

▶ [subscribe](#)▶ [contact us](#)▶ [submit an article](#)▶ [rational.com](#)▶ [issue contents](#)▶ [archives](#)▶ [mission statement](#)▶ [editorial staff](#)

## ▶ From Use Cases to Role-Based Security Components

by [Enricos Manassis](#)

Technical Architect

Rubus



*The sophisticated, built-in security functionality of application servers such as COM+ and J2EE has greatly simplified the development process. The crucial and often complex decision-making process surrounding security role assignments within an organization, however, still puts the development process, along with project timetables, at risk. This article describes a systematic method<sup>1</sup> for assigning role-based security when installing a component-based application in an application server. The method is based on the application's use cases and a UML System Model, which can be constructed with Rational Rose.*

### Security: Benefits of Built-In Server Functionality

Modern software systems are typically implemented as applications running within the controlled environment of an application server. This control takes the form of a series of services, typically transaction monitoring, connection thread and object pooling, queued components, loosely coupled events, and security.

For application developers, the primary benefit in using a controlled environment is that you do not have to worry about implementing these services; the application server can implement them better and more effectively. You can focus on the business logic of the application and leave the rest to the server. In the past, it was common for software engineers to develop, in system after system, a means of access control to the various functions of the system. Obviously, very few developers today enjoy developing from scratch things like security mechanisms for their applications. Most experienced developers know how difficult and tricky this can be, so they welcome ready-made solutions. And that is exactly what an application server like COM+ and J2EE offers for role-based security.

Every object within an application has a series of publicly accessible methods. The principles of role-based security are as follows:

- Application developers identify roles that can access the objects. A role defines a category of actual users.
- Each role is associated with a list of one or more actual system users. Conversely, an actual user can be associated with more than one role, reflecting different roles the user plays within the organization.
- Each object and each method of the application is associated with a specific role.

All this is done using the configuration tools that come with the application server. The application developer can work on the application without concern for the security services. When the application is installed in the application server, the role-based security configuration can be completed. The application server will monitor all calls to the objects and their methods and enforce security access according to the configuration. The application server has the responsibility to identify the actual user who is making a particular call. The outcome of this identification depends solely on the configuration of the application in the application server.

## **The Fly in the Ointment: Organizational Decision-making**

Having this built-in functionality can certainly make the development and deployment process nice and easy, but there are usually obstacles along the way. The biggest one is typically the decision-making that developers must undertake in cooperation with business managers concerning security role assignments. This involves:

- Deciding which roles are actually needed for the application.
- Deciding which objects and methods can be accessed by each role.

Unfortunately, these decisions can engender endless debate and slow down development and deployment. The method I'll discuss below can help an organization avoid these obstacles by removing guesswork and systematizing the decision-making process. By cutting back on "freeform" decision areas, it can help developers not only improve the effectiveness of the process, but also achieve consistent quality throughout the application.

The method is based on the analysis of use cases and a UML System Model based on those use cases. To be fully effective, it requires that each external service (third-party product to be integrated into the application) be component-based in the same technology (COM+ or J2EE) as the application itself.

## **Overview: A Method for Designing Role-Based Security**

The first step in this method for designing role-based security is to identify roles. We do this by reviewing the UML System Model specified by use cases. Every UML model has a Use Case View that shows the Use Case Model and defines the actors. Each actor, in turn, defines a role in the Role-Based Security Model. Inheritance relationships between actors define a hierarchy of role definitions, which may or may not be implemented in the application server. If the hierarchy is not implemented, then each terminal role will translate to one role in the application server.

The next step is to **decide which objects and methods each actor can access**. This requires a careful review of the UML System Model's sequence diagrams. A well-designed UML System Model clearly associates each sequence diagram with the use case from which it originated. Each sequence diagram, in turn, clearly identifies the actor as the originator of the interaction. To assign the correct role to an actor, you must ensure that all the objects with which the actor needs to interact directly are accessible by that role.

And finally, you should look at the UML Design Model to **determine how the application is separated into layers**, typically presentation, business, and data layers. An actor should be able to access the presentation layer but not the business or data layer. Our method introduces a number of extra roles that do not map to any actor identified in the use case analysis. These roles represent the various layers of the application and enforce an extra level of security.

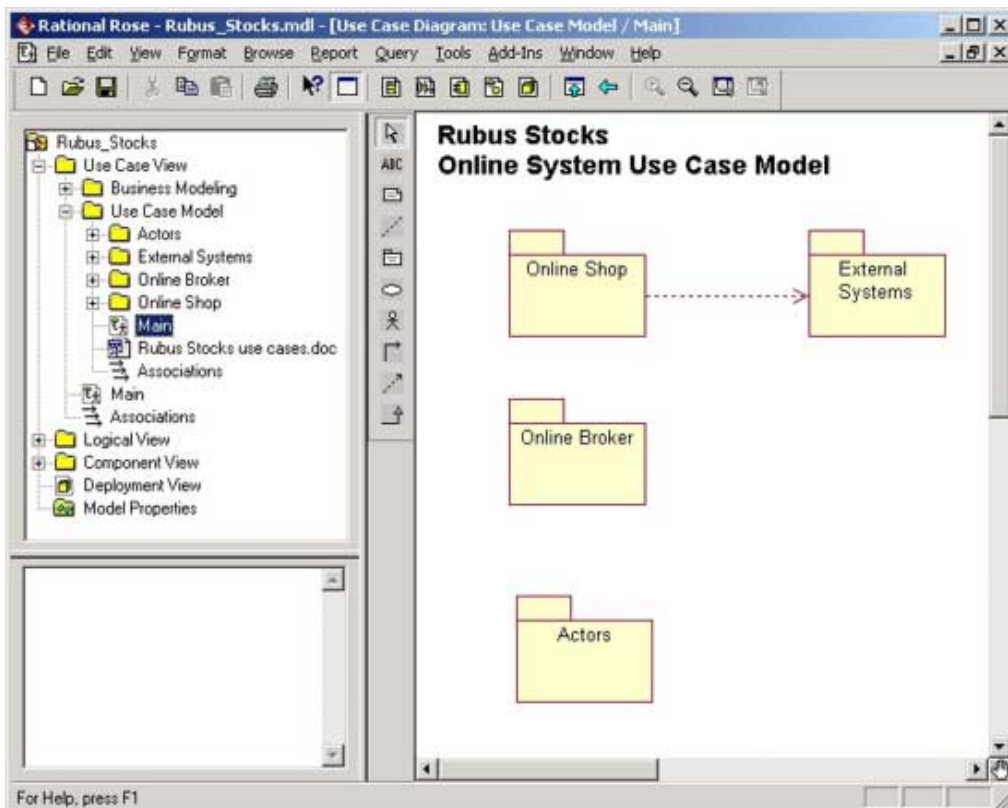
We can represent findings from these analyses using a matrix. The first column lists the system's objects, and the second column lists all the methods for each object. The first

row lists the roles we identify. If a role has access to a method of a particular object, then we place an X in the box where the role column and the method row intersect.

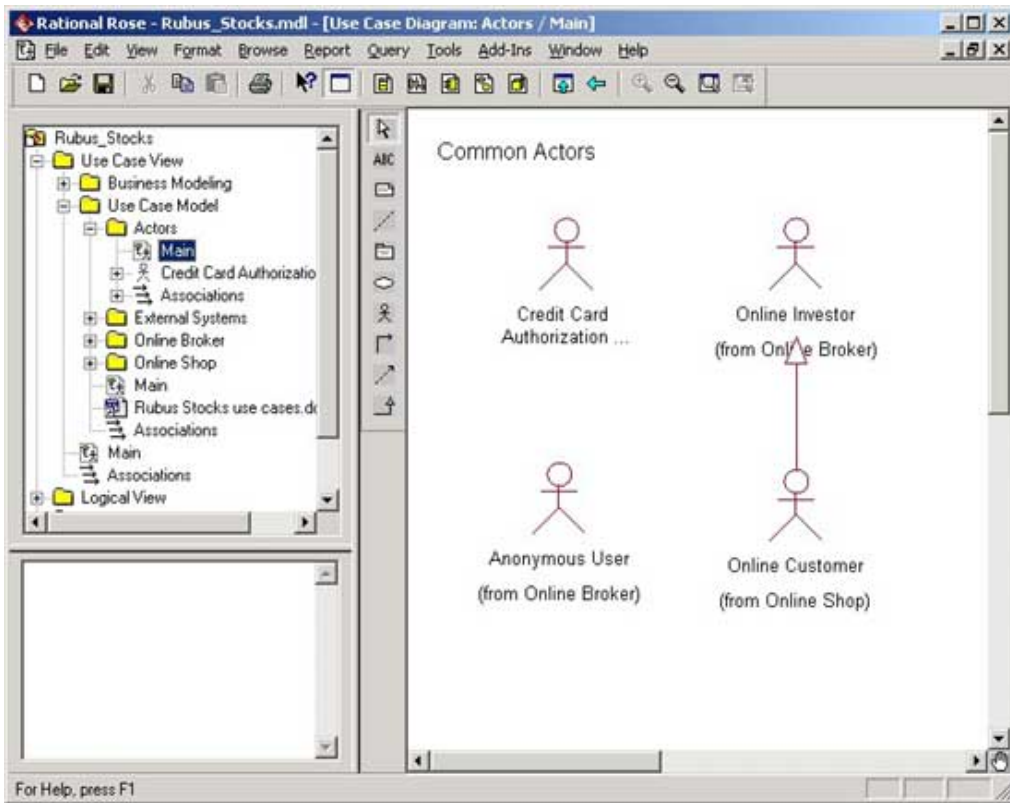
<b>Object</b>	<b>Method</b>	<b>RoleA</b>	<b>RoleB</b>	<b>RoleC</b>
Object1	Method1	X		
Object1	Method2	X	X	
Object1	Method3	X		X
Object2	Method1	X		
Object2	Method2		X	

## Identify Roles

As we saw in the overview, this is the easy part of the process. First we review the Use Case View of the UML System Model and then check the Use Case Model for definitions of the actors (see Figures 1 and 2).



**Figure 1: Use Case View of a UML Model**



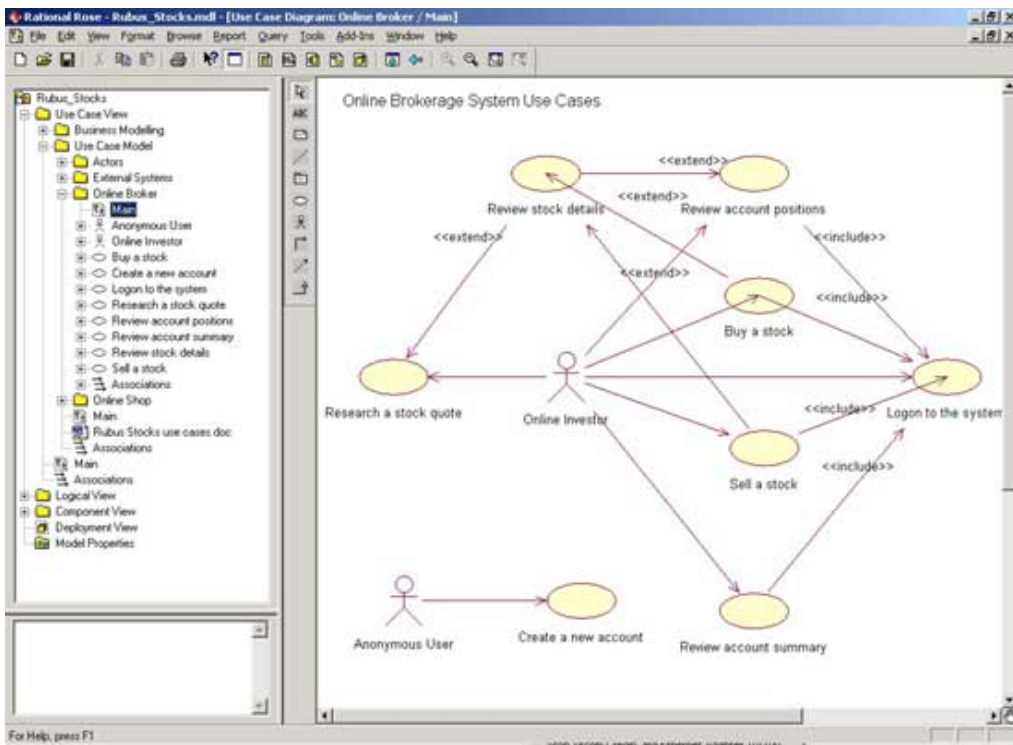
**Figure 2: Actors in a Use Case Model**

By looking at Figure 2, we can see that our system should include four roles:

- Online Investor
- Online Customer
- Anonymous User
- Credit Card Authorization System

The three first roles are internal to the system, but the fourth represents an external system. For role-based security to work well within an application server, the external system should have a boundary component that integrates with the application server. This is the only way the application server can control the complexity of the security set-up within its boundaries.

The "Anonymous actor" associated only with the "Create Account" use case represents all users who have not yet registered with the system but want to create a new account. Reviewing the use case model in Figure 3, we can also observe that the "Create Account" use case is associated only with that actor. This means that other actors cannot create new accounts. In other words, an actor must have an "anonymous" status in order to create a new account.



**Figure 3: Online Brokerage System Use Cases**  
 (View [full size graphic](#) in new window)

Figure 3 also shows us that the "Online Customer" actor inherits from the "Online Investor." This means that any function accessible by the "Online Investor" will also be accessible by the "Online Customer." Note, however, that in association with the "Logon to the System" use case (the "Online Customer" must logon before using any shopping feature), the converse is not true: the "Online Investor" cannot access customer features. This observation is important for the security model. It means that the role representing the "Online Customer" should systematically be granted access to all objects and methods that "Online Investor" can access.<sup>2</sup>

## Decide Which Objects and Methods Each Actor Can Access

As we saw in the overview, the next step is to decide, via a careful review of the UML System Model's sequence diagrams, which objects and methods to make accessible to each actor. This involves reviewing each sequence diagram that represents the interaction between an actor and the system's classes in the context of a use case. The underlying assumption is that all use cases are modeled with interaction diagrams.

We should consider two levels of sequence diagrams:

- The analysis level that shows interactions between the actors and boundary types of classes.
- The design level that shows interactions between the actors and classes of:
  - The presentation layer (for an interactive user)
  - The business layer (for an external system)

For the purposes of this discussion, we will assume an interactive user. For an external system, a business layer would access another business layer, possibly with a presentation layer at the boundaries of the external system.

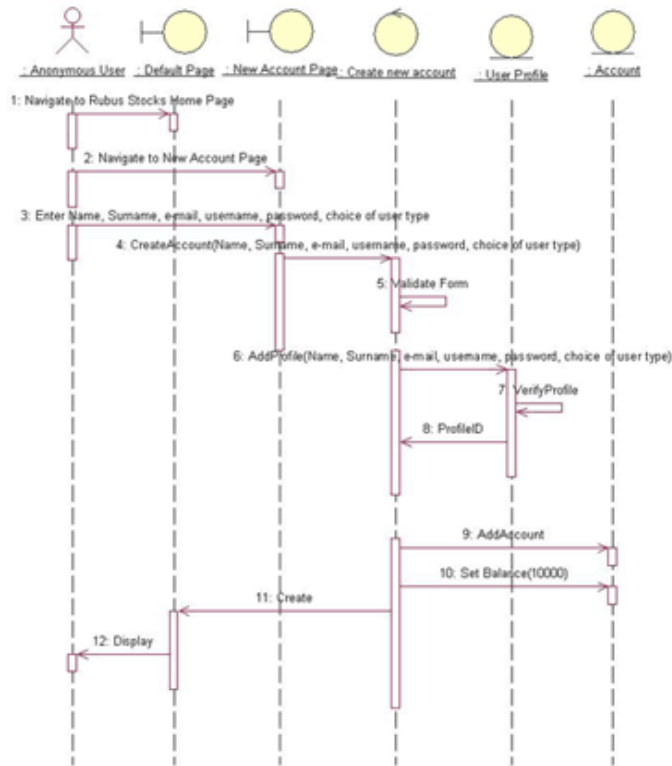
It is also good practice to build two levels of role-based security models:

- An analysis level that shows all the classes the actor can access. There is no need to list the methods in detail. The assumption is that the security model at this level is coarse, and that the design model will refine it. If a role accesses an object in the analysis model, it could well be that later, in the design model, the role will access only part of that object or will not access it at all.
- A design level that accurately reflects the object that will be visible during the configuration of the application in the application server. For each role, this model specifies all the details regarding access to objects and methods.<sup>3</sup>

### Analysis Level Security Model

In our sample system, let's consider the "Create New Account" use case. Figure 4 is the sequence diagram for the UML Analysis Model. The "Default Page" and the "New Account Page" are the two boundary classes with which the "Anonymous User" directly interacts. But for the Analysis Security Model we also consider the other classes involved in the interaction. The security matrix below reflects the level of detail covered by the UML Analysis Model.

<b>Object</b>	<b>Anonymous User</b>
Default Page	X
New Account Page	X
Create New Account	X
User Profile	X
Account	X

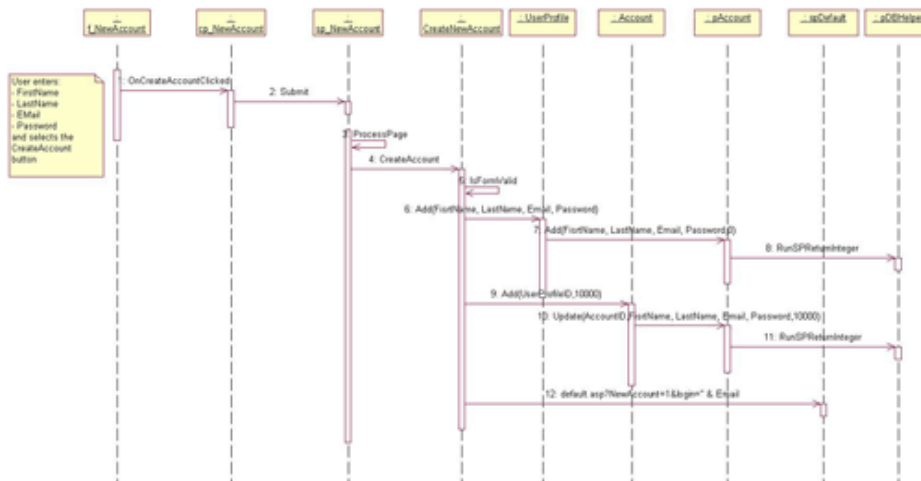


**Figure 4: Sequence Diagram (Analysis) -- Create New Account**  
 (View [full size graphic](#) in new window)

The next step in the process is to review the UML Design Model.

### Design Level Security Model

Following the same process as for the Analysis Model, we next examine all the sequence diagrams that were developed to cover use cases. At this level we also list all the methods for the objects, considering only public methods. Figure 5 presents the design model sequence diagram for the "Create New Account" use case. The actor is not represented at this level of detail; it is understood to be the "Anonymous User."



**Figure 5: Sequence Diagram (Design) -- Create New Account**  
**(View [full size graphic](#) in new window)**

As a first cut, we could decide that all the objects in the sequence diagram can be accessed by the role representing the actor who originates the interaction. At this level of detail we also specify which methods of each object can be accessed. The matrix below shows the methods of all the objects that participate in the collaboration:

<b>Object</b>	<b>Method</b>	<b>Anonymous User</b>
f_NewAccount		X
cp_NewAccount	OnCreateAccountClicked	X
sp_NewAccount	ProcessPage	X
spDefault		X
CreateNewAccount	CreateAccount	X
UserProfile	Add	X
UserProfile	VerifyUser	
Account	Add	X
Account	Update	
Account	GetAccountInfo	
Account	GetSummary	
Account	ListPositionsForSale	
Account	ListPositions	
pAccount	Add	X
pAccount	Summary	
pAccount	VerifyUser	
pAccount	GetAccountInfo	
pAccount	Update	X
IpDBHelper	GetConnectionString	
IpDBHelper	RunSPReturnRS	
IpDBHelper	RunSQLReturnRS	

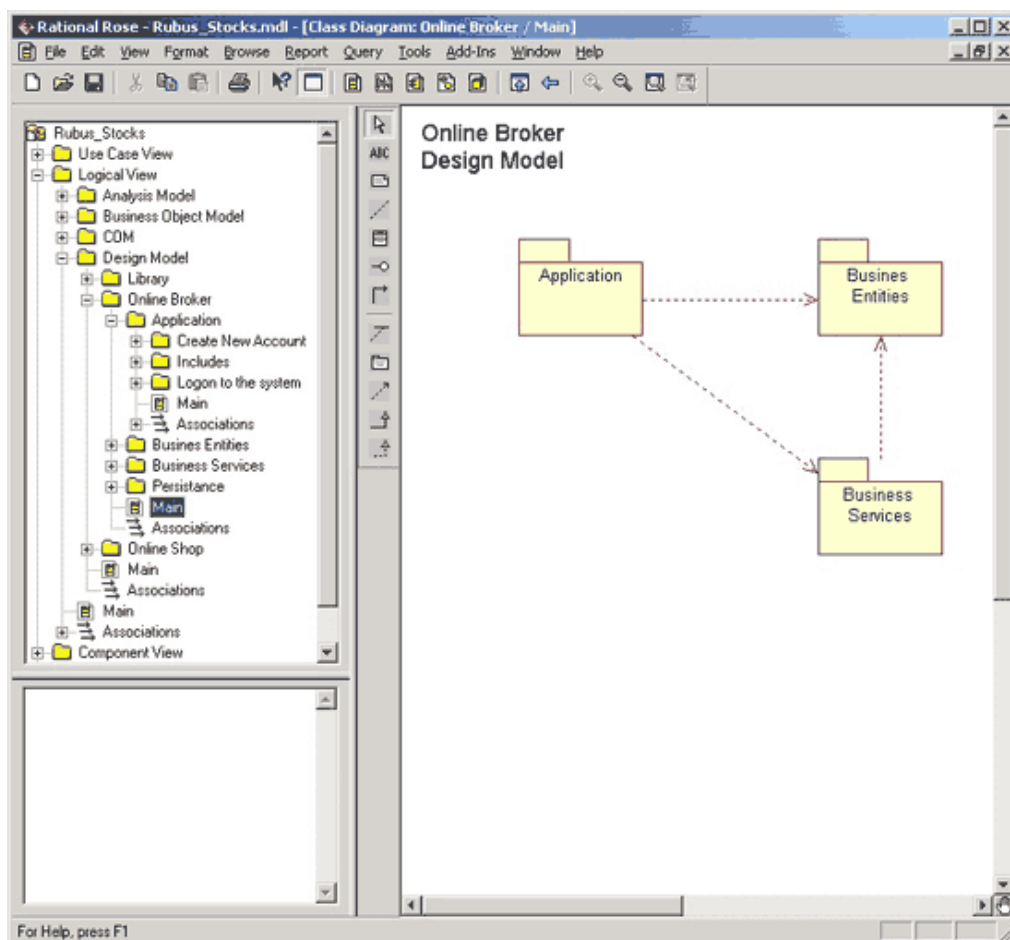
IpDBHelper	RunSPReturnRS_RW	
IpDBHelper	RunSQLReturnRS_RW	
IpDBHelper	RunSP	
IpDBHelper	RunSQL	
IpDBHelper	RunSPReturnInteger	X

Note that we can already select exactly which methods of the objects to be installed in the application server need to be accessed by the particular role.

## Additional Security for Layered Architectures

We must also consider another dimension of modern system design. An application is segmented into a number of layers -- typically the presentation, business, and persistence layers. Furthermore, the presentation layer can be split into a controller and a view layer when implementing the Model View Controller (MVC) approach.

In our model, we can analyze the system's structure by examining the UML Design Model (see Figure 6). We can immediately see that the actor must be able to access the presentation layer but not the other layers.



**Figure 6: A UML System Design Model**  
(View [full size graphic](#) in new window)

We can also see that there are two major subsystems: Online Broker and Online Shopper, which correspond to two major roles we defined: "Online Investor" and "Online Shopper." In reviewing the Online Broker subsystem, we can clearly identify the following system segments:

- Application
- Business entities and business services
- Persistence

Note that the organization for the application segment echoes the divisions for use cases. We can now add corresponding classes from our previous security matrix to the list of segments:

- Application: f\_NewAccount, cp\_NewAccount, sp\_NewAccount, spDefault, CreateNewAccount
- Business entities and business services: UserProfile, Account
- Persistence: pAccount, pDBHelper

To achieve a higher level of security configuration, we can separate the roles that are allowed to access each of these segments. Clearly, the application segment will be accessible by roles associated with use case actors. In our example, the role/actor is "Anonymous User," and the part of the security matrix that covers the application segment will not change for that role.

For the other segments, we must introduce two new roles that do not correspond to any use case actor. We will assign them distinctive names to avoid confusing them with roles that do correspond with use case actors. The "sysApplication" role will be granted access to the business entities and services segment. Each class in the Application segment will be configured to play that role when it collaborates with the business entities and services segment. We can conceptualize the classes of the application layer as one actor (hence the role sysApplication) that interacts with the system defined by the business entities and services segment. Then, in the same way, we will define "sysBusiness" as a role that can access the persistence segment.

We can now review the security matrix, adding the new roles and shifting the access rights to the correct role, based on the segment to which the class belongs:

<b>Object</b>	<b>Method</b>	<b>Anonymous User</b>	<b>sysApplication</b>	<b>sysBusiness</b>
f_NewAccount		X		
cp_NewAccount	OnCreateAccountClicked	X		
sp_NewAccount	ProcessPage	X		
spDefault		X		
CreateNewAccount	CreateAccount	X		
UserProfile	Add		X	
UserProfile	VerifyUser			

Account	Add		X	
Account	Update			
Account	GetAccountInfo			
Account	GetSummary			
Account	ListPositions ForSale			
Account	ListPositions			
pAccount	Add			X
pAccount	Summary			
pAccount	VerifyUser			
pAccount	GetAccountInfo			
pAccount	Update			X
pDBHelper	GetConnection String			
PDBHelper	RunSPReturnRS			
PDBHelper	RunSQLReturnRS			
PDBHelper	RunSPReturnRS_RW			
PDBHelper	RunSQLReturnRS_RW			
PDBHelper	RunSP			
PDBHelper	RunSQL			
PDBHelper	RunSPReturnInteger			X

In the application server, each component will be installed to run with a specific identity (actual user account). For the "User Profile" class to be accessible by "Create New Account," as documented in the sequence diagram, we need to configure "Create New Account" to run under an identity that will be part of the list of actual users assigned to the "sysBusiness" role. In addition, we should avoid associating users who will be associated with the "Anonymous User" role with the "sysBusiness" role; otherwise, we defeat the purpose of having two roles.

At this point the security matrix has to be implemented as a configuration for an application server. How this will actually be done depends on the application server used for the implementation.

## **Automating Specifications with Rational Rose**

This systematic approach for deciding how to configure role-based security when installing a component-based application in an application server facilitates the work of technical architects and introduces a rational dimension to this activity. Because it works from the system design specified in detail within a UML System Model, it would be highly desirable to have a tool that could automatically analyze such a model and configure the application server with the correct roles and access rights for the implemented interfaces.

Actually, Rational Rose comes close to achieving this. Rational Rose exposes all its functions as COM objects and can also be extended with Visual Basic for Applications (VBA). If COM+ is the target application server, then we can access its COM interface

from VBA and thus create a Rational Rose add-in that will directly install the components with the correct security settings in COM+. This brings us one step closer to the objective of a completely automated system specification based on design.

---

<sup>1</sup> The method described in this article is based on work done for an e-commerce sample system for Rubus, named Rubus Stocks. This system, in turn, is based on a sample system named FM Stocks, a Web application commissioned by Microsoft and developed by Vertigo Software, Inc. as part of a Windows Distributed interNet Architecture (DNA) performance and scalability study. Complete documentation for this original sample can be found at Fitch & Mather Stocks 2000: [Introduction and Article List](#).

<sup>2</sup> This sample was designed to exemplify various modeling techniques. The rationale behind the usage may not necessarily reflect a typical e-business scenario, but it is internally consistent.

<sup>3</sup> This detailed matrix could be used as the input for an automated configuration program. This would be particularly useful if the application server supports configuration through an API (e.g., COM+ defines a set of COM+ objects to support the configuration features that are also exposed through system tools).



***For more information on the products or services discussed in this article, please click [here](#) and follow the instructions provided. Thank you!***

Copyright [Rational Software 2001](#) | [Privacy/Legal Information](#)