/powerdesigner>

Description: Full round-trip engineering support to PowerDesigner.

Rose Secant Extreme Link from Secant Technologies

URL: <http://www.secant.com>

Description: Object-to-RDBMS/ORDBMS and multi-tier Application Server code generation from UML models.

Rose VERSANT Link from Versant Object Technology

URL: <http://www.versant.com>

Description: Round-trip development support with Versant-enabled C++ and Java header files.

Rose Visual FoxPro Link from Microsoft Corp.

URL: <http://www.microsoft.com/vfoxpro>

Description: Visual Modeler Connection Wizards supports reverse engineering and code generation to Microsoft Visual Modeler 1.1.

Document Generation and Reporting

Now that you've got your model, what are you going to do with it? If you need to generate documentation or possibly publish to the web, these should be of interest. Some add-ins even support an iterative approach with modifications on the WYSIWYG side. Rational Software supplies the add-ins **SoDA** (Software Document Automation), **Web Publisher**, and **Format Wizard Add-In**.

DocEXPRESS/Factory from ATA, Inc.

URL: <http://www.docexpress.com>

Description: Automation for documentation, project Web sites, and Component Data books - over 200 templates.

DocEXPRESS/Reporter from ATA, Inc.

URL: <http://www.docexpress.com>

Description: Produces documentation for Word, RTF, and HTML. Spell checks documentation fields and diagram notes.

DocEXPRESS/Word from ATA, Inc.

URL: <http://www.docexpress.com>

Description: Microsoft Word add-in for publishing Rose data. Custom report authoring tool for Reporter and Factory.

Model EyeQ from Synergex

URL: <http://www.synergex.com>

Description: Extracts graphical and textual information from Rose to create custom documents.

Rose Word Link from Ensemble Systems, Inc.

URL: <http://www.ensemble-systems.com>

Description: Two-way integration between Rational Rose and Microsoft Word.

Frameworks and Application Development

What distinguishes an IDE from a framework or application development tool? In general, the latter adds a substantial number of libraries and infrastructure towards the development and deployment of an application. Because of this general qualification, there are quite a few add-ins here. Rational Software also adds a few to this list with support for **Forte**, **Visual Basic**, **Rose for RealTime**, and **Rational Suite**. Rose for RealTime comes to Rational from the authors of the **ROOM** (Real-time Object Oriented Methodology) at ObjecTime Limited. Check their listing for more details. Rational Suite (i.e., AnalystStudio, DevelopmentStudio and TestStudio) is an offering that arms team members in specific roles with an arsenal of integrated tools. Rose is an integral part of that arsenal.

Rose Arc InTENSE Link from ARC Information Technologies

URL: <http://www.arcit.com>

Description: Users can customize this electricity trading system through Rose models.

BEA M3 from BEA Systems, Inc.

URL: <http://www.beasys.com/action/press/press122.htm>

Description: Accelerate the deployment of BEA M3 applications - under development.

BEA TUXEDO Builder (a.k.a. Rose Expert) from BEA Systems, Inc

URL: <http://www.beasys.com>

Description: Accelerate creation of TUXEDO-based applications.

BOCA/CDL from Data Access Technologies

URL: <http://www.dataaccess.com/dat>

Description: Rose extensions for Business Object Component Architecture (BOCA) and generates Component Definition Language (CDL). Downloadable beta available.

Rose Bold for Delphi Link from BoldSoft AB

URL: <http://www.boldsoft.com>

Description: Bold for Delphi is the first development product built with the Bold Architecture - optimized for OO.

Rose BOMA Link from SES Software Holdings, Ltd.

URL: <http://www.sesh.com>

Description: Extensive library of Java business objects for global value chains for intranets and the Internet.

Rose Centura Link from METEX Systems Inc.

URL: <http://www.metex.com/products>

Description: Facilitates round-trip engineering to Centura Team Developer 1.x.

Rose DAE Link from Information + Graphics Systems, Inc.

URL: <http://www.igs.com>

Description: Model to data model mapping. Framework and support middleware to share data across the enterprise.

Rose Dynasty Link from Dynasty Technologies, Inc.

URL: <http://www.dynasty.com>

Description: Imported into DYNASTY's Business Application Framework for automatic generation and deployment.

FastUI from Syrinx Corp.

URL: <http://www.syrinxcorp.com>

Description: Generate a prototype UI from use case. Includes various stereotypes. Under development.

Rose GDMO Link from ObjectStream, Inc.

URL: <http://www.objectstream.com>

Description: Design and implement agents, managers, applications, and interfaces for telecommunications networks. Forward and reverse to ITU-T standard GDMO/ASN.1 documents.

Gen-it for C++ from Codagen Technologies Corp.

URL: <http://www.codagen.com>

Description: Generates C++ code for components participating in a

framework. Flexible code generation. 4Q99

Gen-it for Java from Codagen Technologies Corp.

URL: <http://www.codagen.com>

Description: Generates Java code for components participating in a framework. Flexible code generation. 3Q99

IONA Library Object Models from IONA Technologies, Inc.

URL: <http://www.iona.com>

Description: Rose object models are available for IONA's libraries.

Meta Integration Works from Meta Integration Technology, Inc.

URL: <http://www.metaintegration.com>

Description: Create applications that migrate legacy data and share of meta-data. Generates C++ with Data Connectors SDK for 2-tier, 3-tier and web apps.

ObjectGEODE RoseLink from VERILOG

URL: <http://www.verilogusa.com>

Description: Import into ObjectGEODE - support for SDL from the ITU, analysis, design, verification and validation through simulation, code generation and testing.

ObjectSwitch AddIn from ObjectSwitch

URL: <http://www.objectswitch.com>

Description: Generate robust, operations-critical applications from Rose. Generate apps for for ObjectSwitch Network Resident Servers.

ONTOS* Integrator from ONTOS, Inc

URL: <http://www.ontos.com/rational.htm>

Description: Object-oriented middleware products allow integration and management of information sources across the enterprise.

PowerTier Link for Rose from Persistence Software Inc.

URL: <http://www.persistence.com/Products/index.html>

Description: Generate a complete EJB application server or complete C++ application server from Rose object models.

Rational Rose RealTime from ObjecTime Limited and Rational Software

URL: <http://www.rational.com/products/rosert/index.jttml>

Description: Extends Rose with model execution and code generation from ObjecTime Limited. Not an add-in, this is a separate Rational product.

Rogue Wave Libraries, Rogue Wave Software, Inc.

URL:

<http://www.roguewave.com/company/newsletters/win98/code.html>

Description: Use Rose to visual model and assemble Rogue Wave

components.

Rose MIB Link from ObjectStream, Inc.

URL: <http://www.objectstream.com>

Description: Design and implement agents, managers, applications, and interfaces for telecommunications networks. Forward and reverse to SNMP MIB files.

Rose PowerBuilder Link from METEX Systems Inc.

URL: <http://www.metex.com/products/>

Description: Supports iterative and incremental development by facilitating power script and reversed engineering from PowerBuilder.

Rose RescueWare Link from Relativity Technologies

URL: <http://www.relativity.com>

Description: Analyzes and transforms legacy applications from COBOL into modern languages.

RoseOSimLink from OriginalSim Inc.

URL: <http://www.originalsim.com>

Description: Simulation application developers can export models into OSim Framework for model behavior development, implementation and refinement. Supports process compliant with HLA.

Rose SF Bridge from METEX Systems Inc.

URL: <http://www.metex.com/products/>

Description: SanFrancisco, from IBM, is a flexible, Java-based foundation for business applications.

SuperNova/Visual Concepts from SuperNova, Inc.

URL: <http://www.supernova.com>

Description: First platform, data source, and language-independent environment for the management, assembly, and deployment of component-based applications.

Rose Synergy Link from Synergex

URL: <http://www.synergex.com>

Description: Provides the core foundation for mission-critical solutions - extensive GUI, logic, and data tools.

Rose UNIFACE Link from Compuware Corp.

URL: <http://www.compuware.com>

Description: Forward and reverse object models to UNIFACE to construct and assemble large-scale applications.

Rose VA Generator Link from Unity Software Systems, Inc.

URL: <http://www.unity-software.com>

Description: Forward and reverse to VisualAge Generator code frameworks for high-volume transaction processing system.

Rose Vision Link from Unify Corporation

URL: <http://www.unify.com>

Description: OO 4GL product for building solutions integrating the internet with legacy apps - e.g., e-commerce.

VISION* Domain from VISION* Solutions - a unit of SHL SYSTEMHOUSE

URL: <http://www.gis.shl.com>

Description: An AM/FM/GIS product that is seamlessly integrated with Rose. See "When Vision Becomes Reality" in the premiere issue of Rose Architect.

Windchill Information Modeler from Parametric Technology Corporation

URL: <http://www.ptc.com>

Description: Windchill is a suite of 100% Java, Web-centric applications providing enterprise-wide Product Data Management (PDM).

Rose Zeus Frameworks Link from DHR Technologies, A division of OAO Technology Solutions Inc

URL: http://www.oaot.com/oaot_web/it_es.html

Description: A family of pattern languages for the development of robust enterprise systems. Frameworks: ritual project, application system, and performance support.

Integrated Development Environments (IDEs)

Once upon a time, IDEs consisted of a debugger, a compiler, an editor, and a fairly fundamental library. These days, IDEs are much more than that. If you don't find what you need here, please check frameworks and languages. Rational Software supplies add-ins to support **Visual C++**, Rational **Apex C/C++** and **Rational Apex Ada 95** (and **Ada 83**). Apex is a high end IDE that addresses both Ada and C/C++.

Rose Delphi Link from Ensemble Systems, Inc.

URL: <http://www.ensemble-systems.com>

Description: A two-way integration to Borland Delphi. Key features: "markerless" Delphi code and code sync.

DSW RoseLink for Delphi from DSW Group, Ltd.

URL:

<http://www.thedswgroup.com/Products/RoseLink/Roselink.htm>

Description: Forward and reverse models to maintain consistency between source code, third party source libraries, and Rose artifacts. Includes a full VCL Rose model.

Rose JBuilder Link from Ensemble Systems, Inc.

URL: <http://www.ensemble-systems.com>

Description: Two-way integration to Borland JBuilder. Key features: "markerless" Java code and code sync.

Rose JDeveloper Link from Ensemble Systems, Inc.

URL: <http://www.ensemble-systems.com>

Description: Two-way integration to Oracle JDeveloper Suite. Key features: "markerless" Java code and code sync.

Rose PARTS for Java Link from ObjectShare

URL: <http://www.objectshare.com>

Description: Enhances Rose 98i's Java add-in to support a tight integration to ObjectShare's Java IDE - PARTS for Java.

Rose Sniff+ Link from TakeFive Software, GmbH

URL: <http://www.takefive.com/partners/integrations2.html>

Description: Add-In for Rose/C++, SNiFF+ is a Source Code Engineering tool for software developers in large-scale projects.

Rose Visual Café Link from Ensemble Systems, Inc.

URL: <http://www.ensemble-systems.com>

Description: Two-way integration to Symantec Visual Café. Key features: "markerless" Java code and code sync.

Languages and Standards

This category addresses various languages and emerging standards that may not have as widespread adoption across all disciplines as C++ or Java. As a result, some of the add-ins listed here might as easily fit under IDEs. Rational Software provides add-ins for ANSI C++, Java, CORBA IDL and Rational Apex Ada 95 and Ada 83. Since Rational Software was born as an Ada software engineering company, it's no surprise that Rational still dominates this market. Apex Ada is the product that continues that tradition.

Rose EXPRESS Extension 1.1 from Conformics AB

URL: <http://www.conformics.com>

Description: Conversions to the information modeling language EXPRESS of ISO (ISO 10303-11). Enables Extended Global Enterprising based ISO 10303 (or STEP).

Rose Formaliser Link from Headway Software

URL: <http://www.headway-software.com>

Description: Full round-trip engineering to Z notation. Features: structured editing and checking of OO variant of the formal specification language

Gen-it for Smalltalk from Codagen Technologies Corp.

URL: <http://www.codagen.com>

Description: Generates Smalltalk code for components participating in a framework. Flexible code generation. 4Q99

Rose JTemplate Link from PERGAMON GmbH

URL: <http://www.pergamon-sw.de>

Description: Template-based approach to code generator that supports an iterative development process, company standards, and recurring code fragments.

Syrinx Java from Syrinx Corp.

URL: <http://www.syrinxcorp.com>

Description: Java code generation with complete package analysis and build files. (Under development.)

Rose for VisualAge Smalltalk from Ensemble Systems, Inc.

URL: <http://www.ensemble-systems.com>

Description: Two-way integration to VisualAge Smalltalk. Includes code synchronization.

VisualWorks RoseLink 1.0 from ObjectShare

URL: <http://www.objectshare.com>

Description: Bi-directional link between VisualWorks and Rational Rose98.

VisualWorks RoseLink 5i from ObjectShare

URL: <http://www.objectshare.com>

Description: Bi-directional link between VisualWorks 5i and Rational Rose98 and 98i

Rose VDM++ Link from IFAD - Institute of Applied Computer Science

URL: <http://www.ifad.dk/Products/downloads.htm>

Description: An integration to VDM++, a formal specification language based on the ISO language VDM-SL.

Rose <-> XMI <-> UREP from Unisys Corp.

URL: <http://www.marketplace.unisys.com/urep>

Description: Uses XMI 1.0 in a two-way exchange to the Universal Repository (UREP) - manages metadata and business data. (Beta today, release in October.)

Xtend:Specs for Rose from ICON Computing, Inc.

URL: <http://www.iconcomp.com>

Description: Validation of model specifications written in OCL (Object Constraint Language). Features: consistency checking HTML spec/doc generation and embedded browser.

Meta Data Management

Meta Data, data about data, is a very general term. Since most add-ins in this guide have something to do with development artifacts, almost all the add-ins in this guide could be listed in this category. The three add-ins listed here are specifically targeted to meta data — i.e., repositories and exchanging meta data between modeling

tools. For this category, Rose provides an integration with the **Microsoft Repository**. If you've been unfortunate enough to use another CASE tool, an importer to Rose may be available, ask your local Rational Software representative.

Meta Integration Model Bridge from Meta Integration Technology, Inc.

URL: <http://www.metaintegration.com>

Description: Migrate legacy data between design tools, and various metadata repositories. Other tools include ERwin, Designer 2000, PowerDesigner, COOL: GEN, IE: Advantage, Microsoft Repository/XIF, and XMI.

Rose Toolbus Link from Reischmann Informatik GmbH

URL: <http://www.reischmann.com>

Description: Toolbus facilitates exchanging model information between CASE tools (e.g., Rose and IDS Scheer) and repositories (e.g., ROCHADE and Softlab's ENABLER).

Universal Repository from Unisys Corp.

URL: <http://www.marketplace.unisys.com/urep>

Description: Integration with the Universal Repository (UREP) - an extensible OO repository that defines, integrates, and manages metadata and business data.

Project and Process Management

The success, or failure, of a project is dependent upon how, or if, that project is managed and the enforcement of a development process. Beyond scheduling programs like Microsoft Project, this category is usually overlooked. Fortunately, many of our RoseLink Partners consider this area their field of expertise. Rational Software also contributes to this category with a defect management tool (**Rational Summit**), a generalized process (**Rational Unified Process — RUP**) and a configuration management tool (**ClearCase**).

CKO: Active Project Portal from Pagoda Corp.

URL: <http://www.pagodacorp.com>

Description: Combines Intranet-based project, process, and knowledge management through bi-directional integration with Rose, Requisite Pro, Rational Unified Process and Microsoft Project.

GlobalMANAGER™ from Global Objects, Inc.

URL: <http://www.globalobjects.com>

Description: Access and integrate crucial information (e.g., cost, schedule, development tool, metrics) in order to make informed decisions and reports about s/w development projects.

Mesa Vista for Rational Rose from Mesa Systems Guild, Inc.

URL: <http://www.mesasys.com>

Description: Web integration of product team with process & project management automation. Also supports Microsoft Project and Teamwork.

Metrics One from Number Six Software, Inc.

URL: <http://www.numbersix.com/metricsone>

Description: Computes object-oriented metrics from your Rose model. See Bryan Lyons' article in the premiere issue of Rose Architect.

Rose Planner Link from Ensemble Systems, Inc.

URL: <http://www.ensemble-systems.com>

Description: Two-way integration to Microsoft Project. Keep project schedules and actual architecture being implemented in sync.

RoseTalk from Syrinx Corp.

URL: <http://www.syrinxcorp.com>

Description: Facility to manage design/development discussions for each model element. Integrates with other Syrinx add-ins. (Under development.)

Requirements Management

By the time a developer delivers an application, how does the customer know quantifiably (besides sheer faith) that the end product actually meets the initial requirements? The tools in this category help customers define, organize, manage and trace their requirements. Rational's contribution to this category is the product **RequisitePro**. RequisitePro, along with the add-ins listed below, are the leading tools in this market.

DOORS Rose Link from Quality Systems and Software

URL: <http://www.qssinc.com/products/doors/roselink.html>

Description: Establish and maintain the traceability of a software system from requirements definition through object-oriented analysis and design.

icCONCEPT from Integrated Chipware Inc.

URL: <http://www.chipware.com>

Description: Requirements management tools that fully support the dynamic and diverse processes found in most successful business environments. Formerly known as RTM.

Rose Productivity and Others

This category is an assortment of add-ins that helps users make the most of their time with Rose. These tools include miscellaneous scripts, **patterns** and even CDROM-based **training**. I've also

added **Testing** tools into this assortment. Rational Software has a rather large collection of add-ins for this category - configuration management (**ClearCase**), version control (**Visual Source Safe**), Rose model templates (**Framework Wizard**), reverse engineering COM servers (**TypeLibImporter**) and functional testing (**SQA Suite**). Finally, an add-in from Number Six is worth a mention. Their add-in saves and restores your Rose desktop (model, open diagrams and all). Check out their Web site for more details.

BPT RoseScripts from Blueprint Technologies, Inc.

URL: <http://www.blueprint-technologies.com>

Description: A collage of Rosescripts for extending and manipulating Rose.

Framework Studio from Blueprint Technologies, Inc.

URL: <http://www.blueprint-technologies.com>

Description: Framework Studio enables content (i.e., patterns and frameworks) storage, searching, and reuse. Implements a UML repository. See Joanne Garlow's article in the Winter 1999 issue of Rose Architect.

Mastering UML with Rational Rose from ICONIX Software Engineering, Inc.

URL: <http://www.iconixsw.com>

Description: This tutorial provides an easy introduction to the UML, using Rose. See the RoseLink Product Review in the Spring 1999 issue of Rose Architect.

Rose 98 for Power Users from ICONIX Software Engineering, Inc.

URL: <http://www.iconixsw.com>

Description: This tutorial teaches how to optimize your project's usage of Rose 98 in a use-case driven UML modeling process. Topics also include: Rose extensibility, and productivity tips.

TestExpert to Rational Rose from Silicon Valley Networks

URL: <http://www.svnetworks.com>

Description: Generates well-defined frameworks for test cases from key information collected from specific software components in Rose. (in development)

Guns for Hire

If you didn't find the add-in that you need, there is also some shareware available on the net. Check out Rational's repository of user-created scripts at

<http://www.rational.com/products/rose/support/download.jtmpl>.

Another public repository of scripts is hosted by QOSES at

<http://www.qoses.com/downloads.html>.

One final alternative to writing your own is to hire one of the RoseLink Partners to either create a completely new add-in or to customize one of their commercial add-ins. Your best choices are RoseLink Partners who have created an add-in similar to one you're looking for. The partners who explicitly seek this sort of business are ATA, Blueprint Technologies, Conformics AB, Data Access Technologies, DSW Group, Ensemble Systems, GT Sum, Headway Software, Meta Integration, METEX Systems, Number Six Software, and Syrinx. More often than not, partners who offer consulting will be glad to help you out. Maybe you'll be lucky enough to have them roll your suggestions into their commercial product.

Summary

This add-in list is the most comprehensive available and was compiled from information on Rational's Web site, from the respective RoseLink Partner's Web site and from the Partners themselves (thanks for your help!). Since this is a growing body of work, this list may already be outdated by the time of publication. Please refer to Rational's Web site for new partners, new add-ins and partner press releases for your most up-to-date options. So if you don't see the add-in you want in this article, just wait — somewhere, somehow, someone may already be working on it.

Note: This list of partners and products is current as of the time we go to press. To keep this list as accurate as possible, we will update it as changes occur. Future revisions of this list will be found on this website.

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

[Magazine](#)

>[Back Issues](#)

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

[Scripts & Models](#)

[Product Reviews](#)

[Book Reviews](#)

[About Rose Architect](#)

[Contact Us](#)



[About Rational Rose](#)



[About the UML](#)



by Veronica Herrera



Veronica Herrera is a Web Marketing Manager for Rational Software and is also the Web Editor for the Rose Architect web site.

Since www.rosearchitect.com debuted last fall, we've had thousands of visitors and hundreds of e-mails asking questions, offering advice, contributing material to the Web site, and making us feel good with "fan mail." Of special interest and use to us has been your responses to our Web survey, which we will use to aid us in the redesign of our site. We'll have a new look and feel starting in October.

But until then, because the site is still updated monthly, there always something new to see and do there. One of our most popular features has been the contest of the month. Everything from software to shirts has been given away as prizes, and we've tested your knowledge on everything from the UML to a quiz judging how well you know the Amigos.

Also something new, which you might not know about if you haven't visited us in a while, are downloadable scripts and models that complement our articles (and save you time typing in all that code!). And to help you find articles of particular interest, we now have a search capability to help you explore our back issues.

We have many exciting ideas planned in the coming months, so I hope you'll join us as we continue to make www.rosearchitect.com a place where you enjoy visiting often.

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

Magazine

> **Back Issues**

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

Scripts & Models

Product Reviews

Book Reviews

About Rose Architect

Contact Us



[About Rational Rose](#)



[About the UML](#)



The Web Publisher — To Publish Or Not To Publish

by **Naveena Berenyu**

Naveena Berenyu is the Product Engineer for Rational Rose. She has a Master's degree in Computer Science from Arizona State University and five years of experience using, supporting, and marketing object-oriented methods and tools. She can click a mouse around a Rose diagram faster than your eyes can follow.

Forgive the title but I'm still reeling from the fact that they gave *Shakespeare in Love* the best movie Oscar.

In my last article I talked briefly about the Web Publisher feature of Rose 98i. Since then, I've received a couple of questions about it so I thought I would take this opportunity to go into more detail.

The Web Publisher is a utility that ships with Rational Rose 98i Enterprise Edition. It is an enterprise class feature, which is extremely useful if you are involved in enterprise development with large teams. The Web Publisher allows you to take your entire UML model in Rational Rose — your diagrams, the details of each element in the model, and the documentation associated with each of the elements in your model — and convert it to HTML. For a quick demo of this cool feature, check out

<http://www.rational.com/products/rose/prodinfo/98ifeatures.jttml#iwpl>.

What can the Web Publisher help *you* achieve? Depending upon your particular role in a project team, this feature can provide many benefits. Let's see how it can help you depending on your role in a project team:

1. As an analyst working on a large project with a distributed team, who may not all be using Rose.
2. As an architect finishing your design/architecture of the system and passing it off to development.
3. As a developer working on a large design which has been passed off to development.

4. As a project lead needing to update and publish your model.
5. As a documentation editor working as part of a team of developers and analysts.

Analyst. Analysts are usually tasked with defining and driving the requirements. The Web Publisher makes the iterative process of gleaning requirements easier by providing a common medium for sharing the requirements as they evolve. You can use Rational Rose to draw your use case models and scenario diagrams, which capture the business processes in the system. Then you can publish these models to HTML and provide them to the customer and other members of your team.

Architect. As an architect, you can design/architect the system, and then publish the design model to HTML. You may then hand this model off to development to be implemented. This model can then be used by the developers who now have a concrete view of the architecture that they are implementing.

Developer. How can it help you as a developer? The most obvious benefit is that you don't have long spec documents to deal with. Instead, you have a visual model of the system you are implementing. You can easily see which attributes and operations are necessary for the classes you are currently implementing. Plus you have a view of the overall functionality of these classes and how they work with the rest of the system. What an idea!

Project Lead. If you're a project lead, then your job is made easier using the Web Publisher as well. During development when you hit a major milestone, you can update your model with the implementation details, and publish the new model. Now you have a published model available for your team that is up to date with your code and will continue to be iterated on as the implementation continues.

Documentation Editor. If you are the documentation editor/maintainer of a project, your job just got a lot easier with the Web Publisher. You can use the published model of the architecture/design as part of your documentation, or *all* of your documentation.

And the best part about using the Web Publisher is that your team can be geographically distributed. You can publish models on your Intranet and share them across countries where other members of your team can view it using the browser of their choice.

Hopefully I've been able to show you how the Web Publisher can help you based upon the role you play in your project team. I'd love to hear how you've been using the Web Publisher at your company, and to see published models in different languages as well. If you have examples you'd like to share, please send the URL of your published models to naveena@researchitect.com.

Until next time, (*Saving Private Ryan* should have won...)

À Bientôt
Naveena

[Figure 1](#)

[Figure 2](#)

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.

[Magazine](#)

> [Back Issues](#)

[Premiere Issue](#)

[Winter 1999](#)

[Spring 1999](#)

Summer 1999

[Table of Contents](#)

[Scripts & Models](#)

[Product Reviews](#)

[Book Reviews](#)

[About Rose Architect](#)

[Contact Us](#)



[About Rational Rose](#)



[About the UML](#)



META Integration Model Bridge (MIMB): ROSE/ERwin BRIDGE

by Chandrika Shankarnarayan

Chandrika Shankarnarayan is a Program Manager for Rational Rose. Chandrika has 2 years of experience using Rational Rose extensively, supporting it and, more recently, managing the development of the product. Before joining Rational, she spent two years developing real-time embedded software at Baxter. She can be reached at chandrika@rational.com

As the Rational RoseLink Partner program expands, so do the number of Rose complementary products available to help extend the power of Rose. To assist you in learning more about these products, we present the RoseLink Review.

Introduction

Meta Integration™ Model Bridge (MIMB) is a stand alone utility provided by Meta Integration Technology Inc. (MITI). It's a metadata movement utility that converts object or data models from one design tool format under a certain methodology to another. This tool is targeted to database developers and software developers who want to move their metadata between different design tools, metadata repositories, data registries, and database and software development environments. Some of the key reasons for this include legacy metadata migration, re-engineering metadata when moving from a relational database to an object database, and metadata integration.

MIMB currently supports various object modeling tools, data modeling tools, RDBMS and OODBMS databases, and metadata repositories. The purpose of this product review is to look specifically at the object-data modeling bridge between Rational Rose and Platinum's ERwin.

As you are aware, Rose 98i supports the UML methodology while ERwin supports IDEF1X. The Model Bridge supports both exporting and importing of metadata between these two design tools. [Figure](#)

[1](#) shows you the import/ export functionality of MIMB. The particular example illustrates importing an ERwin 3.5 model, reading it in MIMB and exporting it to Rose 98.

The basic functionality includes the ability to:

1. Perform consistency checks on the model to be imported for integrity reasons (e.g., detect disconnected attributes or links in the original export file format). This optional step is useful in limited cases.
2. Import the model from the desired design tools; in this case, import from Platinum ERwin/IDEF1X.
3. View the imported model via the "Model Viewer" and search for desired elements in the imported model. This optional feature is very useful to visualize the input model if you don't have a license for its original design tool format.
4. View the model that you are importing or have imported in the original design tool directly from MIMB. This optional step requires you to have installed the original design tool.
5. Subset the model before exporting it. This allows you to export only parts of a model based on various criteria such as, one file per package, one file per diagram, selected package or diagram, or selected classes. This optional feature is very useful to reuse only a part of a model, or to convert very large models that may exceed the capacity of the destination design tool.
6. Export the imported model into any design tool, in this case Rational Rose 98/UML.
7. View the exported model in the design tool directly from MIMB. This optional step requires you to have installed the destination design tool.

Summary

Today, MIMB is used primarily for legacy metadata migration and metadata integration. It is mainly used in the DoD market by agencies such as Health Affairs (HIRS), National Security Agency (NSA), Air Force Electronic Systems Center (AF-ESC), and the Defense Logistic Agency (DLA). As a result, the bridge has gone through intensive testing against very large model conversion exercises between different design tools, particularly Rose and ERwin.

Some of the highlights of the tool include the speed of model import and export, which is ten times faster during import/export operation when compared to the time it takes to just load the same model in a design tool like ERwin or Rose. To illustrate the performance of the tool, the DoD Health Affairs standard data model (known as FAM-D) developed under Visible Corporation's IE: Advantage is used. This model contains about 200 packages/diagrams, 2,000 classes, 7,000 attributes, and 2,200 associations. The total size of their export files (.imp) is about 5MB. It takes MIMB only about thirty seconds to import the model (regular Pentium PC). It takes MIMB only about fifteen seconds to export the model to Rational Rose 98. It takes Rose about five minutes to load the generated 14MB Rose file (.mdl).

The model Viewer in the Model Bridge is a very useful feature because it allows you to view the imported model in a tree-view display of packages, classes, attributes/types, and operations in the model. You can selectively display the classes/types, search for specific classes from within the Model Viewer and even browse to the desired class/type from the search results. This is especially useful when dealing with large models. There is also the option, while importing or exporting, to view the model in the design tool that you've chosen to import the model from or export the model to. When you choose this option, MIMB launches the design tool in question and displays the model in the tool. This is a very neat feature. Currently, the Rose/ERwin Bridge supports ERwin version 3.0 and 3.5.x and Rose 4.0 and Rose 98 only. At the end of the MIMB installation, the InstallShield program prompts the user to add "Open with Model Bridge" to the menu of all design tool export file formats. For example, if ERwin is installed, it associates the ERwin executable as "Open" to the (.erx) files by double-click, and "Open with Model Bridge" is an alternate open on right click. If you don't have ERwin or any of the other design tools, a double click on any export file format automatically opens Model Bridge, pre-imports the model and is ready to export the model to the design tool of your choice.

The Meta Integration™ Repository (MIR) fully implements and integrates all concepts of IDEF1X Data Modeling and UML Object Modeling Class Diagrams. All the other UML diagrams like Use Case, State, and Interaction Diagrams are not yet implemented. The basic model elements that MIMB supports are class diagrams, classes, packages, relationships, attributes, and operations. The MIR Metamodel that is published on the Meta Integration web site defines how data and object models, model mappings, and model management information are represented.

Evaluation Criteria

Ease of installation and configuration

The Setup Wizard provided with the MIMB is very effective and does a good job of automating the installation and configuration. It is very clean and fast.

Ease of use

The MIMB user interface is easy to use and pretty straightforward. There are fewer options provided in the GUI when compared to the design tools themselves, which reduces the complexity. Navigation through the tool is logical and on-line help is available from within the tool. It would be nice to have tool mentors for each of the options available so that the users do not have to launch the on-line help and browse through it when they need more information about a certain menu item.

Integration with Rose

The MIMB supports import/export of Class Diagrams and the associated model elements only. They have focused only on full implementation and integration of object modeling as defined by UML class diagrams with IDEF1X data modeling. The primary reason for this is that Use Case, Interaction, and State diagrams have no real equivalent for non-UML tools like Platinum ERwin, Oracle Designer 2000, Sybase PowerDesigner, Sterling Software Cool:Gen, Visible Corporation IE:Advantage, etc., which are some of the design tools MIMB supports. Furthermore, IDEF0 activity modeling tools like Platinum BPwin also have no direct mapping in UML.

Adherence to standards

The MIMB fully supports UML class diagrams. All of the modeling elements specified for class diagrams in the UML standard are provided in the MIMB. If users of a particular design tool need more UML support than provided by the tool, then, this converter might not be appropriate for their needs.

Value Added

The converter is certainly a valuable addition to Rose because it supports one of the eight basic diagrams defined in the UML standard: the class diagram. The legacy metadata migration and metadata integration capabilities provided by MIMB are a significant value add because of the ease of metadata migration between different object modeling and data modeling tools. Also, the Model Bridge is customizable to suit each design tool's needs.

Vendor support

Meta Integration Technology provides consulting services and technical support for the MIMB product. The on-line help provided with the tool itself is fairly comprehensive. The Meta Integration Web site carries a lot of information on current products. Given that the tool is actually a converter that converts metadata between two design tools, Meta Integration works with the design tool vendors for product sales, training classes, and other field-related activities.

Future Directions

A key feature planned by Meta Integration for the next version of MIMB is to support XML standards for metadata movement (both OMG XMI and Microsoft XIF). As a potential future enhancement, Meta Integration also plans to invoke the Model Bridge from within Rose as an import/export menu option. Furthermore, MIMB is the win32 packaging of the model converter for metadata movement. They are now focusing on data movement through their Meta Integration' Works (MIW) product written in Java. MIW provides a model manager, a model comparator, a model mapper, and a data bridge builder generating C++ code.

Technical Environment/Requirements

The MIMB operates in Windows 9X and Windows NT environments.

Pricing

The price of Meta Integration Model Bridge is entirely determined by the set of bridges you acquire:

- \$500.00 per import bridge
- \$750.00 per export bridge
- \$5,000.00 for a version of MIMB with all the important and export bridges
- a minimum of 20 license purchases for developing a new bridge.

Evaluation copies of the product are available for download from MITI's web site — <http://www.metaintegration.net>. Or you can purchase MIMB online at

<https://commerce6.ba.best.com/~miti/Buy/Products.html>

[Back to Contents](#)

Copyright © 1999 [Rational Software](#) and [Miller Freeman, Inc.](#), a [United News & Media](#) company.