IBM Cúram Social Program Management

Working with the Cúram Model in Rational Software Architect

Version 6.0.4

Note

Before using this information and the product it supports, read the information in Notices at the back of this guide.

This edition applies to version 6.0.4 of IBM Cúram Social Program Management and all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this guide is to detail how *IBM® Rational® Software Architect* is used to work with the *IBM® Cúram Social Program Management* UML model. *Rational Software Architect* primarily as a tool for UML modeling, analysis, and design. A key reason that *Rational Software Architect* is used for these functions is because of its support for domain specific languages and *Eclipse* extensibility, which enables the use of a powerful and intuitive user interface.

Although *Rational Software Architect* is used for UML analysis and design, a discussion of these topics falls outside the scope of this document which focuses on specific tasks that can be performed with the Cúram model in *Rational Software Architect*.

1.2 Audience

This guide is intended for any reader who will be using *Rational Software Architect* to perform common modeling tasks on the Cúram UML model.

1.3 Prerequisites

Readers should have a good working knowledge of UML, *Java*®, and *Eclipse*.

Note

*Rational Software Architect*Rational Software Architect is third-party software. Please refer to the *Cúram Supported Prerequisites* document for more information on the supported versions of third party tools.

The *Cúram Modeling Reference Guide* should be consulted as reference for further information on the *IBM Cúram Social Program Management* modeling elements.

1.4 Chapters in this Guide

The following list describes the chapters within this guide:

The *Rational Software Architect* Workbench

This chapter describes the *Rational Software Architect* workbench and the various views that make up the workbench. It also covers *Rational Software Architect* basics including creating, opening and closing a model, basic tasks for model elements, and creating and absorbing model fragments.

Using *Rational Software Architect* with the Cúram model

This chapter describes in detail specific Cúram model elements and how they are manipulated in *Rational Software Architect*.

How *Rational Software Architect* Differs from *IBM® Rational Rose®*

This appendix compares and contrasts the differences between modeling in *Rational Rose* and *Rational Software Architect*.

Right Click Context Menu Options for Model Elements

This appendix describes what can be added from the right click context menu for each model class in the *Rational Software Architect* project explorer window.

Chapter 2

The Rational Software Architect Workbench

2.1 Introduction

This chapter details the main parts of the *Rational Software Architect* workbench that you will use when working with the *IBM Cúram Social Program Management* model. In *Rational Software Architect*, a perspective is a particular layout of views, an editor and tool bars. Here the *Rational Software Architect* modeling perspective is described. This perspective allows you to view, create, and maintain elements of the Cúram model.

The most common tasks that are performed in *Rational Software Architect* are also detailed in this chapter. These include:

- Creating, opening, and closing a model
- Basic maintenance tasks common to all model elements
- Working with model fragments
- Searching in the model

Note

Rational Software Architect can be used as a modeling tool or as a plug-in for *Eclipse*. For the purposes of this guide, the focus will be on using the *Rational Software Architect* standalone tool.

2.2 Integrating the Cúram Model into Rational Software Architect

The Cúram model uses a small subset of the range of functionality provided by *Rational Software Architect*. In order to simplify the use of *Rational Software Architect* for the Cúram model, a number of techniques are used to tailor the tool to the Cúram model.

A Cúram profile is provided for working with the Cúram model. The Cúram profile defines what UML stereotyped elements and values can be defined within the Cúram model.

When you create a model in *Rational Software Architect*, the Cúram model template is used. This combines the Cúram profile with a filtering capability to remove unnecessary or unsupported functionality from menus and options in the *Rational Software Architect* workbench. The Cúram model template is automatically used when you open an existing Cúram model. You should always select it when creating a new model.

Note

Some options which are not supported in the Cúram model cannot be hidden from the user. If the user performs an unsupported action, a validation message is displayed and the action is reversed. For more information on this please refer to Section 2.5.3, *Modifying an Element*.

2.3 The Modeling Perspective

The *Rational Software Architect* Modeling Perspective is used for UML modeling and consists of four main views: the Project Explorer view, the Properties view, the Diagram Editor view, and the Model Editor view.

View	Description
Project Explorer	Allows you to see all the related parts of the Cúram model in a navigable tree structure
Properties	Allows you to view and maintain information about a selected model element.
Diagram Editor	Allows you to create, view, and edit model diagrams using the custom Cúram palette.
Model Editor	Allows you to view and edit a model's configuration in a tabbed view.

Table 2.1 Modeling Views

2.3.1 Project Explorer View

The Project Explorer view allows you to view all the related model elements, diagrams, and children of a selected model element. The right-click context menu in this view provides a range of options that can be performed for the selected element.

2.3.2 The Properties View

The Properties view allows you to view and edit the properties for the selected model element. It allows you to configure general and stereotype proper-

ties for an element, set element relationships, manage element documentation, etc.

The tabs in the Properties view used for Cúram model development are the General tab, and the Documentation tab, and the Cúram tab. These are described in the table below.

Tab	Description
General	The General tab allows you to maintain the base UML configuration of an element in a model, e.g., name, visibility, etc.
Documentation	The Documentation tab allows you to create, view, and edit documentation relevant to a specific element.
Cúram	The Cúram tab holds the properties that are relevant to a stereotyped element in the Cúram domain. These properties are specific to the Cúram model and are used to enhance the configuration of an element.

Table 2.2 Description of tabs in the Properties view used in Cúram Modeling

2.3.3 The Diagram Editor

The Diagram Editor is used to create, view, and edit diagrams. It is split into two areas, a Diagram view and a Palette. The Palette contains a Cúram "drawer" which contains a number of the most commonly used Cúram model elements for dragging and dropping into the Diagram Editor. The Diagram view is where you can view and modify a model element in relation to other elements. A right-click context menu is also available for the addition of model classes in the Diagram Editor.

2.3.4 The Model Editor

The Model Editor allows you to view and edit information related to a model or sub-unit fragment. To open a model in the model editor right-click on the model in the project explorer, select Open With, and then Model Editor.

The Cúram Profile can be viewed in the Details tab of the Model Editor. The profiles and model libraries that are used for working with the Cúram model are pre-configured for the Cúram model. In general, you should not need to alter the profiles and/or model libraries supplied with the Cúram model.

There are 5 tabs in the Model Editor view: Overview, Details, Diagrams, References, Fragments. These are described below.

Tab	Description
Overview	The Overview tab contains general information related to the selected model. You can also edit documentary information related to the model here.
Details	The Details tab allows you to maintain the profiles and model libraries that are applied to the model you are currently viewing.
Diagrams	The Diagrams tab allows you to view a list of available diagrams for the selected model.
References	The References tab provides a list of other models and profiles from the workspace referenced by the selected model. It also provides a list of other models from the workspace that reference the selected model.
Fragments	The physical resources associated with the model elements are called fragments and are essentially separate files. The ability to divide a model into fragments is particularly useful in large development projects. This is done by extracting packages into physical sub-units, or fragments. The physical location of the model elements are transparent, and the fragments remain a logical part of the original model. The Fragments tab contains a list of fragments that are included in the model. You can choose to search for and absorb fragments into the containing model from this list. For more information on absorbing fragments, please see Section 2.9.2, *Absorbing a Fragment*.

Table 2.3 Description of Tabs in the Model Editor

2.4 Working with the Model

This sections below describe how to create, open, close, and navigate a model in *Rational Software Architect*.

2.4.1 Creating a Model

Rational Software Architect provides the Create Model Wizard to assist you in creating new models from stored templates. A Cúram template is provided in the Cúram plugin for *Rational Software Architect*. When using the template to create a new model, the model capabilities are set to what is appropriate for that template. Using the Cúram model template to create

your model ensures that the menus and options are available while modeling are those supported by the Cúram model.

There are two ways of invoking the Create Model Wizard in *Rational Software Architect*:

- Right-click on the model directory and select Create Model.
- Select File from the topmost menu bar, New, and then UML Model.

To create a Cúram model in the Create Model Wizard:

1. Name the model and specify a location.
2. Select the Standard Template option.
3. Select Cúram in the Categories pane (check the 'Show All Templates' option to see the Cúram category).
4. Select Cúram model in the templates pane.
5. Select the model capabilities. By default, Cúram capabilities are selected. It is recommended that you use the default capabilities.
6. Select the referenced models, if there are any.

The '< Back' and 'Next >' buttons allow you step forward and backward through the Create Model wizard steps. You can exit the Create Model wizard at any time by clicking on the Finish button. The model will be saved at the point you exit the wizard.

2.4.2 Opening a Model

You can open a model in the Modeling perspective within *Rational Software Architect* or *Eclipse*. The default settings for this perspective display the Project Explorer view on the left-hand side. When a model exists in your project, the Project Explorer contains two additional folders (Diagrams & Models). Expanding these, and the model underneath, opens the Model at that point.

Rational Software Architect uses a form of lazy loading whereby the full model will not be opened initially and each portion is opened as it is navigated to. Alternatively you if want to force a load of the full model you can right-click on the top level model file and select Open All Sub-Fragments.

2.4.3 Closing a Model

To close a model, right-click on the model and select Close or Close All from the right-click context menu.

2.4.4 Navigating the Model

To move through the packages and elements of a model, select the expand

'+' option for the package or element where children exist in the Project Explorer. If a package is not currently loaded, it will be loaded and the icon will change. The package will be expanded to display the child elements. Modeling elements can be added to the model in the Diagram Editor using the Model Palette and the Cúram Drawer.

2.5 Working with Model Elements

The UML Model elements utilized by Cúram include Packages, Classes, Attributes, Operations, Parameters and Relationships.

- Packages are containers for classes.
- Classes define the Business Processes, Value and Rules Objects or Database Schema e.g. Facade, WebService, RDO, Entity, etc.
- Attributes define fields on the value or rules objects or database entities.
- Operations represent the business or SQL functions on relevant to a parent class.
- Parameters are the input or return arguments to a parent operation.
- Relationships define bonds between the various classes that make up the application e.g. aggregation of structs, foreign keys between entities.

The sections below describe how to view elements, add elements to a model, modify elements, remove elements from a model, and create relationships between classes.

2.5.1 Viewing an Element

To view an element, ensure that the Properties view is opened in the foreground and select the element in the Project Explorer. The element will then be opened in the Properties view and a number of tabs relevant to that element will be available to maintain that element.

The right-click context menu in the Project Explorer allows you to access functionality for that element as well as view information related to the element.

Selecting to expand a modeling element in Project Explorer will load that element's sub-fragments. The modeling element's icon will change in the Project Explorer to mark that its sub-fragments have loaded.

2.5.2 Adding an Element to the Model

There are a two main ways to add an element to a model in *Rational Software Architect*, using the Project Explorer right-click context menu or the Diagram View and Palette. These are described below.

Using the Project Explorer

1. Right-click on the existing parent element in Project Explorer.
2. Select the relevant element type from the list of child element types of your selected element.

Using the Diagram View

1. Open the diagram you wish to add the new element to in the Diagram view.
2. Double click on the required element in the Cúram Palette, or right-click in the Diagram view and select the element you wish to add from the right-click context menu.

2.5.3 Modifying an Element

A model element's name can be modified from the right-click context menu in the Project Explorer or the Diagram view. More extensive modification, including an element's documentation, can be performed through the properties view and the tabs available there.

Element properties that are specific to Cúram are managed on the Cúram tab in the Properties view. Depending on what is being modified some modifications require two separate changes to be made, one on the Cúram tab and one on another tab in the Properties view. For example, changing the return type of an operation from an entity shadow class to a handcrafted struct requires you to change both the return type on the General tab and the Shadow Type on the Cúram tab. For more information on Shadow Types see Section A.2, *Shadow Classes*. Similarly changing the primitive type for a domain definition from SVR_INT16 to SVR_STRING requires you to change both the primitive type on the Attributes tab and add a Maximum_Size entry on the Cúram tab.

Important

When you attempt to perform an action that is not supported by the Cúram model, a validation will be displayed and the action will be reversed. Due to a bug in *Rational Software Architect*, the view may not be updated until refreshed for example; navigating to another element and back to the element being updated. The use of the operations, attributes and parameters tabs to add elements is not currently supported for use with the Cúram model as these tabs do not provide the correct Cúram stereotypes.

2.5.4 Creating a Relationship between Elements

To create a relationship between elements:

1. Select the element you wish to create a relationship with in the Project Explorer.
2. In the Properties view, navigate to the Relationships tab.
3. Search for the other element for the relationship and select the source or target.
4. Select the type of relationship you wish to create between the two elements.
5. The General Tab of the Relationship can then be used to specify a name, multiplicity, etc.

Relationships can also be created in the Diagram view using connector handles. To do this:

1. Hover over an element in a diagram.
2. Drag one of the available arrowheads onto another element in the diagram to create a relationship. The two different arrowheads signify whether the element being dragged from is the source or target of the relationship.
3. The General Tab of the Relationship can then be used to specify a name, multiplicity, etc.

2.5.5 Removing an Element from a Model

Model Elements can be deleted in the Project Explorer or Diagram Editor. To delete an element in the Project Explorer, right-click the element and select Delete from Model. Elements can also be deleted in the same way in the Diagram Editor.

In the Diagram view, you can choose to delete an element just from the diagram, or from the complete model. Deleting an element from the model means that the element will be unavailable to other parts of the model and will be removed from the Project Explorer tree.

Important

References to other elements in *Rational Software Architect* are maintained by an internal identifier system. Each element is given a unique identifier on creation and references are made to this unique identifier. This differs from *Rational Rose* where references could be determined both by id and qualified name. It is therefore important to note that if a class is removed, recreating the class with the same name is not sufficient to correct any broken references and the broken reference resolution process will be required to reconnect broken references.

2.5.6 Copying and Pasting

To save time and effort, you can use the *Rational Software Architect* Project Explorer to copy and paste Classes, Operations and Attributes. Operations and attributes can only be copied in the same class or between classes of the same type. An example of how to copy and paste between classes is:

- Select the attribute(s) in the tree control of the *Rational Software Architect* Project Explorer.
- Right-click on the selected attribute(s) to be copied and choose 'Copy' from the context menu.
- Then right-click on the class (of the same type) to receive the new attribute and choose 'Paste'.

You can use a similar technique for moving attributes via 'Cut' and 'Paste'.

Note

If you try to copy/paste across different class types you will receive an error dialog indicating: "The requested action violates the integrity of the model."

2.5.7 Attribute Order

Be aware that by default *Rational Software Architect* displays the order of attributes alphabetically. Attribute ordering is significant for Entity and Struct classes when they are used to define indexes as the DDL that's generated for index creation relies on this ordering. You can view the attribute ordering via the Attribute tab of the class. You can also change the default behavior of *Rational Software Architect* from its default ordering of 'Stereotyped Type then Alphabetically' to 'Storage Order' by selecting the Windows menu and Preferences submenu. From the resulting dialog navigate to Views, Modeling, and Project Explorer where you can use the Project Explorer settings, Sort By drop-down to change the ordering; click OK to save your changes.

If you need to change the order of the attributes the Attributes tab provides 'Move up' and 'Move down' buttons as appropriate.

2.6 Searching in Rational Software Architect

The sections below describe how to search in a model, search for element references in a model, and search for elements using the type browser.

2.6.1 Searching the Model

The Search option is a powerful text search tool that can be used to search the model using a broad range of criteria. The Model Search functionality can be used to see how an element is related to the rest of the model. To search for an element in the model:

1. Select the Search option from the main menu bar and click on the Model Search tab.
2. Specify your search criteria. There are a range of search criteria that can be specified that allow you to narrow your search.
3. Search results are displayed in a tab beside the Properties view. Double clicking on a search result listing will cause the project explorer to jump to that element.

2.6.2 Searching for References to an Element

You can search for references to an element. To do this:

1. Select the element you wish to search for references to.
2. From the right-click context menu, select the `Modeling References` option. You can choose to search for references in the enclosing model, the enclosing package, the workspace, or you can define a custom working set.
3. The results of your search will then be displayed in the `Search Results` tab as seen below. Double clicking on a search result listing will cause the project explorer to jump to that element.

2.6.3 Searching for Elements using the Type Browser

The Type Browser is used during the creation of an element that requires the specification of a type or element reference. It is used to search for the type of model element you wish to create, for example, the parent of a Domain, the Return Type of an Operation, etc. When you open the Type Browser, you can enter the name of the element to search for, or you can also browse for the element directly in the model. The `Modify Search Scope` option will control the scope of the search. Searching is based on an index that *Rational Software Architect* will maintain across sessions and does not require the complete model to be opened. Please note that the first search will be longer due to the creation of this index.

2.7 Specialized Tabs and Wizards

Some specialized tabs and wizards are provided to support assisting frequent tasks or management of specific complex content.

2.7.1 Foreign Key Tab

The Foreign Key tab offers a tab to allow for the definition and maintenance of a foreign key's name and mappings. This tab is visible on the Properties View of a Foreign Key relationship.

The Name field is used to manage the foreign key name and manipulates the label entry on the General Tab.

The Table contains two columns; child and parent and these columns indicate the direction of the foreign key and name the entities on either end of the relationship. The table serves as a widget to edit the foreign key mappings which are stored on the appropriate role fields on the General tab.

Rows on the table relate to the mappings in the foreign key where a entry in the child column will be mapped to an entry in the parent column. The entry in the row can be selected by utilizing a drop-down on the row which lists the applicable attributes for that entity. Rows can be removed by setting the drop-down to blank and can be re-ordered using the 'Move up' and 'Move down' buttons on the right of the table.

2.7.2 Secure Field Tab

The Secure Field tab offers a tab to allow for the definition and maintenance of a Facade class operation's Secure Fields. This tab is visible on the Properties View of a Facade owned operation.

The tab contains two columns; field name and Security Identifier (SID) Name. The field name entries are computed from walking the available fields for the return type of the operation. SID Names can be entered, edited or deleted from the right column as required. This table serves as a widget to edit the Secure_Fields property on the Curam tab.

2.7.3 Manage Operation Parameters Wizard

The Manage Operation Parameters Wizard is to create and maintain the parameters and return type of an operation. The Parameters and Return Type frames will be visible where the operation allows addition of such.

The Parameters Frame offers tabular listing of the parameters where parameters can be added, deleted or re-ordering using the buttons to the right of the table. The table offers direct in-place editing for the name, type and Shadow Type of the parameter. For more information on Shadow Types see Section A.2, *Shadow Classes*.

The Return Type Frame offers the ability to select the type and manipulate the Shadow Type of the return value.

2.7.4 Operation Wizard

The Operation wizard is to create operations. The parameters and return type frames will be visible where the operation allows addition of such.

The wizard utilizes the same layout and functionality as the Manage Operation Parameters Wizard, with additionally providing a field to enter the name of the operation.

2.7.5 Entity Operation Wizard

The Entity Operation Wizard is to create standard and non-standard database operations where the input/output structures can be determined from the entity. The wizard contains a list of input and/or output attributes which is used to specify the attributes that form a struct class which is generated by the wizard.

The generation firstly checks whether a struct exists in the same package with the same attributes and prompts whether to use this struct or generate a new struct to promote re-use of existing structs.

The naming pattern for this generated struct class is:

```
<Entity Name><Key (Input)/Dtls (Output)>Struct<Unique Number>
```

e.g. PersonKeyStruct1

2.7.6 Domain Definition Wizard

The Domain Definition Wizard is used when creating a Domain Definition and offers a simple process for creating a Domain Definition class.

The wizard allows the ability to set the name of the Domain Definition and browse for the type.

Optionally, the Max Size field will be editable when a Domain is chosen that is based on a SVR_STRING or SVR_BLOB to allow for the regular size value to be set.

2.8 Working with Class Diagrams

To create a class diagram:

1. Right-click on the package in which you want to create a class diagram in the Project Explorer.
2. Select Add Diagram and then Class Diagram from the right-click context menu. The new diagram is then created and opened in the Diagram Editor.
3. Elements can then be dragged from the Project Explorer onto the Diagram Editor.

Modeling elements can be added to the model in the Diagram Editor using the Model Palette and Cúram drawer. For more information on using the Diagram Editor, see Section 2.3.3, *The Diagram Editor*.

2.9 Working with Fragments

The Cúram model is a collection of elements that are logically related but physically separated.

When you open a model that contains fragments, the fragments do not load automatically, they load when you open them or when you access functionality that requires artifacts from the fragments. When you load a fragment, the parent resource is also loaded.

The sections below describe how to create a fragment and absorb a fragment into the parent model.

2.9.1 Creating a Fragment

To create a fragment:

1. Right-click on the package that you wish to create a controlled fragment from.
2. Select Refactor, and then Create Fragment.
3. You will be prompted to name the fragment and select the location where you wish to save the fragment.
4. Once the fragment is saved, another dialog appears in which you must ensure that you have the 'Update references to elements in new fragment' option set. If not set you risk breaking references to child elements contained in this fragment.

Once the fragment has been created, the icon for the package changes to signify that it is a controlled fragment.

2.9.2 Absorbing a Fragment

Occasionally you may want to remove a fragment by absorbing it back into its containing fragment or model. To do this:

1. Right-click on the fragmented package and selecting Refactor.
2. Choose the Absorb Fragment option.
3. When absorbing a fragment, you must ensure that a tick is placed in the 'Update references to elements in the fragment' box. This ensures that existing references in the fragment are not broken in the process of absorbing the fragment.

It is also possible to absorb all the fragments in a model at the same time. To do this:

1. Right-click on a model and select Refactor.
2. Select the Absorb All Sub-Fragments option. All the fragments in the model will then be absorbed. You should also ensure that you update element references when absorbing all the fragments in a model.

2.10 Validating a Model

Rational Software Architect offers the ability to validate a model. A model can be validated by right-clicking on the model and selecting validate.

The validation reports a summary in the console panel and describes any warnings or errors found in the Problems View. The problem description should indicate the issue and link to the location found in the model.

Chapter 3

Using Rational Software Architect with the Cúram Model

3.1 Introduction

This chapter also provides detailed instructions on how to model *IBM Cúram Social Program Management* elements in *Rational Software Architect*.

The sections below describe typical development tasks with some of the example model element types that make up the application. Each section takes a model element type and describes how it is used in the model. These element types include:

- Domain Definitions
- Entities
- Structs
- Aggregations
- Processes
- Facades

3.2 Working with Domain Definitions

The datatypes of attributes in *IBM Cúram Social Program Management* are modeled as Domains. Domains are defined in terms of a fundamental datatype such as a string or an integer or in terms of another already existing application domain. Domains have application-specific type names such as SOCIAL_SECURITY_NUMBER, PAYMENT_AMOUNT, etc. Domains can have associated validations defined for them such as uppercase, range

checks, code tables, pattern matches, or custom validations.

3.2.1 Creating a Domain Definition

New domain definitions can be added to the model using the right-click context menu in the Project Explorer. With *Rational Software Architect* you are not restricted in terms of the package structure. Domains can be added to any existing named package or combined with other elements in the same package. For consistency care should be taken to preserve standard structure. This allows them to be easily managed and re-used across the application.

You can create a domain definition using the following steps:

1. In the Project Explorer, navigate to the package where you want to create the new domain definition.
2. Right-click on the package and navigate to the Add Class Menu and select Domain Definition.
3. In the Create Domain Definition Wizard, enter the name of the domain and select a domain definition type. If the type you select is SVR_STRING you must also specify the maximum size.

   

   Note

   When searching for the base Domain Types e.g. SVR_STRING, you will have to modify the Search Scope and select the `Search non-imported UML libraries`. The base types exist in a plugin delivered with the *SDEJ* and can only be searched for and cannot be browsed to.
4. Choose the domain type. This can be done in two ways: through the type browser or by searching the model. Once you have selected the domain type, click Finish.

3.2.2 Renaming a Domain Definition

You can rename a domain definition in one of two ways:

1. Right-click on the domain definition in the Project Explorer and rename it.
2. Select the domain definition in the Project Explorer and then edit it in the General tab of the Properties view.

Important

When you rename a domain definition, you must also rename its single attribute to the same name. This can be done in the Attributes tab in the Properties view for the domain definition.

Unlike *Rational Rose*, the process of renaming a Domain Definition

will maintain any references to that Domain.

3.2.3 Modifying a Domain Definition

A domain definition contains a single attribute whose type represents the domain it inherits from. To modify a domain definition do the following:

1. Navigate to the Attributes tab in the Properties view and double-click the Type cell for the single attribute.
2. Search for and select the domain type in the Type Browser.

3.3 Working with Entities

Entities are objects which represent the persistent storage of the application. They have attributes which are defined as domains. They can have primary keys and index and foreign key relationships.

Create, read, update, and delete style operations are defined on entities as stereotyped methods. The signatures of these operations are implied by the stereotype. Other operations can be defined on entities by defining their signatures in the model. Operations requiring complex database queries can be specified in SQL.

3.3.1 Creating an Entity

To create an entity, select the package where you want to create it and from the right-click context menu, choose Class, then Entity.

As an example, consider the Person entity in the Cúram model. Once it is added to the Person package, the required attributes are created for it. Entity operations are also added which handle the data passing to and from the database tables.

3.3.2 Adding an Attribute to an Entity

Attributes are required in order to store information related to an entity. For example, in the Person entity, the CountryOfBirth attribute is used to store the country of birth for a person. The domain definition for this attribute is COUNTRY_CODE.

An entity will generally have at least one attribute that contains a unique identifier. This is identified by the key attribute. The Person entity contains a key attribute concernRoleID.

To add an attribute to an entity:

1. Select Add Attribute from the right-click context menu for the entity.
2. Select Key or Details as required. This opens the Create Attribute Wizard. Here you can name the attribute and select its type.

3.3.3 Adding an Operation to an Entity

Operations are added to entity classes via the right-click context menu. To add an operation to an entity:

1. Select Add Operation from the right-click context menu and choose the stereotype for the operation you want to create.
2. Accept the default name of the operation which will match the stereotype you selected.

Most of the operation stereotypes do not require you to model the arguments or return types. If the stereotype you choose does require a return type to be modeled it must be a struct. To do this:

1. Select the return type in the wizard using the 'Select Type' button.
2. If the return type you select is an entity, you must also select the Shadow Type from the drop down to identify the actual struct that will be used. If the return type you select is a struct, do not select a Shadow Type.

3.3.4 Adding a Return Type to an Entity Operation

Some of the entity operation stereotypes do not require you to model the return type as it is implied by the stereotype. You can set the return type on an entity operation when you create the operation. If you want to add a return type later or change the return type, do the following:

1. Select the operation in the Project Explorer.
2. Select the General tab in the Properties page and select 'Set return type'.

If the return type you select is an entity, you must also select the Shadow Type. To select the Shadow Type:

1. Open the Cúram tab, select the required Shadow Type value for the Shadow_Type property.
2. If you change the return type on an operation and the new return type does not require a Shadow_Type, make sure that the ShadowType on the Cúram page is set to unspecified.

3.3.5 Adding an 'ns' Operation to an Entity

Complex database operations are modeled as 'ns' type operations. To add this type of an operation to an entity:

1. Right-click on the entity and select Operation.

2. Choose the stereotype of the operation from the list of available stereotypes.
3. You will then be presented with the Create 'ns' Operation Wizard where you can name the operation, it's parameters and select the return type. If the parameter, return type you select is an entity, you must also select a Shadow Type.

To add the SQL for the operation, navigate to the Cúram tab of the Properties view, and edit the SQL property string value.

3.4 Working with Structs

Method arguments and return types on operations and entity classes are modeled as structs. A struct is a value object. Attributes of structs are specified as Domain Definitions. The following sections describe how to create a struct and add attributes to it.

3.4.1 Creating a Struct

To create a struct, do the following:

1. Right-click on the package you wish to create a struct in, and select Struct from the right-click menu option for the package.
2. Provide a name for the struct in the properties view of the General tab.

3.4.2 Adding Attribute to a Struct

Attributes describe the data that is contained in the struct. To add an attribute to a struct, do the following:

1. Select the struct you wish to add an attribute to in the project explorer.
2. Select Add Attribute, and select Default from the right-click context menu.
3. In the the Create Default Attribute Wizard, name the attribute and choose its type from the list of available types.

3.5 Working with Aggregations

Relationships are bonds between classes. A number of different relationship types can be modeled such as aggregation (one class contains another), assignable (attribute values of one class may be copied to the other), foreign key (for referential constraints), index and unique index (to define database indexes on entity classes).

3.5.1 Creating an Aggregate Relationship

An aggregation relationship is used to model a relationship between objects where one object contains another. In *IBM Cúram Social Program Management* this relationship will always be between two structs.

In the Project Explorer:

1. Select the struct which will be the containing struct in the relationship.
2. In the Properties view, select the Relationships tab for the struct. Choose to add a relationship originating from this element.
3. Select the object to be contained, as the target of the relationship and select Aggregation as the relationship type.

If the contained object is an entity you must pick the Shadow Type to identify the actual struct to be contained. This can be done in the Cúram tab of the Properties view.

On the relationships page, right-click the aggregation you have just created and select Navigate from the context menu. This opens the Properties view of the newly created aggregation. Verify that your aggregation is correct by viewing the diagram on the General tab.

In the diagram, the 'diamond' should appear beside the containing struct. In the Cúram tab, type a role name for the contained struct and set the multiplicity of the relationship. The multiplicity of the container struct must be 1. The multiplicity of the contained struct can be 1..* (for a 1 to many relationship) or 1 (for a 1 to 1 relationship).

3.6 Working with Process Classes

Business functions are represented in the Cúram model as methods of process classes. The arguments and return type for methods are modeled as structs or domain types. The model defines the interface for process class methods, but not their implementation. Process classes can call on entity classes to perform database operations as required.

3.6.1 Creating a Business Process Class

To add a business process class to a package, select Add Class, and then Process from the right-click context menu and name the class.

3.6.2 Adding Operations to a Process Class

Operations are added to process classes via the right-click context menu. To add an operation to a process class:

1. Select Operation from the right-click context menu and choose Default.
2. This opens the Create 'default' Operation Wizard where you can name the operation, add it's parameters and select its return type.

3.6.3 Adding an Argument to a Process Operation

Arguments for process operations are defined as structs or domain types. To add an argument to a process operation:

1. Right-click on the process and select Manage Parameters.
2. In the Manage Operation Parameters Wizard, name the parameter and select the parameter type. If the type you select is an entity, you must also select the Shadow Type.

3.6.4 Adding a Return Type to a Process Operation

The return type from a process class operation is a struct or domain type. You can set the return type on an process class operation when you create the operation.

If you want to add a return type later or change the return type:

1. Select the operation in the Project Explorer.
2. In the General tab in the Properties page for the operation, select 'Set return type'.

If the return type you select is an entity, you must also select the Shadow Type. To select the Shadow Type:

1. Open the Cúram tab and the select the required Shadow Type value for the Shadow_Type property.
2. If you change the return type on an operation and the new return type does not require a Shadow Type, make sure that the Shadow_Type on the Cúram page is set to unspecified.

3.7 Working with Facade Classes

Some business process functions are invoked from the client application while others provide utility functions not directly available to the client. A facade class is a process class whose interface is visible to the client.

3.7.1 Creating a Facade Class

To add a facade class to a package, select Add Class, Facade from the right-click context menu and name the class.

3.7.2 Adding Operations to a Facade Class

Operations are added to Facade classes via the right-click context menu. To add an operation to a Facade class:

1. Select Operation from the right-click context menu and choose Default.
2. In the Create 'default' Operation Wizard, name the operation, parameters and select its return type.

3.7.3 Adding Arguments and a Return Type to Facade Operations

Arguments and return types are added to facade operations in the same manner as they are added to process classes. Please refer to Section 3.6.3, *Adding an Argument to a Process Operation* and Section 3.6.4, *Adding a Return Type to a Process Operation* respectively.

Appendix A

How Rational Software Architect differs from Rational Rose

A.1 Introduction

This chapter describes the differences between modeling in *Rational Rose* and *Rational Software Architect* for *IBM Cúram Social Program Management*. Each section details a specific aspect of the modeling process in *Rational Rose* and then describes how it differs in *Rational Software Architect*.

A.2 Shadow Classes

In *Rational Rose*, shadow classes are placeholders created for classes which are not visible inside the model but are produced when the server code is generated. An example of such a class is the standard entity key struct, e.g. the PersonKey struct for the Person entity. In this case the shadow class PersonKey created inside the model is used to represent the future generated class.

In *Rational Software Architect*, server shadow classes are not used. A reference must always be be directed to an existing class in the model. Instead of using shadow classes, a Shadow Type property has been introduced for the following model types:

- Operations - to signify the return type
- Parameters
- Relationships

A.2.1 Specifying a Shadow Type for a Parameter or Operation Return Type

In *Rational Rose*, the Merlin Toolbar listed all possible future generated classes as available types when setting the parameter type or operation return type.

When setting a future generated class in *Rational Software Architect* as a type of a Parameter, you first specify the parameter type as the class from which the future generated class is created. The Shadow_Type stereotype property, found on the Curam tab, can then be set to represent the generated class type.

When setting a future generated class (for example, the standard entity details struct) as an operation return type in *Rational Software Architect* you must do the following:

- Add an operation
- Specify return type as the class from which a future generated class is created
- Set the Shadow_Type stereotype property for the operation as the future generated class type.

A.2.2 Adding an Relationship between a Shadow Class and a Class in the Model

In *Rational Rose*, in order to add an relationship between a future generated class and an existing class or between two future generated classes; shadow stereotyped classes which represent future generated classes needed to be created. The relationships could then be drawn between them.

In *Rational Software Architect*, when adding an relationship between a future generated class and a class in the model, you must add a relationship between the base classes and set the `Left_Class_Shadow_Type` or `Right_Class_Shadow_Type` stereotype property depending on the direction of the relationship as per the Relationships General Tab.

A.3 Server Components

In *Rational Rose*, Server component classes are used to signify client visibility of process classes outside of the model. This relationship was stored in the `Curam.mdl` and `.cat` file containing the process class.

In *Rational Software Architect*, while moving to a multi-model solution it was necessary to remove elements that bound the model into a single model. Instead of assigning <<process>> classes to server components the classes must have a particular stereotype applied.

A.4 Modeling Facade Classes

In *Rational Rose*, all process classes assigned to a Server Component with a

stereotype of <<ejb>> become client-visible classes for the application. Adding a class to a server component with this stereotype also makes it visible to the webclient.

To add a client-visible class for the application in *Rational Software Architect*, a Facade class should be chosen.

A.5 Generating Function Identifiers for Model Classes

In *Rational Rose*, all classes assigned to a component with a blank stereotype result in the generation of Function Identifiers for that class. No EJB or webservice components are generated.

To add this type of class in *Rational Software Architect*, a Process class is created with the value of the stereotype property Generate_Fids set to True.

A.6 Modeling Web Service Classes

For *Apache Axis2* web services:

- *Axis2* web services do not exist in previous versions, so there is no analogue in *Rational Rose*.

For *Apache Axis* 1.4 web services:

- In *Rational Rose*, all classes assigned to a component class with a stereotype of webservice are also visible to the client.

  In *Rational Software Architect*, to add a web service class for the application, a Web Service class should be chosen.

A.7 Assignable Relationship Field Mappings

Assignable relationship field mappings are used, for example, in an explicit field assignment where fields with different names are matched. In *Rational Rose*, assignable field mappings are created by adding keys/qualifiers to one of the Association roles.

In *Rational Software Architect*, assignable field mappings are maintained on the General tab of the assignable relationship's properties. The mapping is maintained by defining the fields involved in the Role option of each class. Additional fields can by specifying using a comma separated entry.

A.8 Class Abstract Options

The Abstract option specifies that the class is abstract. In *Rational Rose*, the Abstract option is available along with the options for entity or process classes.

In *Rational Software Architect*, this option is not listed along with the other

IBM Cúram Social Program Management-specific stereotype properties in the Properties tab. Instead, the standard *Rational Software Architect* abstract option is used for this. In order to specify that the class is abstract you need to place a check in the 'Abstract' checkbox which can be found in the General tab.

A.9 RDO Description Stereotype

Child attributes of RDO and ListRDO classes are handled a bit differently: In *Rational Rose*, RDO and ListRDO classes used to have two stereotypes of attributes: <<dataitem>> and <<description>>. The description stereotype was used to identify which attribute should be used as the description for that RDO/ListRDO.

In *Rational Software Architect*, the description stereotype is no longer used for these attributes; instead the dataitem stereotype has a boolean `description` property to indicate that it is the description for the RDO/ListRDO. As with the `description` stereotype in *Rational Rose*, only one child attribute of the RDO/ListRDO should have its `description` indicator set to true.

Appendix B

Right Click Context Menu Options in the Project Explorer View

B.1 Introduction

This appendix describes what can be added from the right-click context menu for each model class in the *Rational Software Architect* project explorer window.

B.2 Child Options for Class Types

The table below describes the specific attributes and operations that are available to be added to each class from the right-click context menu in the project explorer.

Class	Available Attributes	Available Operations
audit_mappings	audit_mappings	n/a
facade	n/a	default, wmdpactivity, qconnector, batch
webservice	n/a	default, wmdpactivity, qconnector, batch
wsinbound	n/a	default, wmdpactivity, qconnector, batch
process	n/a	default, wmdpactivity, qconnector, batch
struct	default	n/a
entity	key, details	batchinsert, batchmodify, insert, modify, nkmodify, nkread, nkreadmulti, nkremove, ns, nsinsert, nsmodi-

Class	Available Attributes	Available Operations
		fy, nsmulti, nsread, nsread-multi, nsremove, read, read-multi, remove, default
rdo	dataitem	n/a
listrdo	dataitem	n/a
loader	n/a	n/a
do-main_definition	n/a	n/a
extension	default, dataitem, key, de-tails	batchinsert, batchmodify, insert, modify, nkmodify, nkread, nkreadmulti, nkre-move, ns, nsinsert, nsmodi-fy, nsmulti, nsread, nsread-multi, nsremove, read, read-multi, remove, default, wm-dpactivity, qconnector, batch

Table B.1 Right Click Context Menu Options for Classes

B.3 Other Options

Option	Applicable Parent	Applicable Children
Package	Package, Model	Any class type
Manage Parameters	Any applicable operation	n/a

Table B.2 Additional Right-Click Context Menu Options

Appendix C

Broken Reference Resolution

C.1 Broken Reference Resolution

C.1.1 Background

A resource reference is where an element refers to another element either in the same file or another model/file. An example of this is the type of an attribute/parameter or relationship e.g. association/index.

In *Rational Rose* references were resolved through a two stage look-up process;

1. Firstly the qualified name of the reference was used to find the element;
2. Where the element could not determined by qualified name, the id of the element was used.

If the element still could not be found the element reference was considered as broken and needed to be manually resolved by the model owner.

C.1.2 Rational Software Architect changes in References

In *Rational Software Architect* only the id of the element is used to resolve a reference which has led to the possibility that there will be more instances of broken references requiring manual intervention.

This possibility is due to support of previous product versions. When an element is created in a model it gets a unique id (adding an element across multiple product lines, which is sometimes the case required to introduce a new feature into the product) can subsequently introduce multiple unique IDs for the same added element. If a customer refers to this added element in their model and then later jumps product stream, the reference will then be broken from the customer's model to new Cúram model.

A broken reference can be reported during two phases:

1. Opening your model in *Rational Software Architect*, here the *IBM® Rational®* automated resource reference resolution process will be invoked but may be unable to find a resolution and will report any failures in the Problems View;
2. Extracting the model using the command line build tooling, here errors will be reported in relation to the type of a attribute, parameter or relationship not being found.

C.1.3 Extension to Broken Reference Resolution

To account for this possibility of references being broken during a product upgrade and avoid the requirement for manual intervention, an extension to the *Rational*-provided resource resolution process is provided. This extension requires a map of the previous model's IDs is extracted and then is used to resolve references in the current model. This map is processed to look up the broken ID and determine the qualified name of what it was previously referring to and from this resolve the breakage through discovery of the id in the new model for the qualified name found.

As the map needs to be extracted from the previous model an export option has been introduced into *Rational Software Architect* which should be run against the previous model and it should be called as follows:

1. Navigate to `File > Export > Curam > Qualified Name Map`
2. Select the project to export e.g. EJBServer.
3. `Browse` to a location to save the file.
4. Clicking `Finish` will invoke the export process.

The output of this task is a model map that needs to be referenced when opening a new upgraded model.

To reference the map a Preference page is used within *Rational Software Architect* as follows:

1. Navigate to `Window > Preferences > Curam > Qualified Name Map`
2. Browse to the map created earlier.

C.1.4 Resource Reference Resolution Process

If an error is found indicating a broken reference. The model containing the broken reference should be opened and a dialog will pop-up indicating a broken reference.

The repair process should then resolve and correct the reference.

If the process fails and the reference remains broken it will become an error in the Problems view. Here there is a right-click option offering an additional `Search or browse for a valid reference` which can be used as a last resort.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law.

IBM Japan Ltd.

1623-14, Shimotsuruma, Yamato-shi

Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typograph-

ical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources.

IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products

should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This publication documents intended programming interfaces that allow the customer to write programs to obtain the services of IBM Cúram Social Pogram Management.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/us/en/copytrade.shtml.

Apache is a trademark of Apache Software Foundation.

Java and all Java-based trademarks and logos are registered trademarks of Oracle and/or its affiliates.

Other names may be trademarks of their respective owners. Other company, product, and service names may be trademarks or service marks of others.