Rational Process Workbench™

Process Developer's Guide

VERSION: 2002.05.00

PART NUMBER: 800-025093-000

WINDOWS 98 SE, ME, NT, 2000, XP

Rational®
the software development company

support@rational.com
http://www.rational.com

Part Number: 800-025093-000

Version Number: 2002.05.00

Contents

Figures

Preface

This document is the user's guide for the Rational Process Workbench™ (RPW), which is a tool for customizing and publishing the Rational Unified Process® or RUP® based process.

This manual describes in detail how you use RPW to develop and model your own process, and how to publish Web sites based on your own process definition. The RPW assists process engineers by accelerating their delivery of a customized software development process, allowing them to visually model process using Unified Modeling Language (UML).

This tool is for use with Microsoft Windows 98 SE, ME, Windows NT, Windows 2000, and XP operating systems.

Audience

This manual is intended for process engineers who are familiar with UML as an object-oriented modeling language and with object-oriented design using Rational Rose.

Other Resources

You can view online Help whenever you need assistance by doing the following:

- From any RPW display, press the F1 key and select an option from the Help menu.

This manual is available in both printed and PDF formats. See the Rational® Suite Documentation CD for the PDF file.

For information on developing process plug-ins using Rational Process Workbench (RPW), refer to the RUP Resource Center (www.rational.net/rupcenter/). The RPW Tutorial is also located on the RUP Resource Center.

For an introduction to RPW and information about installing this tool, see the Rational Process Workbench manual titled *Getting Started*.

Rational Process Workbench Documentation

This document covers the following topics:

- It presents and describes the underlying semantic model, which is both the foundation of RPW's operation and the model that defines the *terminology* of the process modeling language.
- It describes RPW's operation, and the steps involved in developing and publishing processes.
- It provides a complete command reference for RPW.

For anyone interested in process engineering, in general, and how this product applies to your organization, please read:

- Chapter 1, An Overview of the Rational Process Workbench
- Chapter 2, Modeling Elements and Principles
- Chapter 3, Process Content Libraries
- Chapter 4, Working with Process Models and Publishing Web Sites

Of particular interest to process engineers are:

- Chapters 1 through 4, introduced above
- Chapter 5, Working with Rational Process Workbench
- Chapter 6, Command Reference
- Appendix A for information on RPW commands in HTML files

A Glossary of terms is also provided.

Contacting Rational Technical Publications

To provide feedback about documentation for Rational products, please send e-mail to our technical publications department at techpubs@rational.com.

Contacting Rational Technical Support

If you have questions about installing, using, or maintaining this product, contact Rational Technical Support as follows:

Your Location	Telephone	Facsimile	E-mail
The Americas	(800) 433-5444 (toll free) (408) 863-4000 Cupertino, CA	(781) 676-2460 Lexington, MA	support@rational.com
Europe, Middle East, Africa	+31 (0)20 4546 200 Netherlands	+31 (0)20 4546 202 Netherlands	support@europe.rational.com
Asia Pacific	+61 2 9419 0111 Australia	+61 2 9419 0123 Australia	support@apac.rational.com
World Wide Web	www.rational.com		

Note: When you contact Rational Technical Support, please be prepared to supply the following information:

- Your name, telephone number, and company name
- Your computer's make and model
- Your operating system and version number
- Product release number and serial number
- Your case ID number (if you are following up on a previously-reported problem)

An Overview of the Rational Process Workbench

1

RPW is a new type of tool that brings process engineering to its next level of maturity. Your ultimate goal in using this product is to publish process Web sites that describe your software engineering processes.

RPW introduces *process modeling* to manage the structural complexity of software engineering processes and to allow specification of process at a higher level of abstraction than the process text itself.

With RPW you use both the expressiveness of UML as a modeling language and Rational Rose's visual modeling support. RPW defines a specific language for process modeling and provides support for modeling when using this language.

There are four entities involved in developing process with RPW.

- **Process Model** — In the process model, you define your process elements; that is, your Roles, Artifacts, Disciplines, and so forth. This is also where you express how they relate to each other and specify how they collaborate during the course of the process. Process Models are defined in Rose's logical view.
- **Component Model** — The component model contains the definitions of process in terms of process components. This is where you select those process elements you want to include in your own process. Component Models are defined in Rose's component view.
- **Process Content Library** — The Process Content Library is the collection of HTML files, including their descriptions and any other material that will ultimately make up the Web site. This is where you author the text that describes your own process.
- **Process Web Site** — Process Web sites are published by RPW. The process Web site is the final result of your development activities using RPW.

The underlying metamodel recognized by RPW is an extension to the Unified Modeling Language (UML). RPW uses the UML's standardized extension mechanisms to specify its modeling concepts and operates as an add-in to Rose.

Configuration Management

RPW separates the process structure (model) from its content (process text).

When several process developers work in parallel to develop the same process model, the basic principle of dividing the process model into *controlled units* is applied, just as it's described for Rational Rose usage in general. To keep your process modeling situation simple, we recommend that only one person is assigned the responsibility of actually *modeling* the process inside of the process model.

This basic principle of configuration management is also used to control branching and merging of process models: the RUP® process model is delivered as a separately "controlled unit" file, and your customizations of that model needs to occur in your own separate process model. This pattern may be repeated. For example, a large-scale organization may want to customize their process at multiple organizational levels. This is achieved through each organizational level developing its unique customization in its own, separate process model, which is based on the process customization of its enclosing organizational unit. This forms a chain of dependent process models, with each process model developed independently of the others, and where each process model is a refinement of its parent model.

Configuration management of a Process Content Library is not integrated with the configuration management of its process model. The Process Content Library is a directory structure where the files and folders may be units of configuration management. Each process model is associated with its own Process Content Library. Therefore, in an environment where multiple process models are used, multiple Process Content Libraries will also exist, each of which may be configuration-controlled separately.

To maintain the integrity of the RUP Process Content Library, it must be treated as a read-only library.[1] In the same way that customizations occur in separate process models, their accompanying Process Content Libraries reside in their own file directory.

1. See the section titled *Setting Up and Managing the RPW Workspace* on page 40 for details.

Modeling Elements and Principles

2

This chapter defines the elements and principles that you use to define your own process models. Collectively, these definitions are referred to as the *semantic model* of the RPW.

A fundamental premise for RPW, and its semantic model, is that it's entirely based on UML—each element in the semantic model is derived from a standard UML element. RPW uses stereotypes to designate the UML elements to their particular (process modeling) kind. Therefore, RPW defines a "small language" that's specific for process modeling based entirely on UML and for which the "words of the language" are given by the stereotypes used.[1] In fact, this "small language" uses only a small subset of UML to define the dozen, or so, modeling concepts supported by RPW.

Figure 1 illustrates the semantic model, not the organization of your process model, using three packages to separate and logically group its concepts.

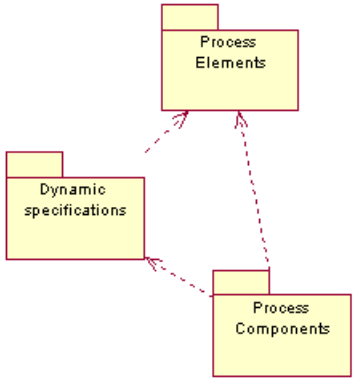

Figure 1 Overview of the semantic model

Process modeling concepts are divided into three categories, depending on their purpose in process models:

1. From a UML perspective, this semantic model and its stereotypes form what UML calls a *profile*.

- **Process elements** define the core concepts used in process modeling and how these are associated to each other. This is where concepts like *Role*, *Artifact*, *Activity*, and *Discipline* are defined.
- **Dynamic specifications** define concepts that specify how the process elements collaborate in a process. UML activity diagrams and sequence diagrams are used for these specifications.
- **Process components** define concepts that specify how process elements are grouped into components to form "chunks of process" that will be deployed as collective units.

This chapter refers to the two Rose views used by RPW —**logical view** and **component view**. Process elements and dynamic specifications are defined in the logical view, and process components and process closures are defined in the component view.

Figure 2 shows the Rose modeling space where you'll be developing your process models. Process models are developed in Rose's logical view and component models are developed in its component view.

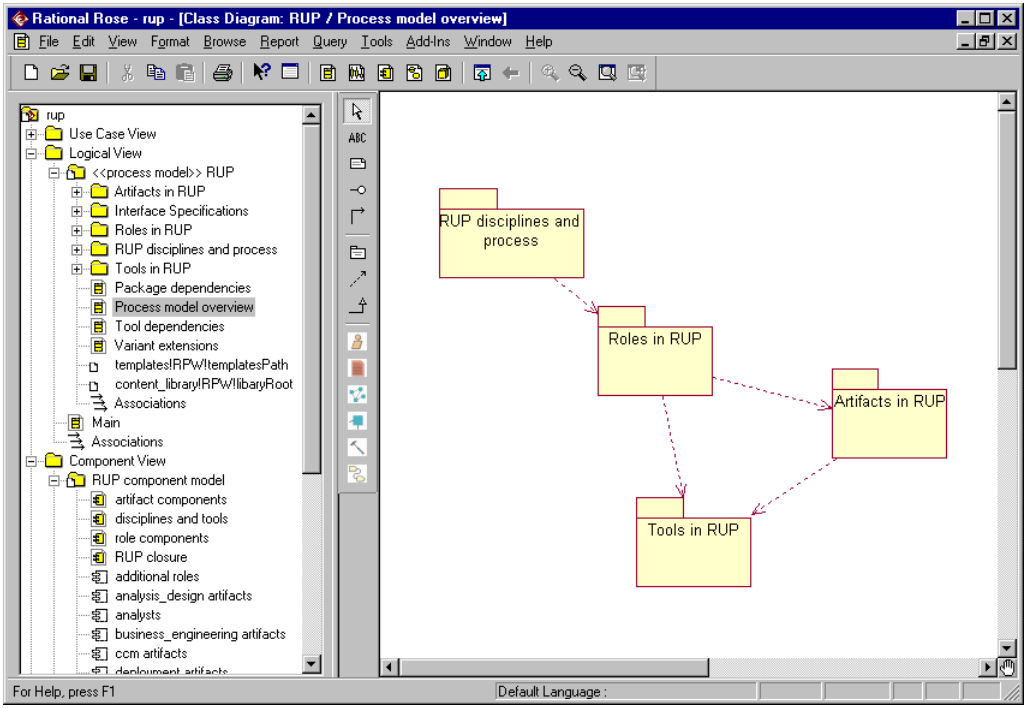

Figure 2 Rational Rose modeling space

The Use of Stereotypes

RPW's entire operation is based on using UML stereotypes as the means of assigning process elements to their particular kind. RPW defines stereotype icons for the different concepts, and you specify the model element kind by setting the stereotype value for the modeled element. **This is key to the whole operation of RPW.**

Process Elements

Process elements are defined using UML classes and operations, which are stereotyped to carry their specific meaning. In addition, the various relationships between elements are defined using associations, also stereotyped to their specific kind.

Figure 3 presents all process concepts recognized by RPW. In this diagram, which is extracted from the semantic model shown in Figure 1, the stereotypes show the UML constructs that are used for their modeling.

Although Figure 3 may appear complex at first glance, here's how to read it—the "things" in the diagram are the "things", or process concepts, that you use to build your process models. In a way, these are the terms of the "process modeling language".

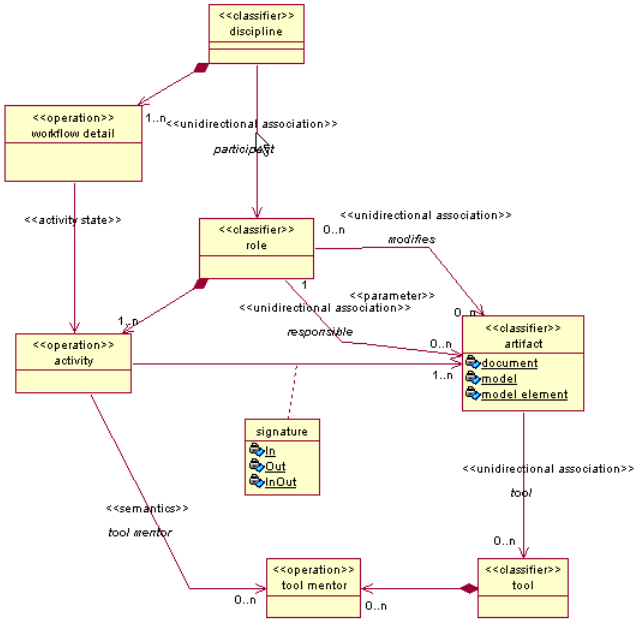

Figure 3 Process elements of the semantic model

Each process concept defines how it's modeled (operation or classifier) and its associations to other process concepts. For example, Role elements are modeled as classifiers, and they define operations to represent their Activities; Role elements can define associations to Artifact elements through «modifies» and «responsible» associations.

Figure 4 displays an example of a model segment showing the use of stereotypes in your process model.

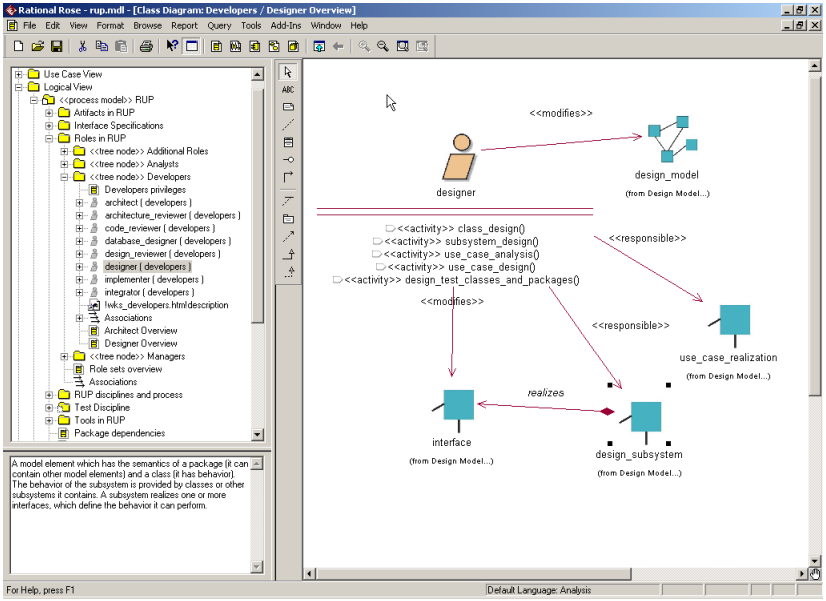

Figure 4 A sample model segment

Artifacts

Artifacts are modeled as classes. RPW recognizes three stereotypes for Artifacts: «model», «model element», and «document».

This classification of Artifacts is not currently used by RPW and they're all treated the same. However, it's good practice to determine what an Artifact's type is when it's created and what relationships exist between the different Artifacts specified.

Roles

Roles[2] are modeled as classes, stereotyped to «role».

Roles are assigned responsibility over Artifacts through «responsible» associations.

Roles define associations to those Artifacts that the Role is authorized to modify through «modifies» associations. Only these Artifacts can be specified as being created or modified by Activities defined on the Role.

2. Formerly known as "workers" in previous versions of the RUP and RPW.

Activities

Activities are modeled as operations, stereotyped to «activity» on their performing Roles.

Activities operate upon Artifacts and the UML signatures of **{In, InOut, Out}** are used to specify the way a particular Activity uses and modifies its parameter Artifacts. Only Artifacts that the performing Role is «responsible» for or has «modifies» authority over are eligible for **InOut** or **Out** assignment.[3] Artifacts within the visibility scope of the Role are eligible for **In** assignment.

Tool Mentors are associated to Activities and this association is managed by the RPW Overview Dialog.

Disciplines

Disciplines[4] are modeled as classes, stereotyped to «discipline».

Disciplines are process elements that define distinct boundaries within a process. Each Discipline identifies a set of Roles that participate in the Discipline and defines a set of Workflow Details that specify the collaborations of these Roles, and their Activities, within the Discipline.

The Roles that participate in a particular Discipline are associated to that Discipline through «participant» stereotyped, unidirectional associations from the Discipline to its Roles.

Note: One Role may participate in multiple Disciplines.

You specify how its Workflow Details engage as the Discipline is conducted in an activity diagram, associated to the Discipline class itself. This diagram provides an overview of the Discipline itself, and the details of the collaborations of the Discipline are defined entirely by its Workflow Details and their respective activity diagrams.

The activity states in the Discipline's activity diagram designate the Workflow Details.[3.]

Workflow Details

Workflow Details are modeled as operations, stereotyped to «workflow detail» on their Discipline classes. Workflow Details do not specify any operation parameters.

A Workflow Detail specifies one specific collaboration within its Discipline.

---

3. A special-purpose RPW dialog supports the insertion of this information in process models.
4. Formerly known as "workflows" in previous versions of RUP and RPW.

The collaborations of Roles and their Activities in a Workflow Detail are specified in an activity diagram associated to the Workflow Detail. Only Roles that participate in a Discipline are eligible for being involved in its Workflow Details.

Tools

Tools are modeled as classes, stereotyped to «tool».

Tools represent a particular development tool used in an organization.

Tool Mentors

Tool Mentors are modeled as operations, stereotyped to «tool mentor», defined on Tool classes. They do not specify any parameters.

A Tool Mentor provides a recipe for how to perform a certain Activity using a particular Tool.[3.]

Processes

Processes, as static elements[5], are modeled as classes, stereotyped to «process». Process is the model element that contains the definition of the Web Site Form, which is the invariable part of your process Web site.

Dynamic Specifications

The dynamic behavior of a process is described by the collaborations that occur among its participants. Such collaborations exist at different levels and with different levels of "exactness and formality".

Figure 5 provides an overview of the diagram types used to specify the dynamic behavior of a process and the respective process elements that they describe.

- **Discipline workflows** provide an abstract overview of the collaborations that occur within their Disciplines.
- **Activity Overviews** provide a concrete, more detailed overview of individual collaborations within Disciplines.

5. "Process" also exists in the context of components, which is a different thing. For details, refer to the information under the heading *Process Components*, in the subsection titled *Process components*.

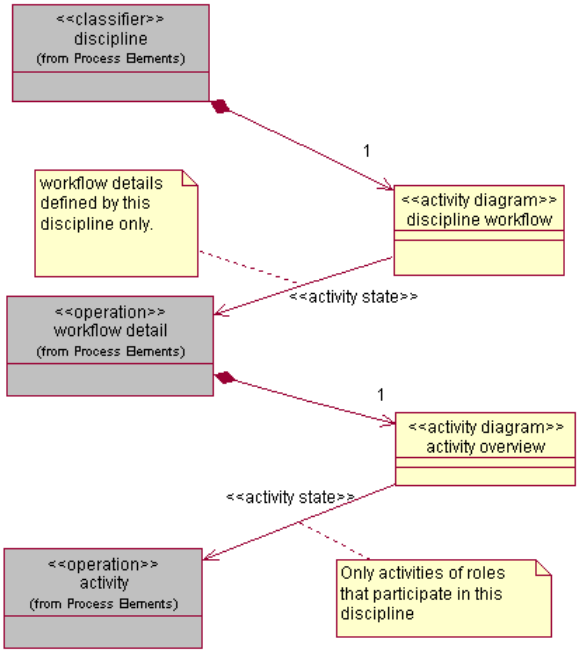

Figure 5 Dynamic specifications of the semantic model

Discipline Workflows

Discipline workflows are *abstract workflows* that describe the overall Activity model of their Disciplines.

Each activity state in a Discipline workflow diagram is associated to one particular Workflow Detail operation, defined on the Discipline.

Figure 6 illustrates how Discipline workflows are modeled using activity diagrams.

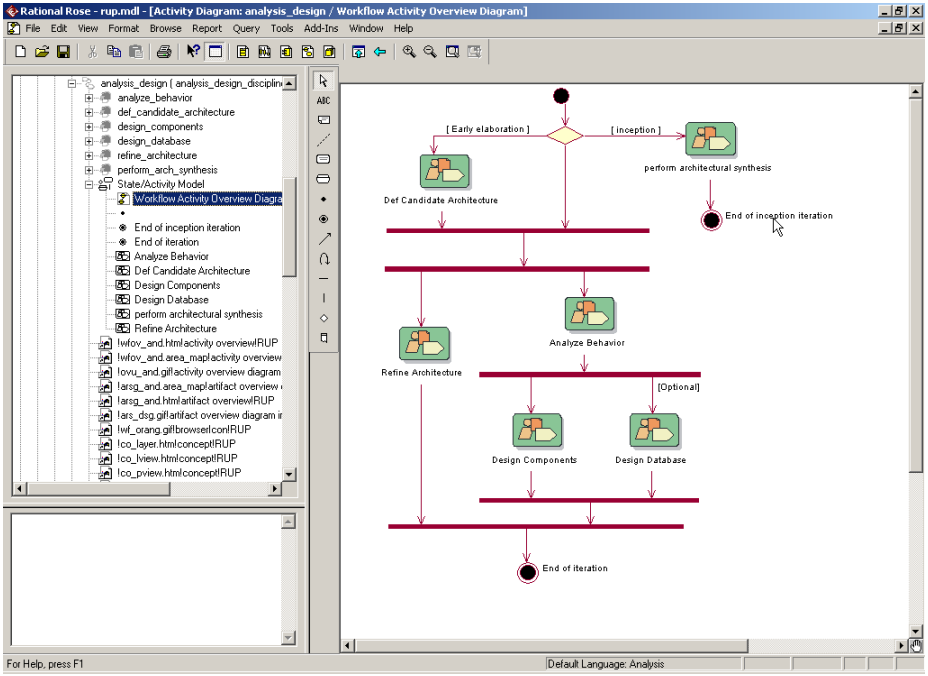

Figure 6 Modeling a Discipline workflow

Activity Overview

Activity Overviews are activity diagrams that describe the collaborations of individual Workflow Details.

Each activity state in an Activity Overview diagram is associated to one Activity operation. Only Activity operations performed by the Discipline's participating Roles are eligible for association.

Figure 7 shows how Workflow Details are modeled using activity diagrams. It also shows the technique that RPW uses for associating activity states to their Activity operations.

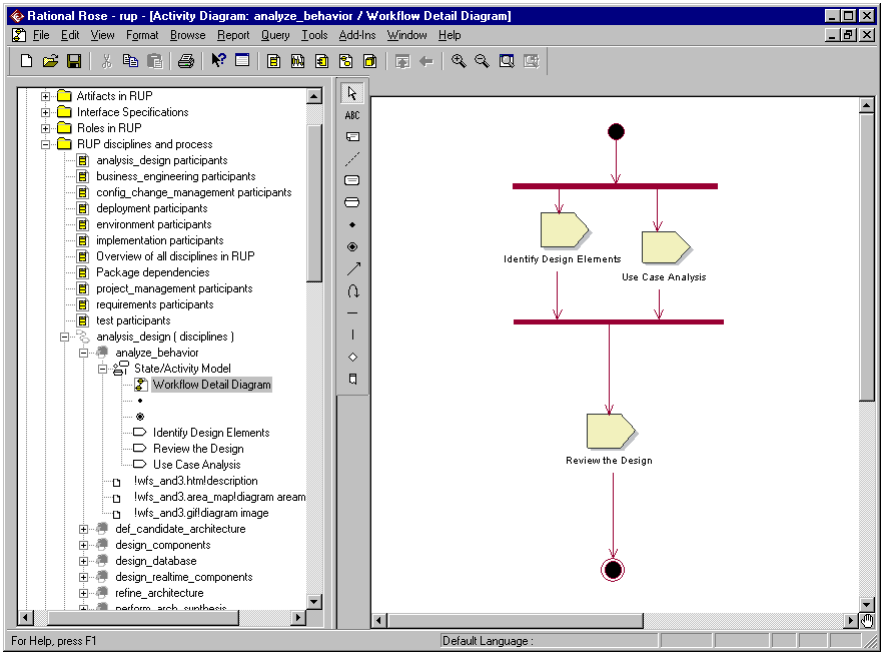

Figure 7 Modeling Workflow Details

Process Deployment Specification

RPW bases the publishing of processes on process components, which are components specified in the **component view** of Rose. A collection of components specifies the closure of a given process.

A process component realizes one or more process elements from one or more process models. Process components represent non-arbitrary sets of process elements that are internally consistent and may be reused with other process components to assemble complete processes. A natural criteria for defining a component is that its realized elements should form a natural group of process elements that are strongly related to each other.

The component view of Rose is used for specifying how processes are deployed to their resulting process Web sites.

The concept of component as supported by Rose is used for specifying which process elements that should be realized as “atomic units of process deployment”.

A special kind of process component, «process», represents a complete, end-to-end process. It is distinguished from other process components in that it does not realize any process elements itself, but instead it identifies the component models that collectively constitute its particular configuration.

Component models package process components into configurable units.

Figure 8 shows an example of a process configuration and how it is specified in terms of its included component models.

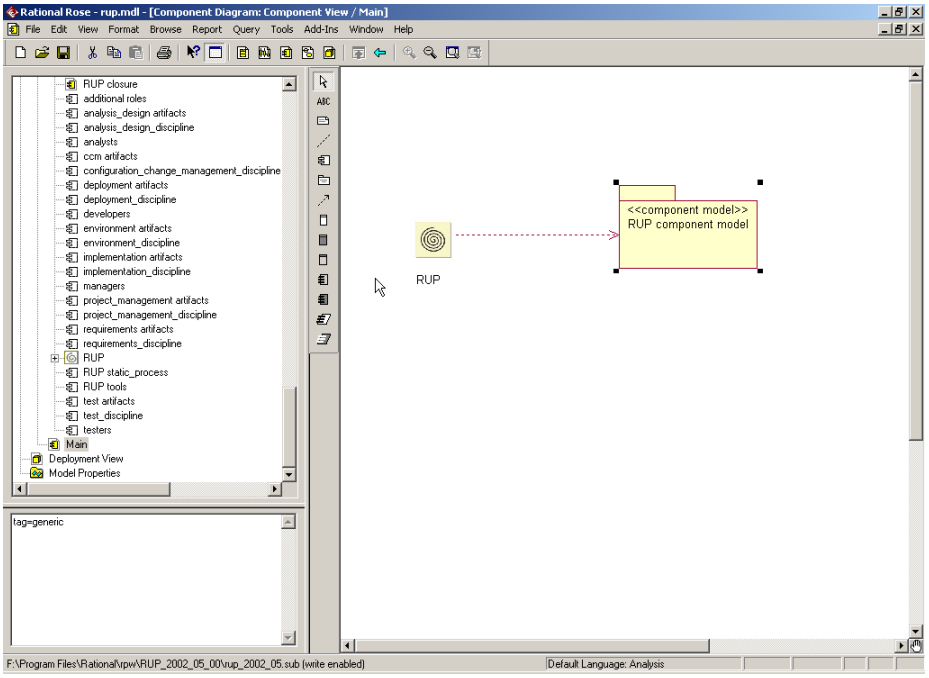

Figure 8 RUP components defined in Rose’s component view

Figure 9 shows the section of the semantic model that specifies the concepts involved in process deployment.

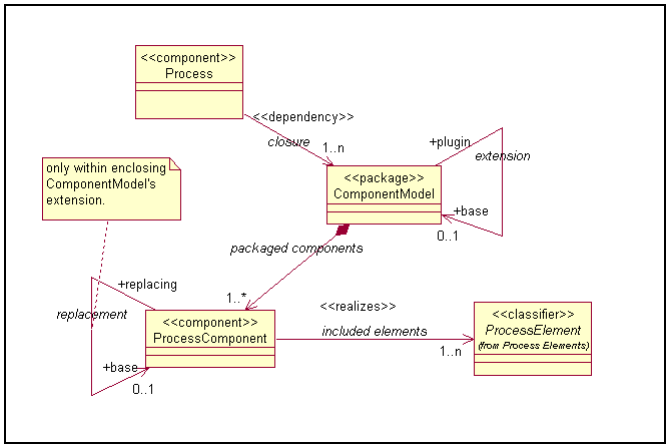

Figure 9 Process components of the semantic model

Process Component Models

Process Component Models package process components into process plug-ins, and process configurations are specified in terms of such component models.

A process component model can be a *base model*, which means that it specifies a standalone and self-contained process closure—an *extension model*—it extends an existing component model and adds or replaces components to the set defined by its base. An extension component model requires the presence of its base component model to form a complete closure.

Extension component models can also extend other extension component models to any depth forming more specialized plug-ins.

Each extension component model specifies the relationships between its contained components and the components in its base. The default is that an extension model receives *all* components from its base component model in addition to its own components.

An extension component model, however, can also specify *replacement* dependencies between its own components and components in its base, in which case the replaced component from the base is suppressed in the extension component model and only

the replacing component will be included. This mechanism, therefore, provides the means to replace base elements with more specialized ones in processes (see Figure 10).

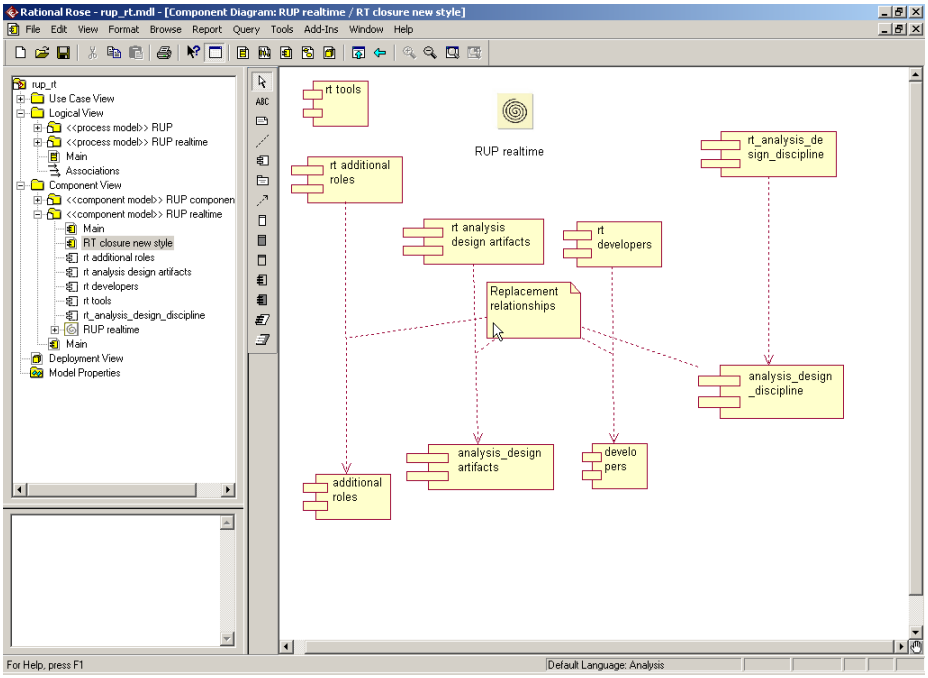

Figure 10 An example of component dependencies

Assess the closure of a process configuration

The special-purpose component type—designated with the «process» stereotype—represents the configuration of a process. A «process» component is associated to one or more component model packages, each of which contains a set of process components.

The closure of a process configuration is transitively defined through the contained components of its associated component models, and to their realized process elements.

Therefore, «process» components can be regarded as the roots from which closures of process are calculated. It is through the process' configuration that you determine the exact set of process elements that will be deployed into its published Web site at the time of generation.

A process configuration's closure can be checked for well-formedness to determine if it is correct or not. A well-formed process configuration is one where all realized process elements' expectations, in terms of other elements, are satisfied and where no ambiguity exists.

Model Management

Process models can accommodate any structure and organization of their contained process elements. This includes the use of packages, in general, to manage the complexity of the process model.

However, two types of packages that have special meaning in RPW exist—«process model» and «tree node» packages. RPW's operation is based on the visibility scope of process elements, established at their enclosing package. For example, in a well-formed process model, a «role» element can only reference «artifact» elements that are within the visibility scope of the «role» element. You use package dependencies to control visibility.

Figure 11 displays an overview of the RUP process model.

The «process model» package is a mandatory process element that provides the top-level container for the elements of a process model.

«Tree node» packages are elements that structure their contents with respect to their organization in a published Web site and their presentation in the tree browser.

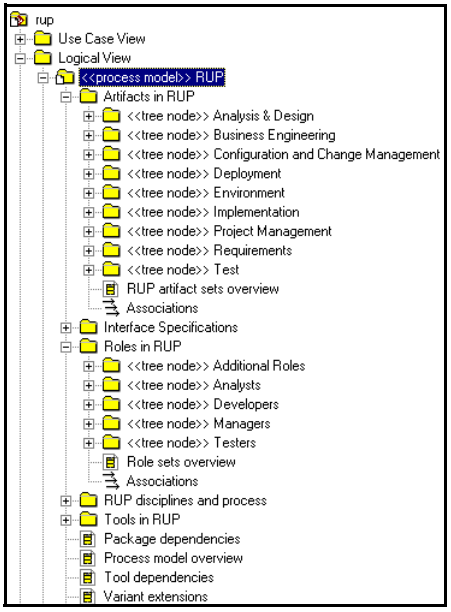

Figure 11 An overview of the RUP process model

Figure 12 illustrates these two package types used for structuring process models. For further details, please refer to the section titled *What determines the content and organization of the tree browser?*, found in *Chapter 4, Working with Process Models and Publishing Web Sites*.

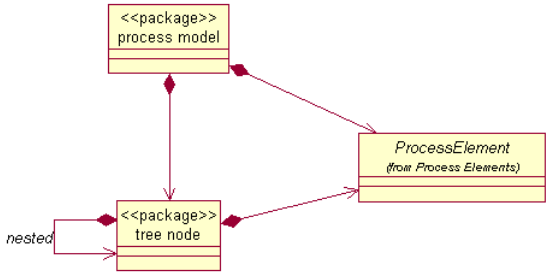

Figure 12 Elements for managing the process model

Using «process model» packages

The term *process model* is somewhat confusing because it's used both to loosely describe "the whole entity that RPW operates upon" and one specific package type inside such "whole entities". This section describes the latter—the special purpose package type that exists in Rose's logical view and that constitutes the mandatory top-level container of process elements.

Process models are modeled in packages in the logical view, stereotyped to «process model». To be valid, every process element must reside inside of a process model.

Within a «process model», its contained process elements may be arbitrarily structured using regular packages to manage the complexity of the model, as well as the «tree node» packages to specify the organization of the tree browser. Process elements obey the visibility rules defined between packages.

Using «tree node» packages

«Tree node» packages are used for structuring your process elements as you want to see them organized in published Web sites. They appear as intermediate nodes in the published tree browser.

Deriving process models from existing process models

A Rose model may contain multiple «process model» packages simultaneously. This is the case when, for example, a customized process model is defined that's based on an existing RUP process model.

The general structure for working with process model derivations—that is, to create a customized version of the RUP—is that the base process model is maintained in its own separately controlled unit (`.cat` file) and the derived process model in another `.cat` file. For a detailed description on how to work within this structure, see *Chapter 4, Working with Process Models and Publishing Web Sites*, under the heading titled *Branching and merging process models*.

Modeling Techniques

The modeling language, defined by RPW, employs only a small subset of the full modeling language as defined by UML, using stereotypes to specify the process elements to their particular kind.

In addition to these specific element types, RPW supports the use of *inheritance*[6] to define new process elements based on existing ones and it supports the use of *interfaces* to separate specification from implementation of process elements. It also supports the inheritance paradigm in general; for example, to use *operator overloading* as the technique to replace existing process elements, modeled as operations, with more specialized ones.

Achieving variability

Variability of process is achieved by varying the underlying "implementation" of process elements behind their corresponding interfaces, similar to how you model object-oriented (OO) software. Such variation is best achieved using interfaces as modeling elements, but may also be achieved using inheritance or combinations thereof.

Using interfaces in process modeling

Interfaces may be used anywhere that classifiers are expected in process models, and can represent Roles, Artifacts, Tools, and Disciplines.

RPW uses regular UML interfaces in process modeling, specified to their exact kind[7]. For example, an interface defining a Role is modeled with operations stereotyped to «activity» and may be assigned «modifies» and «responsible» responsibility over Artifacts in the process model, just like their corresponding classes. Role classes that realize a particular interface receive the same responsibilities as the interface.

---

6. Single inheritance only.
7. Rational Rose does not support stereotyping of interfaces—it uses the stereotype compartment to specify the fact that a classifier is an interface—and a set of special constructs needs to exist in a process model to support modeling with interfaces.

As in software modeling, interfaces are involved in two aspects of modeling:

- they provide a specification perspective of elements in the logical view and, therefore, enable variability of a process model
- they support the *composition* of a process in terms of its constituent components

Interfaces are not process elements per se and are, therefore, only a concern in the modeling space, not in the authoring or publishing of process. In particular, they are not visible in the generated Web site and they do not have text files associated to them.

In the logical view of Rose, interfaces can be used to specify the various responsibilities of the process model. Here interfaces are applied just like when designing software: interfaces are used to enable variation of its elements.

Due to the limitations of Rose with respect to stereotyping interfaces—Rose uses the concept of "class" stereotyped to 'interface'—RPW has invented a special modeling technique to specify the particular kinds of interfaces supported by RPW: *RoleInterface*, *ArtifactInterface*, *DisciplineInterface*, and *ToolInterface*. Every interface present in a model should be derived from one of these interface elements.

In the component view, process components are assigned the responsibility to realize interfaces defined in the logical view.

Figure 13 demonstrates the modeling of Roles using interfaces. Two capabilities—"Designer" and "RealtimeDesigner"—are modeled with IDesigner and IRealtimeDesigner respectively.

The two Disciplines specify their participants: both capabilities are involved in the real-time analysis and design, whereas only the standard "IDesigner" capability is involved in the standard Analysis & Design Discipline.

The particular subsection of a process model shown in Figure 13 illustrates that the capabilities promised by the two interfaces can be realized in different combinations: either by the ComboRealtimeDesigner Role alone or by the two individual Roles Designer and OnlyRealtimeDesigner.

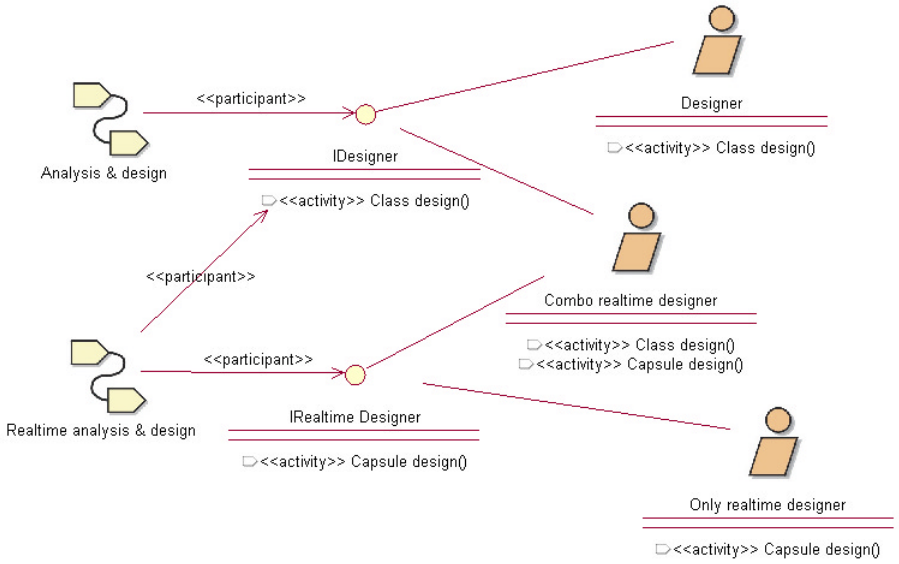

Figure 13 An example of how to use interfaces

Using inheritance

Process elements can be defined using *inheritance* to base definitions of new process elements on existing ones. Inheritance can occur across the boundaries of process models and this common technique is used to derive new customized specializations from existing process elements.

UML's definition of inheritance also applies in process modeling and inheriting process elements may substitute for their inherited elements with retained well-formedness of their process closures. The inheriting Role assumes the Activities defined on the inherited Role, along with all of its associations. In a given process closure, either class can be realized to represent the base Role.

Using operator overloading

Process elements that are modeled with operations cannot alternate by themselves—redefining an operation means you have to redefine its class as well.

RPW supports operation variability using the principle of *operator overloading*: a new operation with the *same name*[8] replaces an existing one. The new operation is defined on a derived class.

This applies to Activities, Workflow Details, and Tool Mentors.

8. Note that overloading is determined by the name only and does not involve parameters.

Process Content Libraries

3

RPW is designed to work with any file organization and uses an *explicit* file reference schema to associate model elements in process models with their content files. The explicit file reference schema means that file references are explicitly modeled as properties of process elements inside process models.

All process elements have one or more files associated to them and these are the files by which the process element is represented in a published process Web site. Files are included in a published Web site under the condition that one or more process elements included in the closure of the Web site has an association to them.

Figure 14 shows an example of the Process Content Files dialog used for associating files to process elements.

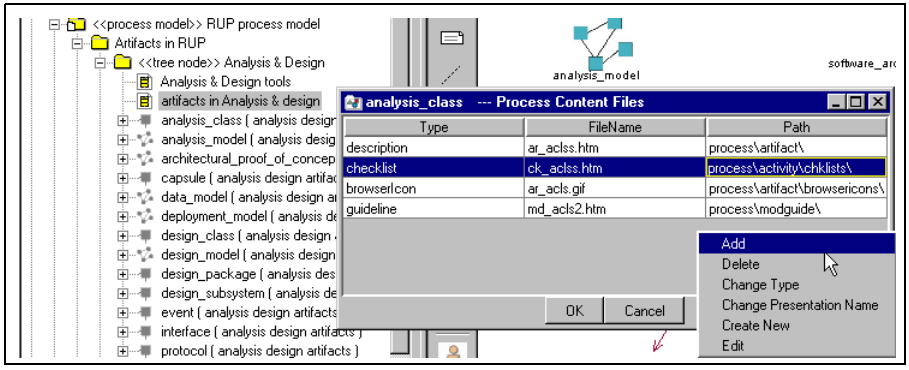

Figure 14 Process Content Files dialog box

File references are always relative to the Process Content Library root specified for the enclosing «process model» element. This means that you can associate the same model to alternate Process Content Libraries as long as they have the same internal structure and filenames.

Presentation Names

The concept of *presentation name* is central to the operation of RPW. Each file in a published Web site is represented by its presentation name, which is defined inside of the file itself. This means that any generated hyperlinks or existing hyperlinks will adopt the defined presentation name as the anchor text in hyperlinks designating a file.

Presentation names are modified from within RPW. The Presentation Name dialog, which is invoked from the Process Content Files dialog, is displayed in Figure 15.

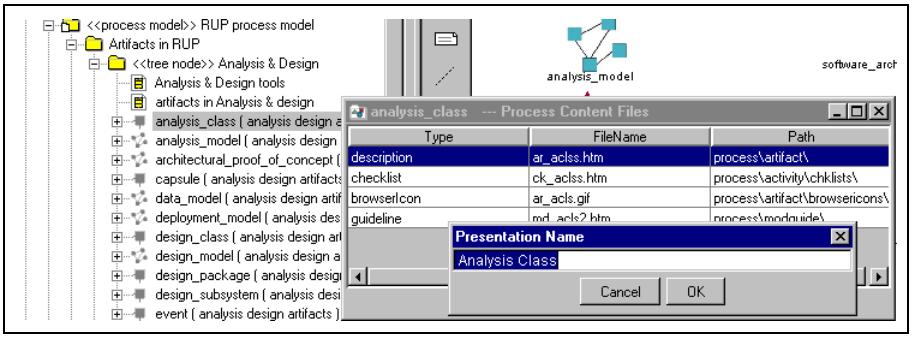

Figure 15 Presentation Name dialog box

Common Files

A common set of file types is defined for all types of process elements, and additional file types are defined for some process elements to give them a broader description:

- Description files
- Browser icons
- Folders
- Diagram files
- Anonymous files

Description files

Each process element must have one **description** file associated with it. A description file provides the fundamental process description of that particular process element.

Browser icons

Each process element may specify an individual icon for its representation in the tree browser of the published Web site. This feature is used in the RUP for Artifacts and Tools, but may be used for any element. Browser icon files are expected to be `.gif` files.

Browser icon files are optional and RPW uses default icons to represent elements that don't specify their own browser icon files.

Folders

All process elements can have any number of folders associated to them. Folders are file elements that allow for the association of a process element to an entire folder in the content library, regardless of the files it contains. When published, the entire folder is included in the resulting Web site.

Folders can contain additional HTML files that represent decomposition of large description files or other files such as `.doc` files that are referenced from description pages.

Diagram files

Some element types present certain overview information in a graphical format. Some of this information can be generated by RPW as it publishes Web sites. You can override this generation and use your own handcrafted graphics by associating an element to its override graphics. Each diagram is represented by two files: the image file and an areamap that defines the clickable regions in the file. See *Using custom-designed graphics in your Web site* on page 70.

Anonymous files

Process elements can be associated with any number of anonymous files. Just like folders, they are included in published Web sites, but have no further semantics.

Additional Files

Artifact, Discipline, and Activity elements types define additional sets of files, which help to describe their elements in a more detailed way. They're listed in the next three subsections.

In the following subsections, the information in the bracketed text—for example, (zero or one)—signifies the number of files for each type.

Artifact file types

Artifacts define the following additional file types:

- Checkpoint file (zero or one)
- Guidelines files (zero or more)
- Diagram `.gif` symbol file (zero or one)
- Report file (zero or one)
- HTML template file (zero or one)
- Word template file (zero or one)

Discipline file types

Disciplines define the following additional file types:

- Concept files (zero or more)
- In addition, a set of six files constitutes the sophisticated presentation structure for Disciplines in the RUP:
  - Introduction (zero or one)
  - Workflow Detail overview (zero or one)
  - Activity overview (zero or one)
  - Artifact overview (zero or one)
  - Guidelines overview (zero or one)
  - Concepts overview (zero or one)

Activity file types

Activities define the following additional file types:

- Concept files (zero or more)
- Guideline files (zero or more)
- Checkpoints (zero or more)

Templates

The `rpw\template` folder in Process Content Libraries contains HTML template files for those files you can create using RPW.

For details on modifying these templates, please refer to *Customizing templates* on page 39.

Web Site Form

«Process» classes specify the contents of Web Site Forms, which are the collection of HTML files and directories that constitutes the framework into which process Web sites are published.

The Web Site Form is comprised of the following file types:

- Folders (one or more)
- Anonymous files (zero or more)
- A *pointer* to a **tree browser folder** (one)

  The tree browser folder is assumed to exist in one of the folders defined in the Web Site Form; that is, it does not *add* a folder, but only *points* to an existing one.

Working with Process Models and Publishing Web Sites

4

The Structure and Content of Process Models

The only mandatory element in a process model is the package that represents the process model itself. This package is stereotyped to «process model» and serves as a container of process elements. Within this package, a process model may have any organization and structure.

Process models, which are essentially free-form design models, are developed using the modeling capabilities of Rational Rose with some additional constraints and capabilities added by RPW. For the most part, RPW operates within any process model structure and it "successfully ignores" any elements in the model space that do not concern it, operating only on the ones it recognizes. This means that a process model can also contain additional model elements beyond those recognized by RPW. For example, you may want to define Artifact dependencies so you can provide a graphical view over all of your Artifacts. Such modeling information is maintained by Rose, but ignored by RPW in its support for modeling and generating process Web sites.

In cases where you want to model using interfaces, RPW dictates the existence of the four model elements that represent the four supported interface types, described earlier in *Chapter 2, Modeling Elements and Principles*, under the heading titled *Using interfaces in process modeling*.

A general UML modeling element is the **package,** which is used to manage the complexity of design models. RPW bases some of its operation upon UML's definition of **visibility** of packages to support modeling and determine the validity of process modeling constructs.

Maintaining parallel process models

A Process Content Library may exist in multiple versions; for example, one for each translation into a different language. An important property across such multiple process models is that they use the same internal file organization and that the files carry the same names. If this structure is maintained, RPW can associate the same process model to any of its Process Content Libraries, and it can publish a Web site using HTML files from the currently associated Process Content Library.

Branching and merging process models

Branching and merging are common terms for the activities involved in supporting parallel development streams. Although this is most common in software development, it's also applicable in process development.

The following scenario demonstrates situations where branching and merging occurs:

1. RUP is delivered as a process model with an accompanying content library. (New versions come out approximately every six months with an updated process model and content library.)
2. Using RPW, customers customize the RUP to suit their specific needs. This entails both customizing the process model and the content library. For this purpose, they use one RUP version as their baseline process and create customizations relative to this baseline process.
3. A new version of the RUP becomes available, and customers who had already customized RUP now want to update their customizations to use the updated RUP as their baseline.

The activity of creating a customized process model is called branching and the activity of updating to a new baseline process is called merging.

A simple rule exists to facilitate branching and merging in a controlled fashion — **any customizations should occur in a space that is separate from the RUP**. This means that a separate process model, in Rose's workspace, should be established in which all new and derived process elements are defined, and that a separate content library should be established in which all customized text material is stored.

Technically, Rose's concept of separately controlled units enables the separation of a customized process model from the baseline RUP process model. By accommodating the RUP process model in its own package and making this package separately controllable, it's possible to include a newer version of the RUP process model without disturbing the customized process elements.

Central activities in merging process models are:

- to verify the synchronization of the customized process model with the updated RUP process model to ensure that all expected elements still exist in the RUP process model
- to determine what changes in the RUP process model affect your customized process model
- to re-establish component realizations into the RUP process model

Within this scheme of separately controlled process models (`.cat` files), a base process model and its derived process models can be assessed and modeled independently, yet with maintained integrity between the models, inside the same Rose modeling space. When a subsequent release of the RUP process model is received, the new model is loaded into the Rose modeling space and the derivation path is assessed using RPW's **check syntax** feature. This feature helps detect missing elements in the new base model.

Figure 16 represents a process model derived from the RUP, showing that you derive both the process model and the component model.

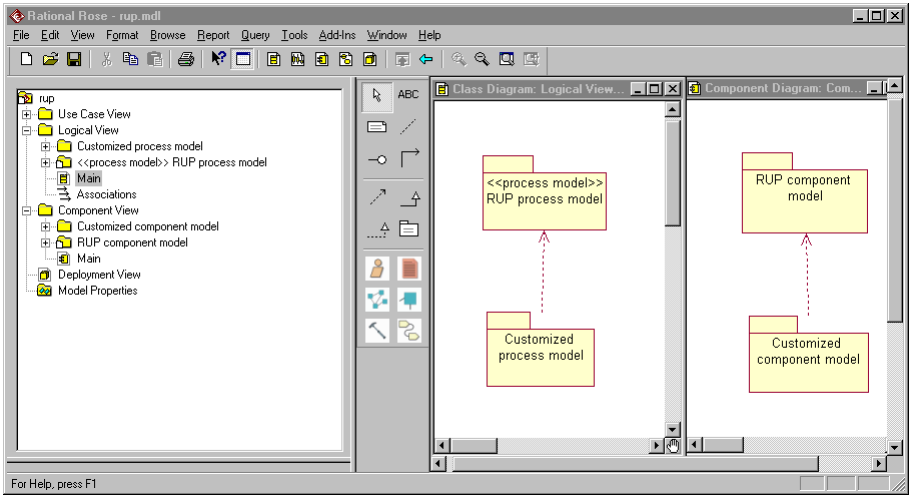

Figure 16 Process model derived from the RUP

Publishing Web Sites

The ultimate goal of process development with RPW is to publish a process, resulting in a Web site that contains your customized process. Publishing a process is simply the act of generating the set of HTML files and graphics that constitute your process using RPW. You can generate a published Web site into any selected location in the file system.

Hyperlinks and presentation names

One important property of RPW is that *files in a published Web site preserve their internal structure and organization relative to their enclosing library root*. This means that any hyperlinks in a content library, that were defined relative to its root, are preserved and still function in a published Web site.

The concept of *presentation name* is central to the operation of RPW. They are the names by which their elements are represented in published Web sites. Presentation names are defined for each of the files contained in the content library.

What determines the organization of the files in a published Web site?

Process publishing preserves the organization of files relative to their respective content library root.

A published Web site may draw its contained files from several parallel content libraries in cases where multiple process models contribute to the published process. In such cases, their file structures are superimposed on one another in the resulting Web site.

What determines the content and organization of the tree browser?

The published tree browser contains two parts: the static part, which does not vary depending on the process that's published, and the variable part, which varies with the particular published process.

The generic structure of the tree browser defines four compartments that contain the four types of process elements, modeled using classifiers: Roles, Artifacts, Disciplines, and Tools. Each type of element is presented in its own subtree in the tree browser.

Five menu files created by the publishing process contain the complete definitions of the tree browser's organization, as shown in Figure 17.

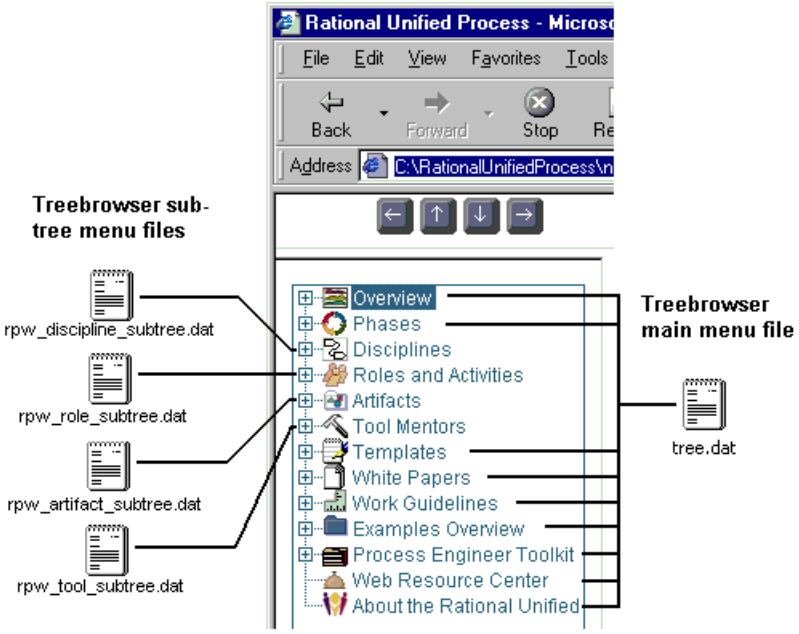

Figure 17 Tree browser menu files

Inside each subtree, the process elements are organized with the preserved structure of «tree node» stereotyped packages in the process model, relative to their respective «process model» package.

A published Web site may draw its process elements from several process models. Process publishing superimposes the «tree node» structures from the constituent process models into the single tree browser. In the case of two «tree node» packages from different process models having the same names, they will be considered the same and only one tree node will be created into which elements from the two packages will be stored.

Once generated, you can reorganize the generated tree files from their default alphabetical order, which was implemented in the first generation, in any order you like. This order will be preserved, with the same location, over subsequent generations.

Figure 18 displays the process model from the RUP, and shows how its process elements are structured using a mix of «tree node» packages and regular packages. In the generated RUP Web site, only the «tree node» stereotyped packages are included as intermediate nodes in the tree browser.

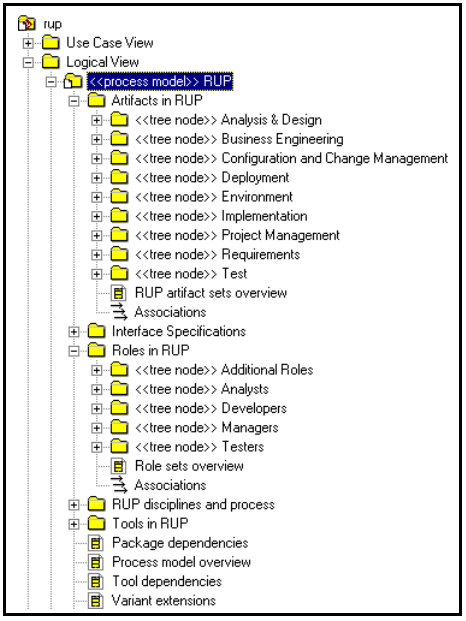

Figure 18 RUP process model

Exporting Component Models

Component models can be exported to the file system, into what is referred to as **Configuration Unit** files. These files can then be used in the **RUP Builder** application. RUP Builder represents a lightweight mechanism used to customize process based on these configuration units, instead of through process modeling in the RPW workspace. Using RUP Builder you define process configurations, which can then be published to the process Web sites.

RUP Builder operates on the same concepts as RPW, and publishing a configuration from RUP Builder yields the same result as when it's published using RPW. The only difference is in the terminology: RUP Builder talks about its process material in terms of *base process* and *process plug-ins*, whereas RPW uses the more generic terms *component model and configuration units* for the same things.

From the component view of Rose in the RPW workspace, export your component models to individual configuration unit files in the file system. Those files are transportable and can be inserted into the RUP Builder workspace separately.

The same associations as defined in the component view are preserved in the configuration unit files, and are used to determine the correctness and combinational properties of configuration units in the context of one another.

Working with Rational Process Workbench

5

A successful installation of RPW on your computer is a prerequisite to proceeding with this chapter.

This chapter contains descriptions of tasks you will perform in the course of your work. Those tasks include:

- Setting Up and Configuring the RPW Tool
  - Enabling and disabling the RPW add-in to Rational Rose
  - Customizing templates
- Setting Up and Managing the RPW Workspace
  - Managing the process model and Process Content Library from the RUP
  - Creating your customization workspace
  - Sharing process material among developers
- Developing Process Models
  - Defining new process elements
  - Deriving new process elements from existing elements
  - Creating variation points in your process model
  - Replacing Activities
  - Replacing Workflow Details
- Managing Process Content
  - Creating and editing content pages
  - Synchronizing a Process Content Library with its process model
  - Translating a Process Content Library into a different language
  - Managing Web Site Forms

- Defining a Custom Process
  - Creating your own process model
  - Authoring your process text
  - Organizing your Process Content Library
  - Creating your own process components
  - Defining your own process closure
  - Defining how the published tree browser will be organized
  - Upgrading to a new version of RUP
- Publishing a Process
  - Assessing the closure of your process
  - Publishing a Web site
  - Using custom-designed graphics in your Web site

Setting Up and Configuring the RPW Tool

RPW is an add-in to Rose, and requires that Rose is installed and operational on your system before RPW can be installed. All RPW functions become available when the add-in is enabled in the Rose add-in menu.

RPW helps you develop process models that are maintained as separate files in the file system, process component models that are maintained as separate files in the file system, and content libraries that are regular file structures in the file system.

For details on installing RPW, refer to *Chapter 2* in the *Getting Started with Rational Process Workbench* manual.

Enabling and disabling the RPW add-in to Rational Rose

RPW operates as an add-in component to Rose and installs itself as that during installation. As such, RPW can be enabled or disabled using Rose's Add-in Manager.

- Select Add-ins > Add-in Manager …
- Locate and select to enable or deselect to disable RPW in the presented list.
- Apply the change.
- Close the Add-in Manager window.

Customizing templates

Each Process Content Library contains a template section that provides the HTML templates for any new content files you create. These template files are themselves standard HTML files, which can be modified using your favorite HTML editor to include or exclude sections, your logo, and so forth.

Once a template is modified, files subsequently created and based on that template will include those modifications.

1 Edit file templates

Template files are located in the `rpw\templates` folder in the Process Content Library. A file exists for each type of file that can be created by RPW.

Each time you request the creation of a new file in the Process Content Files dialog, the corresponding template is copied from this directory into your specified location to provide the starting point for your file. RPW's *General HTML commands*[1] will be processed by RPW during the creation of these new files.

Note: A word of warning—the template files contain certain RPW commands to support the publishing operation that may not be modified. These commands determine where, and what, information is generated during publishing. Commands categorized as *General HTML commands* are processed by RPW during the creation of new files to resolve references. Therefore, creating new files outside of RPW, based on these templates, may cause unexpected results.

1. See Appendix A for details of these commands.

Setting Up and Managing the RPW Workspace

RPW's workspace is comprised of one or more process models and one or more component models inside the Rose modeling space. Each model is contained in a separate package, in either Rose's logical view or component view.

In your modeling space, you'll have both:

- an existing process model and component model from the RUP
- your own process model and component model that accommodate your customization

Each process model is associated to a Process Content Library, which contains the process text for those elements in the model.

Managing the process model and Process Content Library from the RUP

The process material from the RUP is considered to be read-only[2] and, therefore, must be kept separate from your customizations. This separation facilitates future updates of the RUP without disturbing your customizations.

The RUP material consists of the process model, component model, and Process Content Library. The first two are separately controlled Rose units, maintained in their own files, whereas the Process Content Library is a regular directory structure in the file system.

The files represent the RUP packages that exist in Rose's logical view and component view respectively, and they are loaded into your workspace as separately-controlled units.

1 Insert the RUP process and component models into your workspace

The RUP is delivered in the form of two separate model files—one containing the process model and one containing the component model.

- When you receive updates of the RUP, bring the process model into your Rose model by using the Rose load unit function and specify its location.
- If your Rose model already has a previous RUP process model loaded, you must unload it first.

2. This is not physically read-only because certain modeling can only be achieved if the model is write-accessible.

2 Associate the RUP process model with the RUP Process Content Library

- Once you've successfully loaded the new RUP process model into your Rose model, associate it with its Process Content Library by using the Associate Content Library command on its «process model» package, and then select the root of the new Process Content Library in the presented File Selection dialog.
- Invoke the check files command on the process model to verify that all of its files are present in the Process Content Library.

3 Insert the RUP component model into your workspace

- Insert the RUP component model into your workspace in a similar way to which you inserted the RUP process model into your Rose model.
- If your Rose model already contains a RUP component model package, you unload the current file, if loaded, and then load from the new file.
- If your Rose model does not contain a RUP component model package already, create a package in the component view and name it *exactly* "RUP Component Model". Make the package controlled and unload it. Load the newly received component model in its place.

Creating your customization workspace

All of your process customizations must be maintained separately from the RUP material. The basic structure of your customization material will be the same as that of the RUP material. You define your customized process elements in your own process model, specify your customized processes in your own component model, and store all process text you create in your own Process Content Library.

1 Create your own process model

- Create your process model at the top-level of Rose's logical view, parallel with the RUP process model. Create a package and stereotype it to «process model».
- If you already have a Process Content Library, create an association from your process model to it using the Associate Content Library function.

2 Create your own Process Content Library

- Invoke the Associate Content Library function on the «process model» package and select the `root` folder of your Process Content Library, if you already have one.

- If you don't have a Process Content Library already, create a new folder in the file system and populate it by copying files from the RUP content library.

  Note: All files except the files in the process folder in RUP constitute the invariable Web Site Form, and will serve as an excellent starting point for your own Web Site Form, if you want to create one.

  Also note that you can reuse Web Site Form files and folders in the RUP content library without copying them—you just reference the files directly in the RUP content library.

3 Create your own component model

- Create your own component model at the top-level of Rose's component view, parallel with the RUP component model. A component model is a package used to control the model separately. Stereotype your new package to «component model».
- Create a dependency from your new component model package and RUP's component model package. This dependency enables reuse of RUP components in your closures.

Sharing process material among developers

Depending on the size of your process development and customization effort, you might want to organize your process material to allow for parallel, yet controlled, process development.

We advise that you avoid sharing your process model among multiple developers, although it's technically possible. We recommend you organize your team in such a way that one person is responsible for the whole process model, or that the team members share the responsibility of the model, but only one developer works on it at a time. This also applies to the component model.

The core activity in your process development is authoring the process contents. As such, it's important that you set up your Process Content Library so multiple developers can simultaneously work in it.

1 Share a process model

A process model can be shared, using the same techniques you use when sharing Rose design models, in general. The basis for this is the concept of separately-controlled packages. Refer to the *Work in Teams* topic in Rose's online Help for instructions on separately controlled packages.

We recommend that you assign only one person the responsibility for updating your process model. This avoids any extra complexity that may be introduced when supporting multiple users.

2 Share a component model

Component models can be shared among process modelers, using the same techniques used when sharing process models. Again, refer to the *Work in Teams* topic in Rose's online Help for a description of how to do it.

For the same reasons as for process models, we advise against sharing of component models.

3 Share a Process Content Library

A Process Content Library is a regular hierarchy of files in the file system, possibly controlled using a configuration management strategy.

Authoring of process contents occurs outside of RPW and the actual strategies you use for sharing the content library are determined by factors in your environment. For example, you could divide the authoring work along the package boundaries in your process model, but other strategies could be adopted.

Developing Process Models

This section describes how to define new process elements, in addition to those elements already defined in the process model found in the RUP, and how to derive new process elements from existing RUP elements. Basic modeling principles are described in *Chapter 2, Modeling Elements and Principles* in this manual.

RPW uses the Rose workspace to develop process models and includes a process model from the RUP.

A common way to customize the RUP is to remove one or more of its Disciplines and to add custom-defined Disciplines that will exist in parallel to those Disciplines defined in the RUP.

Other types of customization occur at more detailed levels, where you modify individual Roles, Activities, and Artifacts to better suit your needs.

Defining new process elements

Any customization of the RUP you make must be maintained in your own process model. When you work inside of your process model, you use the notation supported by RPW to define new process elements that support your specific process needs.

Defining new process elements is a general-purpose, object-oriented, modeling exercise that uses the specific process modeling concepts RPW has introduced in the Rose workspace.

1 Create your own process model

Process models, described in detail in *Chapter 4*, are represented by top-level, stereotyped packages in the logical view. Each process model defines its process elements, and their classifications and relationships.

- Create a package in Rose's Logical View and stereotype it to «process model».
- Right-click on the created Process Model and select Associate Content Library.
- In the displayed File Selection dialog, select the folder that represents the root of your Process Content Library. If you don't have a Process Content Library at this point, that's fine. Your subsequent modeling activities do not require this association to exist, and you can always establish this association later on.
- Right-click again and select Set Template Directory.
- In the dialog that displays, select the directory inside your Process Content Library where RPW templates are stored.

2 Create new Roles and Activities

Inside a process model, create a class to represent your new Role. This can be achieved in a number of ways using Rose's functions. For example, in a class diagram, create a new Role class using the Role symbol from the toolbar. (You may need to customize your toolbar to add the process elements to it.) Give the created Role class an appropriate name.

Note: This name is a *modeling name*, which is different than its presentation name.

You can also create a role by selecting create new class from the context menu of the enclosing package and stereotyping it to «role» in its specification dialog.

- Define the new Role's responsibilities in terms of owning or modifying Artifacts.

  In a class diagram where the Role is present, include the Artifacts that the Role is responsible for and create Unidirectional Associations going from the Role class to the Artifact classes. Stereotype the association to «responsible».

  For other Artifacts where the Role should be granted update privileges, create unidirectional associations and stereotype them to «modifies».

- To create an Activity, create an operation on its performing Role, stereotype the new operation to «activity».

- To define the Activity's Artifacts, right-click and select Overview. In the Artifacts tab, populate the panes to the right by double-clicking the Artifacts from the list presented to the left. The three lists present those Artifacts eligible for inclusion in the respective categories of In, Out, InOut signature.
- To assign Tool Mentors to the created Activity, select the Tool Mentor tab, right-click on it, and then select Add.

  A dialog is presented where you first select the Tool, and then the Tool Mentor.

3 Create new Artifacts

Inside a process model, create a class to represent your new Artifact:

- In a class diagram, create a new class and stereotype it to either «model», «model element», or «document».
- Give the created Artifact class an appropriate name.

  Note: This name is a *modeling name*, which is different than its presentation name.

4 Create new Tools and Tool Mentors

Inside a process model, create a class to represent your new Tool:

- In a class diagram, create a new class and stereotype it to «tool». Give the created Tool class an appropriate name.

  Note: This name is a *modeling name*, which is different than its presentation name.
- To create a Tool Mentor, create an operation on its Tool, and then stereotype the new operation to «tool mentor».

5 Create new Disciplines and Workflow Details

Inside a process model, create a class to represent your new Discipline:

- In a class diagram create a new class and stereotype it to «discipline». Give the created Discipline class an appropriate name.

  Note: This name is a *modeling name*, which is different than its presentation name.
- To create a Workflow Detail, create an operation on its Discipline, and then stereotype the new operation to «workflow detail».

The activity models for Disciplines and Workflow Details are both expressed using activity diagrams. Please see the information in *Chapter 2, Dynamic Specifications* for more details.

6 Structure your process model

Process models may be structured internally to manage their complexity by using packages as grouping concepts. Packages may be nested to any depth and a process model may contain any number of packages. A special type of «tree node» package controls the way a published process' tree browser is composed and presented.

- Create a Tree Node package inside your process model by creating a package and stereotyping it to «tree node».
- Populate the «tree node» package by moving its process elements to physically reside in the package.

Deriving new process elements from existing elements

If you have the process model from the RUP in your Rose workspace, in parallel to your own process model, you can reuse existing process elements in the context of your customization. For example, you can create a derived Role from one of the RUP Roles, and then extend it with additional Activities. The resulting derived Role assumes the Activities of both the RUP Role and the new ones. You would do this to reuse existing definitions without having to recreate them, as well as being able to use them in their original context. This technique applies when you want to reuse elements within the same process model also.

In steps 1, 3, 4, and 6, the topic of inheritance is discussed. For further details, refer to *Chapter 2, Modeling Techniques*, under the heading *Using inheritance*.

Steps 2, 5 and 7 refer to the topic of operator overloading and you can find more information on this in *Chapter 2, Modeling Techniques*, under the heading *Using operator overloading*.

Note: It needs to be noted that associated files are ***not*** part of what is inherited. When you associate files in the Process Content Files dialog for an inheriting element, you can include files from both the RUP content library and your own content library, as well as create new files in your own content library.

1 Derive a Role

New Roles can be created using the concept of inheritance.

- Create a new Role class to represent the new Role.
- In a class diagram, create an inheritance association from the new Role to the existing Role from which it will inherit properties.

2 Redefine an Activity

Existing Activities can be redefined using what's commonly referred to as "operator overloading". For details on this topic, please refer to *Chapter 2, Using operator overloading*. This implies that a derived Role exists and can be assigned the redefined Activity.

- On an inherited Role, create a new Activity operation and give it the same name as the one it will replace.
- Specify the Artifact signature and Tool Mentors for the new Activity in the Overview dialog.

3 Derive an Artifact

New Artifacts can be created using the concept of inheritance.

- Create a new Artifact class to represent the new Artifact.
- In a class diagram, create an inheritance association from the new Artifact to the existing Artifact from which it will inherit properties.

4 Derive a Discipline

New Disciplines can also be created using the concept of inheritance.

- Create a new Discipline class to represent the new Discipline.
- In a class diagram, create an inheritance association from the new Discipline to the existing Discipline from which it will inherit properties.

5 Derive a Workflow Detail

Workflow Details can be redefined using what's commonly referred to as "operator overloading". This implies that a derived Discipline exists, which can be assigned the redefined Workflow Detail.

- On an inherited Discipline, create a new Workflow Detail and give it the same name as the one it will replace.
- Specify the Activity overview for the new Workflow Detail.

6 Derive a Tool

New Tools can be created using the concept of inheritance.

- Create a new Tool class to represent the new Tool.
- In a class diagram, create an inheritance association from the new Tool to the existing Tool from which it will inherit properties.

7 Derive a Tool Mentor

Tool mentors can be redefined using what is commonly referred to as "operator overloading". This implies that a derived Tool exists, which can be assigned the redefined Tool Mentor.

- On an inherited Tool, create a new Tool Mentor and give it the same name as the one it will replace.

Creating variation points in your process model

RPW supports using interfaces to create variation points in your process model. A variation point is a point in your model that can assume any variant of a construct, provided it obeys the contract of the variation point and leaves the remaining model intact. As modeling elements, interfaces provide the specification perspective of such constructs, behind which its variation occurs. Variation points can be formed around the process elements that are modeled using the class construct Roles, Artifacts, Disciplines, and Tools.

For more information on interfaces, refer to *Chapter 2, Modeling Techniques*, under the heading *Using interfaces in process modeling*.

1 Create an interface to represent the variation point

Create a new interface where appropriate in the process model. Specify the type of the new interface by assigning it an inheritance association to one of the four predefined interfaces, named:

- RoleInterface
- ArtifactInterface
- DisciplineInterface
- ToolInterface

These are defined in the Interface Specifications package in the process model.

2 Specify the interface's properties

Interfaces assume the same specification properties as the concepts they represent, except for associated files.

After an interface has been given its type as instructed in step 1 above, you can model it as if it was a real class, and the appropriate dialogs become available for specification of the interface's properties.

3 Associate a class with its interface

In a class diagram, create a «realizes» association from the class to its interface. It's important that the realizing class includes the exact profile of the interface. One class can realize multiple interfaces.

Replacing Activities

A common customization is to redefine existing Activities to use and produce a different set of Artifacts.

The modeling technique used to achieve this is commonly referred to as operator overloading. For more details on this topic, refer to *Chapter 2, Modeling Techniques*, under the heading *Using operator overloading*. In process modeling, this means that a new Activity, with the same name but a different Artifact list is modeled. The new Activity can then substitute for the existing Activity in a process closure.

Following our recommendation to separate your process model from the one in the RUP, it becomes necessary to create a new Role on which the replacing Activity can be defined. This is where you use the inheritance construct to associate the new Role with the original one; an association through which the new Role assumes the original Role's other Activities and responsibilities.

1 Create a replacing Activity

To create the new Activity, you first have to create a Role that performs the Activity. Replace an Activity by giving it the same name as the original.

- Create a «role» class that will perform the replacing Activity.
- Create an inheritance association from the new class to the «role» that defines the original operation.
- Create a new «activity» operation on the new class.
- Name it the same as the original «activity».
- Define the Artifact list for the new Activity.

- Using their Process Content Files dialog, specify the description files for the new Role and the new Activity. You can add the same files as the original Role and Activity, or you can create new files to be used instead.

2 Include the derived Role in your process' closure

A process closure can only include one class from the same inheritance structure. This means that a derived Role can take the place of its superclass Role, but they cannot reside in the same closure at the same time. Use the component specification realization to substitute a derived Role in a closure.

For more details regarding process closure, please refer to *Chapter 2*, under the headings *Process Components*, *Process closure as a special type of component*.

Replacing Workflow Details

Another common customization is to replace existing Workflow Details with new ones that have the same purpose, but different definitions. You use operator overloading to replace Workflow Details in the existing RUP.

A replacing Workflow Detail can have a different Activity overview, which employs a different Activity set in a different way. To accommodate your replacing Workflow Detail you need to create a new Discipline on which the new Workflow Detail is created.

1 Create a replacing Workflow Detail

- Create a «discipline» class that will perform the new Workflow Details.
- Create an inheritance association from the new class to the «discipline» that performs the original Workflow Detail.
- Create a new «workflow detail» operation on the new Discipline.
- Name the new Workflow Detail operation exactly the same as the original «workflow detail».
- Define the activity diagram for the new Workflow Detail. This activity diagram can be different from the original one, and can exclude existing Activities and include others.
- Using their Process Content Files dialogs, specify the description files for the new Discipline and the new Workflow Detail. You can add the same files as the original Discipline and Workflow Detail, or you can create new files to be used instead.

2 Include the derived Discipline in your process' closure

A process closure can only include one class from the same inheritance structure. RPW supports single inheritance, and this means that a derived Discipline can take the place of its superclass Discipline, but they cannot reside in the same closure at the same time. Use the component specification functions to substitute a derived Discipline in a closure.

For more details regarding process closure, please refer to *Chapter 2*, under the headings *Process Components, Process closure as a special type of component*.

Managing Process Content

This section describes how to manage your content when you develop your own process material.

RPW allows you to create new HTML files based on provided templates. This allows you to author process content—either inside or outside of RPW's control—with your preferred HTML editor. Most process HTML files have presentation names. These are the names by which files are represented in published Web sites. RPW supports changing the presentation names in a separate user interface.

During publishing RPW processes hyperlinks encountered in the text material and updates them with their appropriate presentation names. This means that you can use spontaneous hyperlinks in your process text. If a particular process closure, however, does not include the file designated by a hyperlink, RPW disables the hyperlink as it processes the file during publishing.

Web Site Forms provide the "static" portions of generated Web sites, which are those portions that do not vary with the closure of the generated process. RPW supports composing Web Site Forms using the same paradigm it does for content files in general, and allows you to modify the set of files and folders of which it's comprised.

Creating and editing content pages

Content pages provide the text descriptions of process elements. As you develop your customized process material, you need to create new pages to describe the new process elements you create, or you need to add or replace files for existing elements.

The content pages reside in a Process Content Library. Content pages are ordinary HTML pages with some additional RPW constructs embedded in them. These constructs are clearly marked in the files and must not be modified or removed.

RPW does not include an HTML editor. It does, however, support template-based creation of new pages in the Process Content Library, as well as navigation from the Rose modeling space to an HTML editor. This allows you to create new content files from your modeling context, and to navigate back and forth between your modeling and authoring activities.

1 Create a new file in your Process Content Library

Using this capability, you request the creation of a new page and are prompted for its type and destination location. RPW provides a prepopulated file containing all necessary constructs to make the file operational in a published Web site.

Conversely, if you have existing files that you want to incorporate into your Process Content Library, such files need to be updated to include the necessary RPW constructs. We recommend that all new material finds its way into the Process Content Library using the provided RPW functions. (Any reuse of such text material occurs using the cut-and-paste functions provided with your HTML editor.)

2 Edit a content page

Editing content files is an activity that occurs outside of RPW's control, using your preferred HTML editor.

RPW allows you to associate the HTML editor you choose and supports invocation of that editor on individual files in the process model.

3 Change the presentation name of a page

The presentation name of a content file is a notation of the name by which the file is represented in a published process Web site. This feature allows you to change a presentation name in one place, with the effect that the new name is used the next time you publish its process. RPW provides a function for changing a presentation name from the context of process elements in the model.

In the Process Content Files dialog, select the file, and then select change presentation name. This displays a text entry dialog where you type the new name.

Note: The presentation name is a general property defined for a number of file types, not only description files. Also note that the presentation name of a file is stored in the file itself and changing its presentation name causes an update of the file itself, which may violate your configuration management strategies.

4 Insert hyperlinks in your text material

You can use regular hyperlinks anywhere in your process text material to provide spontaneous navigation between process elements in your Web site.

At the time of publishing, RPW processes such hyperlinks and resolves them to the correct file in the Web site, provided the actual designated element exists inside of the closure of the published process. If not, RPW will visually indicate such links as non-operational.

Note: RPW always automatically generates the regularly structured hyperlinks that appear in the overview pages of process elements.

Synchronizing a Process Content Library with its process model

A Process Content Library and its process model are considered to be synchronized if all file references existing in the process model are satisfied by the Process Content Library. RPW provides a function that determines whether a process model is synchronized with its associated Process Content Library, which operates at any level inside of your process model, from the entire process model down to individual process elements.

1 Check files in a process model

On the process model package, use check files to determine whether all files associated from the elements in a process model exist in their expected location in the file system. Check files is applicable at any package level in the process model and on individual process elements.

On a «process model» package, or any package or file therein, select Check Files from its context menu.

2 Associate a new file with a process element

- Use the Add option in the Process Content Files dialog to associate new files to a process element.
- In the presented File Selection dialog, select the file you want to add.
- In the subsequent dialog, specify the type of the file it is.

Note: You can Add files, from both your own content library and from the RUP content library, or from any other derived process models, to your process elements.

Translating a Process Content Library into a different language

A Process Content Library can be translated into different languages, retaining the associations to its process model. Conversely, this means that the same process model can serve as the model for multiple translations of its Process Content Library.

1 Associate a process model with a Process Content Library

- Invoke the Associate Content Library command on the «process model» package.
- In the presented File Selection dialog, select the root of the Process Content Library.

2 Verify the synchronization between the process model and the Process Content Library

On the process model package, use check files to determine whether all files associated from the elements in a process model exist in their expected location in the file system. Check files is applicable at any package level in the process model and on individual process elements.

Managing Web Site Forms

Web Site Form is a collective term for the invariable portions of a published process—those portions that are not published by RPW, but exist there to provide the context for your process. The Web Site Form is comprised of a collection of folders and individual files that reside in the Process Content Library.

Most notably, a `tree.dat` file exists that specifies the layout of the static portions of presented tree browsers.

RPW defines a «process» process element, which is the element type on which Web Site Forms are defined. The Web Site Form File Association dialog is used for this purpose and allows for the association of any number of anonymous folders and files. The only mandatory element in a Web Site Form is the "tree browser folder", which is the folder that contains the definition of the initial pre-generation tree browser and other associated files.

In a generated process Web site, all files referenced from the non-generated portions of the tree browser need to be included in the published material. This applies to all sections except Disciplines, Artifacts, Roles, Activities, and Tools. Every page referenced, directly or indirectly, by the tree browser must be included in the Web Site Form definition.

To facilitate the process development stage, you might want to define an alternative, smaller Web Site Form—one that excludes the many files included in your production-quality Web Site Form.

Note: The `tree.dat` file in your tree browser folder needs to be updated in tandem with any changes to the Web Site Form to ensure that the files that the `tree.dat` file references are actually present in the published Web site. They become present in the published Web site if they were directly specified as a file or indirectly specified through their enclosing folder.

1 Modify the Web Site Form

Essentially a Web Site Form is a set of anonymous files and folders, which together constitute the invariable portions of a published Web site.

- Invoke the Web Site Form command on the «process» class. This displays a list of folders and files that are currently included in the Web Site Form.
- In the presented dialog, you can add and remove files and folders.

2 Modify the Web Site Form portions of the tree browser

The tree browser consists of four parts that RPW automatically generates and one part that is static.

The tree browser is specified in the `tree.dat` file in the folder designated by the tree browser folder in the Web Site Form. This file is a simple text file and you can modify it, using an editor of your choice, to include additional entries, and to modify or remove existing entries.

3 Define a new «process» representation

«Process» classes define Web Site Forms as their properties. Each process closure (which is represented by a «process» component) includes the realization of exactly one «process» class in its components. This provides the Web Site Form for the published Web site.

Defining a Custom Process

This section describes how to define your custom process using the process model supplied with the RUP and the Process Content Library as the starting point for your customized RUP.

You can customize the process model, component model, and Process Content Library included with the RUP to your specific needs using RPW. This work entails defining new process elements, either right from the beginning or derived from existing RUP elements, authoring their associated process text, and specifying the closure of your process.

Creating your own process model

Essentially, a process model is a stereotyped package that resides at the top-level in the Rose workspace's logical view. Process models serve as containers of process definitions and you can have multiple process model packages simultaneously loaded in the same Rose workspace. This allows you to view and share information between the various process models.

Your process customizations are developed in your own process models, where they are separate from the RUP definitions, which are kept in another process model. This separation facilitates future updates of the RUP.

You work inside of the process model, where you use the notation of RPW to define new process elements to support your specific process needs. See Chapter 2, *Modeling elements and principles* in this manual.

In addition to creating new elements, using the object-oriented extension mechanism of inheritance you can derive new process elements from existing ones in the RUP, and you can create variations points using interfaces.

1 Derive new Roles from existing ones

You derive new Roles from existing ones when you want to extend an existing Role definition, but the existing Role is defined in a different process model and, therefore, cannot be directly manipulated. You use *inheritance* to achieve this effect. Inheritance can also be used to refine existing Role definitions in your process model and thereby create similar, but not identical, definitions of Roles.

This is useful when your process model supports multiple process variants, across which related but not identical Role definitions are used. For example, you could have a designer Role and a specialized realtime designer Role. Through an inheritance relationship, an inheriting Role assumes the Activities and «modifies» and «responsible» associations from the inherited Role. An inheriting Role can also define additional Activities and associations.

This is how you do it:

- Create a new Role class in your process model to represent the derived Role.

- Establish a package dependency between this Role's enclosing package (either «tree node» package or regular package) and the enclosing package of the Role from which it will be inherited. This allows your new Role to "see" its parent Role. You create package dependencies in class diagrams.
- Put both your new Role and the inherited Role in the same class diagram, and create a generalization association from your Role to the inherited Role.

The following well-formedness rule applies to modeling using inheritance in Role modeling:

> In a given process closure an inheriting Role can assume the position of its inherited Role, since it possesses all its properties. However, including multiple Roles from the same inheritance structure creates ambiguity in the closure and, therefore, only one Role from the same inheritance structure can appear in a closure.

- Use the Process Content Files dialog on the derived Role to specify its files. You can use the same file as the inherited Role or you can create a new file. A given Role can use files from both the Process Content Library associated to its enclosing process model and the files in the content libraries of derived process models.

2 Create a replacement Activity

Create replacement Activities when you want to redefine an existing Activity in terms of the Artifacts it uses, creates, and modifies. To create the replacement Activity:

- You first inherit the performing Role in your process model to create a new Role that can perform the replacing Activity, as described in the previous step.
- You replace Activities using the technique of *operator overloading*. On the derived Role you create a new Activity and give it *exactly the same name* as the Activity that it will replace.
- Now you can specify the Artifacts that the replacing Activity uses, modifies, and creates by using the Artifacts tab in its Overview dialog. Make sure that your inherited Role defines «modifies» and «responsible» associations to accommodate the demands of your new Activity.
- On the Tool Mentor tab, specify the Tool Mentors that support the Activity.
- Using the Process Content Files dialog, you can also redefine the set of files that the Activity uses.

3 Create a replacement Discipline

Create a replacement Discipline when you want to alternate its specification in terms of performing Activities. This type of replacement can occur both at the level of adding and replacing entire Workflow Details, as well as modifying the specifications of individual Workflow Details.

- Use the same techniques as described in step 1 for Role and step 2 for Activity to create the replacement Discipline, and its added and replacing Workflow Details in your process model. Workflow Details are modeled as operations, just like Activities.
- Now you can redefine the properties of both your Discipline and its defined Workflow Details in terms of their Activity overviews, participating Roles, and the content files associated to the Discipline and Workflow Details.

4 Use several process models to achieve multi-tier customizations

The derivation schema from process model to process model may be repeated to achieve derivation in several steps. This is useful, for example, when you're deriving a process model from the RUP to serve your organization level, and then refining this process model individually for individual project types inside of your organization. In this scenario, you'll have one organization process model and one project process model for each project type.

- Use the same techniques as described above to derive new and replacing process elements in your process models. A given process model can use elements from any process model in the derivation chain, as long as the proper process model dependencies exist.

Authoring your process text

Each process element in your process model is accompanied by one or more HTML files that constitute the element's process text.

Process text is maintained in the file system as regular HTML pages and you can use your preferred HTML editor to author the text. RPW enables you to create new process text files using templates, which populate your new files with their initial structure. This includes inserting the required RPW constructs in the HTML material, which are the foundation for generating the process, as well as creating the initial outline for your new file.

RPW is only involved in the initial creation of the process text, therefore, you may prefer to invoke your HTML editor from outside of RPW environment.

Use the Process Content Files dialog to create files for all of the different element types. The dialog is tailored to recognize the various file types defined for each process element type.

Refer to *Chapter 3, Process Content Libraries* for a complete description of the types of files that can be defined for the different types of process elements.

1 Associate the HTML editor to be used for HTML editing

RPW does not include an HTML editor itself, but allows you to associate to your favorite HTML editor, which it then invokes when you request the editing of HTML text from the Process Content Files dialog.

- Go to Rose's Tools menu and select Process Workbench > Options.
- The resulting dialog allows you to set your HTML editor. This setting remains in effect between sessions.

2 Create process text for Roles and Activities

Both Roles and Activities specify ***description*** files, which is the mandatory file type for all elements. In addition, Activities specify Concepts and Guidelines, which can be used to expand its description and provide more detailed how-to information.

3 Create process text for Artifacts

In addition to the mandatory ***description*** file, Artifacts specify a variety of file types that provide more specific descriptions, as well as a means to specify templates files that can accompany the Artifact in a generated Web site. Those file types include:

- Guideline
- Checkpoint
- Report
- Microsoft Word template
- HTML template

4 Create process text for Disciplines and Workflow Details

Disciplines specify a set of files, which collectively constitute the elaborate presentation of Disciplines as found in the final Web sites (see *Chapter 3*, under the headings *Additional files, Discipline file types*):

- Artifacts overview

- Concepts overview
- Guidelines overview
- Workflow Details overview
- Activity overview
- These are all mandatory files that need to exist in the process model. However, they do not contain any edited text so creating them from the Process Content Files dialog, as RPW creates them, is sufficient.
- Both Disciplines and Workflow Details are specified in terms of activity diagrams. Following the general diagram management direction given later in this chapter, under the heading titled *Using custom-designed graphics in your Web site*, you can override the diagrams generated by RPW by inserting your own into the model.

5 Create process text for Tools and Tool Mentors

Tools and Tool Mentors do not specify any file types in addition to the mandatory description file type.

You can add or create their description files from their Process Content Files dialog.

6 Insert hyperlinks

You can use regular hyperlinks anywhere in your process text material to provide spontaneous navigation between process elements in your Web site.

At the time of publishing, RPW processes such hyperlinks and resolves them to the correct actual file in the Web site, provided that the actual designated element exists inside of the closure of the published process. If not, RPW will visually indicate such links as non-operational.

Organizing your Process Content Library

Your Process Content Library stores the many files that provide the process text, graphics, icons, and so on, for your process Web site. To facilitate the incorporation of future updates of the RUP, you must maintain your own process text in its own Process Content Library, separate from the RUP material.

Although RPW does not impose any particular organization of your Process Content Library, it does require that some information exists there to operate correctly.

1 Associate your process model with its Process Content Library

- On the «process model» package, select Associate content library and specify the root of the Process Content Library in the presented File Selection dialog.
- The created association defines the path through which all file associations are resolved between a process model and its files. You can successfully establish such associations only after you have associated a Process Content Library to your process model.

2 Create the `rpw` directory

The `rpw` directory contains all of the things that RPW requires to work correctly. This directory contains one sub-directory, the `templates` directory, which contains the templates that RPW uses when creating new HTML files in the Process Content Library.

The `rpw` directory needs to reside at the top-level of content libraries.

Populate the `templates` directory with the templates from the RUP content library as a starting point for your template customizations.

3 Create a `tree browser folder` directory

The `tree browser folder` is a directory that is treated somewhat specially. RPW uses the contents of the tree browser folder as the basis for published tree browsers. A tree browser folder is a regular folder inside the content library, and it can have any name.

In the Web Site Form dialog, on «process» elements, you point out the tree browser folder that will be used for publishing that process.

See Chapter 3, *Web Site Form* for a description of this folder.

Creating your own process components

When you've defined your customized process elements and created their accompanying process content files, it's time to create the process components that specify how these elements will be grouped together for publishing. See the section titled *Publishing a Process* later in this chapter for more details.

The component view in Rose is used to define both process components and process closures. RPW uses regular components stereotyped to process components, and uses their realization properties to specify those process elements to include.

1 Create a new component

Create a new process component using Rose's support for component creation. Process components, in general, are not stereotyped.

One special type of component, «process» component, exists which is the representation of process closures. For more details on this, refer to the information under the heading *Defining your own process closure*.

2 Specify which process elements will be realized by the component

Process components *realize* sets of process elements. The process elements realized by a particular process component are all included in a process closure where the component is included. In this way, process components represent units of deployment into process Web sites.

The set of process elements realized by a process component can be assessed and determined whether or not it's correct.

The following rules apply:

- Only one class (Role, Discipline, Artifact, Tool) can appear from an inheritance structure. Realizing two classes from the same structure creates an ambiguous specification that RPW cannot resolve.
- The component must realize exactly one class for each interface it realizes—Role, Discipline, Artifact, Tool.
- The same element can only be realized by one component in the closure.

As you specify the realization of a process component, you can use the Assess Component command on the «process» component to determine its correctness.

Defining a process closure

A process configuration is a collection of component models that collectively constitute a complete process.

The closure of a process is represented by a «process» stereotyped component in Rose's component view and its configuration is defined in terms of the component models that it includes.

Component models are either "base models", in which case they are self-contained and can be deployed as-is, or "extension models" that extend an existing base model with additional elements. Extension models may extend other extension models.

The closure of a process is specified by created dependencies from the process closure component to its included component models.[3]

The process elements associated to those process components included in the process define the closure of a particular process configuration. It is a transitive closure, meaning that all process elements associated to already-included process elements are also included. For a process configuration to be considered as having closure, all of its associated and, therefore, expected, elements must exist inside of its component models. This is a prerequisite for publishing a process configuration. See the section titled *Publishing a Process* for more details.

Figure 19 shows the RUP process closure.

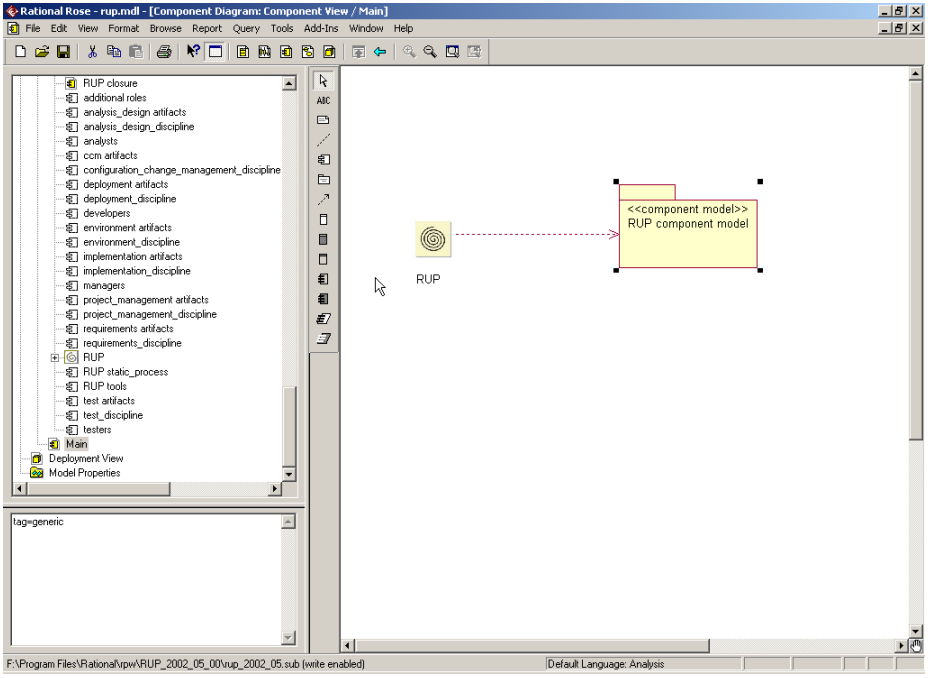

Figure 19 RUP Process Closure

1 Create a component to represent a new process

In Rose's component view, create a new process component using Rose's support for component creation. In general, process components are not stereotyped—the special components that represent process closures are.

- Create a new component in your component model.

---

3. If you are familiar with the RUP Builder application and the way it defines *process configurations* in terms of their base process and process plug-ins, this is how they were modeled in RPW before they were exported to their respective plug-in files.

- Stereotype it to «process».

Note: «process» components can reside anywhere inside of Rose's component view. We recommend you maintain them inside of their respective component models.

2 Specify component models to be included in the process

The configuration of a process is specified in terms of its included component models.

Specify those component models that should be included by pointing to them using model dependencies from the «process» component to all component model packages that it includes.

For each included extension component, the component model that provides its base must also be included in the closure.

- Create a component diagram in the component view if you don't already have one.
- Drag your «process» component onto the diagram.
- Drag each «component model» to be included onto the diagram.
- Establish dependencies from the «process» component to the component model packages.

3 Assess the completeness and well-formedness of the process' closure

On the «process» component, select the **Assess closure** command. This determines the correctness and completeness of the included components and their resolution within the closure.

The following rules apply in assessing closures

- All expected classes are represented in the configuration.
- All expected classes are represented exactly once in the closure.
- Exactly one «process» class—not component—is realized inside the configuration. The «process» class is where the Web Site Form is defined.

Defining how the published tree browser will be organized

You can control how the published tree browser is organized in the four predefined compartments into which published information is inserted—namely the Disciplines, Roles, Artifacts, and Tools sections—by using a «tree node» package stereotype to structure the process elements in your process model. This stereotyped package type may exist in arbitrary structures, which contain process elements, in your process

model. When published, the structure of Tree Node packages is preserved for each process element and is superimposed on the existing structure in its designated compartment in the tree browser.

Additionally, you can override a published tree browser organization, which defaults to be ordered alphabetically, with your own preferred order. Such rearrangements are preserved over subsequent generations into the same location.

Those portions outside of the four predefined compartments of the tree browser are configured by editing a text file, instead of from the modeling space.

1 Create a Tree Node package in your process model

Tree nodes are packages in process models with a special meaning: they are used for providing structure to published tree browsers. A given process element is published under the same folder path, in the tree browser as the path formed by its enclosing tree node packages in the process model. The path for each element is computed relative to the enclosing process model.

- Create a tree node package by creating a package in the process model.
- Stereotype it to «tree node».

2 Insert new process elements into the existing tree browser folders

Consider the following example: let's say you want to insert new elements into an existing Artifact set defined in RUP.

RPW supports this operation by superimposing tree node paths from the various elements, using the model names of the tree nodes to determine equality.

- Following the example above, insert an element in the existing structure by creating «tree node» packages in your process model.
- Give them the same names as the ones in the RUP process model.
- Insert the elements in their respective tree node.

3 Create new folders in the tree browser

- Create new folders in the tree browser by creating «tree node» packages in your process model, and give them unique names.
- You can also create hierarchies of tree nodes by nesting «tree node» packages in the process model.

Upgrading to a new version of the RUP

New versions of the RUP are released at regular intervals. If you've followed our recommendations to keep your customized process models and content library separate from the base RUP material, merging a new version of the RUP with your customized material is straightforward.

To upgrade to a new RUP version, you need to bring the new process model into your Rose workspace, synchronize it with your customized process elements, connect the new RUP process model with its accompanying Content Library, and capture and reestablish your component realization from the RUP process model. A combination of Rose and RPW functions support you in determining if a received process model is compatible with its predecessor and, if not, where the differences exist and reestablish your realization dependencies.

1 Capture your current realization dependencies into the RUP process model

Before you load the new RUP process model into your workspace, you need to capture existing dependencies you have established to its predecessor.

On your Component Model, select Capture component realizations.

2 Bring the new process model from the latest version of the Rational process model into your Rational Rose workspace

The RUP is delivered in the form of two separate model files—one containing the process model and one containing the component model. These two models are represented in the RUP model that was included in RPW installation kit: the RUP process model in the logical view, and the component model in the component view of your Rose model.

- When you receive updates of these models, bring them into your Rose model by using the Rose load unit function and specify the location of the RUP model. If your Rose model already has a previous RUP process model loaded, now you can unload it.
- If your Rose model doesn't already contain a representation of the RUP model, go to the File menu and select Units.
- In the presented File Selection dialog, select the new RUP process model's `.cat` file or the component model's `.sub` file.

3 Inject captured realizations into the new RUP model

The initially captured dependencies in the RUP process model need to be injected into the new RUP model to reestablish those dependencies.

On the new RUP process model, select **Inject Captured Realizations**.

4 Assess the dependencies from your process model to the RUP's process model

- On your «process model», select the Check syntax command. This command verifies that the new model still supports all associations previously established from your process model to the RUP model. This command takes a moment to execute and presents you with a bar that indicates the progress.
- The command is completed either with an OK message or a list indicating where syntax violations exist.

5 Associate the process model from the RUP with its accompanying content library

Once you have successfully loaded the new RUP process model into your Rose model, do the following:

- Associate it with its Process Content Library by using the Associate Content Library command on its «process model» package and selecting the root of the new Process Content Library in the presented File Selection dialog.
- Invoke the check files command to verify that all files are present in the Process Content Library.

6 Synchronize your process model with the latest Process Content Library in RUP

Verify that all file references from your process model into RUP's Process Content Library are still maintained in the new RUP release, as follows:

Invoke the Check files command on your «process model» package. This command works at any package and element level and invoking it at the process model level will verify the files for all elements inside the process model.

7 Assess your process closures

Assessing the closure of your processes ensures that all realized elements from the RUP process model are still represented in your model.

- Invoke the Assess closure command on the process closure component. This command takes a moment to execute and you are presented with a progress bar that indicates the progress.
- The command is completed either with an OK message or a list indicating where syntax violations exist.

Publishing a Process

This section describes how to publish process processes that have been specified in a process model using RPW.

Publishing your process Web site is the final step in making your customized process available for your software development organization. Publishing creates a complete process Web site, similar to the RUP Web site itself, but tailored according to your specifications.

A Process Content Library can be translated into different languages and retain consistency with its process model. In other words, two or more translated versions of a Process Content Library can exist for the same process model. RPW supports associating the process model to one of its Process Content Library variants and generating its Web site in that language.

Assessing the closure of your process

The closure of a process must be complete and correct before it can be published. A process closure is deemed complete when all expected process elements are represented inside of the closure, and it's deemed correct if there is no conflict between any two process elements in the closure.

This criteria serves the purpose of preventing attempts to publish nonfunctional Web sites. It also supports early detection of ill-defined processes during process development.

RPW provides a function to determine whether a process is correct and complete, giving you instant feedback and guidance to locate violating process elements.

1 Determine the correctness of your process

- Invoke the **Assess Configuration** command on the «process» component that represents the closure.

  This command takes a moment to execute and a progress monitor is presented to indicate its progress until it's finished.

- Incorrect constructs are reported in an error log, indicating the reason for the error and the affected process elements.

2 Correct errors in your process' closure

The following error types are detected during assessment of process closures:

- use of not-recognized element types in process elements
- realization of elements, which are not recognized, in process component

- violation of modeling principles and rules
- unsupported interfaces
- unresolved references

You will correct reported errors differently depending on the type of error. The presented error report indicates the cause of the error and the offending, or offended, process element.

- Modeling errors are corrected by modifying the process model.
- Component realization errors are corrected by modifying the component realizations.

Publishing a Web site

Publishing a process Web site is the final step in creating a customized process for your organization. A published Web site is a fully functional RUP Web site.

RPW gives you options, such as whether you want to represent certain information in graphical or tabular format. This helps you work effectively during process development and allows you to tailor the end result.

1 Publish a Web site

Publish a Web site by selecting the **Publish Configuration** command on the «process» component.

Since process Web site publishing is an operation-intensive task, RPW gives you options as to how complete it should attempt to make the published Web site. Here are the publishing options RPW prompts for:

- spontaneous hyperlinking between process elements
- external hyperlinking
- generating a keyword index file
- generating a site map
- generate the Search database
- generating graphically captured information; basically, whether graphics or tables should be generated

2 Interpret and correct errors reported during the publishing process

Two types of errors are detected during publishing: model errors and file errors. The same error-detection algorithm is executed, as when assessing closure, prior to commencing with the file processing operation.

During the file processing operation, any occurrence of missing or incorrect files is reported. The general mode of operation is that processing continues in an attempt to publish the best possible Web site to facilitate error correction.

Using custom-designed graphics in your Web site

Some information in a process model is specified using graphical notation; for example, the activity diagrams that specify the collaborations of the Disciplines and Workflow Details. RPW exports these diagrams "as-modeled" (that is, the diagram you create in the Rose modeling space is the diagram that will appear in the Web site), then inserts them in their place in the published Web site, and makes them navigable. In this way, RPW creates functional diagrams, albeit less appealing from an artistic perspective. In this way, RPW provides a means for inserting your own graphics that will be used in lieu of those graphics it generates. It also provides a function that notifies you when an inserted graphical image is out of sync with the process model, so you can always maintain your graphics in synchronization with your process model.

1 Specify which graphics to use for a process element

When you initiate a Publish command you are presented with a dialog where you can specify how you would like RPW to process graphical information.

- In the Publishing Options dialog, select the Graphics tab.
- Select Use Existing Graphics if you have valid graphics that can be inserted in lieu of the generated graphics. A prerequisite for such graphics to be included for an element is that its areamap and image files are defined in the element's Process Content Files dialog. If not, publishing will default to tabular (table) generation.
- Select Generate Graphics if you want RPW to generate graphical information based on diagram information in your process model. This option is available for Disciplines and Workflow Details.
- Select Generate Tables for fast generation. Graphically oriented information is then published in tabular format (as a table) to convey the same information as their corresponding graphics, but possibly in a less appealing format.

2 Correct graphics that are out of sync

Graphics are out of sync if the set of process elements used in an existing graphics image (to be precise, in its areamap) is different from the set of process elements that would actually have been published. This only applies to the Use Existing Graphics selection.

- During the publishing operation, any out-of-sync graphics are reported as warnings in the message log, which indicates the location of the discrepancy.
- To remove the use of existing graphics for individual process elements, and to allow for the generation of graphics instead, remove the areamap and image files from their Process Content Files dialogs.
- To update your graphics to match the to-be-generated graphics, you must update your existing areamap to include the correct set of process element references.

Command Reference 6

Almost exclusively, process model development is conducted in the modeling space provided by Rational Rose. Rational Process Workbench adds commands only where native Rose commands are insufficient or cumbersome to use.

RPW operation is based on the stereotypes that designate the different types of modeling elements. RPW also adds menu items, allowing you to right-click to invoke context menus for such elements.

All RPW commands can be invoked either in Rose's tree view or in diagrams, following the same principles as you would in Rose.

Process Element Commands

Process element commands are directed to process elements in Rose's logical view, and support modeling and verifying process elements.

This chapter describes these process element commands:

- Check Syntax
- Overview, including:
  - Artifact overview
  - Role overview
  - Activity overview
  - Tool Mentor overview
  - Discipline overview
  - Workflow Detail overview
- Attach Activity
- Attach Workflow Detail

Check Syntax command

The **Check Syntax** command can be invoked on any package in a process model. This command checks the syntax of each process element inside of the designated package, either directly contained or indirectly contained through nested packages, to verify that the element is well-formed according to the semantic model and that all of its associated elements exist in the model. The **Check Syntax** command can be invoked on any package, at any level, inside of a process model. It can also be invoked on the process model package itself.

Check Syntax helps you detect when expected derived process elements are missing in the derivation base process model. For a description of branching and merging process models, see the information under that heading in *Chapter 4* of this document.

Figure 20 illustrates RPW's **Check Syntax** command.

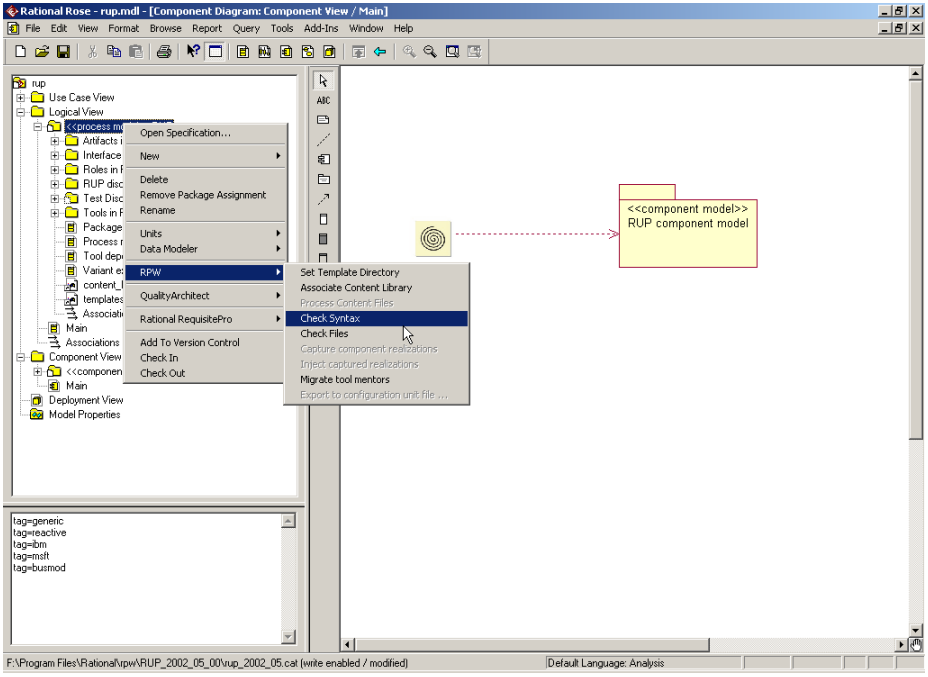

Figure 20 Check Syntax command

The result of the **Check Syntax** command is either a list of syntax errors or a message box indicating that the syntax is correct.

Overview command

The **Overview** command applies to all process element types. This command displays the Overview dialog, which shows the details of the designated process element. This Overview dialog is composed differently depending on the type of process element and shows the specific details of the element types.

The Overview dialog has two purposes:

- it provides an overview of a process element that can otherwise not easily be achieved
- it provides support for input of certain modeling information

Figure 21 illustrates the invocation of the **Overview** command.

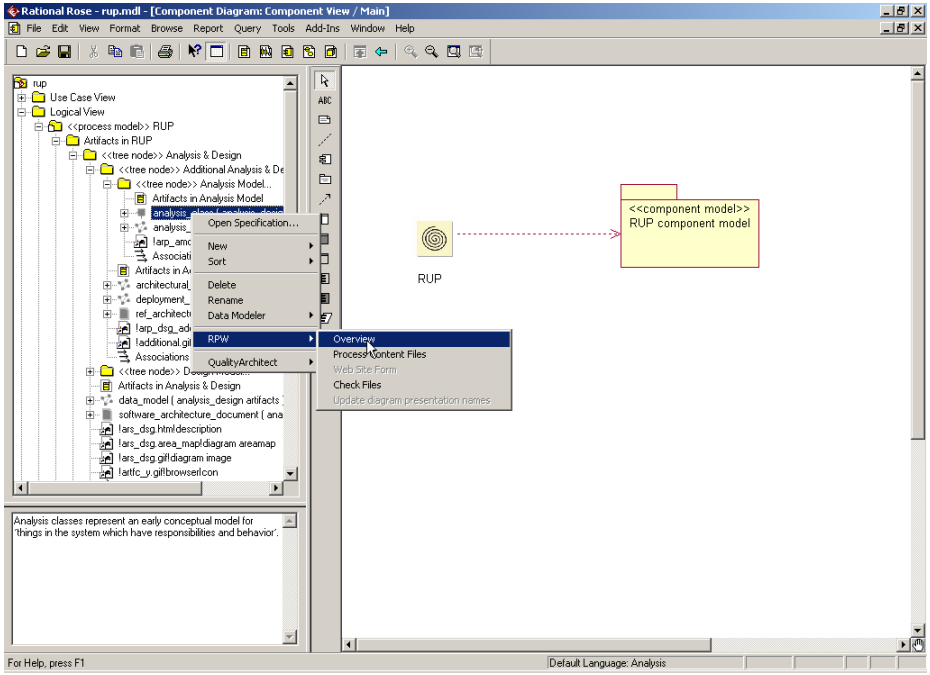

Figure 21 Overview command

The next seven subsections present the various overview dialogs for the different element types.

Artifact overview

The **Artifact Overview**, illustrated in Figure 22, presents, in two tabs, the Activities that use, create, or modify the Artifact.

Note: This information was modeled on the respective Activities and this presentation is **output only**.

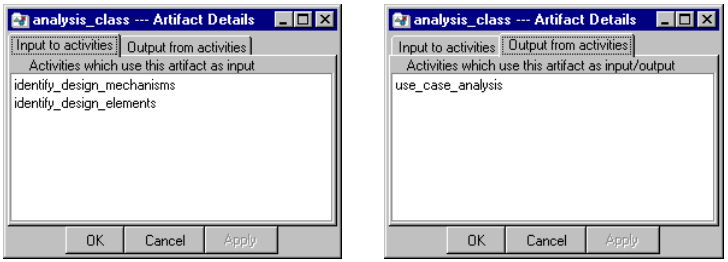

Figure 22 Artifact overview

Role overview

The **Role Overview**, shown in Figure 23, presents the **Artifacts** that the Role has been assigned «modifies» or «responsible» responsibility for in one tab, and the **Activities** performed by the Role in the other tab.

Note: This information was modeled on the respective Activities and this presentation is **output only**.

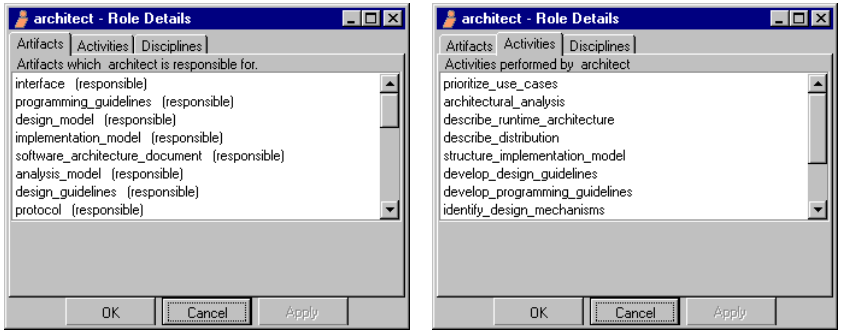

Figure 23 Role overview

Activity overview

The **Activity Overview**, as shown in Figure 24, presents information in its three tabs and supports the entry of modeling information. Figure 25 and Figure 26 illustrate the Tool Mentor and Disciplines tabs, respectively.

The **Artifact Overview** tab presents those Artifacts used, created, and modified by the Activity. The left-hand panel lists all Artifacts that are candidates for being modeled, whereas the right-hand panel lists those Artifacts currently selected in the respective category. *This panel is also an input panel*, where Artifacts are selected or deselected either by double-clicking on the Artifact in the appropriate list or clicking **Add** or **Remove**.

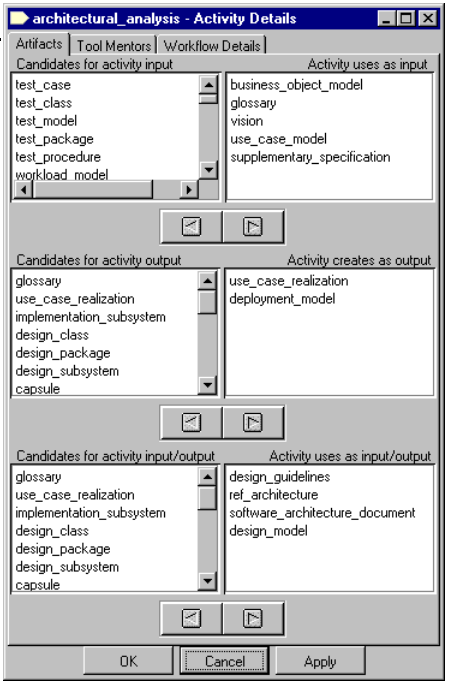

Figure 24 Activity overview

The **Tool Mentors** tab presents those Tool Mentors assigned to this Activity. This panel is also an input panel, letting you add and delete Tool Mentors from the list.

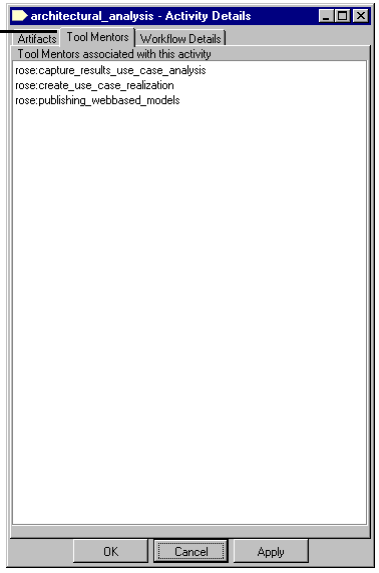

Figure 25 Tool Mentor tab on Artifact overview

The **Disciplines** tab presents the Workflow Details that this Activity is modeled to be part of.
This information was modeled on the respective Activities and this presentation is **output only**.

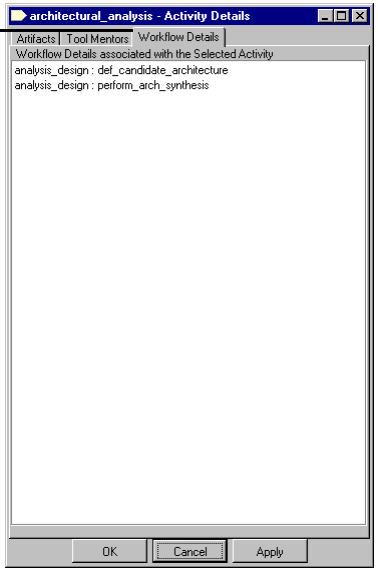

Figure 26 Disciplines tab on Artifact overview

Tool Mentor overview

The **Tool Mentor Overview**, see Figure 27, provides an overview of the associated Activities that have been assigned to this Tool Mentor as Tool Mentor for their task.

Note: This information was modeled on the respective Activities and this presentation is **output only**.

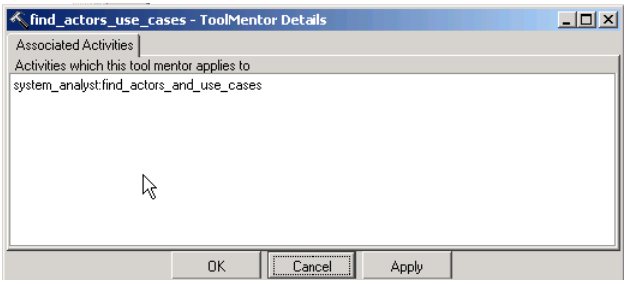

Figure 27 Tool Mentor overview

Discipline overview

The **Discipline Overview**, shown in Figure 28, presents those Roles that have been associated to the Discipline element through «participant» associations. These Roles are eligible for participation in the Discipline's Detail Activity Overview diagrams.

Note: This information was modeled on the respective Activities and this presentation is **output only**.

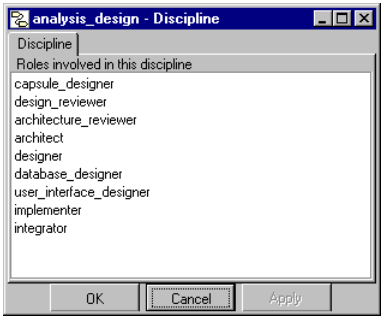

Figure 28 Discipline overview

Workflow Detail overview

The **Workflow Detail Overview**, see Figure 29, presents the Activities that participate in the Workflow Detail, as specified by its Activity Overview diagram.

Note: This information was modeled on the respective Activities and this presentation is **output only**.

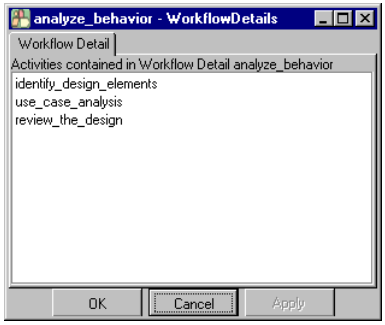

Figure 29 Workflow Detail overview

Attach Activity command

The **Attach Activity** command, shown in Figure 30, applies to activity overviews and displays the Attach Activity dialog, which supports the association of an activity state to its Activity. The selection lists presented in this dialog are populated with the discipline's participating Roles and their Activities.

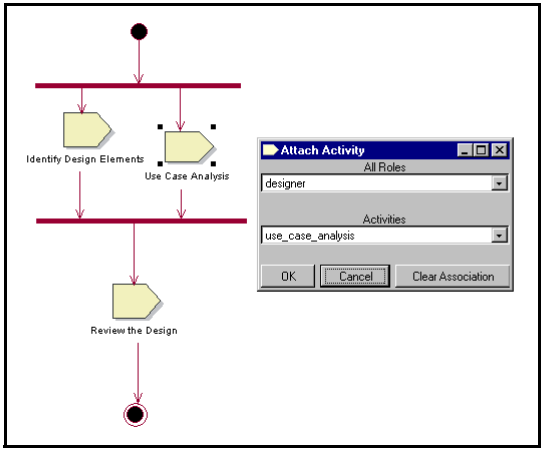

Figure 30 Attach Activity command

Attach Workflow Detail command

The **Attach Workflow Detail** command, see Figure 31, applies to Discipline workflow activity diagrams and displays the Attach Workflow Detail dialog, which supports the association of a Workflow Detail to a Discipline activity state. The displayed selection list is populated with the Workflow Details defined on the Discipline.

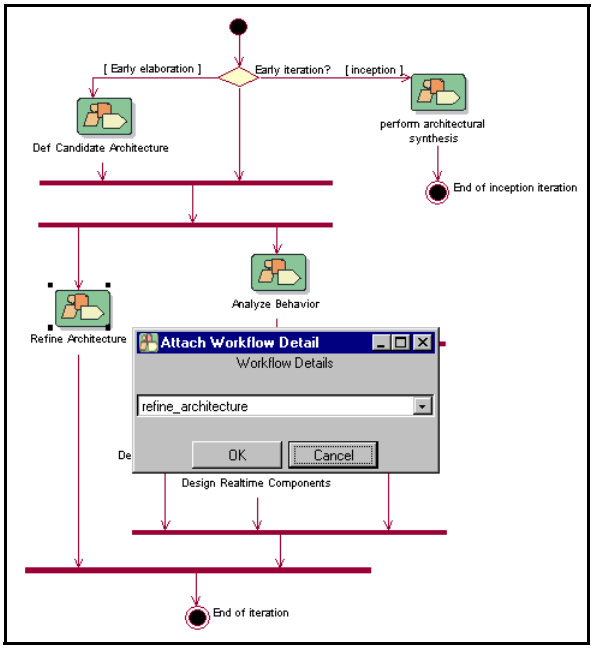

Figure 31 Attach Workflow Detail command

Process Content Library Association Commands

The commands presented in this section support the associations of files to process models.

Associate Text Library command

The **Associate Text Library** command, shown in Figure 32, displays a File Selection dialog where you enter the path to the root of a Process Content Library.

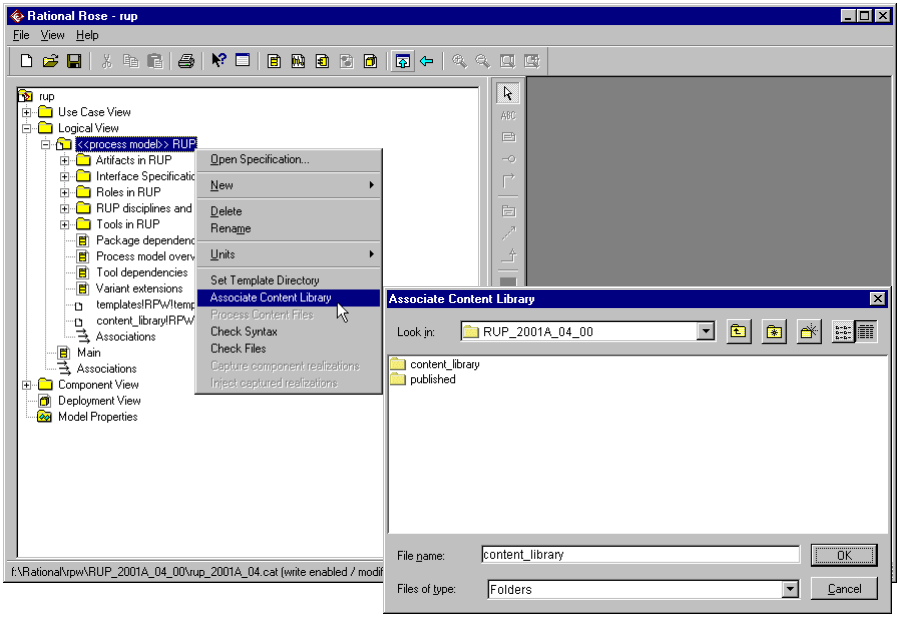

Figure 32 Associate Text Library command

Inject Component Realization command

The **Inject Component Realizations** command, shown in Figure 33, injects previously captured component realizations into a process model. Component realizations are captured using the Capture Component Realizations command. The sequences of that command and this one support the carrying forward of component realizations over consecutive releases of RUP process models, as part of process model merging.

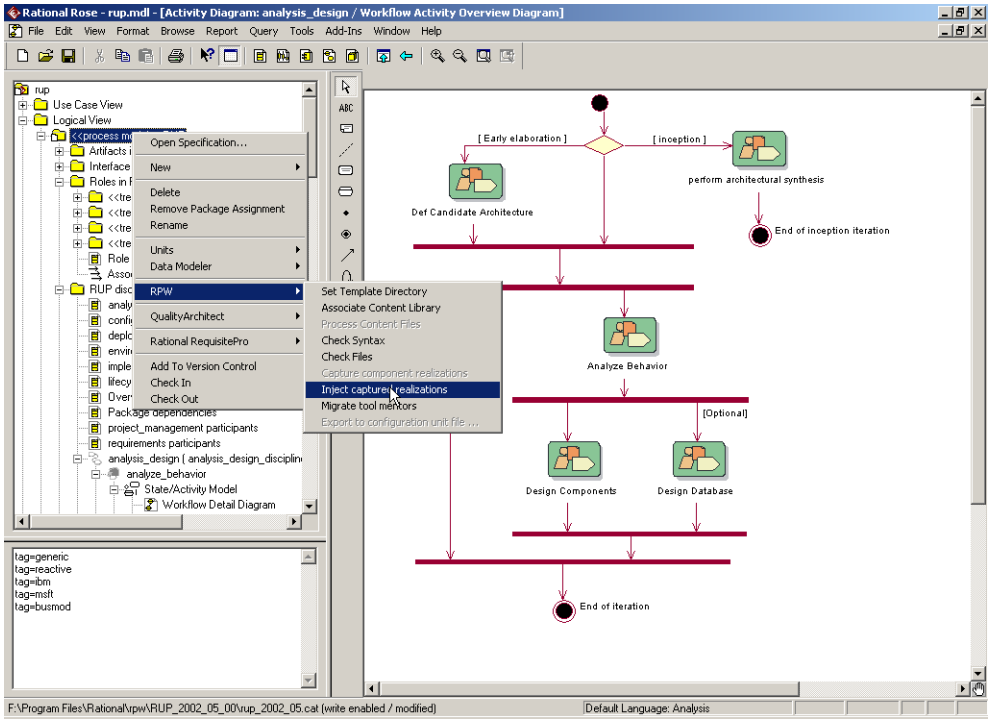

Figure 33 Inject Component Realization command

Process Content Files command

The **Process Content Files** command, displayed in Figure 34, launches a dialog where files can be associated to process elements.

Each process element type defines its own set of file types, which are the files that are relevant in the published process Web site for the element type. For a complete listing of the file types that apply to the different element types, see the discussion in *Chapter 3* of this document, under the headings *Common Files* and *Additional Files*.

Files from either the enclosing process model's associated Process Content Library or from the Process Content Library of a derived process model may be associated to a process element.

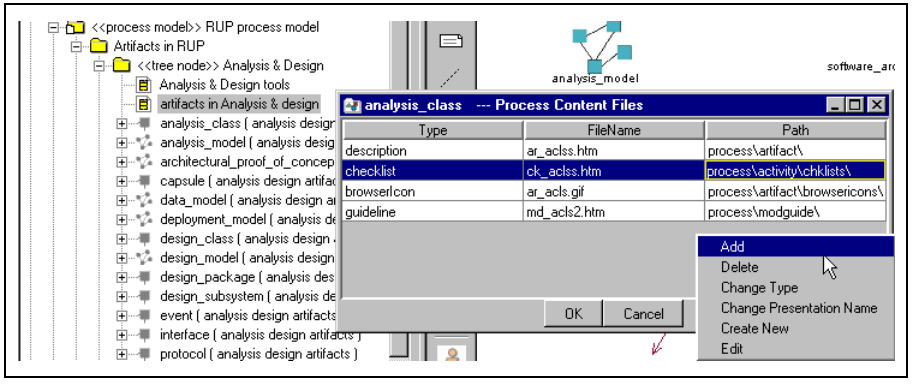

Figure 34 Process Content Files command

Web Site Form command

The **Web Site Form** command, shown in Figure 35, displays the **Web Site Form** dialog, which is a file selection dialog similar to the Process Content Files dialog described previously, except this one defines the Folders and files that constitute the Web Site Form of published processes.

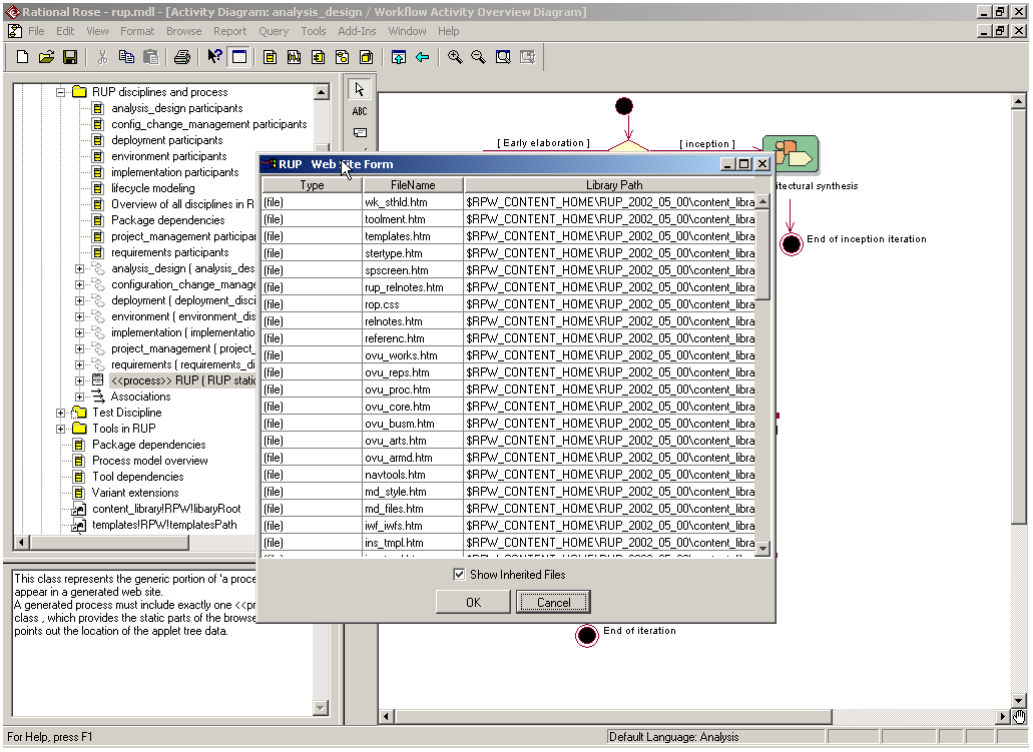

Figure 35 Web Site Form command

Check Files command

The **Check Files** command, see Figure 36, can be invoked at any level, both on individual process elements and on packages. Invoking the command on a package applies the command to all process elements contained in that package.

The **Check Files** command verifies that all files associated to a process element exist in their specified location in the file system.

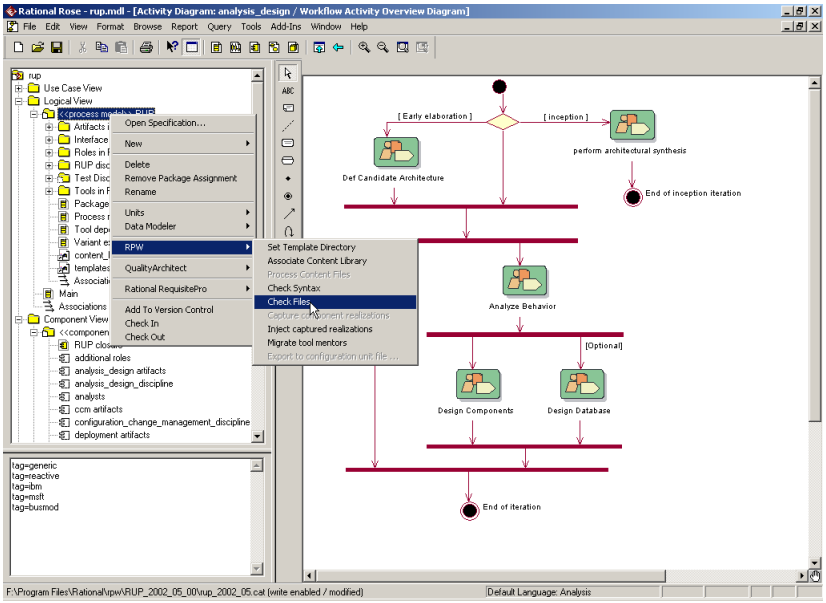

Figure 36 Check Files command

Process Component Commands

Process Component commands assist in the development of the component model, and in publishing of process Web sites. This section describes these process component commands:

- Assess Configuration
- Publish Configuration
- Capture Component Realization
- Export to Configuration Unit

Assess Configuration command

The **Assess Configuration** command assesses the correctness of your process configuration. It validates the process closure defined by your process configuration and reports any violations of the modeling principles and any unresolved or ambiguous element references (see Figure 37).

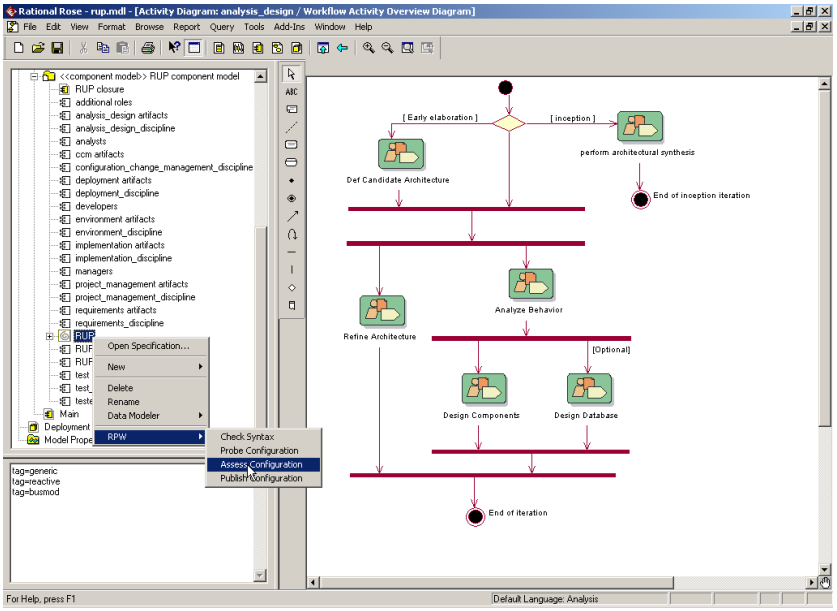

Figure 37 Assess Configuration command

Publish Configuration command

The **Publish Configuration** command, shown in Figure 38, publishes the designated process into the file directory currently associated to the designated «process» component.

Note: Any previous content in the designated directory will be erased.

Publishing a process takes considerable time to complete, depending on its size. For reference, publishing the full RUP takes about 15–20 minutes, depending on the particular configuration of the computer where the publishing takes place.

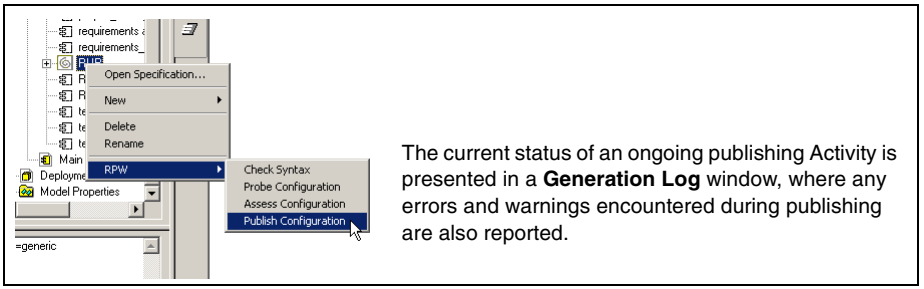

Figure 38 Publish Configuration command

Capture Component Realizations command

The **Capture Component Realizations** command, shown in Figure 39, is performed in preparation for merging a new RUP process model into your modeling space. This command is the second of two commands (the other being **Inject Component realizations**) that support the carrying forward of component realizations over consecutive releases of RUP process models, as part of process model merging.

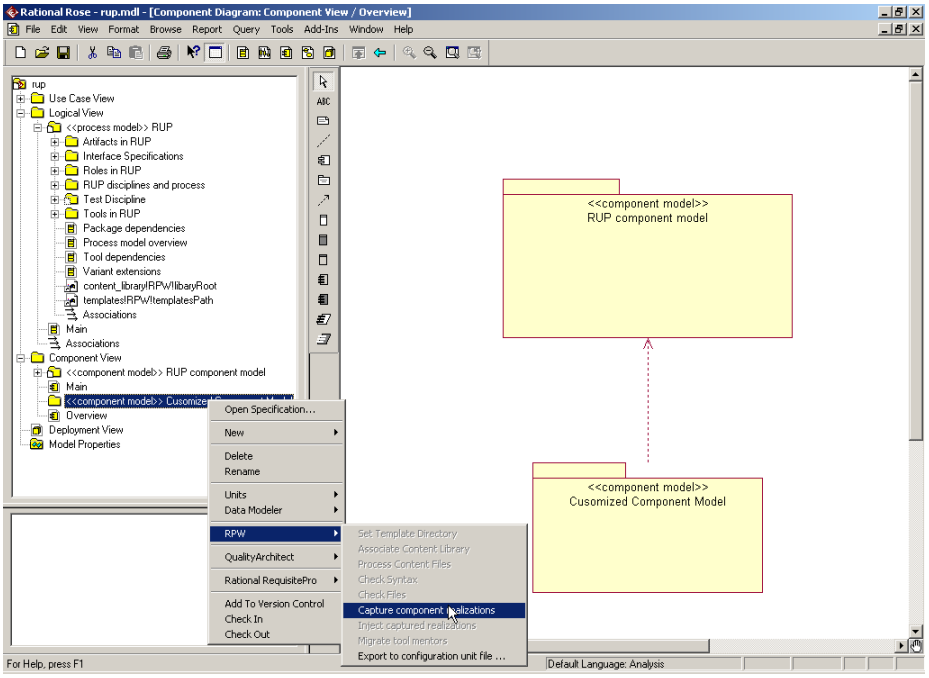

Figure 39 Capture Component Realizations command

Export to Configuration Unit File command

Exporting component models to the file system applies only those users who develop *process plug-ins*, because it assists them in creating their deliverables.

The **Export to Configuration Unit File** command exports the component model, and its included components and their associated process elements, as one entity to the file system. In preparation for export processing, RPW prompts for additional input similar to what happens when you publish a configuration to a process Web site.

For further details on developing process plug-ins (using RPW) and RUP Builder, please refer to the RUP Resource Center (www.rational.net/rupcenter/). The paper is titled *Creating Process Plug-ins Using Rational Process Workbench*.

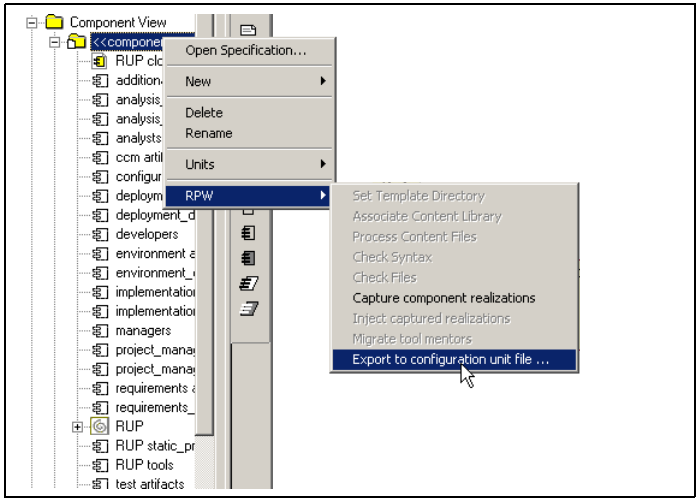

Figure 40 Export to Configuration Unit File command

Rational Process Workbench Commands in HTML Files

A

The generation capabilities of RPW are based on the insertion of commands into the HTML text material. These mark the places where RPW will insert generated information and defines or indicates the type of information to be inserted. These commands are already inserted into the templates provided with RPW and are generally not a concern for you in your process authoring.

When publishing your process Web sites, RPW requires specific HTML tags, recognized only by RPW, to determine the type of information to generate and where in the HTML document to insert this information. All HTML commands used by RPW follow this structure:

```
<rpw name="COMMAND" [attribute name]="[attribute
value]">ARGUMENT</rpw>
```

Note: The `[attribute name][attribute value]` pair can occur from zero to many (0...n) times.

The command, attribute name, attribute value, and argument parameter are all case-sensitive. If the *argument* parameter is not required, RPW will ignore any text entered in this area. All commands used by RPW are described in this format.

Command	*Argument:*	required argument
	Attribute:	required attributes
	RPW Element:	valid elements that the command containing the HTML file can be associated
	Required File Type Association:	required file type that must be associated to the element parameter specified above
	Description:	a description of the command and in some cases an example is provided

Caution: Do not embed RPW commands within each other or within an HTML hyperlink tag.

HTML commands for Artifacts

InsertArtifactImage	*Argument:*	none
	Attribute:	none
	RPW Element:	document, model element, model
	Required File Type Association:	diagram image
	Description:	inserts the associated diagram image file type
InsertArtifactResponsibleRole	*Argument:*	none
	Attribute:	none
	RPW Element:	Artifact
	Required File Type Association:	none
	Description:	inserts a relative link to the Role responsible for this Artifact
InsertArtifactReport	*Argument:*	none
	Attribute:	none
	RPW Element:	Artifact
	Required File Type Association:	report
	Description:	inserts a list of relative links to the associated report file for the Artifact
InsertArtifactMoreInfo	*Argument:*	none
	Attribute:	none
	RPW Element:	Artifact
	Required File Type Association:	checkpoint or guideline or concept
	Description:	inserts a list of relative links to all associated checkpoint or guideline or concept files for the Artifact

InsertArtifactTemplate

Argument: none

Attribute: target (optional)
ul_tags (optional)

RPW Element: Artifact

Required File Type Association: Microsoft Word template and/or HTML template

Description: Inserts a list of relative links to all Microsoft Word templates or HTML templates for this Artifact.

1 *target*

The attribute target is used in the same way as the HTML target attribute. By default, if it's not provided, the target="_blank", which opens the link into a new window.

2 *ul_tags*

The attribute *ul_tags* determines if RPW will start and end the list with the <*ul*> and <*/ul*> tags. All list entries will still be enclosed in <*li*> and <*/li*> tags. Valid values for this attribute are *true* and *false*. If the attribute is not present, the default value is *true*.

InsertArtifactInputActivity

Argument: none

Attribute: none

RPW Element: Artifact

Required File Type Association: none

Description: inserts a list of relative links to all Activity elements that use this Artifact as input

InsertArtifactOutputActivity

Argument: none

Attribute: none

RPW Element: Artifact

Required File Type Association: none

	Description:	inserts a list of relative links to all Activity elements that use this Artifact as output

HTML commands for Activities

InsertActivityGuideline	*Argument:*	none
	Attribute:	none
	RPW Element:	Activity
	Required File Type Association:	guideline
	Description:	inserts a list of relative links to all guideline files for an Activity
InsertActivityConcept	*Argument:*	none
	Attribute:	none
	RPW Element:	Activity
	Required File Type Association:	concept
	Description:	inserts a list of relative links to all concept files for an Activity
InsertActivityCheckpoint	*Argument:*	none
	Attribute:	none
	RPW Element:	Activity
	Required File Type Association:	checkpoint
	Description:	inserts a list of relative links to all checkpoint file for an Activity
InsertActivityToolmentor	*Argument:*	none
	Attribute:	none
	RPW Element:	Activity
	Required File Type Association:	none

	Description:	inserts a list of relative links to all Tool Mentors associated with this Activity
InsertActivityResponsibleRole	*Argument:*	none
	Attribute:	none
	RPW Element:	Activity
	Required File Type Association:	none
	Description:	inserts a list of relative links to the Role responsible for this Activity
InsertActivityWFD	*Argument:*	none
	Attribute:	none
	RPW Element:	Activity
	Required File Type Association:	none
	Description:	inserts a list of relative links to all Workflow Details in which this Activity participates
InsertActivityInputArtifacts	*Argument:*	none
	Attribute:	none
	RPW Element:	Activity
	Required File Type Association:	none
	Description:	inserts a list of relative links to all Artifact elements used as input to this Activity
InsertActivityResultingArtifacts	*Argument:*	none
	Attribute:	none
	RPW Element:	Activity
	Required File Type Association:	none
	Description:	inserts a list of relative links to all Artifact elements that are output by this Activity

InsertActivityMoreInfo

Argument: none

Attribute: ul_tags (optional)

RPW Element: Activity

Required File Type Association: concept, guideline, or checkpoint—all are optional

Description: inserts a list of relative links to any associated file types listed above; each entry will be prefixed with the file type name

HTML commands for Tools and Tool Mentors

ToolmentorOverview

Argument: none

Attribute: none

RPW Element: Tool

Required File Type Association: none

Description: inserts a list of relative links to all Tool Mentors contained in the Tool element

ToolmentorInsertActivityList

Argument: none

Attribute: none

RPW Element: Tool Mentor

Required File Type Association: none

Description: inserts a list of relative links to all Activity elements to which this Tool Mentor is associated

HTML commands for Roles

OverviewRole

Argument: none

Attribute: none

RPW Element: Role

Required File Type Association: optionally diagram image and diagram areamap

Description: performs two functions with two possible outputs, depending on your selection during the generation process:

- Table—which contains a list of Activities this Role is responsible for, with their corresponding Artifacts either created or modified by each Activity.
- Existing Graphics with Area Map—RPW inserts the associated diagram image and diagram areamap files. It reports any inconsistencies that may be present between the associated diagram areamap file and what is present in the Rational Rose model. If either of the two required files are missing, RPW defaults to the table format.

HTML commands for Disciplines and Workflow Details

OverviewWorkflow

Argument: none

Attribute: none

RPW Element: Discipline

Required File Type Association: optionally diagram image and diagram areamap

	Description:	performs three functions with three possible outputs, depending on your selection during the generation process: ▪ List—which contains all Workflow Details contained in this Discipline. ▪ Generated Graphics—RPW inserts the Rational Rose Activity flow diagram contained in this Discipline, complete with relative links to the appropriate Workflow Detail. ▪ Existing Graphics with Area Map—RPW inserts the associated diagram image and diagram areamap files. It reports any inconsistencies that may be present between the associated diagram areamap file and what is present in the Rational Rose model. If either of the two required file types are missing, RPW defaults to the generated graphics.
WorkflowConceptsPage	*Argument:*	none
	Attribute:	none
	RPW Element:	Discipline
	Required File Type Association:	none
	Description:	inserts a list of relative links to all associated concept file types from all elements contained in this Discipline
WorkflowGuidelinesPage	*Argument:*	none
	Attribute:	none
	RPW Element:	Discipline
	Required File Type Association:	none
	Description:	inserts a relative link to all associated guideline file types from all elements contained in this Discipline
WorkflowArtifactsPage	*Argument:*	none
	Attribute:	none
	RPW Element:	Discipline

	Required File Type Association:	optionally Artifact overview diagram image and Artifact overview diagram areamap
	Description:	performs two functions with these two possible outputs, depending on your selection of the output format for Workflow Diagrams during the generation process: ▪ Table—which contains a list of relative links to all Artifact elements contained in this Discipline. ▪ Existing Graphics with Area Map—RPW inserts the associated Artifact overview diagram image and Artifact overview diagram areamap file. If either of the two required files are missing, RPW defaults to the table format.
WorkflowActivitiesPage	*Argument:*	none
	Attribute:	none
	RPW Element:	Discipline
	Required File Type Association:	optionally Activity overview diagram image and Activity overview diagram areamap
	Description:	performs two functions with these two possible outputs, depending on your selection of the output format for Workflow Diagrams during the generation process: ▪ Table—which contains a list of relative links to all Activity elements contained in this Discipline. ▪ Existing Graphics with Area Map—RPW inserts the associated Activity overview diagram image and Activity overview diagram areamap file. If either of the two required files are missing, RPW defaults to the table format.
WorkflowDetailsList	*Argument:*	none
	Attribute:	none
	RPW Element:	Discipline

	Required File Type Association:	optionally wfd overview diagram and wfd overview diagram areamap
	Description:	performs three functions with three possible outputs, depending on your selection during the generation process: ▪ List—which contains all Workflow Details contained in this Discipline. ▪ Generated Graphics—RPW inserts the Rational Rose Activity flow diagram contained in this Discipline, complete with relative links to the appropriate Workflow Detail. ▪ Existing Graphics with Area Map—RPW inserts the wfd overview diagram and wfd overview diagram areamap files. It reports any inconsistencies that may be present between the associated wfd overview diagram file and what is present in the Rational Rose model. If either of the two required file types are missing, RPW defaults to the generated graphics.
WorkflowDetailsDiagram	*Argument:*	none
	Attribute:	none
	RPW Element:	Workflow Detail
	Required File Type Association:	optionally diagram image and diagram areamap

Description: performs three functions with three possible outputs, depending on your selection during the generation process:

- Table—which contains all Roles contained in this Workflow Detail. In addition, all Activities along with all resulting Artifacts are displayed with the appropriate Role.
- Generated Graphics—RPW inserts the Rational Rose Activity flow diagram contained in this Workflow Detail, complete with relative links to the appropriate Activities.
- Existing Graphics with Area Map—RPW inserts the associated diagram image and diagram areamap files. It reports any inconsistencies that may be present between the associated diagram areamap file and what is present in the Rational Rose model. If either of the two required file types are missing, RPW defaults to the generated graphics.

HTML commands for tree nodes

TreeNode

Argument: none

Attribute: none

RPW Element: tree node

Required File Type Association: optionally diagram image and diagram areamap

Description: performs two functions with two possible outputs, depending on your selection during the generation process:

- Table—which contains a list of elements nested one-level deep.
- Existing Graphics with Area Map—RPW inserts the associated diagram image and diagram areamap files. If either of the two required files are missing, RPW defaults to the table format.

HTML commands for diagram areamap file types

ImageHeight	*Argument:*	diagram image height
	Attribute:	none
	RPW Element:	Discipline, Workflow Details, tree node, Role
	Required File Type Association:	diagram image
	Description:	provides the height of the associated diagram image file type
ImageWidth	*Argument:*	diagram image width
	Attribute:	none
	RPW Element:	Discipline, Workflow Details, tree node, Role
	Required File Type Association:	diagram image
	Description:	provides the width of the associated diagram image file type
AreaMap	*Argument:*	HTML tag
	Attribute:	none
	RPW Element:	Discipline, Workflow Details, tree node, Role
	Required File Type Association:	diagram image
	Description:	contains the standard HTML tag area for a hot spot to the associated diagram image file type

General HTML commands

Include

Argument: relative path to a file from the output directory

Attribute: none

RPW Element: any

Required File Type Association: none

Description: inserts the contents of the file specified in the *argument*

RPW will process any *General HTML Commands* that are contained in the included file.

RPW will search for the include file in the following location and in the order specified here:

1 The specified "rpw folder" file type that is specified in your «process» element file collection if your relative path starts with this value

2 The current content library directory

3 If the file name is stripped from the relative path in the "rpw folder" specified in the «process» file collection

In all cases, you will continue through this order. If the last one listed here (3) fails, you'll receive an error message.

Example of the **include** command:

<rpw name="Include">rpw/myincludes/include.inc</rpw>

If the "rpw folder" is in the content_library installation path "**c:\program files\rational\rpw\myProcess\content_library\rpw**" and the current root library directory is "**c:\program files\rational\rpw\rup\content_library**", RPW will proceed as follows:

1 check to see if the file is present in **c:\program files\rational\rpw\myProcess\content_library\rpw\myincludes\include.inc**

2 check to see if the file is present in **c:\program files\rational\rpw\rup\content_library\myincludes\include.inc**

3 check to see if the file is present in **c:\program files\rational\rpw\myProcess\content_library\rpw\include.inc**

InsertRelative HyperLink	*Argument:*	anchor display text
	Attribute:	link
	RPW Element:	any
	Required File Type Association:	none
	Description:	inserts a relative link to the target location specified by the *link* attribute The *link* attribute must be a relative link from the output directory. Example: **<rpw name="InsertRelativeHyperLink" link="toolment/toolment.htm">Toolmentor Overview</rpw>** will produce **<a href="{relative link to <outputdirecotry>\toolment\toolment.htm}"> Toolmentor Overview **
InsertRelative Path	*Argument:*	none
	Attribute:	link, pre, post
	RPW Element:	any
	Required File Type Association:	none
	Description:	inserts a relative path enclosed in the contents of the *pre* and *post* attribute values The following symbols cannot be contained in the *pre* and *post*, but must be entered using the HTML equivalent: '<' -> < '>' -> > '"' -> " Example: **<rpw name="InsertRelativePath" pre="<img src=" post=">" link="images/rup1.gif"></rpw>** will produce **<img src="{relative link to <outputdirecotry>\images\rup1.gif}">**
InsertTemplate FileName	*Argument:*	none

Attribute: link, pre, post

RPW Element: Discipline template file

Required File Type Association: none

Description: inserts links to all of the supporting files for a new discipline

Example: create a new Discipline called newDiscipline.

RPW will automatically create all of the supporting files with the following names:

- newDiscipline_Introduction.htm
- newDiscipline_guideline_overview.htm
- newDiscipline_concepts_overview.htm
- newDiscipline_artifact_overview.htm
- newDiscipline_activity_overview.htm
- newDiscipline_wfd_overview.htm

Once the files are copied, this command is run to insert the names of the supporting files into the links from the overview page.

Warning

Argument: none

Attribute: none

RPW Element: any

Required File Type Association: none

Description: does not perform any function and is removed during the generation process

Glossary

Activity. An activity describes an action that is performed by a role.

Activity overview. Activity overviews are activity diagrams that describe the collaborations within individual Workflow Details.

Artifact. An artifact describes a product of software engineering.

branching and merging. Branching and merging are common terms for the activities involved in maintaining parallel development streams. Although this is most common in software development, it's also applicable in process development.

class. A description of a set of objects that share the same attributes, operations, methods, relationships, and semantics.

class diagram. A class diagram shows the existence of classes and their relationships in the logical design of a system. A class diagram may represent all or part of the class structure of a system.

closure. See *process closure.*

Discipline. Disciplines are process elements that define distinct boundaries within a process.

Discipline workflows. Discipline workflows are abstract workflows that describe the overall activity model of a process.

dynamic specifications. Dynamic specifications define concepts that specify how the process elements collaborate in a process. UML activity diagrams and sequence diagrams are used for these specifications.

inheritance. The mechanism by which more specific elements incorporate structure and behavior of more general elements.

interfaces. In Rational Rose's logical view, interfaces can be used to specify various responsibilities of the process model. Applied when designing process, interfaces are used to enable variation of a given deployment of a process.

operation. A service that can be requested from an object to effect behavior.

process. A software development process—the steps and guidelines by which to develop a system. Process specifies a particular lifecycle and defines the phases that constitute this lifecycle.

process closure. The process elements included in a process.

process components. Process components specify how process elements are grouped into components to form "chunks of process" that will be deployed as a collective unit.

Process components represent non-arbitrary sets of process elements that are internally consistent and may be reused with other Process components to assemble complete processes.

Process Content Library. This is a container of process text descriptions and Web graphics.

process elements. Process elements define the core concepts—like role, artifact, activity, and workflow—used in process modeling and how these are associated to each other.

process model. A process model captures the design of a process. A process model defines a complete set of process elements and any processes that include these process elements in their specifications.

Role. A role describes an agent in software engineering.

semantic model. The definitions of the elements and principles you use to define your own process models is referred to as the semantic model.

sequence diagram. A diagram that describes a pattern of interaction among objects arranged in a chronological order. It shows the objects participating in their "lifelines" and the messages that they send to one another.

stereotypes. A type of modeling element that extends the semantics of the metamodel. Stereotypes must be based on certain existing types or classes in the metamodel. Stereotypes may extend the semantics, but not the structure of pre-existing types and classes.

Tools. A tool element describes the tool mentors for the particular tool used in the software engineering environment. A Tool represents a particular development tool used in an organization.

Tool Mentors. A tool mentor provides a recipe for how to perform specific process activities using a particular software tool.

Web Site Forms. Web Site Forms are the collection of HTML files and directory structure that constitutes the framework into which published process Web sites are generated.

Workflow Detail. A grouping of activities that are performed in close collaboration to accomplish some result.