Project IST-004033

ASSERT

Automated proof based System and Software Engineering for Real-Time Applications

Instrument:	**IST [FP6-2003-IST-2 4.3.2.5]**
Thematic Priority:	**Embedded Systems**
Deliverable	**D4.2.1-3 Software Design Tool Prototype (Final Version)**
Work Package:	**WP4.2**
Due date of deliverable:	M27
Actual submission date:	16/November/2007
Start date of Project:	September 5Th 2004 Duration: 3 years
Organisation name of lead contractor for this deliverable	INTECS
Issue- Revision	I1R1

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

SOFTWARE DESIGN TOOL PROTOTYPE (FINAL VERSION)

Distribution:	DVT Cluster
Prepared by:	Stefano Puri (INTECS), Thanaële Preuss (INTECS), Marco Trevisan (UPD), Sue Maurizio(UPD), Matteo Bordin (UPD), Marco Panunzio (UPD)
Checked by:	Silvia Mazzini (INTECS)
Release type:	Internal Public Release
Status	Passed QPR/IPR

Document Change Record

Issue Revision	Date	Affected Section/Paragraph/Page	Reason for Change/Brief Description of Change
I1R0	05/11/07	All	First issue for IPR
I1R1	14/01/08	Appendix A2, A3.1.1, A8.3, A.4.	IPR comments have been integrated.

Table of Contents

Illustration Index

1. Glossary

- ASN : Abstract Syntax Notation
- ASSERT : Automated proof based System and Software Engineering for Real-Time Applications
- APLC : Application Level Container
- ATL : Atlas Transformation Language
- DVT : Development and Verification Tools (ASSERT cluster)
- Eclipse : www.eclipse.org
- Ecore : meta model defined in the EMF framework
- EMF : Eclipse Modeling Framework
- GMF : Graphical Modeling Framework
- HRT : Hard Real Time
- MDA : Model-Driven Architecture
- MOF : Meta Object Facility
- MOFscript : Eclipse plug-in for model to text transformations
- OCL : Object Constraint Language
- OMG : Object Management Group
- OPCS : Operation Control Structure
- RCM : Ravenscar Computational Model
- RcmCiv : identifies a class diagram in the context of HT-UML2 plug-in
- RcmDeplo : identifies a deployment diagram in the context of HT-UML2 plug-in
- RcmInstance: identifies an instance diagram in the context of HT-UML2 plug-in
- RcmSm : identifies a state-machine diagram in the context of the HT-UML2 plug-in
- UML : Unified Modeling Language
- VM : Virtual Machine
- VMLC : Virtual Machine Level Container

- WCET : Worst-Case Execution Time
- XMI : XML Metadata Interchange

2. Introduction

In the context of the ASSERT project a new methodological approach has been devised that permits to express at model level non-functional properties of interest to a system design and to preserve them throughout the entire development process, from specification to deployment in the execution environment.

The new methodology is described in [CVHN06].

In order to achieve the above described goal, the system model to be produced in the design phase includes the following complementary views:

- the *deployment view* [UEE07] of the system, which specifies the system physical components in terms of nodes (processors, memory, devices), partitions, communication links and protocols and their properties and relations to software components;
- the *data view*, which describes data type components that can be exchanged across system software components;
- the *functional view*, which describes the application layer static and dynamic functional behaviour, in terms of class and statechart diagrams respectively;
- the *interface view* [UEE07]of the system, which specifies the individual software components of the application layer, in terms of application level containers (APLC), embedding the functional view elements. APLCs fully specify the interfaces of the system software component instances, by the addition of hard real-time attributes and the definition of their mutual relationships;
- the *concurrency view* [UEE07], which determines the concurrent architecture of the threads of control that populate the system model and their synchronization points and protocols in a form structurally (i.e. by construction).

The system modeling language is described by the HRT-UML/RCM meta-model [BT07].

The HRT-UML/RCM meta-model, is compliant with the OMG Meta Object Facility meta-model, the standard base language for defining modeling languages in the Eclipse environment and has to be considered as UML2 restriction to adhere to the new ASSERT methodology.

The HRT-UML/RCM toolset, also named HRT-UML2, has been developed as an Eclipse plug-in in order to support graphical editing of the HRT-UML/RCM meta-model, to enforce the HRT-UML/RCM grammar rules as described in [CVHN06] and to integrate feasibility analysis and MDA model transformations, such as the vertical transformation to the VM level (concurrency view) and code generation.

2.1. Purpose of the document

The purpose of this document is to provide a comprehensive guide for the use of the HRT-UML/RCM toolset HRT-UML2. The current version of this guide is related to the toolset version 0.6.x.

2.2. How to read this document

It is assumed that the reader is familiar with the UML2 and the document describing the RCM meta-model [BT07].

Moreover a base knowledge about the Eclipse environment is assumed, in particular about the management and the use of the standard Eclipse views.

In the text the following icons can appear at the start of a paragraph:

ℹ the associated paragraph owns some hint

⛔ the associated paragraph has important information that need to be read carefully

In this document a simple example, concerning the modeling of a controller and actuator case is considered as example and illustrated in the screen shots appearing in the figures.

Moreover the Toy Example is used, that is an RCM model developed in the ASSERT project.

2.3. Acknowledgments

Intecs wish to thank for their contribution and support all the UPD team people involved in the specification of the toolset and the related meta-model. In particular, they wish to thank Tullio Vardanega, Daniela Cancila, Matteo Bordin, Marco Panunzio, Sue Maurizio and Enrico Mezzetti.

Special thanks to Marco Trevisan, Sue Maurizio, Marco Panunzio and Matteo Bordin from UPD for their textual contribution about HRT-UML/RCM patterns which has been integrated in this user manual.

3. HRT-UML2: The HRT-UML/RCM Toolset Eclipse Plug-in

3.1. HRT-UML2 Status

The toolset current version is the preliminary version 0.6.x.

It currently supports full graphical editing of the functional view, interface view and deployment view, the automated transformation into the concurrent view, round-trip analysis and Ada05 code generation.

Functional models can be imported from other functional modeling tools, in particular UML2 tools applying the Framework Profile [CP05] and compliant to the XMI 2.0 standard interchange format. Import of models from Rational Software Modeler by IBM has been verified in the context of the ASSERT project demos.

3.2. The HRT-UML2 Process

The current (possibly iterative) process that is assumed when working with the editor is:

1) definition of the functional model (classifiers and state machines)
2) definition of the class model of the interface view model, referring the functional model
3) definition of the instance model of the interface view and partitions (logical deployment)

The definition of the deployment model, physical and/or logical, can be done independently from steps 1) and 2) while some information, i.e. the logical part, is in common with 3).

3.3. HRT-UML2 Plug-in Environment and Distribution

In this paragraph the details concerning the Eclipse environment and installation are provided.

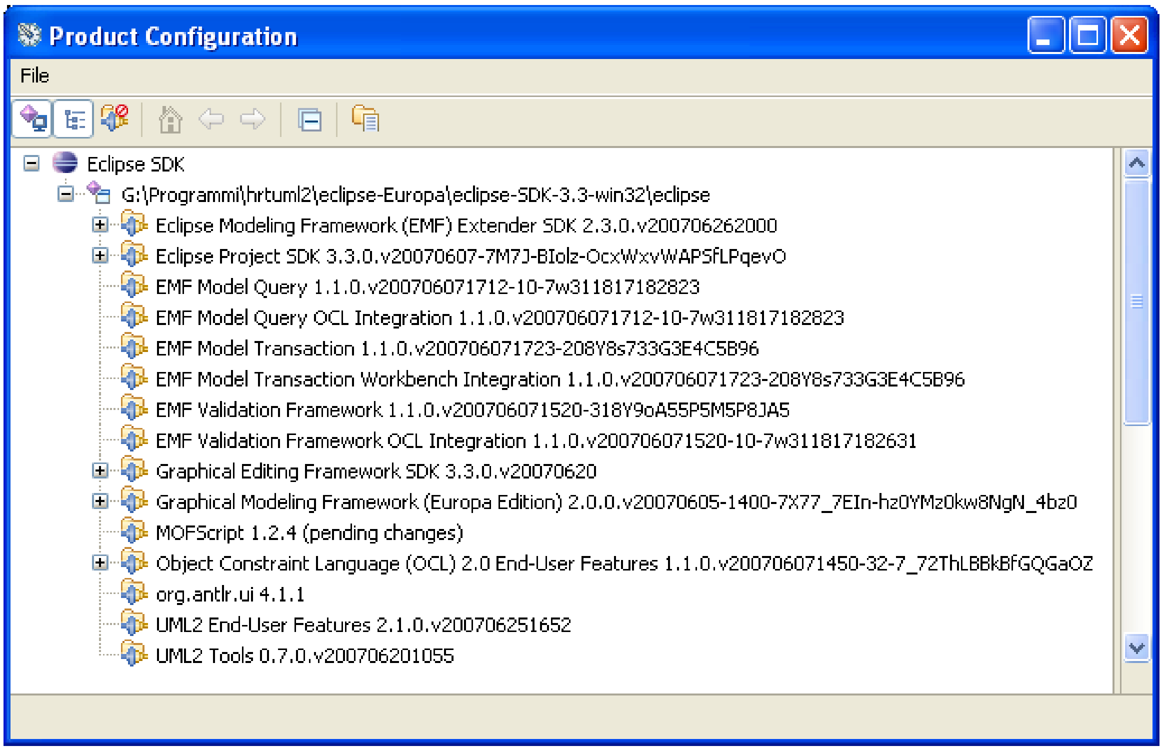
3.3.1 HRT-UML2 requirements

Java1.5 is required to run the round-trip analysis. The following Eclipse environment is recommended:

- Eclipse 3.3 Europa
- GMF 2.0
- ATL, release build February 16, 2007 (http://www.eclipse.org/m2m/atl/download/)
- UML2 2.1.0
- MOFScript 1.2.4 (http://www.eclipse.org/gmt/mofscript/download/)

When a required plug-in is installed manually please refer to its installation instructions to be sure to install also all the needed plug-in.

Figure 1 illustrates how the Eclipse configuration should appear once installed the required plug-in; to open the window select *Help –> Software Updates –> Manage Configuration* from the Eclipse main toolbar.



```
Eclipse SDK
  G:\Programmi\hrtuml2\eclipse-Europa\eclipse-SDK-3.3-win32\eclipse
    Eclipse Modeling Framework (EMF) Extender SDK 2.3.0.v200706262000
    Eclipse Project SDK 3.3.0.v20070607-7M7J-BIolz-OcxWxvWAPSfLPqevO
    EMF Model Query 1.1.0.v200706071712-10-7w311817182823
    EMF Model Query OCL Integration 1.1.0.v200706071712-10-7w311817182823
    EMF Model Transaction 1.1.0.v200706071723-208Y8s733G3E4C5B96
    EMF Model Transaction Workbench Integration 1.1.0.v200706071723-208Y8s733G3E4C5B96
    EMF Validation Framework 1.1.0.v200706071520-318Y9oA55P5M5P8JA5
    EMF Validation Framework OCL Integration 1.1.0.v200706071520-10-7w311817182631
    Graphical Editing Framework SDK 3.3.0.v20070620
    Graphical Modeling Framework (Europa Edition) 2.0.0.v20070605-1400-7X77_7EIn-hz0YMz0kw8NgN_4bz0
    MOFScript 1.2.4 (pending changes)
    Object Constraint Language (OCL) 2.0 End-User Features 1.1.0.v200706071450-32-7_72ThLBBkBfGQGaOZ
    org.antlr.ui 4.1.1
    UML2 End-User Features 2.1.0.v200706251652
    UML2 Tools 0.7.0.v200706201055
```

Figure 1: Eclipse plug-in configuration

3.3.2 HRT-UML2 distribution and installation

The HRT-UML2 plug-in is composed by the the following plug-in's:

- hrtuml2_<version>
- hrtuml2.edit_<version>
- hrtuml2.editor_<version>
- hrtuml2.diagram.piv_<version>
- hrtuml2.diagram.civ_<version>
- hrtuml2.diagram.sm_<version>
- hrtuml2.diagram.instance_<version>
- hrtuml2.diagram.deploy_<version>
- hrtuml2.transformation_<version>
- hrtuml2.validation_<version>
- ch-ethz.fwprofile.RCMvalidation_<version>

hrtuml2, hrtuml2.edit and hrtuml2.editor implement the RCM *Ecore* (meta)model end the corresponding EMF tree editor.

hrtuml2.diagram.civ, hrtuml2.diagram.piv, hrtuml2.diagram.sm, hrtuml2.diagram.instance and hrtuml2.diagram.deploy implement the diagram editor.

hrtuml2.transformation implements the model transformations.

hrtuml2.validation implements RCM model validation.

ch-ethz.fwprofile.RCMvalidation implements the Fw Action Language parser. This plug.in was originally developed by ETH to validate UML models; it has been adapted to validate state machines actions/guards defined in RCM models.

To install the HRT-UML2 plug-in copy these elementary plug-in's into the *plugins* directory of the Eclipse installation.

Moreover the following partial installations of the HRT-UML2 plug-in are still possible:

1. hrtuml2 + hrtuml2.edit + hrtuml2.editor + hrtuml2.transformation + hrtuml2.validation: this allows to work with the RCM metamodel (i.e. to manage RCM models), with the EFM tree editor and with the model transformations.

2. hrtuml2.diagram.piv + hrtuml2.diagram.civ + hrtuml2.diagram.sm + ch-ethz.fwprofile.R-CMvalidation + hrtuml2.diagram.instance + hrtuml2.diagram.deploy + 1. : this allows to use diagrams editor for RCM models.

In order to launch the transformations (i.e. round-trip analysis and Ada code generation) the Ecore metamodel files *rcm.ecore* and *mast.ecore* have to be copied in the MOFscript plug-in repository folder located at:

plugins/org.sintef.mofscript.editor_x.y.z/repository/metamodels

The rcm.ecore and mast.ecore files are available in the hrtuml2.transformation plugin in the following path:

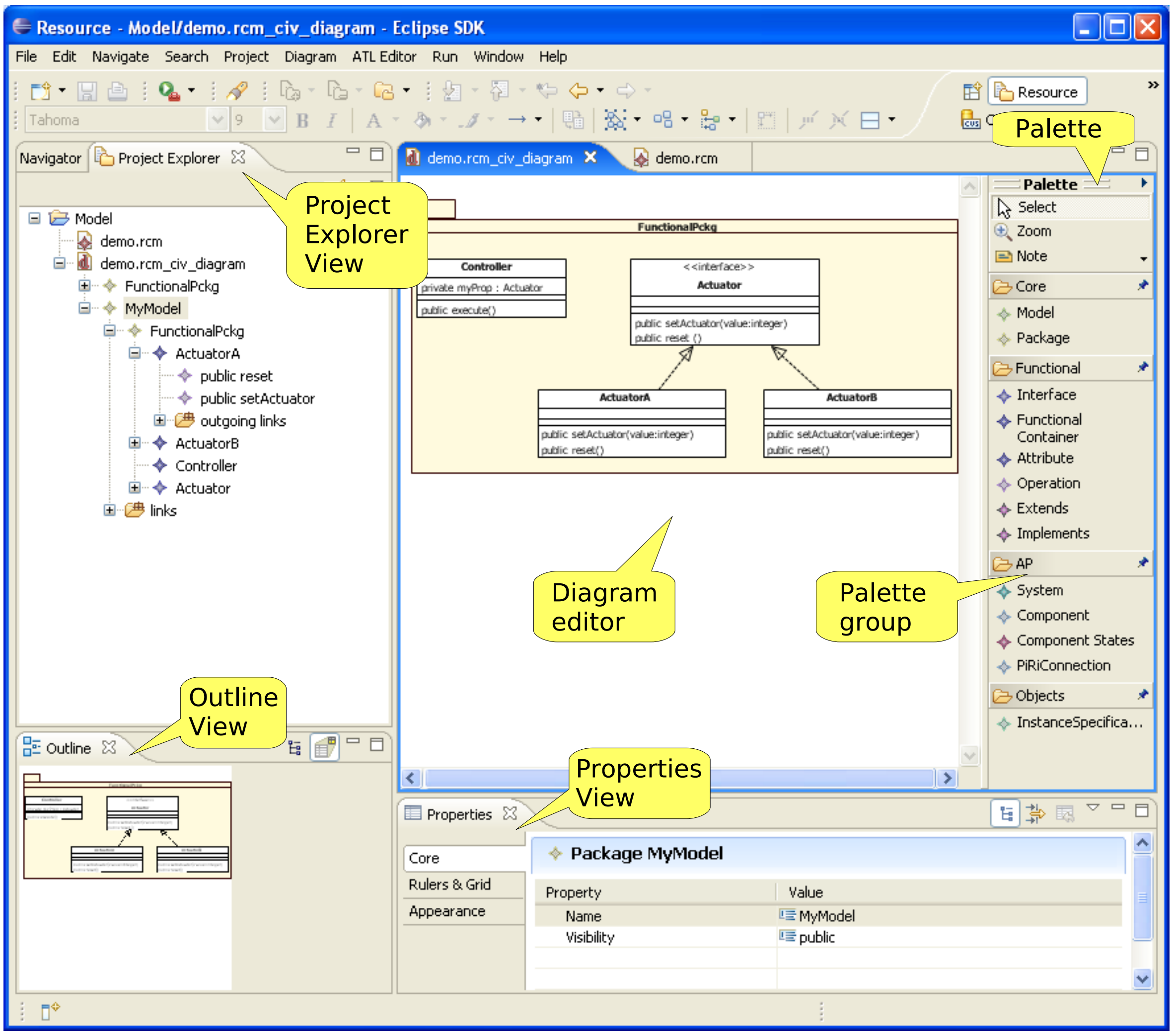
hrtuml2.transformation/metamodels

Also in order to execute the Round-trip analysis the MAST+ tool for the target operating system has to be copied in the following folder.

hrtuml2.transformation/MAST+

3.4. The HRT-UML2 workbench

The HRT-UML/RCM toolset is integrated in the Eclipse platform. Figure 2 illustrates the main editor components, as listed in the following:



Property	Value
Name	MyModel
Visibility	public

Figure 2: HRT-UML2 Eclipse plug-in workbench

- **Project Explorer View**: it shows the contents of the workspace and the HRT-UML2 diagram file in a tree style.
- **Outline View**: when the focus is on a diagram editor, it shows a bird-eye view of the diagram itself
- **Palette**: it is associated to a diagram editor. It allows tool selection and performing of corresponding actions on the associated diagram. Tools in the Palette are grouped according to their semantics.
- **Properties View**: it shows the available properties and associated current values in relation with the entity focused in the diagram editor. Depending on the property, you can edit the associated value.

3.5. Diagrams as resources

The HRT-UML/RCM editor implements diagram editors on top of the RCM meta-model. Currently the following kinds of diagrams are supported:

- UML2 class diagrams,
- UML2 state charts,
- UML2 instance diagrams
- UML2 deployment diagrams.

These different kind of diagrams are identified in the plug-in with different tags, as described in the following:

- class diagram: **RcmCiv** (class-interface view)
- state chart diagram: **RcmSm** (state machine)
- instance diagram: **RcmInstance**
- deployment diagram : **RcmDeploy**

Each diagram has an RCM root entity associated. The association rules are described in the following:

- a RcmCiv diagram may have a Package RCM entity as root
- a RcmSm diagram may have an RCMfunctionalContainer RCM entity as root
- a RcmInstance diagram may have the InstanceSet RCM entity as root
- a RcmDeploy diagram may have the PhysicalArchitecture RCM entity as root.

Each diagram shows the contents of its associated root entity (an HRT-UML2 predefined filter selects visible entities).

A diagram is ***synchronized*** with the related model, i.e. if you modify the model externally, for instance through another diagram or the EFM editor, changes are automatically loaded and the corresponding entities modified in the diagram.

The repository for a HRT-UML2 design model (or project) includes the following files:

1) one or more *.rcm file, representing the *ecore* RCM model
2) zero or many diagrams files: RcmCiv files have *rcm_civ_diagram* as extension, RcmInstance files have rcm_instance_diagram as extension, RcmSm files have *rcm_sm_diagram* while RcmDeploy files have *rcm_deploy_diagram* as extension.

You may create a class or a deployment HRT-UML2 diagram with a new RCM model associated; you may also create a new diagram starting from an existing RCM model, in particular starting from a valid root entity: a Package (RcmCiv), from the InstanceSet (RcmInstance), from an RCM-functionalContainer (RcmSm) or from the PhysicalArchitecture (RcmDeploy). So doing the diagram just created is automatically populated, accordingly to the synchronized nature of the diagram.

To create a diagram with an new empty RCM model associated:

- select the ***File -> New -> Other -> HRT-UML2*** menu from the Eclipse workbench
- select "**RCM Class Diagram**" to create a class diagram or "**RCM Deployment Diagram**" to create a deployment diagram.

To create a diagram starting from an existing model:

- right click the *.rcm model file
- select "*initialize rcm_civ_diagram diagram file*": a Package has to be selected as diagram root element
- "*initialize rcm_instance_diagram diagram file*": the InstanceSet has to be selected as diagram root element
- or "*initialize rcm_sm_diagram diagram file*": a RCMfunctionalContainer has to be selected as diagram root element
- or "*initialize rcm_deploy_diagram diagram file*": the PhysicalArchitecture has to be selected as diagram root element

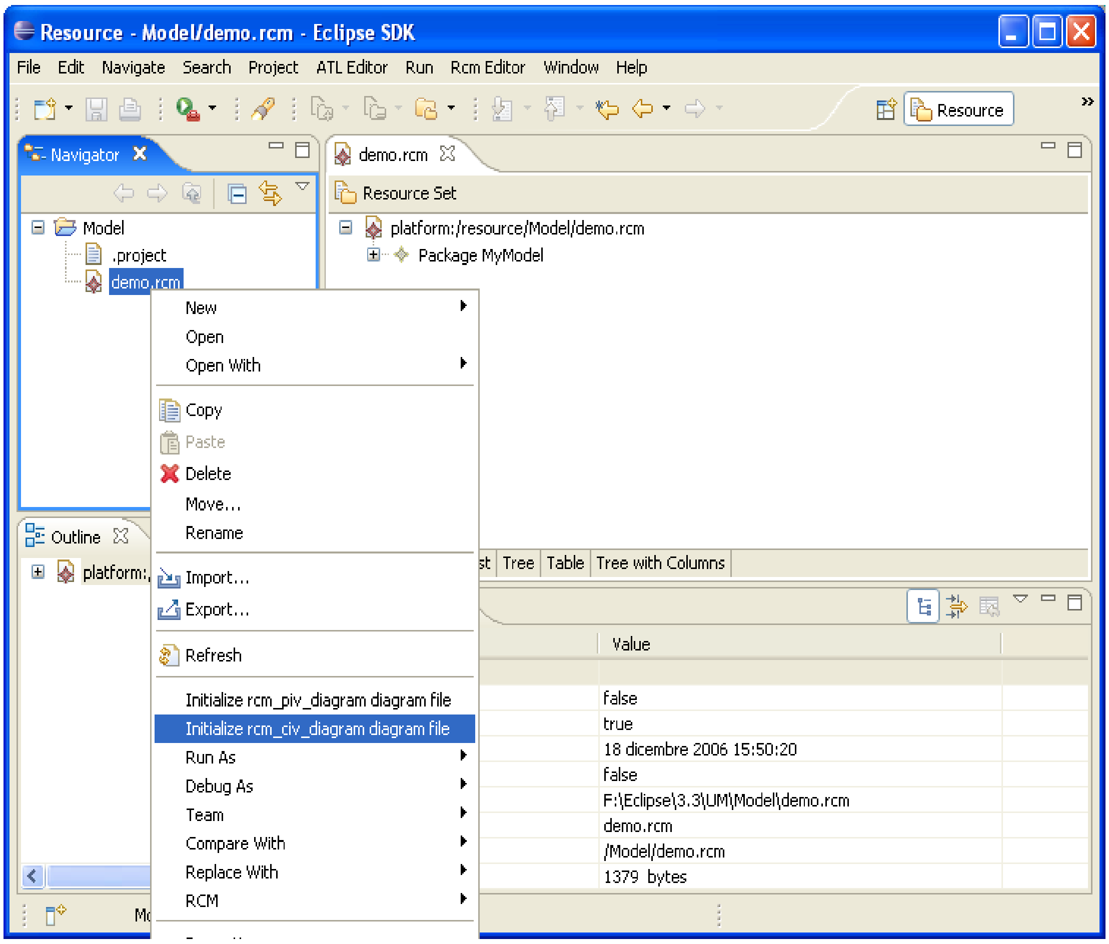
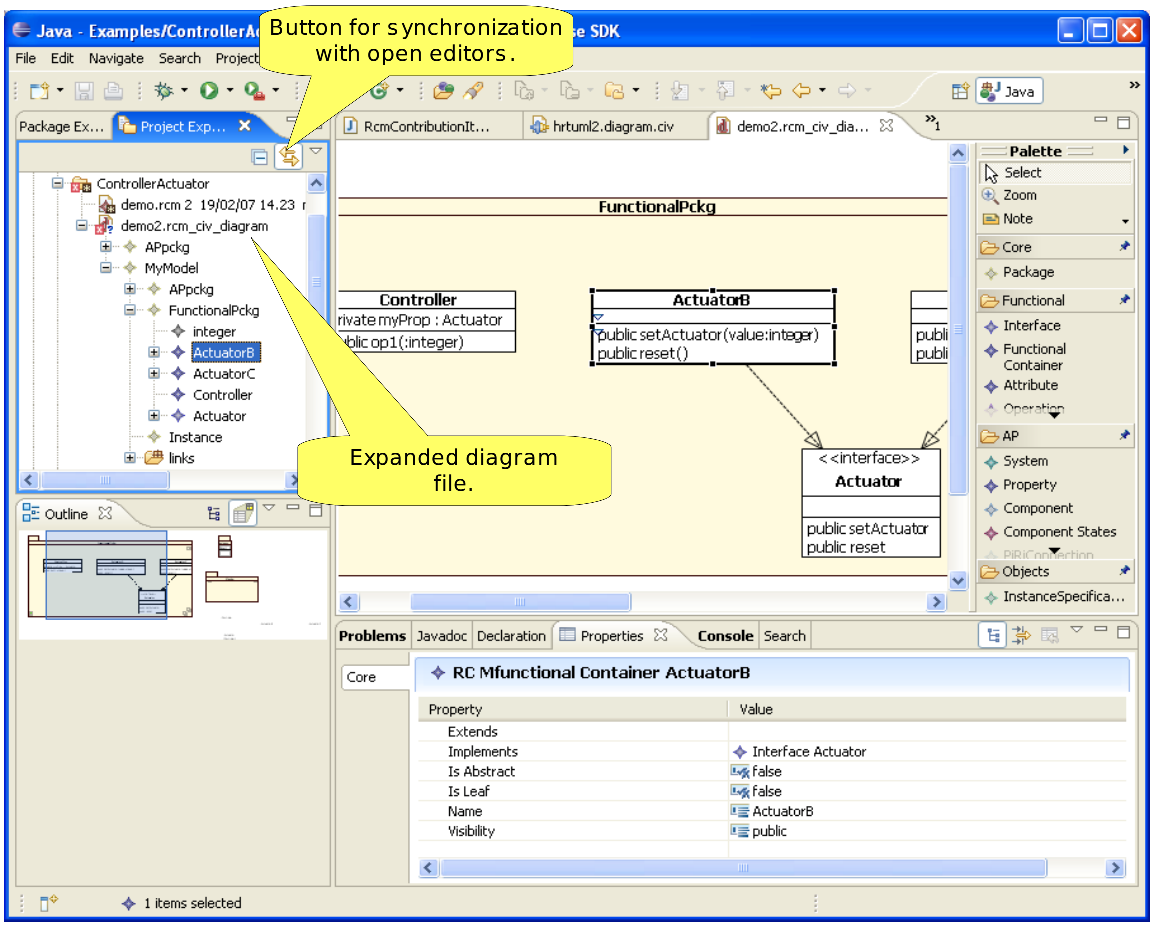

Resource - Model/demo.rcm - Eclipse SDK
File Edit Navigate Search Project ATL Editor Run Rcm Editor Window Help
Resource
Navigator
Model
.project
demo.rcm
demo.rcm
Resource Set
platform:/resource/Model/demo.rcm
Package MyModel
New
Open
Open With
Copy
Paste
Delete
Move...
Rename
Import...
Export...
Refresh
Initialize rcm_piv_diagram diagram file
Initialize rcm_civ_diagram diagram file
Run As
Debug As
Team
Compare With
Replace With
RCM
Outline
platform:
Tree Table Tree with Columns
Value
false
true
18 dicembre 2006 15:50:20
false
F:\Eclipse\3.3\UM\Model\demo.rcm
demo.rcm
/Model/demo.rcm
1379 bytes

Figure 3: initializing a diagram

To open a diagram editor

- double click the diagram file.

You can use the Eclipse Project explorer view to navigate diagram or model file structure and select its owned entities in the open editors, as illustrated in figure 4.



Property	Value
Extends	
Implements	Interface Actuator
Is Abstract	false
Is Leaf	false
Name	ActuatorB
Visibility	public

Figure 4: Navigating the diagram file structure with the Package Explorer view

3.5.1 Printing, Saving a diagram (as an image file)

It is possible to save a diagram or just the current selected figures as an image file: just right click on the diagram area or on the current selected figures and select *File -> Save as Image File...* command.

Also you can print diagrams using the *File->Print...* command.

3.6. Multiple editors

Working with the HRT-UML2 plug in, the following editors can be used for RCM models:

- the EMF editor tree,
- the RcmCiv diagram,

- the RcmInstance diagram,
- the RcmSm diagram,
- the RcmDeploy diagram.

The EMF editor tree is associated to the *.rcm model file.

You can have multiple diagram files, one for each of the different diagrams defined in the same model, attached to the specific root entities.

The HRT-UML2 editor allows also multiple diagram navigation, while having all the diagrams contained just in one diagram file.

To create another diagram associated with a Package or RCMfunctionalContainer, starting from an RcmCiv diagram,

- double click the entity for which the new diagram has to be created.

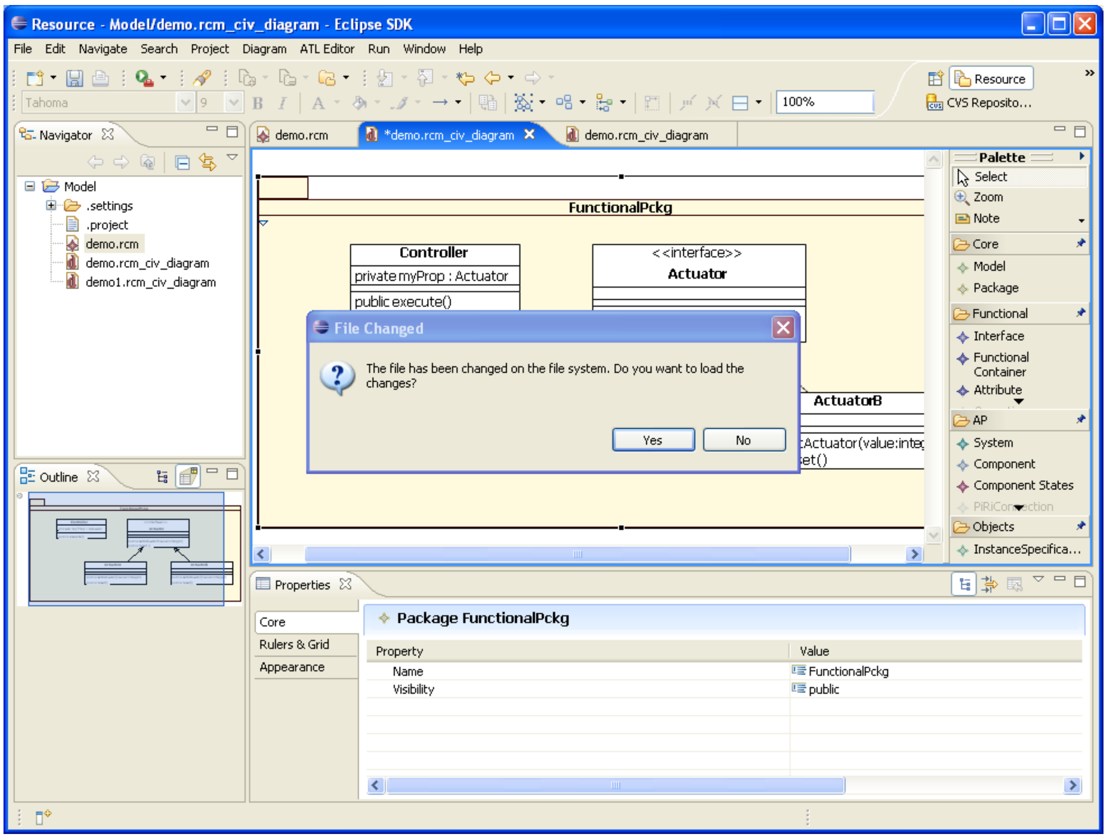
For a Package double click on the top left rectangle of the figure representing the folder, while for an RCMfunctionalContainer double click on the compartment name of the figure.

It is safe to save a diagram before switching to another one.

Different diagrams and the EMF editor tree don't share the same editing domain, so, for instance, in order to see a change made in a diagram file on the EMF editor tree and/or in another diagram, you need to save the change.

Particular attention has to be put while doing modification on different diagrams at the same time while working on the same model; this in order to avoid lost of pending information on a diagram when saving (possible incompatible) modification on a different diagram.

For instance, when using the multiple diagram editor feature on the same diagram file, you may have more than one diagram with pending modifications. In this case, when modification are saved in one diagram, switching to another diagram, a dialog window appears, as illustrated in Figure 5. In this case the answer “Yes” means that **all the local modification will be lost!** You may answer “No” to avoid lost of the local modification and then save the local changes: in this case modification saved before via another diagram will be lost.



File Changed

The file has been changed on the file system. Do you want to load the changes?

Yes No

Package FunctionalPckg

Property	Value
Name	FunctionalPckg
Visibility	public

Figure 5: the dialog window telling about differences between current diagram and the model

3.7. Multiple models

HRT-UML2 plug-in supports working with RCM models stored in different *.rcm* files; in particular from an RCM model *A* it is possible to refer to another model *B* stored in a separated *.rcm* file.

Working with multiple models is not currently supported by the ATL model transformations (i.e. the ones used to perform round-trip analysis and code generation). If you have several models to work with, a possible workaround to this current limitation is to merge their content into a single model: you can easily do it by using the copy and paste feature which is available from the tree editor. For instance you can copy an entire package of one model and then paste it into another one.

From the current RcmCiv diagram you can:

- right click on the diagram background, choose *LoadResource...* command and select the target .rcm model file you want to refer; so doing the elements defined in the target model can be referred from the entities defined in the current model, e.g. through property values.

- right click on the diagram background, choose *Create Shortcut...* command, retrieve the target .rcm model file with the given browser and navigate it to select the element to show in the current diagram.

  Note that the deletion of an imported figure (also called shortcut) doesn't have any impact on the target model, i.e it is only a graphical deletion.

In the property editor, for a label associated to a type, it is possible to have as prefix the name of the resource owning that type (for instance for the type of a property). This can be useful while working with multiple model files, in fact the models could have the same name and it could be difficult from the property editor to identify an element of a particular model. To activate/deactivate this feature:

- select *Window-> Preferences->Run/Debug->String Substitution* from the Eclipse menu and set to true/false the boolean variable named *UseResourceNameForTypePath*.

Appendix A Detail Description On How To Work With The Toolset

A.1 Working With Packages

An RCM model comes with a default top level structure of stereotyped Package to separate and better organize class, instance and deployment modelling. This hierarchy is the following:

- Model: it is the root Package for an RCM model
- Model.Package: it is the root Package for functional and interface class model
- Model.InstanceSet: it is the root Package for instance model
- Model.DeploymentInformation: it is the root Package for deployment model.

During the design of the class model for functional and interface view you are able to create and use Package like in standard UML2 models. For the entities defined in these Packages the visibility rules defined in UML2 apply.

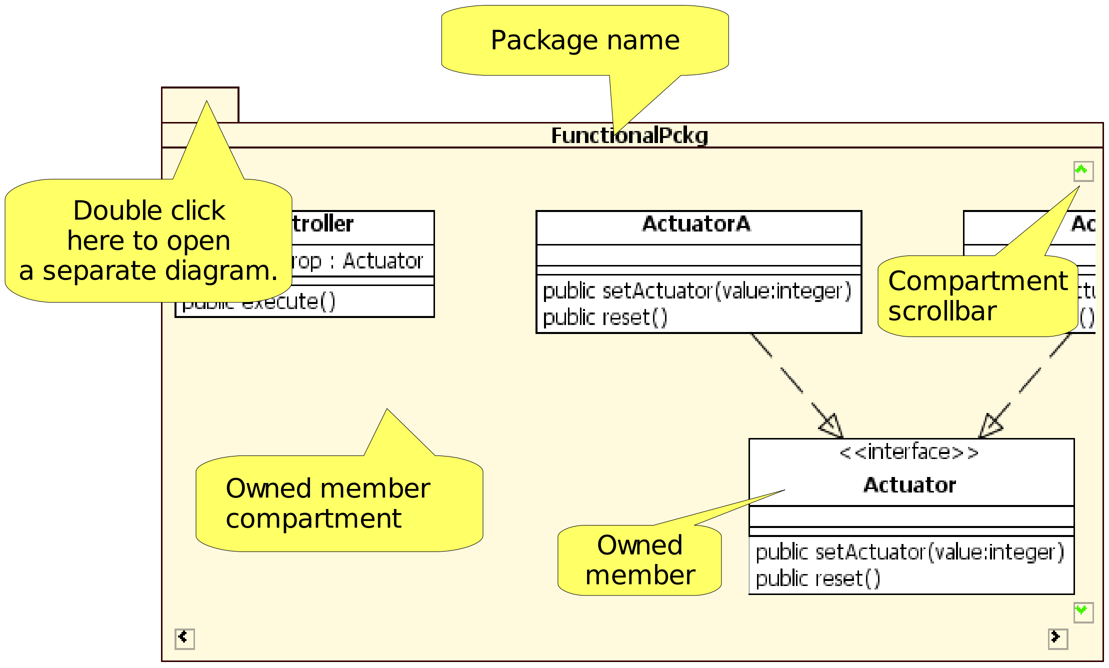


Figure 6: Package expanded presentation

Its presentation in a diagram comes with an owned member compartment, where entities can be directly created and manipulated.

In HRT-UML2 a Package can be the root for an RcmCiv diagram, so you may edit the Package contents in a separate and dedicated diagram.

Through the HRT-UML2 class diagram editor, you may model functional entities and RCM interface entities. You may structure such entities in some package hierarchy or they can be resident on a single root package at the same level.

Compartments for Packages are collapsible, i.e. you may show/hide compartment contents by double clicking the top left part of the compartment itself.

Note that when a compartment is collapsed the outgoing or incoming links are automatically hidden.

For a compartment you can use the ***Arrange All*** command in order to perform an automatic layout of the contained figures: this is particularly useful when a diagram is initialized starting from an existing model.

A.2 Data View Modeling

The Data View is supported by the RCM metamodel and can be designed with the toolset by using the Package and DataType entities available in the functional view diagram (see appendix A.3).

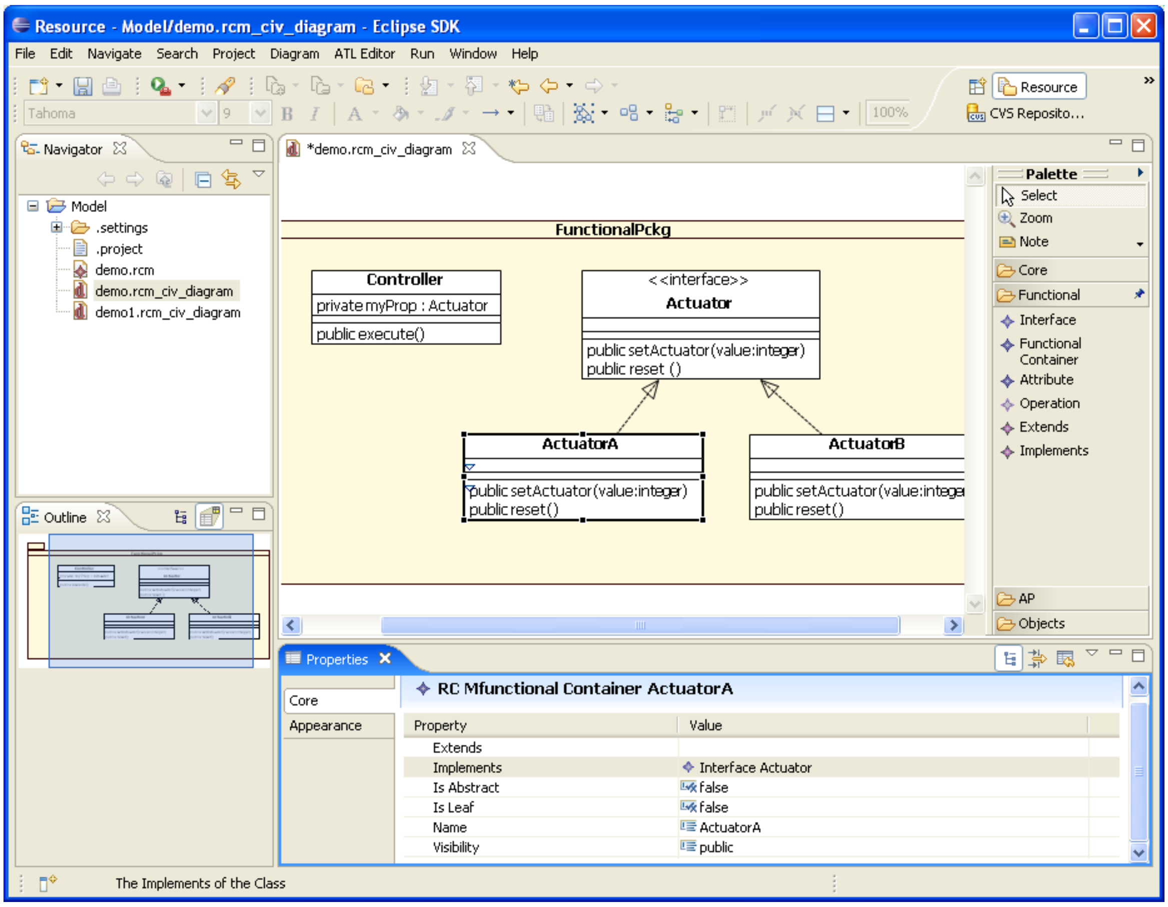
Currently the import from ASN.1 Data View definition into an RCM model can be performed externally with the *asn2uml* tool developed by Semantix.

A.3 Functional Modeling

The HRT-UML2 plug-in allows to create RCM functional entities using an RcmCiv diagram (i.e. the equivalent of an RCM dedicated UML class diagram).

A.3.1 Class Diagram for Functional Modeling

The functional entities can be created on the diagram using the commands available on the *Palette-Functional* command group.



RC Mfunctional Container ActuatorA

Property	Value
Extends	
Implements	Interface Actuator
Is Abstract	false
Is Leaf	false
Name	ActuatorA
Visibility	public

Figure 7: Example of diagram showing functional entities

A.3.1.1 RCMfunctionalContainer (Class)

For the RCMFunctionalContainer the standard UML notation for Class is provided.

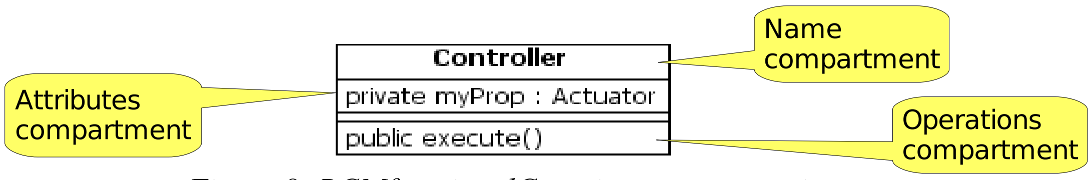


Figure 8: RCMfunctionalContainer representation

Standard UML class properties are available through the property editor associated to the RCM-functionalContainer (e.g. *isAbstract*, *isLeaf* etc). Moreover the RCMfunctionalContainer comes with the enumerated property *Implementation Language*, by default set to *Ada*, which can be used to specify the language (i.e. Lustre, SDL, C++) that will implement the functionalities specified by the current RCMfunctionalContainer (see Appendix A.8.3).

A.3.1.2 Interface

For the RCM Interface the standard UML notation for Interface is provided.

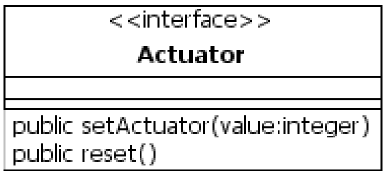
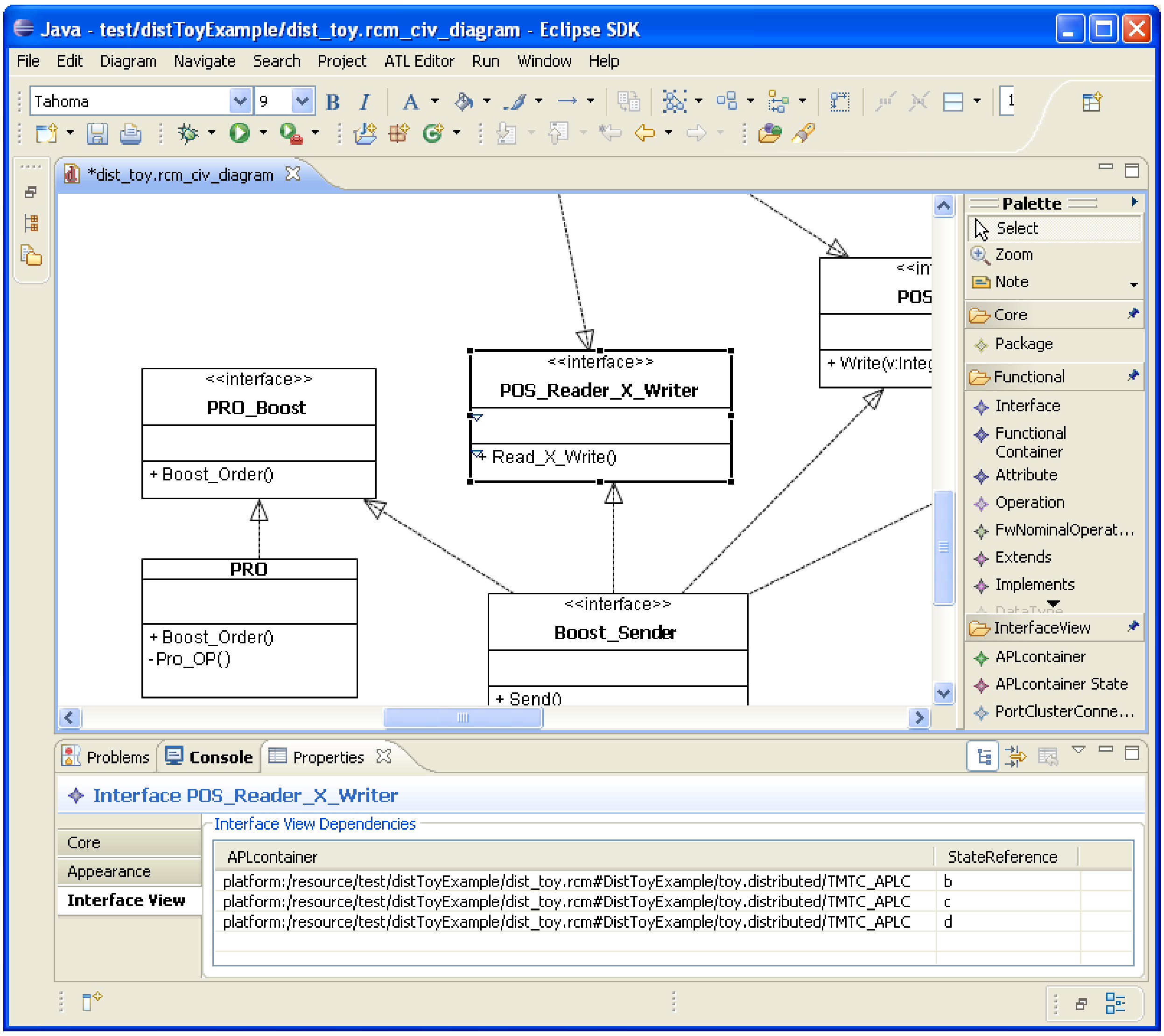


Figure 9: Interface representation

When you select a class or an interface you can see in a dedicated property editor tab all the entities in the interface view that are depending on that functional entity.



Interface POS_Reader_X_Writer

Interface View Dependencies

APLcontainer	StateReference
platform:/resource/test/distToyExample/dist_toy.rcm#DistToyExample/toy.distributed/TMTC_APLC	b
platform:/resource/test/distToyExample/dist_toy.rcm#DistToyExample/toy.distributed/TMTC_APLC	c
platform:/resource/test/distToyExample/dist_toy.rcm#DistToyExample/toy.distributed/TMTC_APLC	d

Figure 10: Interface View Dependencies List

A.3.1.3 DataType, Primitive Type and Enumeration

For the RCM Datatype, PrimitiveType and Enumeration elements the standard UML notation for the analogous elements is provided.

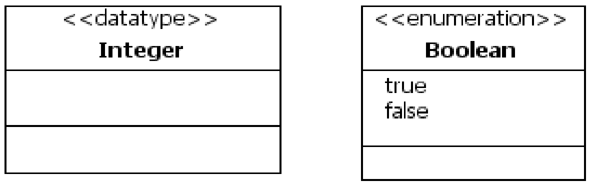


Figure 11: DataType and Enumeration representation

A.3.1.4 Properties

For a Classifier (e.g. RCMfunctionalContainer, Interface) you may create Properties, i.e. typed attributes. The type can be a data type, an enumeration, or another functional Classifier. When assigning the type of a Property the property editor shows the available entities accordingly to the visibility rules.

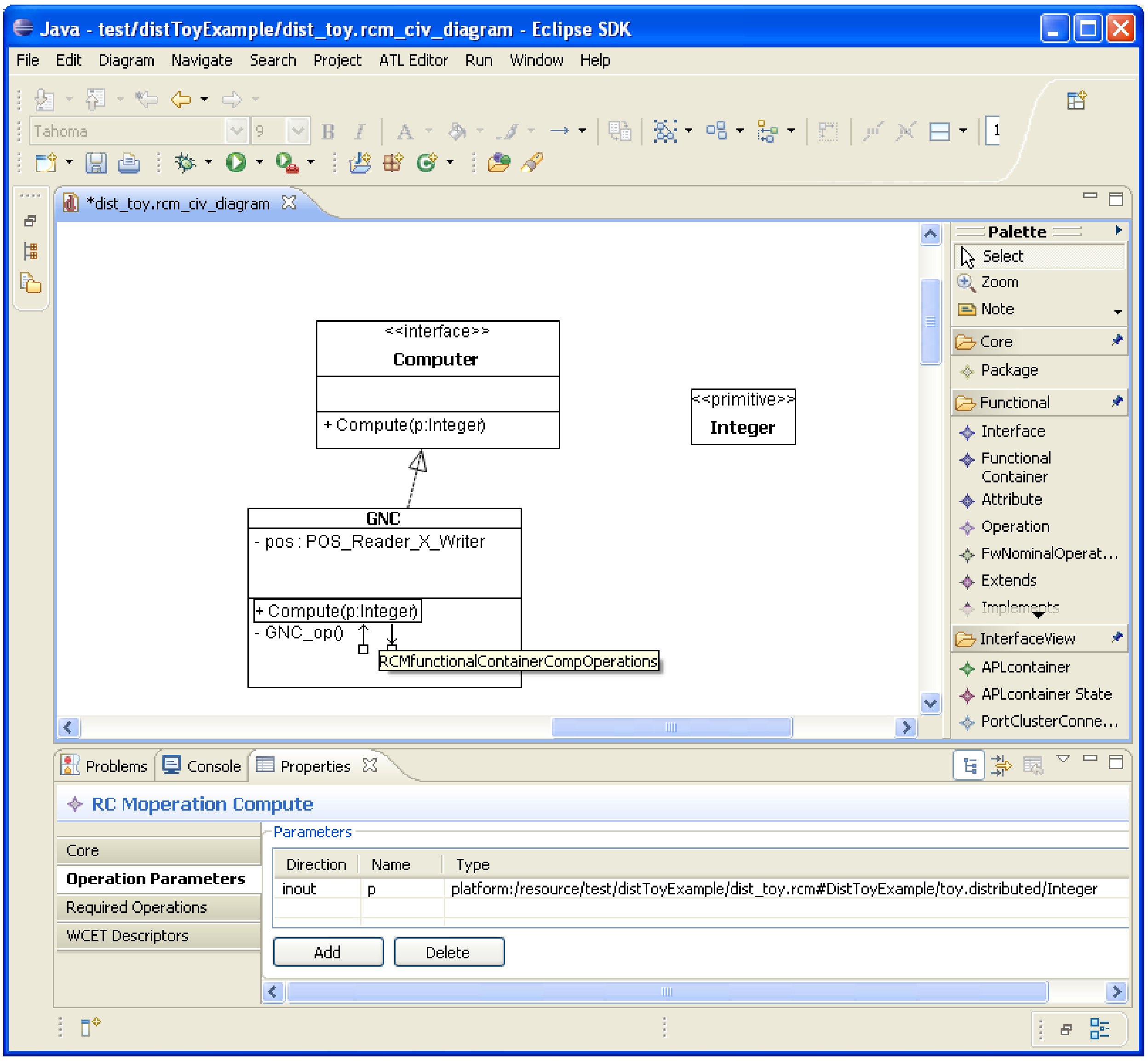
Attributes for a classifier are listed on the middle compartment of the classifier presentation.

Currently the only way to specify UML-like use relations/association between a functional containers and/or interface is to create attributes typed with the target classifier, on the source container.

A.3.1.5 RCMoperation

The list of RCMoperation for a classifier are showed on the bottom compartment of the classifier presentation.

A specialized tab property named “Operation Parameters” has been implemented to edit the operation signature.



RC Moperation Compute

Core
Operation Parameters
Required Operations
WCET Descriptors

Parameters

Direction	Name	Type
inout	p	platform:/resource/test/distToyExample/dist_toy.rcm#DistToyExample/toy.distributed/Integer

Figure 12: Operation Parameters editor

A.3.1.5.1 FwNominalOperation

The tool supports FwNominalOperation creation according to the EthFwProfile [CP05].

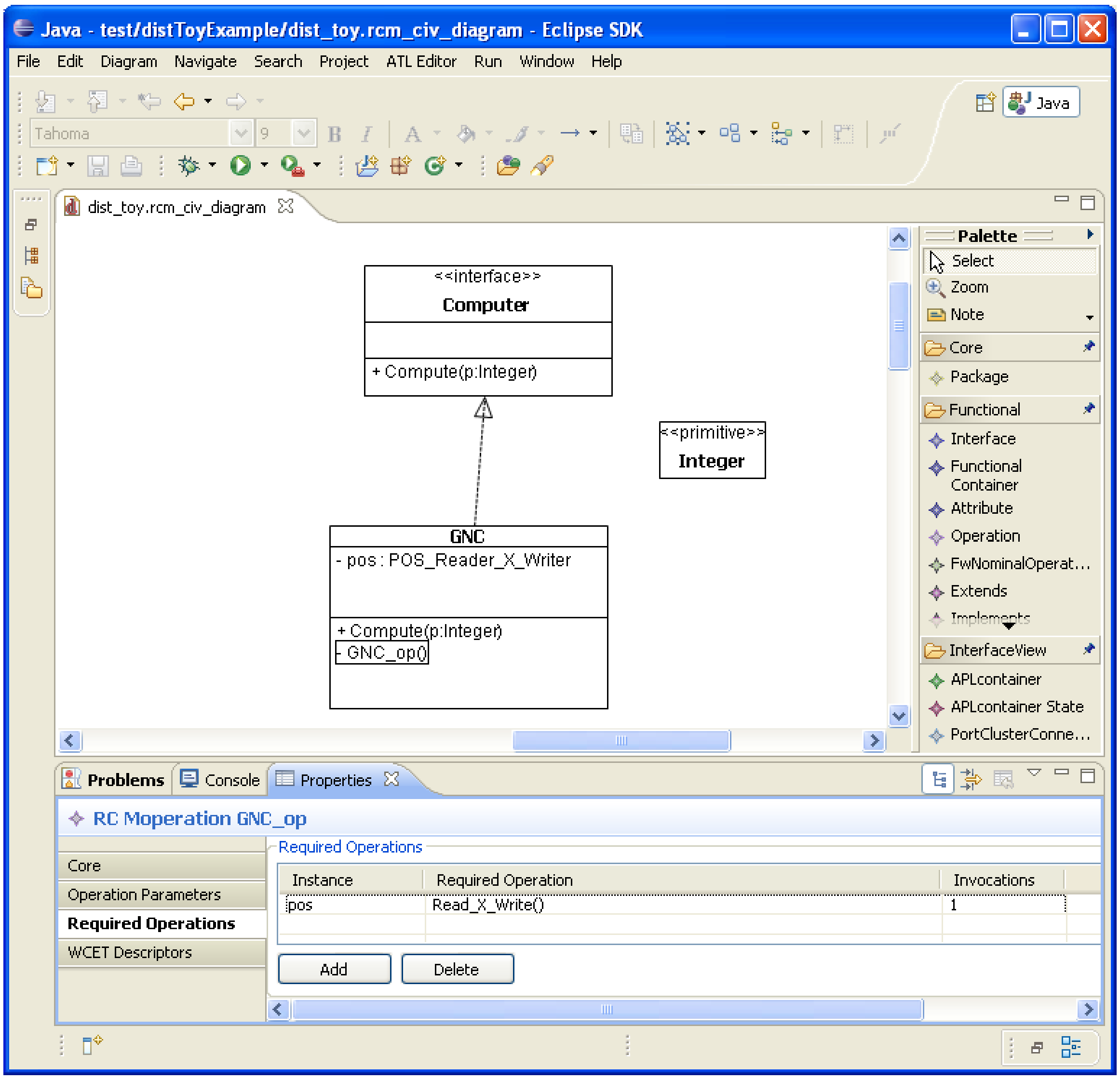
A.3.1.5.2 RequiredOperation

Given an RCMoperation (FwNominalOperation) it is possible to specify which are the required operations it call: this can be done using the RCM RequiredOperation entity defined in the RCM meta-model.

The information about the required operations is necessary because it is used by the tool in the Interface View to build the required elementary ports.

Currently, given an RCMoperation, RequiredOperation relations have to be specified by the user through a dedicated property editor named "Required Operations"; this editor is available in the Properties view when an operation is selected.

Currently it is up to the user to maintain consistency between RequiredOperation's specification and the actions specified in the StateMachine. No automatic creation or checks are currently supported.



RC Moperation GNC_op

Required Operations

Instance	Required Operation	Invocations
pos	Read_X_Write()	1

Figure 13: Required Operation editor

Figure 13 illustrates the Required Operations tab editor: you can click Add or Delete button to create or remove RequiredOperation entity for the current RCMoperation.

For a RequiredOperation entity appearing in the table it is possible to set the following fields:

- *Instance*: represents the target element on which the operation will be invoked. This field is editable through a combo box which shows the attributes currently defined or inherited for the given class. It is possible to use the "*this*" predefined instance to specify self calls.
- *Required Operation*: it is the called operation. This field is editable through a combo box that shows only the operations which are invocable for the selected instance.
- *Invocations*: it is an integer field which specify the maximum number of invocations of the target operation during an invocation of the current RCMoperation.

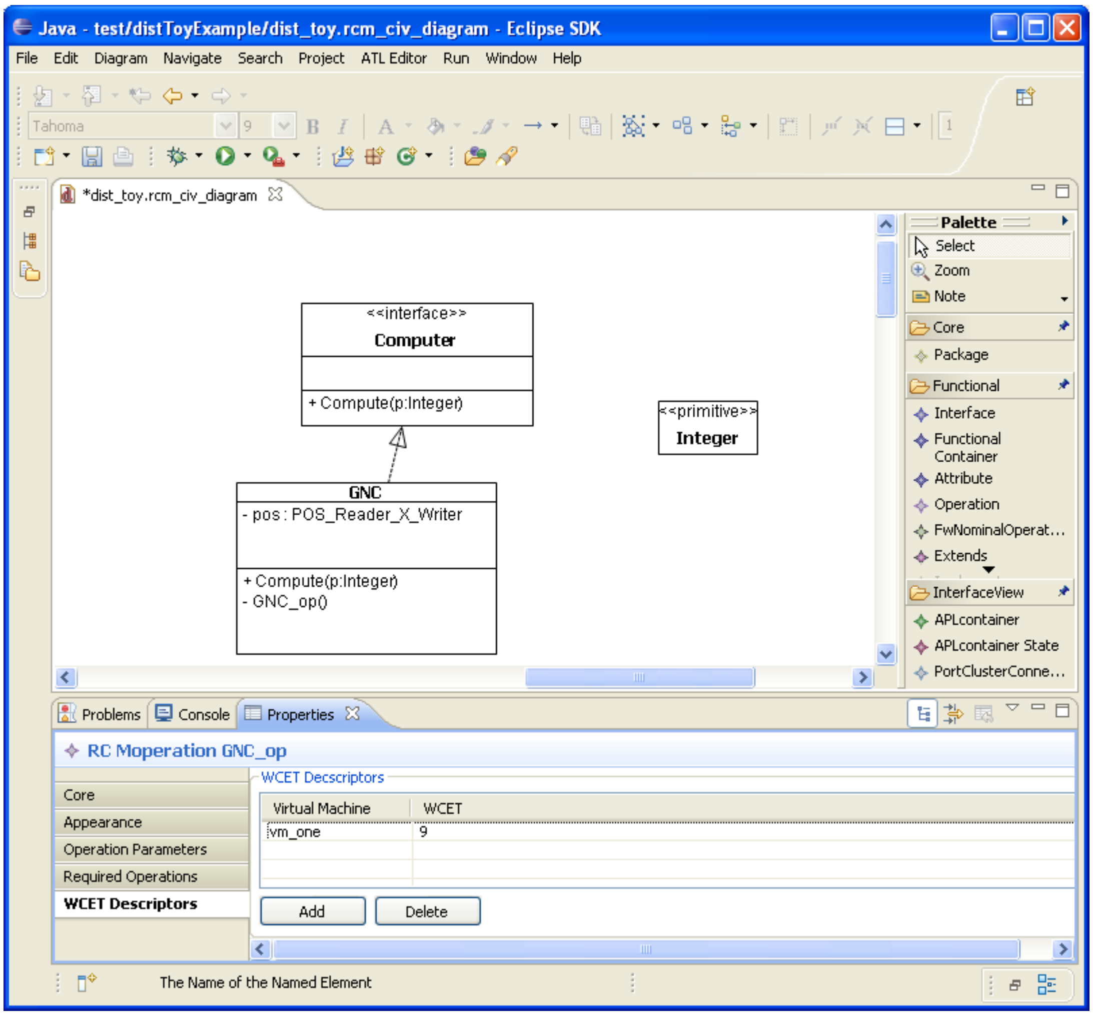
A.3.1.5.3 WorksOn

An operation comes with the *worksOn* relation: this relation allows to specify the state , i.e. the set of properties, on which the operation acts. You can specify this information through the property editor.

Note that this information is particular relevant for the interface view and for the concurrent view transformation.

A.3.1.5.4 WCET descriptor

WCET descriptors for an operation can be edited thought the dedicated property editor tab available when the operation is selected.



Virtual Machine	WCET
vm_one	9

Figure 14: WCET descriptors editor

A.3.1.6 Classifier relations

The editor supports the definition of interface implementation relations and classifier extension relations.

To create an interface implementation relation:

- select the I*mplements* tool from the *Fuctional* palette group
- draw a line starting from the class and ending on the interface.

When an interface implementation is created the tool checks and creates all the operations defined in the interface that are missing in the class.

To create a classifier extension relation:

- select the *Extends* tool from the *Functional* palette group
- draw a line starting from the derived classifier and ending on the base classifier.

A.3.2 State Machine diagram

The RCM meta-model and the HRT-UML2 state machine diagram editor have been defined and implemented accordingly to the EthFwProfile UML profile. Please refer to [CP05] for the specification of the EthFwProfile stereotypes and constraints.

Given a RCMfunctionalContainer it is possible to open its state machine from the class diagram by double clicking the name compartment of the figure; another possibility is to initialize a new RcmSm diagram from the RCM model file selecting the RCMfunctionalContainer as the diagram root entity.

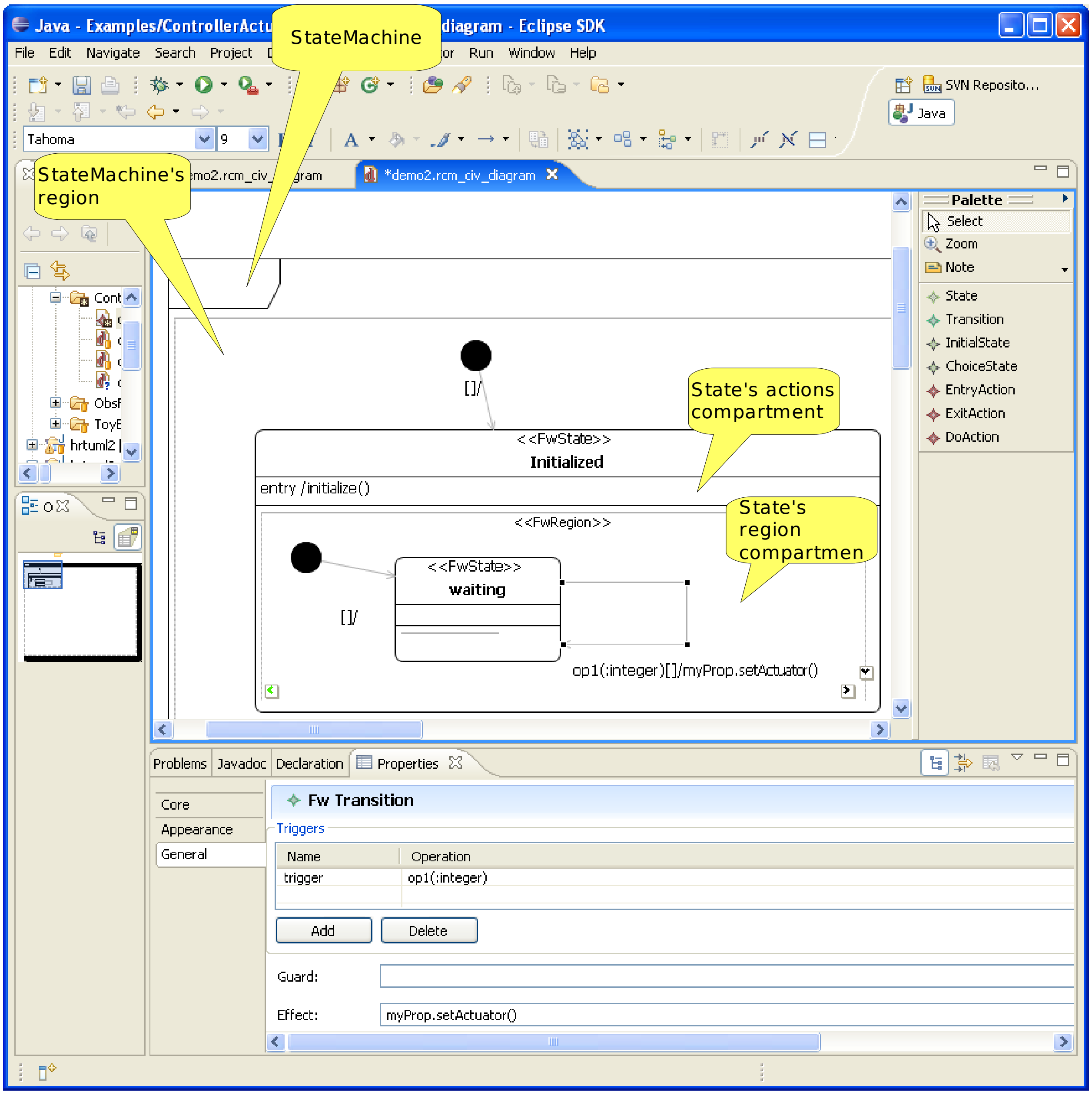
When a state machine diagram is opened for the first time the StateMachine entity and the owned region appear. States (FwState) and transitions (FwTransition) can be created inside the state machine region.

By defnition FwState can own a region, so you can have nesting of state.

Transition has a dedicated property tab that allows to edit triggers, guard and effect. State entry and exit actions, transition guard and effect are features of type string: these string should be expressed using the FwActionLanguage ([CP05]) language syntax.

The hrtuml2 plug-in integrates the Fw Action Language parser developed by ETH to parse and validate the modified state action, transition guard/effect string.

You can enable/disable the Fw Action Language parser check; to do this you have to select *Window-> Preferences->Run/Debug->String Substitution* from the Eclipse menu and set to true/false the boolean variable named *ActionLanguageParser*.



Fw Transition

Triggers

Name	Operation
trigger	op1(:integer)

Add Delete

Guard:

Effect: myProp.setActuator()

Figure 15: State Machine diagram

A.3.3 Importing a functional model

You may populate a new RCM model from an UML model that applies the ETH Framework Profile [CP05].

NOTE: integration of functional models can be also achieved at code generation level (see A.8.3).

To create an RCM model from an existing UML functional model

- open the ***File -> New -> Other -> HRT-UML2 -> RCM from UML*** wizard:

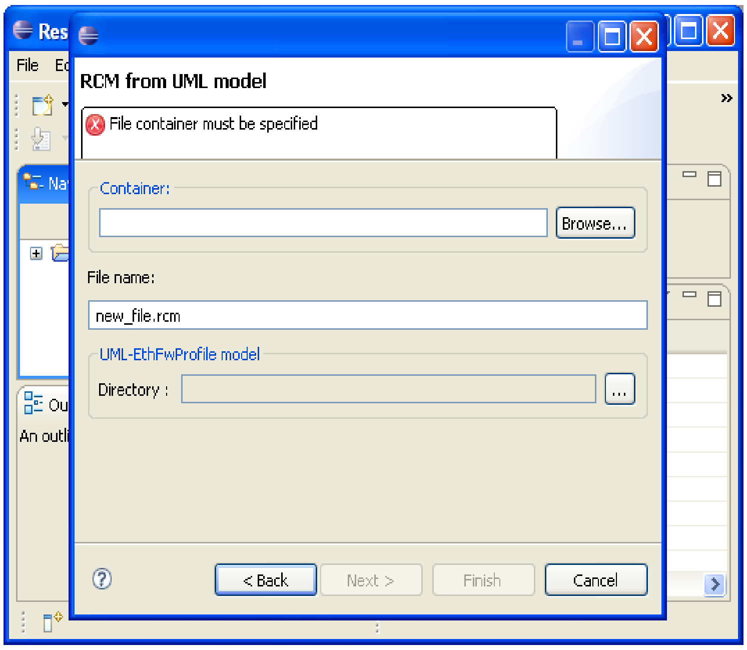


Figure 16: RCM from UML wizard

The wizard come with the following fields to be filled:

-*Container* is the folder where the .rcm model as to be created

-*File name* is the name of the .rcm file to create.

-*UML-EthFwProfile model Directory*: use the navigator to select the UML model from which load the functional model. The model file (.uml or .uml2) has to be on the current workspace.

The result of the import phase is a new RCM model populated with all the model entities (Class, Interface, StateMachine...) stereotyped according to the EthFwProfile. You can navigate the model and extend it for instance by initializing a new RcmCiv editor.

A.4 Interface View Modeling

HRT-UML2 diagram editor supports the interface view [UEE07]design by offering two kind of diagrams:

- class diagrams (RcmCiv), rooted on a RCM.Package model entity,
- instance diagrams (RcmInstance), rooted on the RCM.InstanceSet model entity.

The rationale behind the usage of these diagrams is the following:

- one or several RcmCiv has to be used to define the AP-level container, as *classifiers*, of the interface view
- one instance diagram as to be used to model the collaboration of AP-level container instances, to specify the HRT attributes and to map instances to deployment partitions.

The interface view depends on the functional view, as explained in the following chapters; most of the modification actions on the functional view are automatically reflected by the tool in the interface class and instance view. See chapter 10 for more information about the status of this feature.

A.4.1 Class Diagram

In the following are the Interface View entities that you can manipulate through the HRT-UML2 class diagram.

A.4.1.1 AP-level container

Use *APLcontainer* tool in the *InterfaceView* palette to create an AP-level container classifier (APLcontainer in the RCM metamodel) on the current diagram.

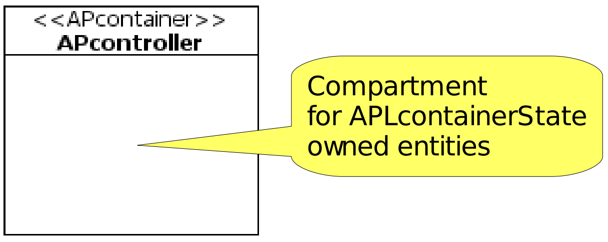


Figure 17: RCMcomponent presentation

An AP-level container is represented as an UML stereotyped component; it can have a state, i.e. It can own specialized RCM Property entities called **APLcontainerState**. An AP-level container may not define operations.

A.4.1.2 AP-level container state (State Reference)

An APLcontainer is a structured classifier so its diagram presentation comes with a dedicated compartment to show the parts (StateReference in the RCM metamodel) it owns; a part represent the state of an APLcontainer. A part of an APLcontainer can only be typed with RCMfunctional-Container.

To add parts to an APLcontainer:

- select the *APLcontainer State* palette command
- click inside the APLcontainer parts compartment.

You may create more than one part inside an APLcontainer.

A type has to be defined for a functional state. It is possible to select the type of the part through the properties view; note that for the selected part the editor shows as available types only function-al containers which are visible to the current APLcontainer.

When the type is selected the the tool automatically performs the following actions on the current APLcontainer:

- creates one or more provided port cluster (RCMpiPortCluster) owning provided element-ary ports (RCMpiPort): a provided port cluster is created for each type provided by the func-tional container typing the APLcontainer state.
- creates zero or more required port clusters (RCMriPortCluster) owning elementary re-quired ports (RCMriPort). Required ports are created according with the RequiredOperation relations specified in the functional view.

The diagram shows the delegation of the created ports to the current internal functional part.

The relation behind the creation of these entities is explained in the following.

The provided port clusters allow the APLcontainer to expose the services provided by the internal declared parts. The APLcontainer provides such services through elementary ports (RCMpiPort), one elementary port for each elementary service; these provided services are grouped into port clusters according to the definition of the types implemented by the functional container typing the APLcontainer state.

There can be the case where a given service is shared among different provided port clusters associated to the same APLcontainer state. In this case, for the related elementary ports, the value for some attributes have to be the same: in particular the tool currently perform check about port kind consistency for all the elementary ports referring the same service imple mented by the same APLcontainer state.

So a port cluster aggregates elementary ports, each elementary port refers to a service provided by the functional container part to which the port cluster is delegated.

An elementary port refers a functional service that can be called through it, moreover it comes with concurrent attributes which are the relevant attributes of the interface view as defined in [CVHN06]: these attributes determine the protocol to use when the related service is invoked. Please refer to [BT07] for an exhaustive explanation of elementary port concurrent attributes and their relations with the attribute defined in [CVHN06].

The required port cluster represents a type that is requested by the functional part to which the port itself is delegated. Currently a required port cluster is created for each attribute, typed RCM-functionalContainer or Interface, declared in the functional container typing the current part: moreover this attribute has to appear in the “Instance” field of a RequiredOperation relation associ-ated to a owned operation. The operation invoked on this property, defined in the functional view through the RequiredOperation's definition, will determine the elementary RCMriPort's.

As for the provided services, each required services in a port cluster is actually requested through an elementary port, an elementary port for each required service. An elementary port has concurrent attributes associated.

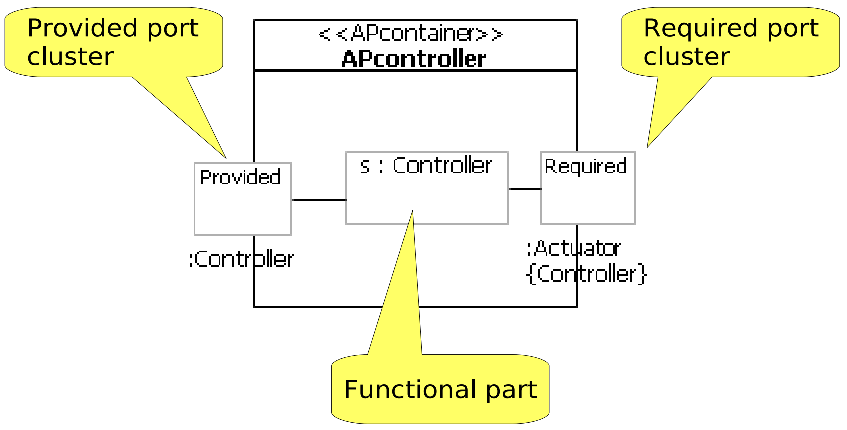


Figure 18: An APLcontainer with state and port clusters

Note that the association of concurrent attributes to ports allows to reuse the same functional service in different concurrent context.

So you may use the same functional container as part more than once in the same or in different APLcontainers. Moreover you may set different concurrent attributes values for the services offered and required by each of these parts.

A.4.1.3 Provided Ports and Port Clusters

For each RCMcomponent, the editor displays on its borders in an identifiable way the set of contained RCMpiPortCluster(s).

For each RCMpiPortCluster, the editor lists the static set of owned RCMpiPorts.

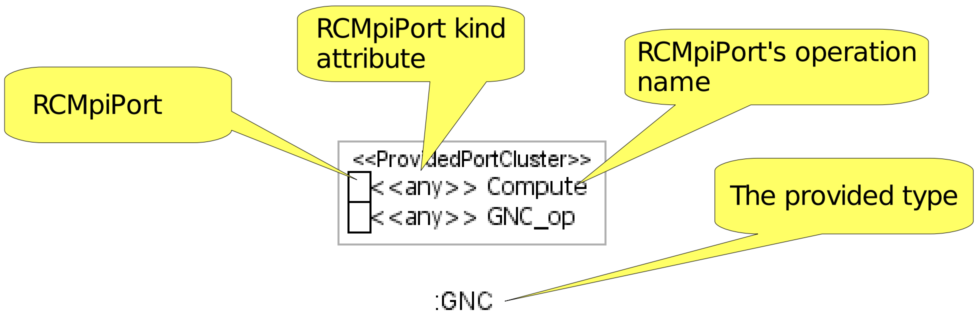


Figure 19: RCMpiPortCluster and RCMpiPorts

You can resize and move the port cluster figure around the APLcontainer border. Moreover the compartment listing the owned elementary ports can be collapsed.

The creation/deletion of these entities is automatically performed by the tool, for instance when the user edits the type of a StateReference (creation/deletion of port cluster) or, from the functional view, when he/she add/remove an operation in the provided type (creation/deletion of RCM port).

The full path of the provided type is available in the port cluster property editor

A.4.1.4 Required Ports and Port Clusters

For each RCMcomponent, the editor displays on its borders in an identifiable way the set of contained RCMriPortCluster(s).

For each RCMriPortGroup, the editor displays the static set of owned RCMriPorts.

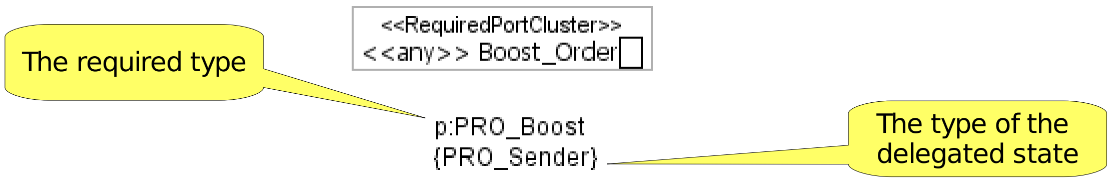


Figure 20: RCMriPortCluster and RCMriPorts

You can resize and move the port cluster figure around the APLcontainer border. Moreover the compartment listing the owned elementary ports can be collapsed.

The creation/deletion of these entities is automatically performed by the tool, for instance when the user edits the type of a StateReference (creation/deletion of port cluster) or, from the functional view, when he/she adds/removes an operation in the provided type (creation/deletion of RCM port).

The full path of the required type is available in the port cluster property editor

A.4.1.5 Port Cluster Connectors

In HRT-UML2 connections between required and provided ports are only allowed to be designed at port cluster level through the PortClusterConnector element.

A PortClusterConnector represents a connector between an RCMriPortCluster and an RCMpiPortCluster. Such connector does not exist in the RCM meta-model, it is a pure graphical element which identifies, the existence of RCMassembly's between the ports owned by the RCMriPortCluster and RCMpiPortCluster.

In the class diagram it is possible to connect port clusters only for the same APLcontainer class: in order to do this:

1) select the PortClusterConnction command from the palette and trace a link from the required port to the provided one.

Automatically all the RCMassembly's between the elementary compatible ports will be created in the model.

The connection can not be started or ended if the cursor is placed on top of an elementary port. See chapter 5.4.3 for more information about port clusters compatibility checks.

A.4.1.6 Managing port cluster visibility on diagram

In RCM, port clusters comes with the **PortClusterVisibility** enumerated attribute which allows you to have some control on the visibility of the services exposed and required by an APLcontainer.

Some graphical features are provided around this visibility attribute.

It is possible to right click an APLcontainer and select one of the following actions:

- show all port clusters
- show public port clusters
- show external restricted port clusters

If the PortClusterVisibility attribute of a provided port cluster is set to ***InternalRestricted*** then the port cluster appears as a port of the APLcontainer state which has generated the port cluster.

For example consider the APLcontainer appearing in figure 21: in this case the provided services related to the internal states s1:POS_Sender and s2:PRO_Sender have been set to InternalRestricted. This makes Dispatcher the unique service provided by the TMTC_AP which can be used by the external environment. Note that the required port cluster related to d:Dispatcher have been set to internal restricted also.

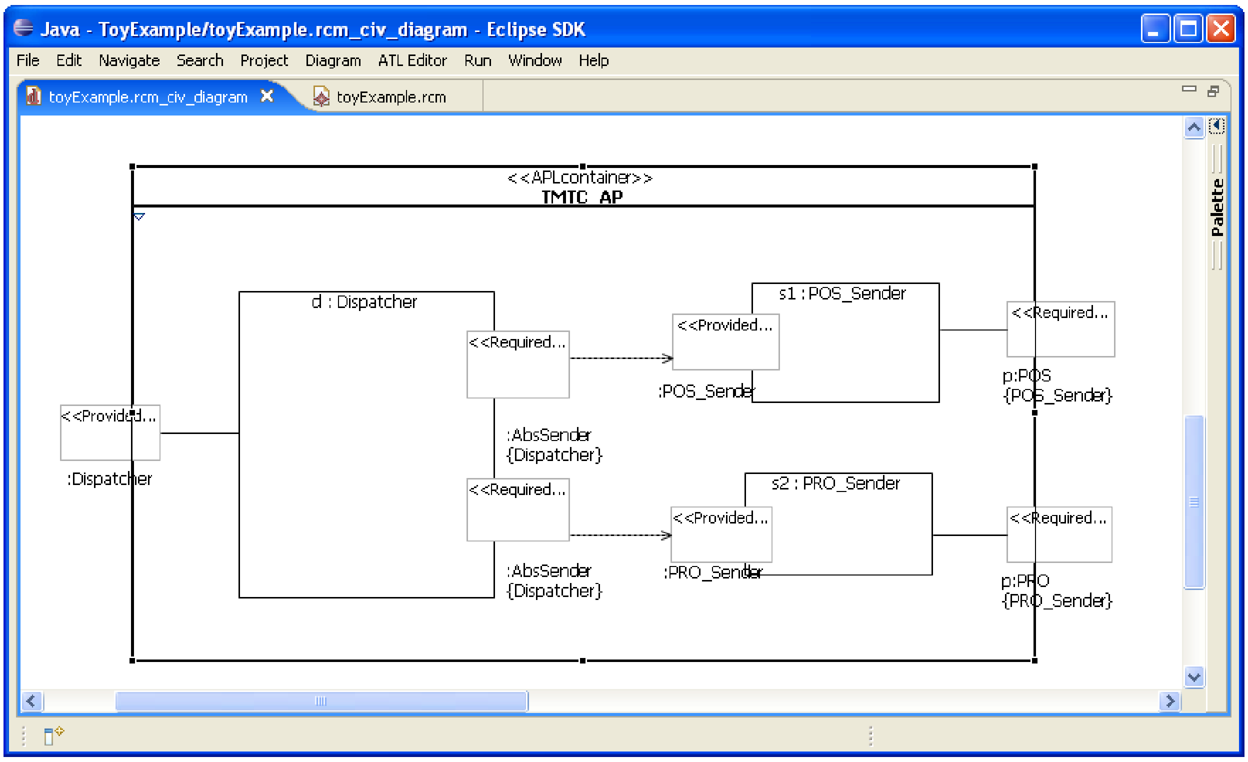


Figure 21: Managing PortClusterVisibility attribute: internal restricted case

Other graphical features for provided port cluster:

- right click a provided port cluster and select "*show related port clusters*"; this action hides all the port clusters that are not in a given relation with the selected one, i.e. it shows only:

1) the required port clusters used by the current provided port

2) the provided port clusters which satisfy the required ports in the previous point

- right click a provide d port cluster and select "Hide Port Cluster" to hide the current port: this can be particularly useful when the port originated by the functional state it is not intended to be used. To make the port appear again a "show al Port cluster" action on the owning APLcontainer has to be performed.

A.4.2 Instance Diagram

According to the RCM metamodel definition, APLcontainer instances are allowed to be defined in a dedicated RCM container entity: the InstanceSet. The HRT-UML2 tool implements the instance diagram editor on top of this entity and allow you to create APLcontainer instances and connect them together. Also a view on the logical deployment is supported.

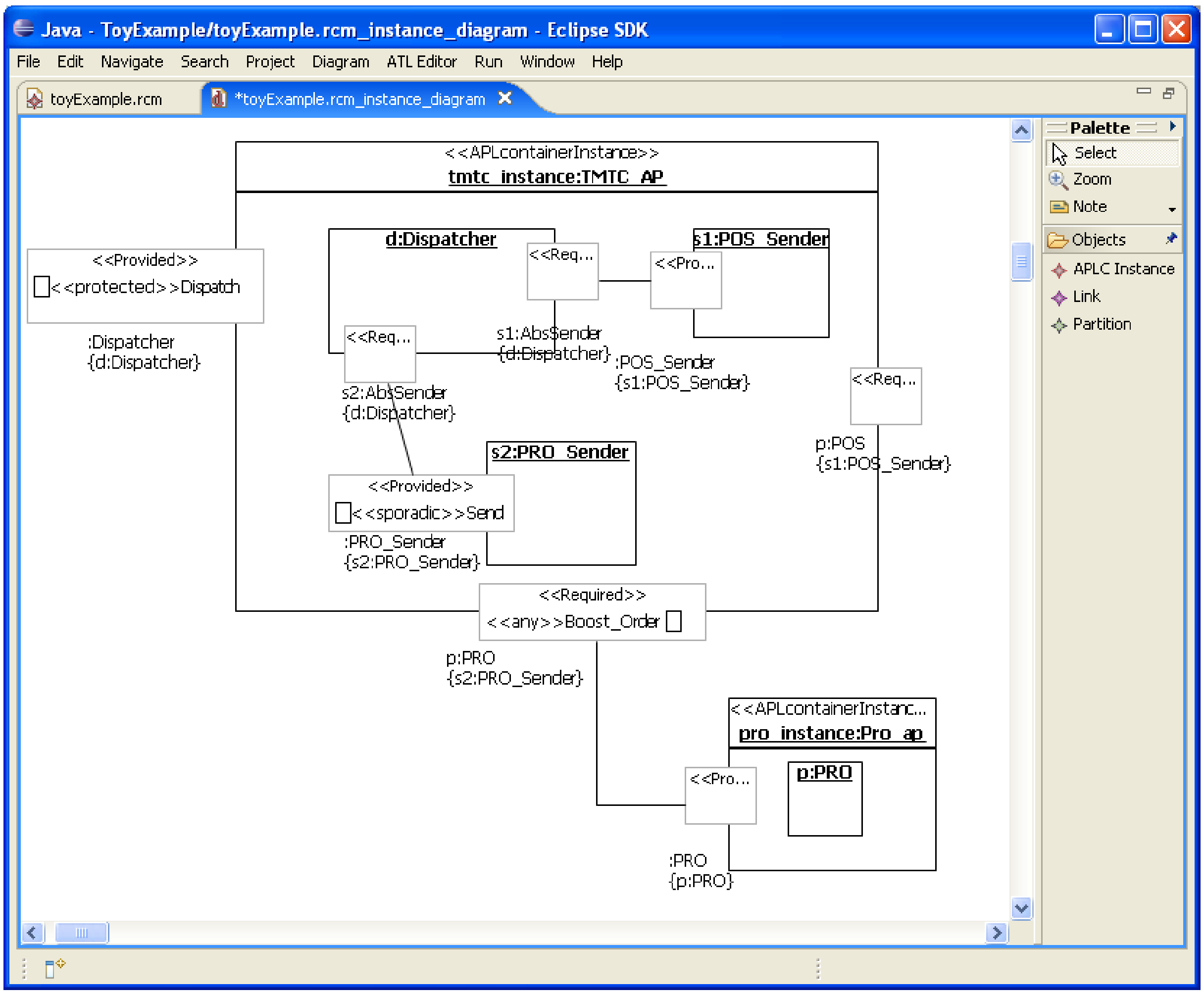


Figure 22: Working with APLcontainer instances

Figure 22 shows a part of the instance diagram for the ToyExample.

A.4.2.1 APLcontainer instance

Through the *Object* palette it is possible to create APLcontainer instances: once an instance has been created you have to specify the Classifier attribute through the property editor to type the current instance to an existing APLcontainer classifier.

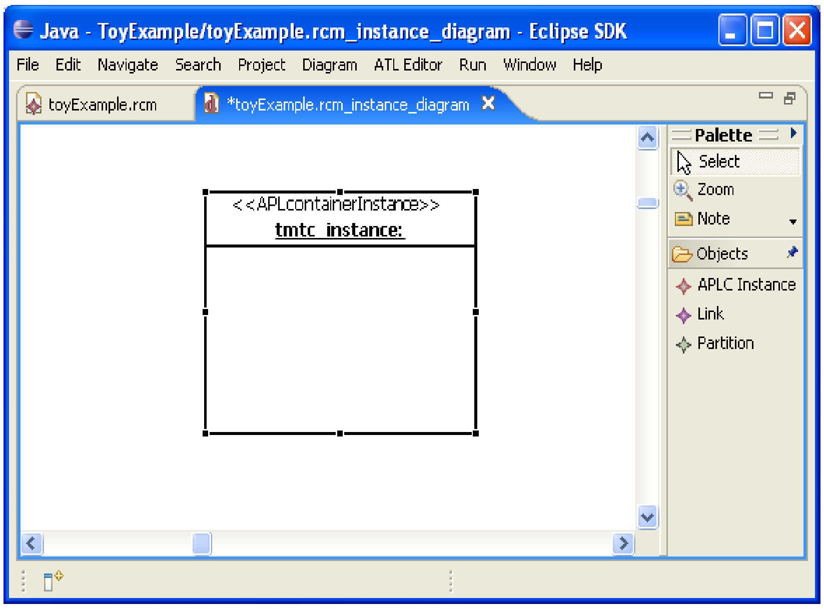
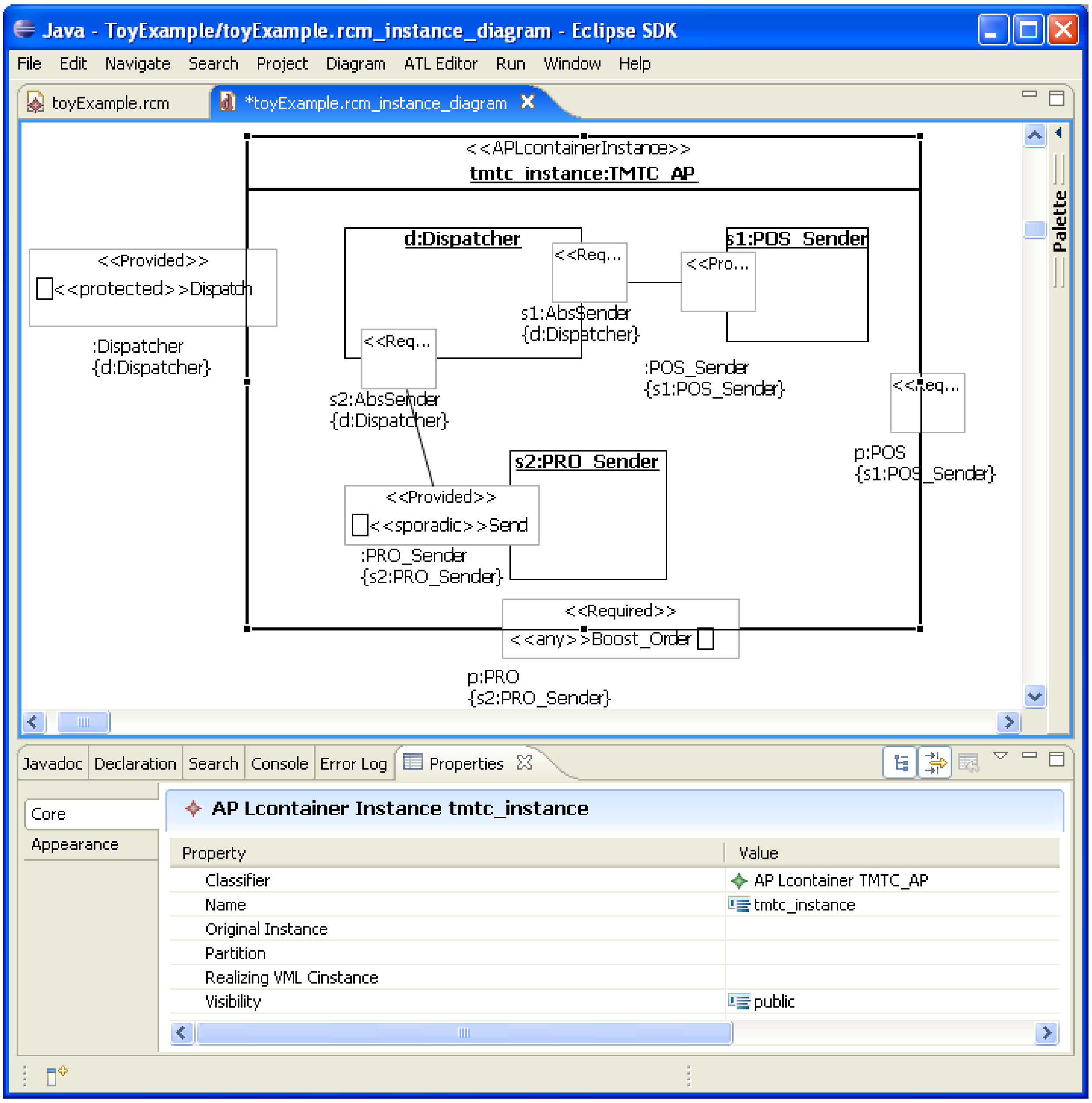

Java - ToyExample/toyExample.rcm_instance_diagram - Eclipse SDK
File Edit Navigate Search Project Diagram ATL Editor Run Window Help
toyExample.rcm
*toyExample.rcm_instance_diagram
<<APLcontainerInstance>>
tmtc_instance:
Palette
Select
Zoom
Note
Objects
APLC Instance
Link
Partition

Figure 23: Untyped instance representation

When you set the Classifier attribute the tool creates the instance's structure according to the structure of the typing classifier; the graphical result of this action is that **instances of port clusters** and **instances of state references** appear for the current APLcontainer instance. Also if port cluster connections are defined at class level, the tool automatically creates links for the corresponding port cluster instances; figure 24 illustrates this scenario for an instance typed to TMTC_AP.


Java - ToyExample/toyExample.rcm_instance_diagram - Eclipse SDK
File Edit Navigate Search Project Diagram ATL Editor Run Window Help
toyExample.rcm
*toyExample.rcm_instance_diagram
<<APLcontainerInstance>>
tmtc_instance:TMTC_AP
d:Dispatcher
s1:POS_Sender
<<Provided>>
<<protected>>Dispatch
:Dispatcher {d:Dispatcher}
<<Req...
<<Pro...
s1:AbsSender {d:Dispatcher}
:POS_Sender {s1:POS_Sender}
s2:AbsSender {d:Dispatcher}
<<Req...
p:POS {s1:POS_Sender}
s2:PRO_Sender
<<Provided>>
<<sporadic>>Send
:PRO_Sender {s2:PRO_Sender}
<<Required>>
<<any>>Boost_Order
p:PRO {s2:PRO_Sender}
Palette
Javadoc Declaration Search Console Error Log Properties
Core
Appearance

AP Lcontainer Instance tmtc_instance

Property	Value
Classifier	AP Lcontainer TMTC_AP
Name	tmtc_instance
Original Instance	
Partition	
Realizing VML Cinstance	
Visibility	public

Figure 24: Typing the instance

A.4.2.2 Port instance and HRT attributes

Port cluster and elementary port instances (i.e. RCM.PiSlot and RCM.RiSlot entities) are automatically created when the Classifier attribute of an APLcontainer instance is set.

According to the RCM metamodel definition (and to UML2 metamodel definition for instances), port cluster instances and port instances are not connected together trough composition relations as in the case of port cluster and elementary ports. However the composite structure of port cluster into elementary ports has been *graphically* reproduced in the instance diagram to help the user to better manage port instances editing and connections.

Each port cluster instance comes with a label: this label shows the type provided or required (in the upper side) and the delegating state reference instance (in the lower side).

Port instances can not be created or deleted by hand.

Port cluster instances and port instances allow to specify HRT attributes.

The following ports have HRT attributes that can be set at instance level only:

1) RCMpiPortCluster with at least one elementary port having *protected* kind,
2) RCMpiPort with *cyclic* concurrent kind,
3) RCMpiPort with *sporadic* kind.

See [BT07] for an exhaustive explanation of the available HRT attributes, also which are editable from the user and which are derived from the analysis.

The attribute **wcet_ri_closure** for provided port instances is automatically derived.

It is worth noting that, given an APLcontainer instance, provided port instances appearing in different provided PortCluster instances can refer to the same operation implementation. In this case the HRT attributes of these port instances have to be the same; the tool manages these synchronizations. For instance, look at figure 25: slots *Boost_Order* in port cluster slots *:PRO* and *:PRO_Boost* resolve to the same operation of state instance *pro:PRO*; in this case they will share the same values for the owned HRT attributes.

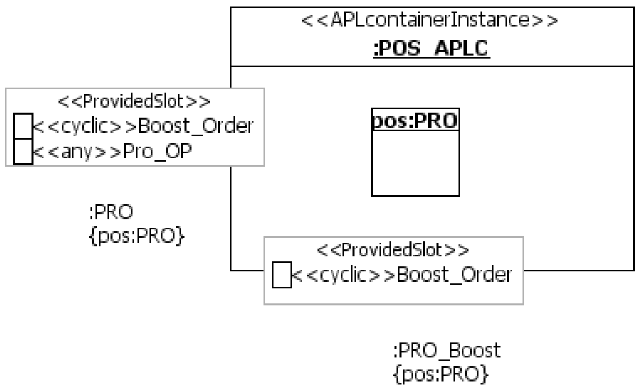


Figure 25: two CyclicSlots resolving to the same operation-state

A.4.2.3 AP instance Link

The editor allows to edit the APInstanceLink between the RCMriPort instances of an RCMriPort-Cluster instance and the RCMpiPort instances of an RCMriPortCluster instance through the editing of a single link between the two port cluster instance.

To create a port cluster instances link

- select the *Link* tool on the palette
- draw a connection from the required port cluster to the provided port cluster.

The connection can not be started or ended if the cursor is placed on top of an owned elementary port instance.

According to the "correctness by construction" approach, the tool prevent the creation of port cluster instances connections when they would invalidate the model.

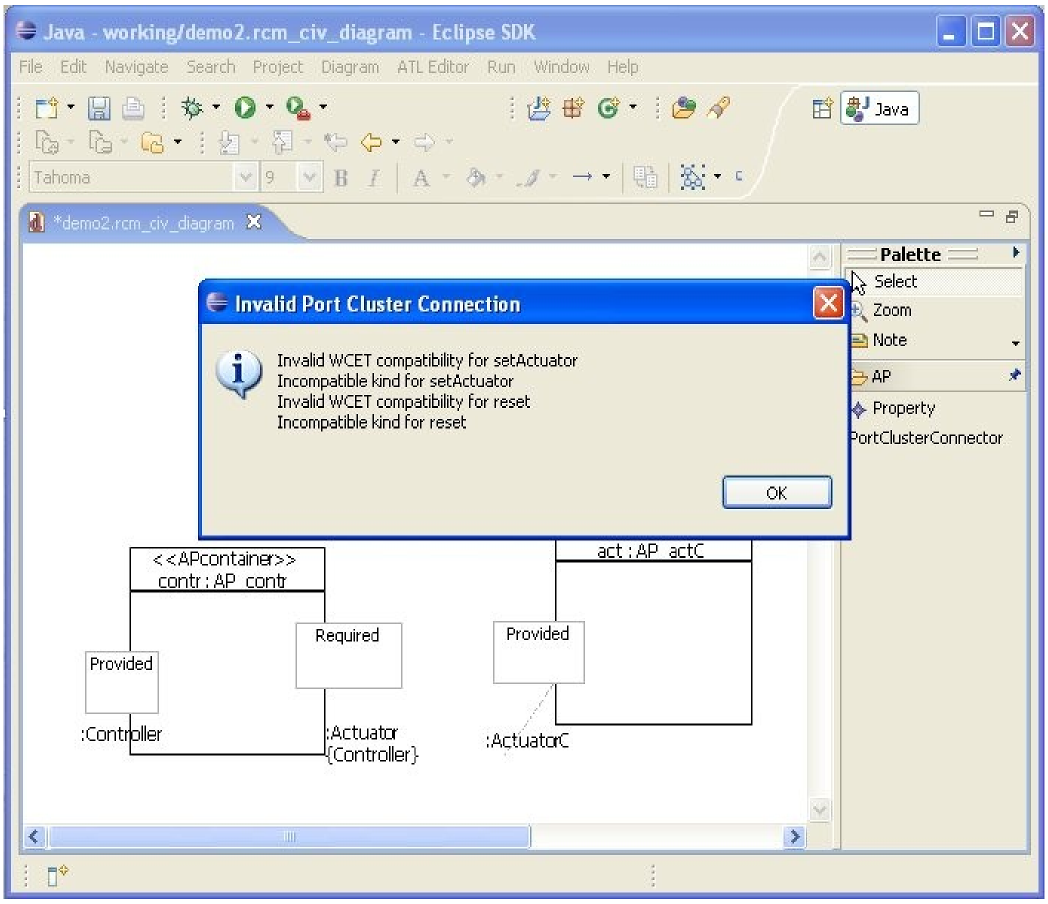
In particular it is possible to create a link between two port cluster instance if:

1) the provided port cluster instance is visible to the required one
2) the two port clusters referred have compatible types (i.e. the type of the provided port cluster is a subtype of the type of the required port cluster): this ensure that a required service has a corresponding compatible target provided services. These two services identify two corresponding, possibly incompatible, elementary ports.
3) matching provided-required referred elementary ports have compatible concurrent kind (see [BT07])
4) for each matching provided-required elementary port instances (piPort -> riPort) the following condition has to be true:

piPort.Wcet_ri_closure <= riPort.Maximum_allowed_execution_time()

where riPort.Maximum_allowed_execution_time = 0 means no restriction.

When trying to create an invalid link an information window appears showing the list of incompatibilities founded.



Invalid WCET compatibility for setActuator
Incompatible kind for setActuator
Invalid WCET compatibility for reset
Incompatible kind for reset

Figure 26: Invalid connection report

In its default configuration the instance diagram only shows port cluster instances connections; RCMassembly links, i.e. links among elementary port instances, are not graphically displayed, even if they exists and managed in the model.

A.4.2.4 A view on the logical deployment

The instance diagram can be used as a view to the logical deployment. In RCM the logical deployment allows to define partitions and logical communications among partitions [CP07].

The Instance diagram shows the current Partition defined in the model (for instance they can be created through the deployment diagram). Moreover it allows you to create partitions and deploy APLcontainer instance into them.

A.4.2.4.1 Partition and LogicalCommunication

You can create RCM Partition entities directly through the instance diagram using the Partition tool command.

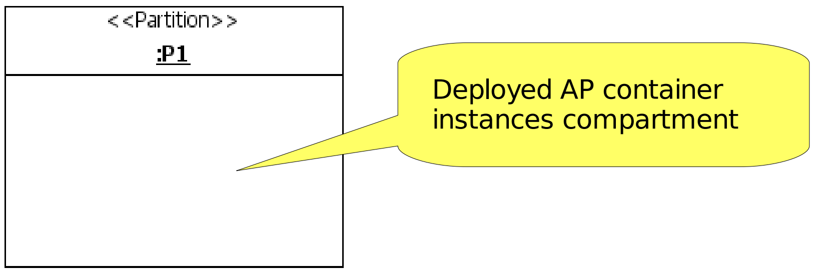


Figure 27: Partition representation in instance diagram

A Partition is represented as a stereotyped instance.

In the RCM model Partition's are owned by the RCM.Model.DeploymentInformation.LogicalArchitecture entity.

For a partition, all the **deployed** APLcontainer instances appears in the dedicated compartment.

When you have partitions then you can create (and so deploy) APLcontainer instances directly into the dedicated compartment of a partition.

You can also drag instances into the partition compartment to deploy them.

When you edit links between deployed APLcontainer instance the tool automatically manages the LogicalCommunication entities existence between the involved partitions, according to the definition given in [CP07]

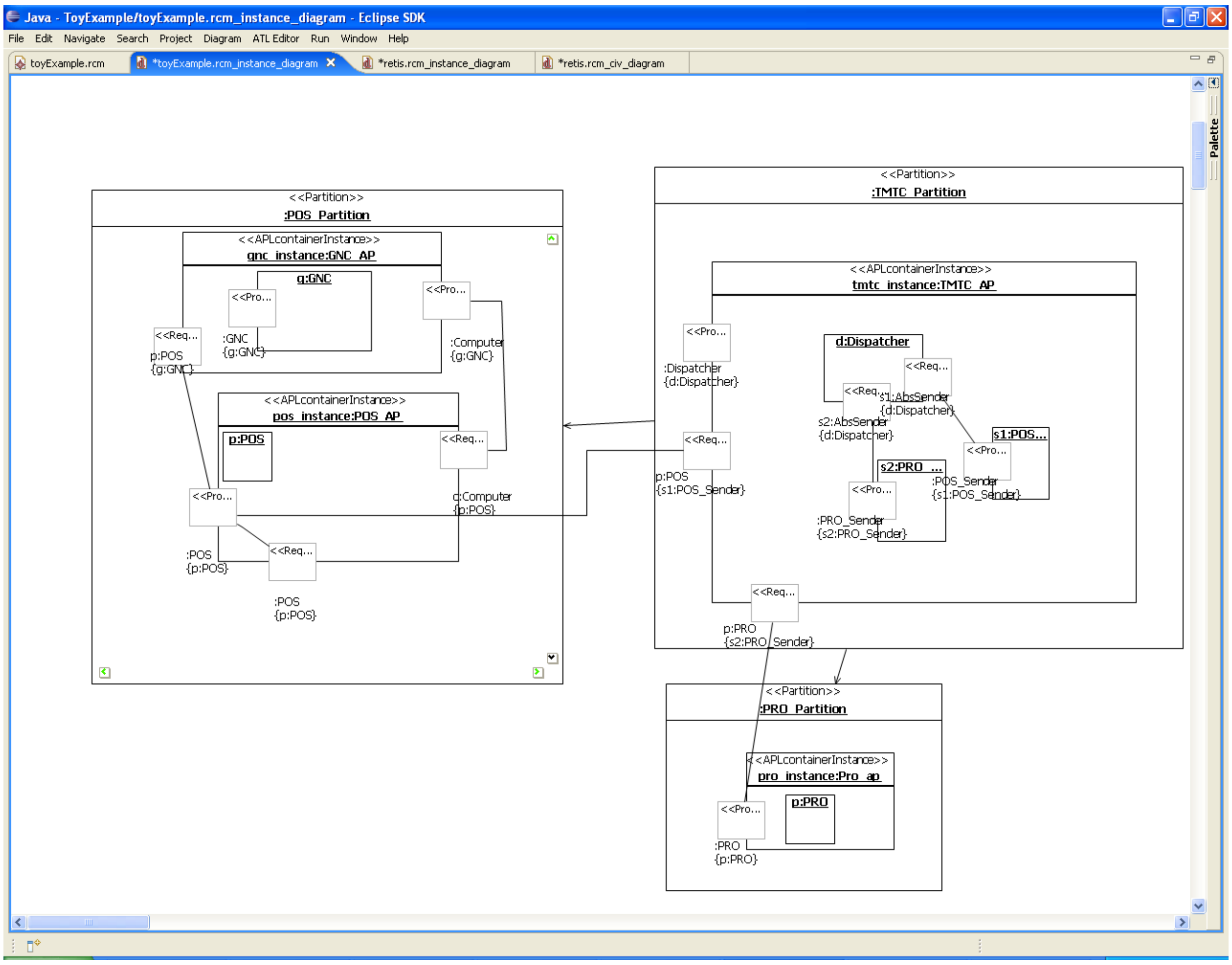


Figure 28: Logical deployment in the instance diagram

When instances are dragged between Partitions, the logical communications figures are not always automatically synchronized with the changes performed in the model: to rapidly synchronize the diagram you can press **F5**. Alternately you can close and reopen the diagram.

A.5 Working with HRT-UML/RCM patterns

This chapter illustrates hot it is possible to use the HRT-UML/RCM patterns during the modeling activity performed with the tool.

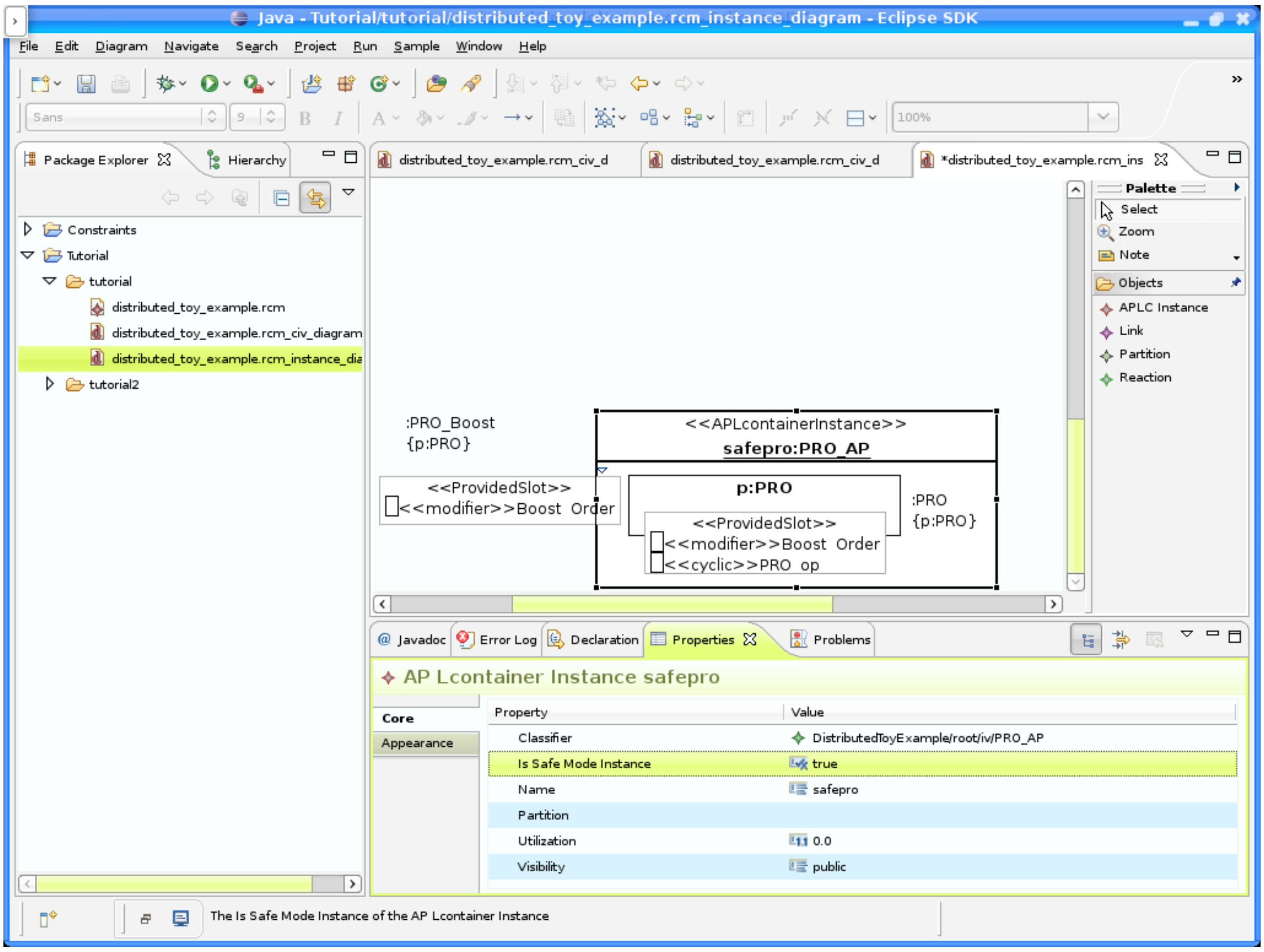
A.5.1 WCET overrun handling

It is possible to mark a number of APLC instances to perform "**safe-mode**" operations within individual logical partitions of the system: in the Instance Diagram, select a single istance of APLC and, on the Properties tab, set the value of the *is_safe_mode_instance* attribute to "True".

All APLC instances marked as "safe-mode" are *inactive* during nominal operation. It is to be noted that the nominal mode is the default mode for the entire system, since support for modeling system modes is *not* presently incorporated in the HRT-UML/RCM.

Each logical partition comprises a Timer Manager which enforces the preservation of the WCET budget assigned to the operations provided by the APLC which compose the partition. Whenever a WCET overrun is detected at run time for any operation of a given logical partition, all "safe-mode" APLC instances *of that partition* are immediately and automatically activated.

The "safe-mode" APLC instances are meant to include operations which perform diagnostic and recovery actions to rectify the violations incurred with the WCET overrun. In order that those operations can be performed without incurring interference from the continuation of the nominal operations (at least one of which had erroneously overrun) the *criticality* attribute of "safe-mode" operations should be set higher than all other nominal operations in the same partition.



Property	Value
Classifier	DistributedToyExample/root/iv/PRO_AP
Is Safe Mode Instance	true
Name	safepro
Partition	
Utilization	0.0
Visibility	public

The Is Safe Mode Instance of the AP Lcontainer Instance

Figure 29: safe-mode instance

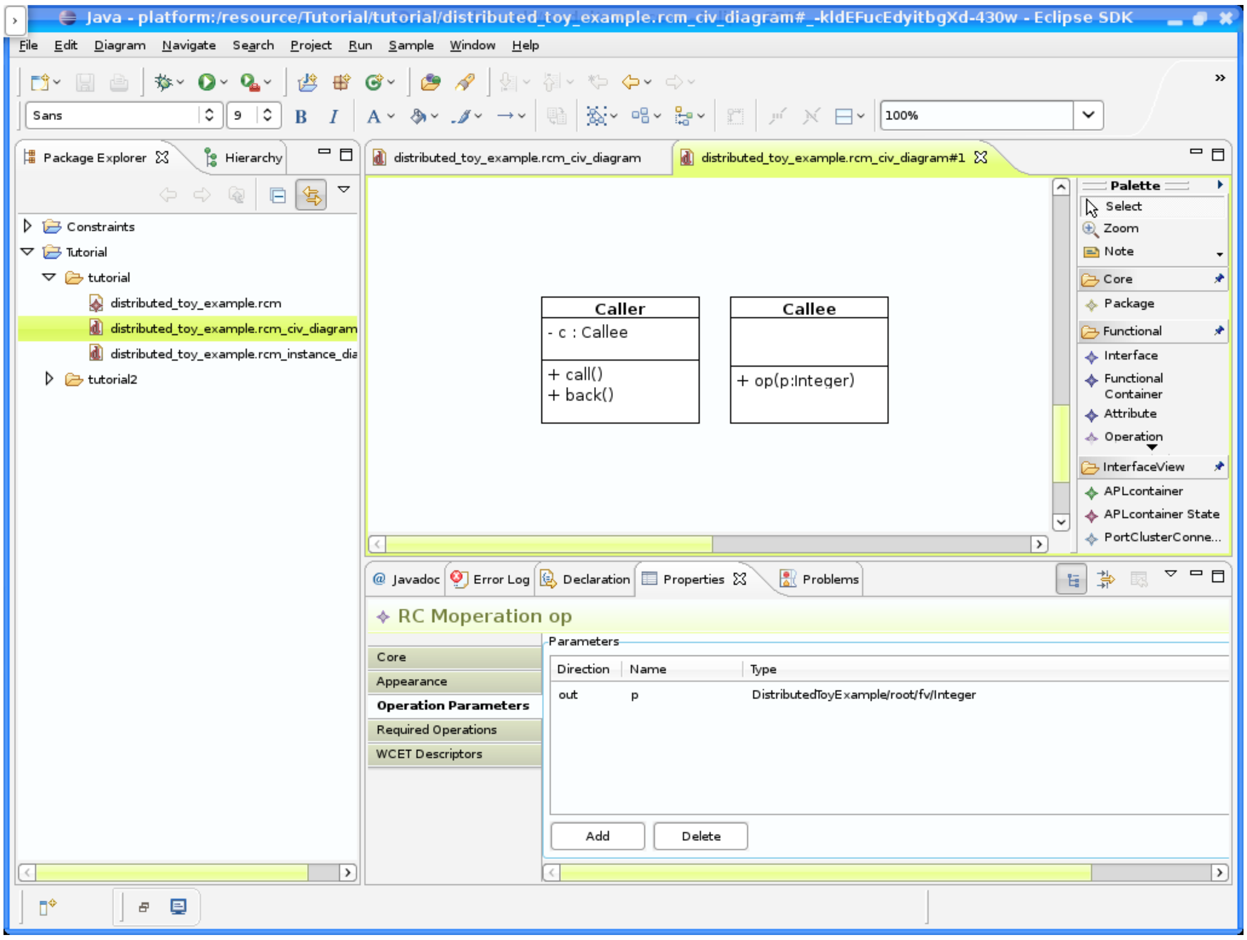
A.5.2 Deferred services with write-mode parameters (and optional time-out)

This pattern allows to model deferred services with write-mode (i.e. out) parameters.

As an important limitation of the present release of the toolset, however, the caller and the callee for that type of deferred operations must be located on one and the same physical node. In other words, this feature does not work yet for remote invocations.

While work is in progress at UPD to incorporate support for this feature (which may possibly complete in time for use in the V3 demonstrators) an alternate solution may be used to model remote deferred invocations with out parameters by using the Reactivity Links pattern provided as part of the current sets of enhancements.

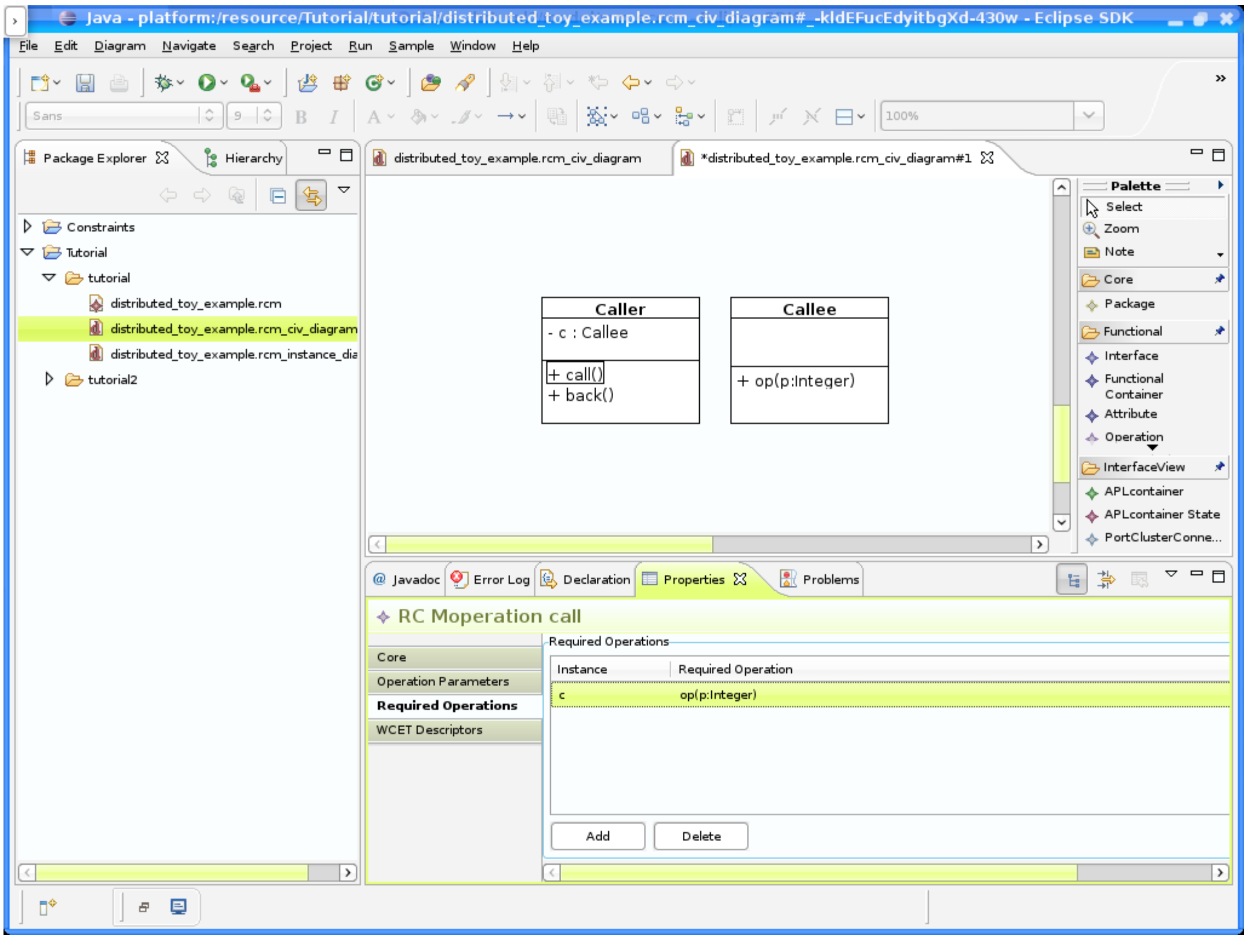
In the class diagram for the functional view, locate an operation with a write-mode parameter. In the picture 30, *op* has a write-mode parameter.



Direction	Name	Type
out	p	DistributedToyExample/root/fv/Integer

Figure 30: deferred with write mode parameters, step 1

Locate an other operation which requires an operation with a write-mode parameter. In the figure 31, *call* has such a required operation.

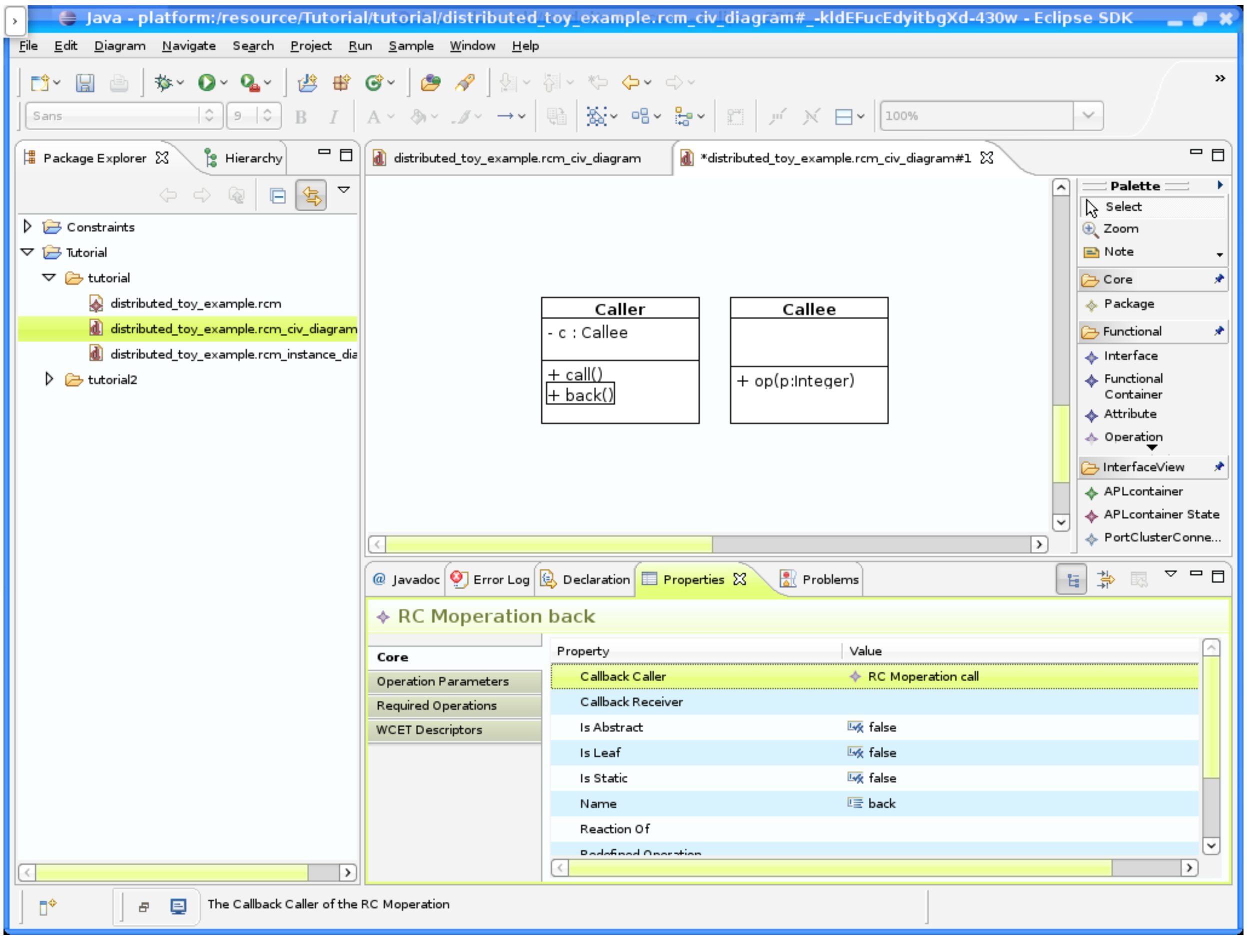


Required Operations

Instance	Required Operation
c	op(p:Integer)

Figure 31: deferred with write mode parameters, step 2

In order to allow the request originated by *call* to be satisfied by a deferred implementation of *op*, we must specify a callback operation for *call*. This is done by creating a new operation in the container of *call*. In the picture at left, the new operation is called *back*. In the properties tab for the new operation, set the value of the callback_caller reference to the caller operation (in our example we set it to *call*). The callback operation will be called when the execution of the required deferred operation will be completed and the write-mode parameter will be available for further computation.

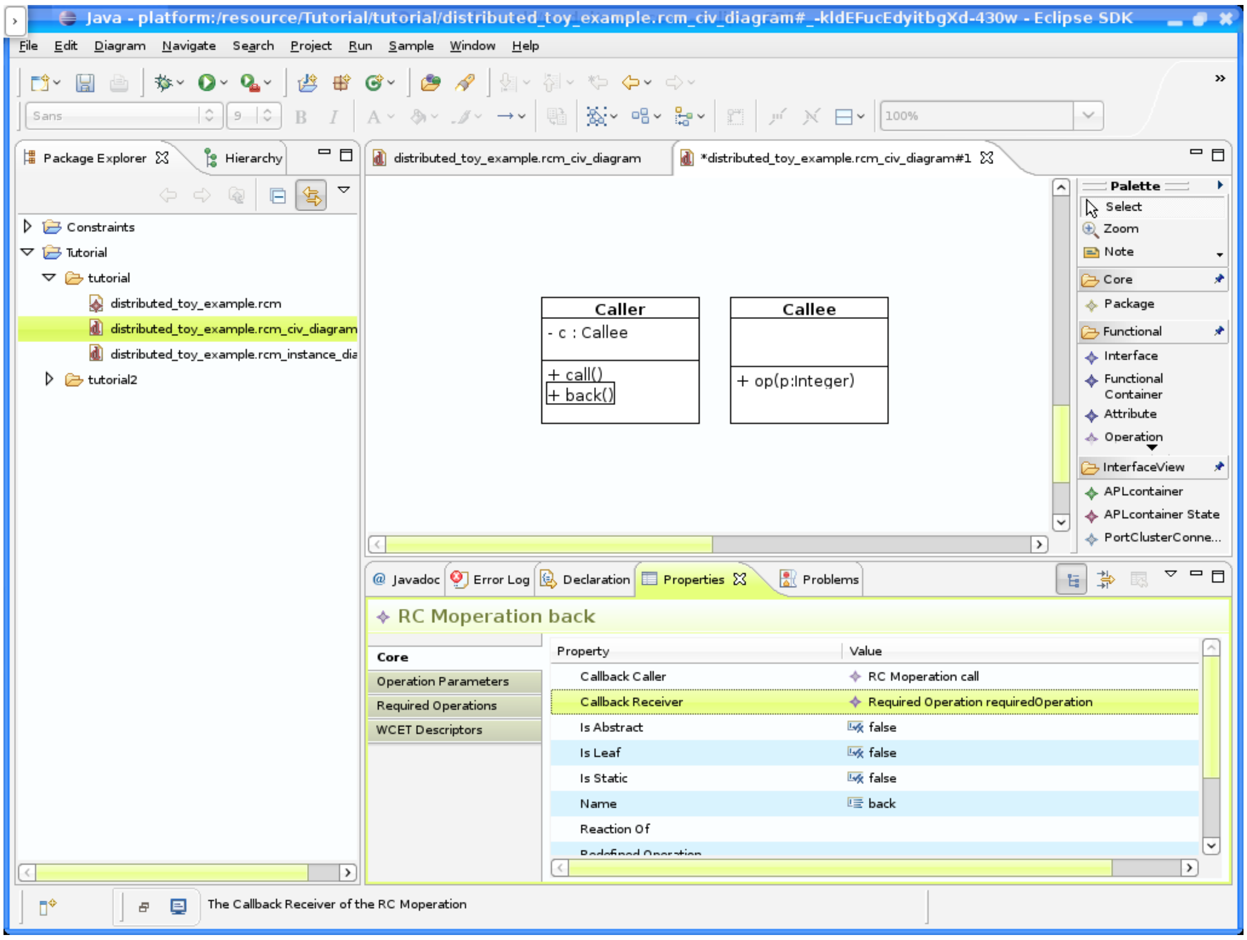


Property	Value
Callback Caller	RC Moperation call
Callback Receiver	
Is Abstract	false
Is Leaf	false
Is Static	false
Name	back
Reaction Of	
Redefined Operation	

The Callback Caller of the RC Moperation

Figure 32: deferred with write mode parameters, step 3

In the properties tab for the new operation, set the value of the callback_receiver reference to the "required operation descriptor" of the caller operation. The "required operation descriptor" to select is the required operation which targets the operation with write-mode parameters. (in our example we set it to the only "required operation descriptor" of *call*)

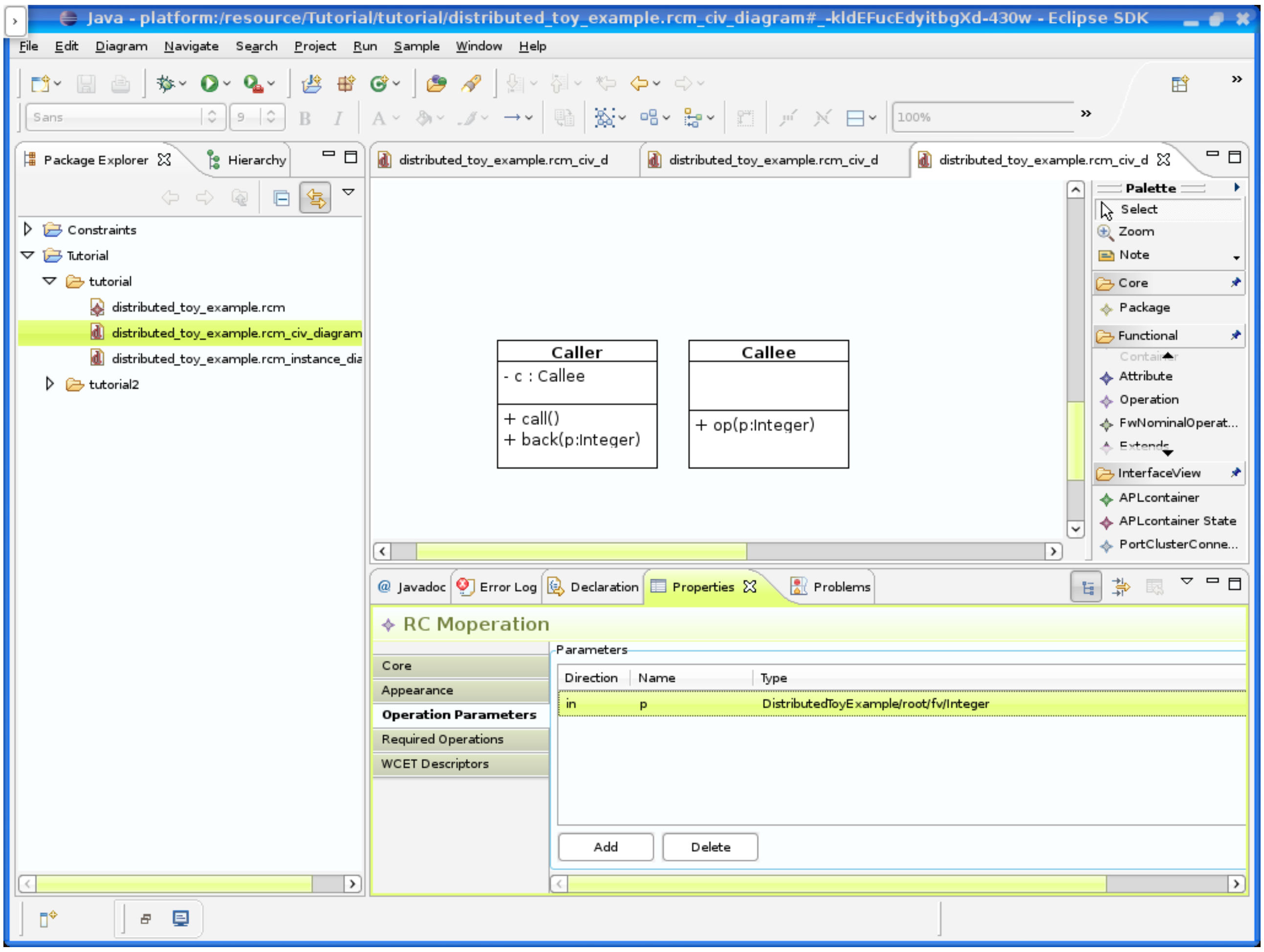


Property	Value
Callback Caller	RC Moperation call
Callback Receiver	Required Operation requiredOperation
Is Abstract	false
Is Leaf	false
Is Static	false
Name	back
Reaction Of	

The Callback Receiver of the RC Moperation

Figure 33: deferred with write mode parameters, step 4

In the properties tab for the new operation, add as many read-mode parameters as needed to match the write-mode parameters of the called service. (In our example we added a single read-mode Integer parameter called *p*.)



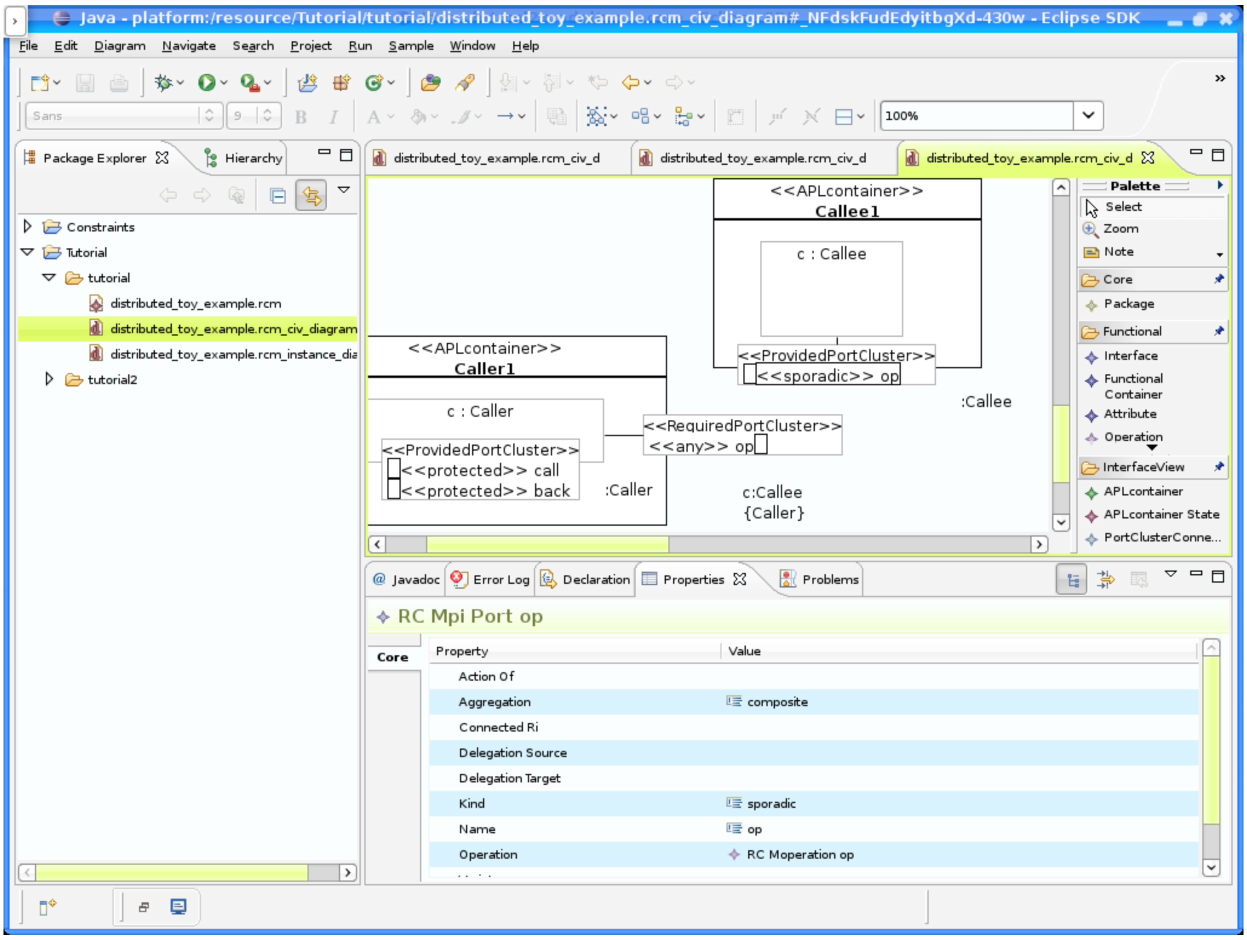
RC Moperation

Parameters

Direction	Name	Type
in	p	DistributedToyExample/root/fv/Integer

Figure 34: deferred with write mode parameters, step 5

In the class diagram for the interface view, locate the APLcontainer providing a port for the operation with write-mode parameters, and set its concurrent type to sporadic, cyclic or modifier.

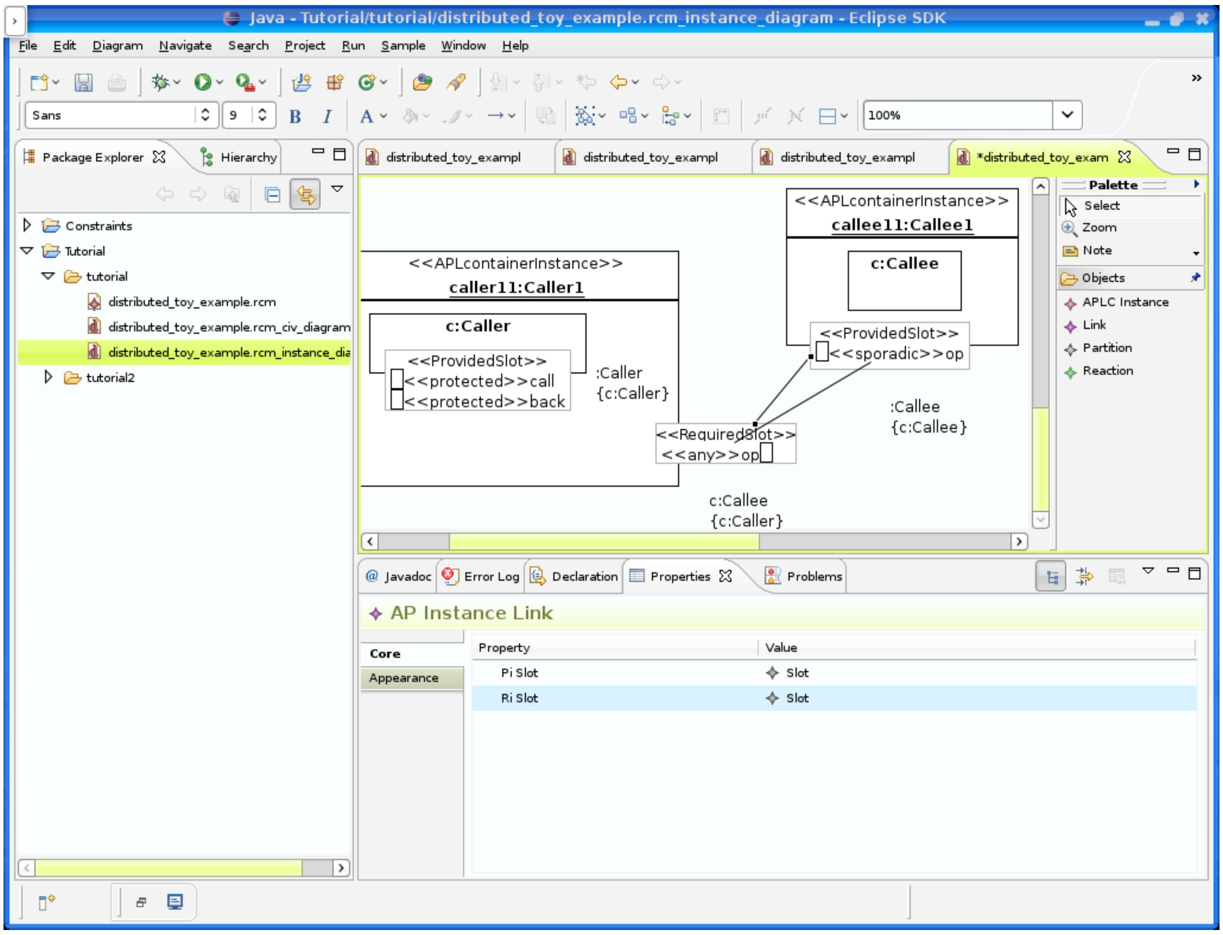

Java - platform:/resource/Tutorial/tutorial/distributed_toy_example.rcm_civ_diagram#_NFdskFudEdyitbgXd-430w - Eclipse SDK

RC Mpi Port op

Property	Value
Action Of	
Aggregation	composite
Connected Ri	
Delegation Source	
Delegation Target	
Kind	sporadic
Name	op
Operation	RC Moperation op

Figure 35: deferred with write mode parameters, step 6

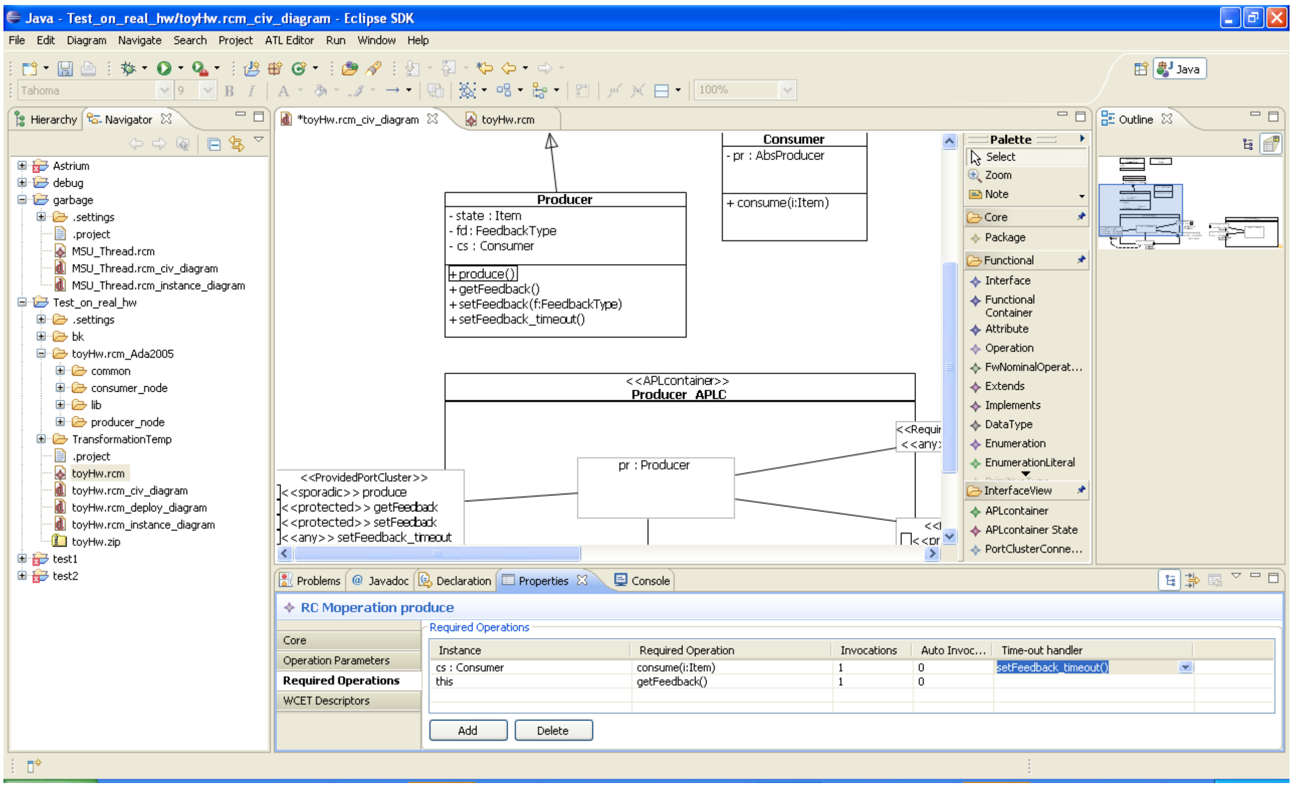
In the instances diagram for the interface view, locate an instance providing a slot whose defining feature is the port which you have just decorated as sporadic, cyclic or modifier. Then locate an instance with a slot whose defining feature is the required port originated by the operation which requires the operation with write-mode parameters. Finally, create a link between the slot for the port cluster containing the latter and the slot for the port cluster containing the former.



Property	Value
Pi Slot	Slot
Ri Slot	Slot

Figure 36: *deferred with write mode parameters, step 7*

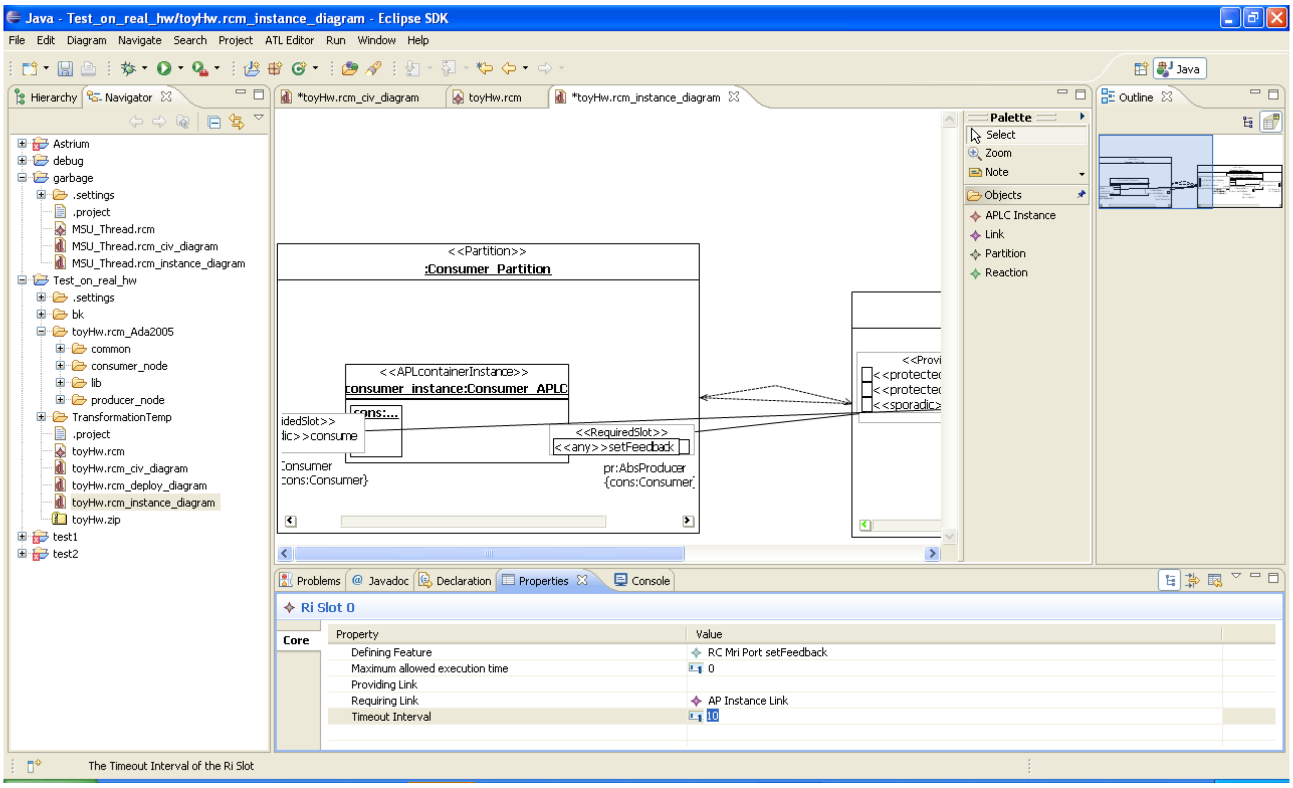
You can specify a **time-out** for the callback operation to *commence*. You may do so by specifying a handler to be automatically called when the time-out should expire: the handler is a method of the invocation object and is set in the column "Time-out handler" of the tab "Required Operations". When that required operation is invoked, a time-out is set on the invocation of its callback; if the time-out expires *before the callback is called*, the time-out handler is automatically invoked and executed.



Instance	Required Operation	Invocations	Auto Invoc...	Time-out handler
cs : Consumer	consume(i:Item)	1	0	setFeedback_timeout()
this	getFeedback()	1	0	

Figure 37: time-out handler

The time value for the time-out is set on the required slot for the desired required port at instance level through the attribute "Time-out interval".



Ri Slot 0

Core

Property	Value
Defining Feature	RC Mri Port setFeedback
Maximum allowed execution time	0
Providing Link	
Requiring Link	AP Instance Link
Timeout Interval	10

The Timeout Interval of the Ri Slot

Figure 38: time-out interval

The creation of a link between two slots should be denied if one of them has a deferred port with write-mode parameters as defining feature, unless a callback operation for the operation originating the required port exists. At present, this restriction is not implemented by the editor.

A.5.3 Reactivity links

It is possible to specify a concatenation of invocations to occur at user-specified time offsets from the arrival of a given deferred invocation. This enhancement feature is termed reactivity link**.**

Reactivity links can be used to program the execution of operations using a sporadic or cyclic operation as the base reference. Given a specific cyclic or sporadic operation, the user may specify a set of operations that have to be performed at a specified time offset from the release of the base reference. The base reference operation is called *action;* each programmed operation to be executed after the *action* are named *reactions*; the relationship between an *action* and one of its *reactions* is named *reactivity link*.

Given the distinction between APLC types and instances in the HRT-UML/RCM, *actions* and *reactions* have to be specified at type level. After this specification has been accomplished, *reactivity links* have to be specified at instance level between slots compatible with the specification established at type level.

A remote deferred operation with out parameters can be modeled using the "Reactivlty Link" model pattern by breaking down that operation in two parts: a "normal" deferred operation (with *no* out parameters) set as the *action*, which does the sending, and an immediate operation with the intended out parameter, set as the corresponding *reactivity link*, which does the receiving at a specified time offset from the relevant sending.

To design reactivity links: select an APLC, show Properties Tab, select Reactivity Links

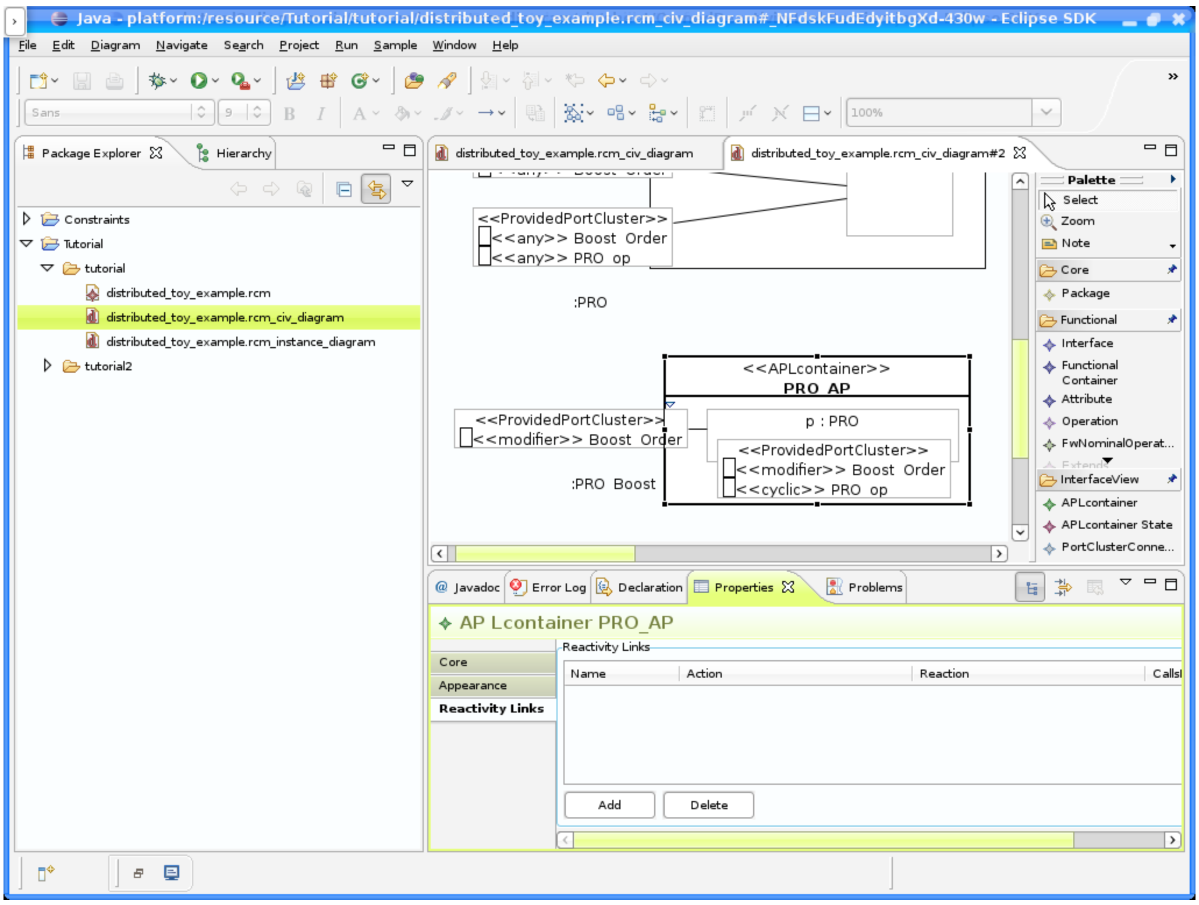
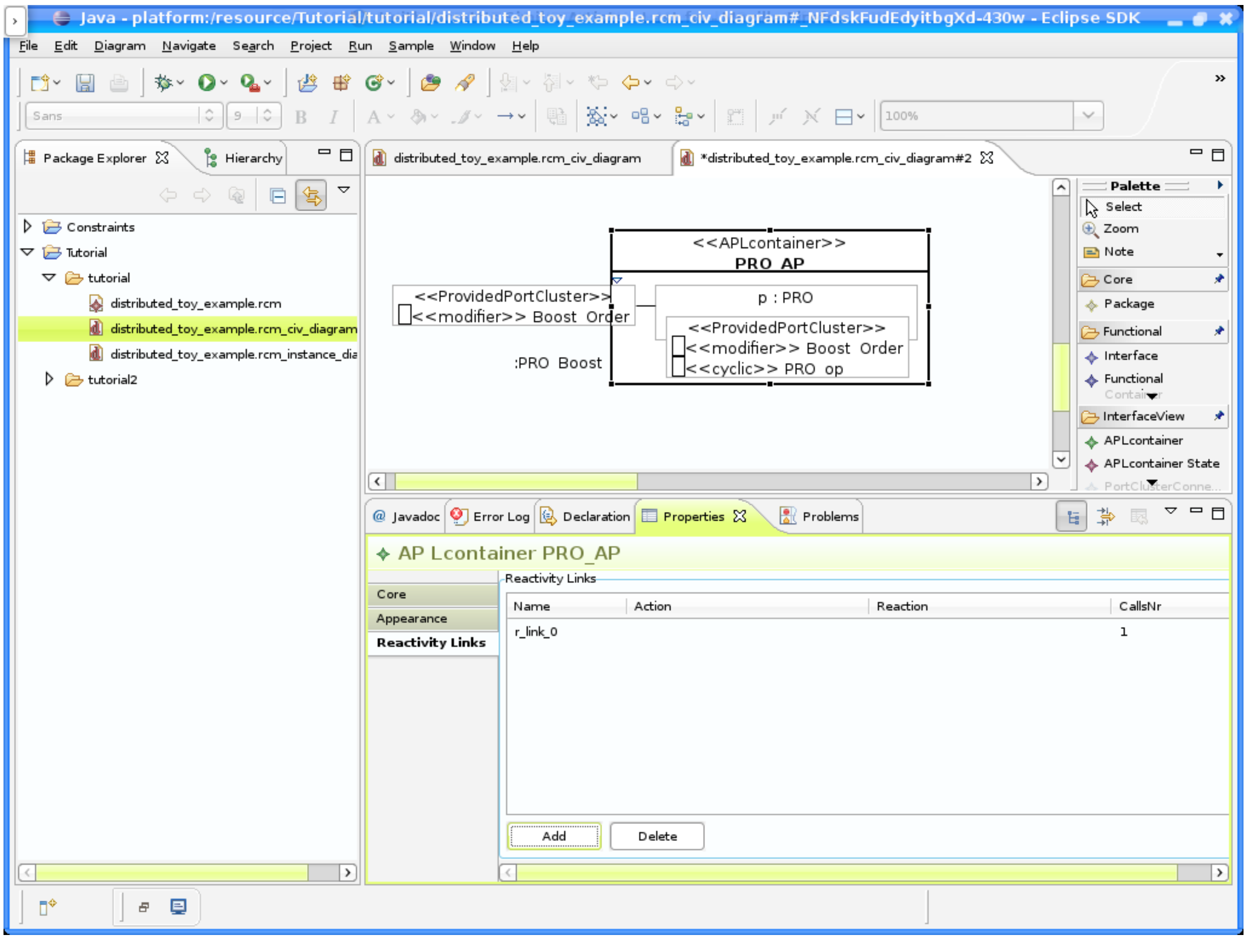


Figure 39: reactivity links, step1

Click on the Add button to add a new reactivity link to the selected APLC. The new link has a default name.

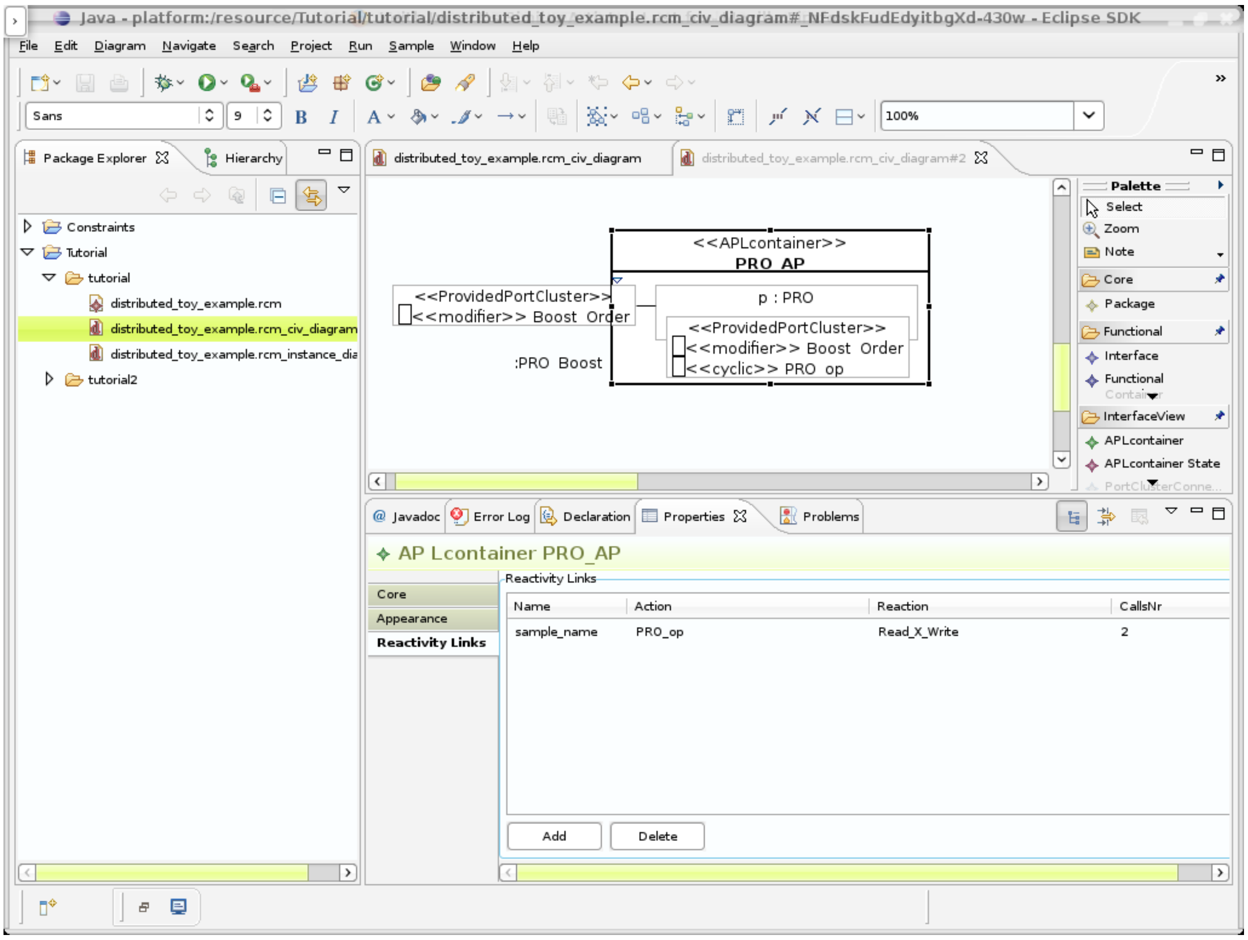


Reactivity Links

Name	Action	Reaction	CallsNr
r_link_0			1

Figure 40: reactivity links, step2

Set the name of the reactivity link, the action, the reaction, and the number of calls. The available actions are all the cyclic or sporadic services provided by the APLC. The available reactions are all the public operations in the whole system. The number of calls must be strictly positive.

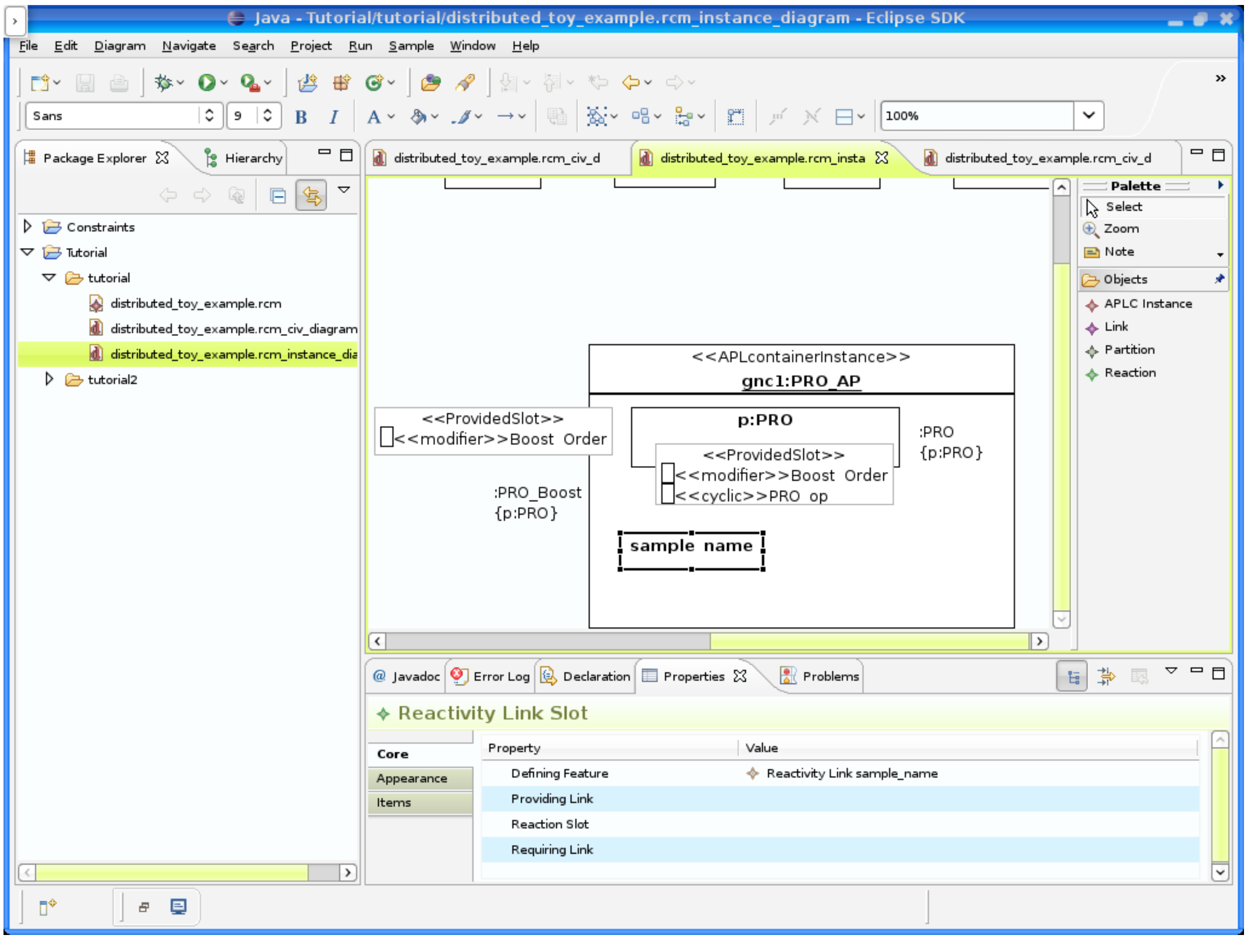


Reactivity Links

Name	Action	Reaction	CallsNr
sample_name	PRO_op	Read_X_Write	2

Figure 41: reactivity links, step3

Open the instance diagram for the model. All instances of the modified APLC will show a new element inside them: a reactivity link slot. If you don't see it, it may be hidden by an other element. It is usually placed in the center of its container.



Reactivity Link Slot

Property	Value
Defining Feature	Reactivity Link sample_name
Providing Link	
Reaction Slot	
Requiring Link	

Figure 42: reactivity links, step4

Select the Reaction tool on the palette, and draw a connection between the reactivity link slot and an elementary port slot of your choice. The tool will visually warn whether the pointed port is suitable for the connection. The destination slot's defining feature must be either a protected or unprotected port providing an operation compatible with that which was specified as "reaction" when the reactivity link was designed.

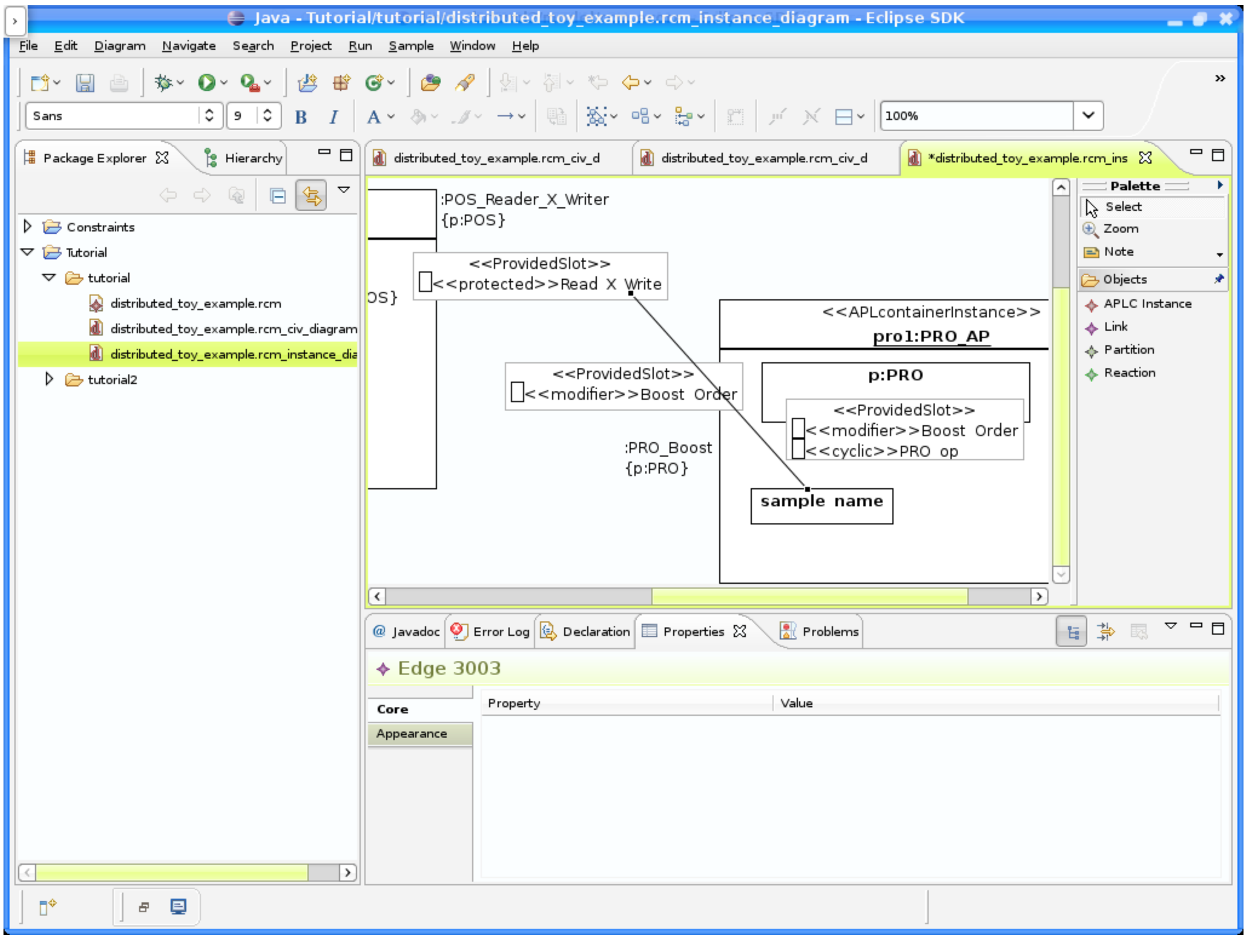
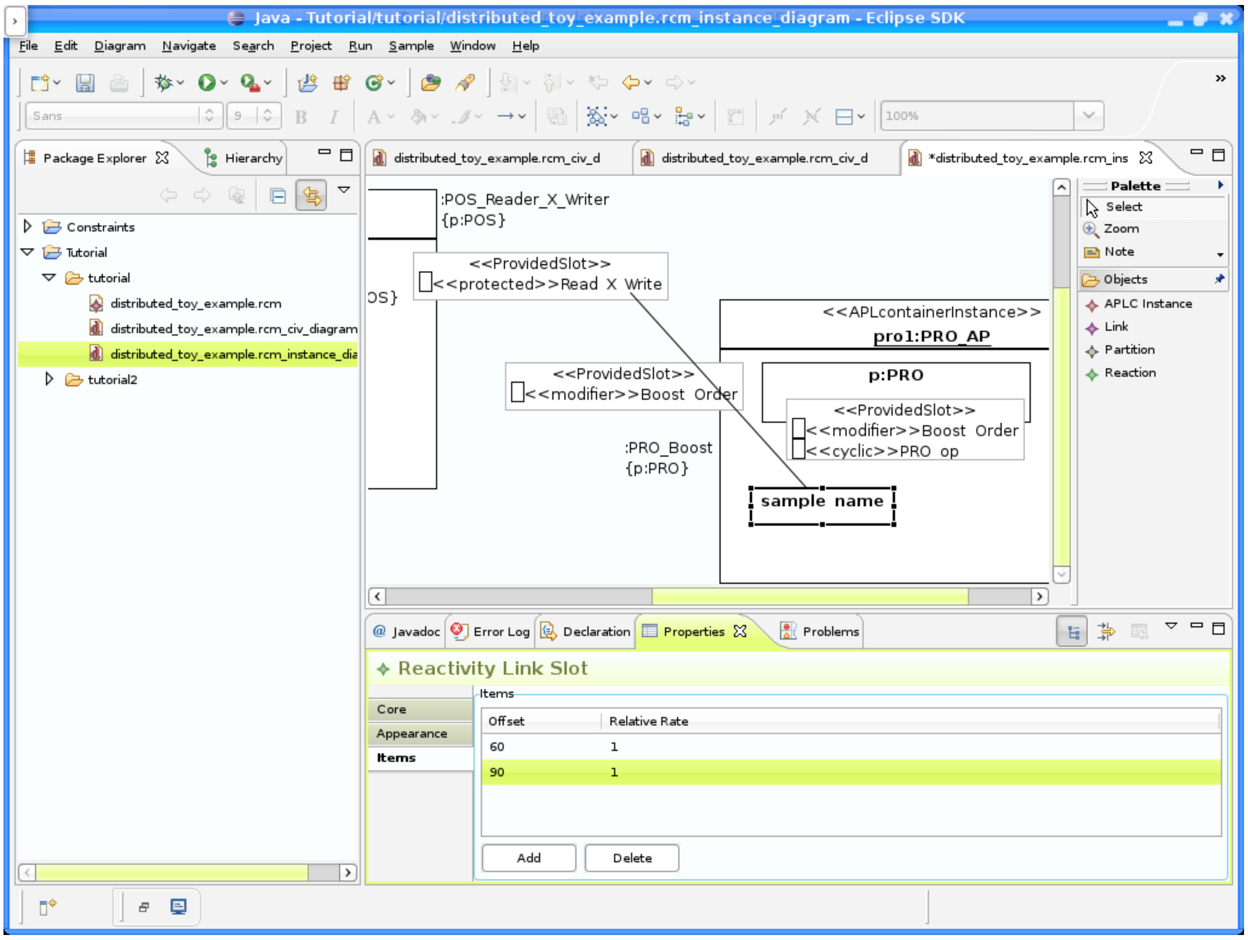

Java - Tutorial/tutorial/distributed_toy_example.rcm_instance_diagram - Eclipse SDK
File Edit Diagram Navigate Search Project Run Sample Window Help
Package Explorer
Hierarchy
Constraints
Tutorial
tutorial
distributed_toy_example.rcm
distributed_toy_example.rcm_civ_diagram
distributed_toy_example.rcm_instance_dia
tutorial2
:POS_Reader_X_Writer {p:POS}
<<ProvidedSlot>> <<protected>>Read X Write
<<APLcontainerInstance>> pro1:PRO_AP
<<ProvidedSlot>> <<modifier>>Boost Order
p:PRO
<<ProvidedSlot>> <<modifier>>Boost Order <<cyclic>>PRO op
:PRO_Boost {p:PRO}
sample name
Palette
Select
Zoom
Note
Objects
APLC Instance
Link
Partition
Reaction
Javadoc
Error Log
Declaration
Properties
Problems
Edge 3003
Core
Appearance
Property
Value

Figure 43: reactivity links, step 5

Finally select the reactivity link slot and click on the Items tab in the property sheet. The editor displays as many items as specified in the CallsNr field when the reactivity link was designed. For each item, specify the desidered offset from thread activation and the desidered relative rate.

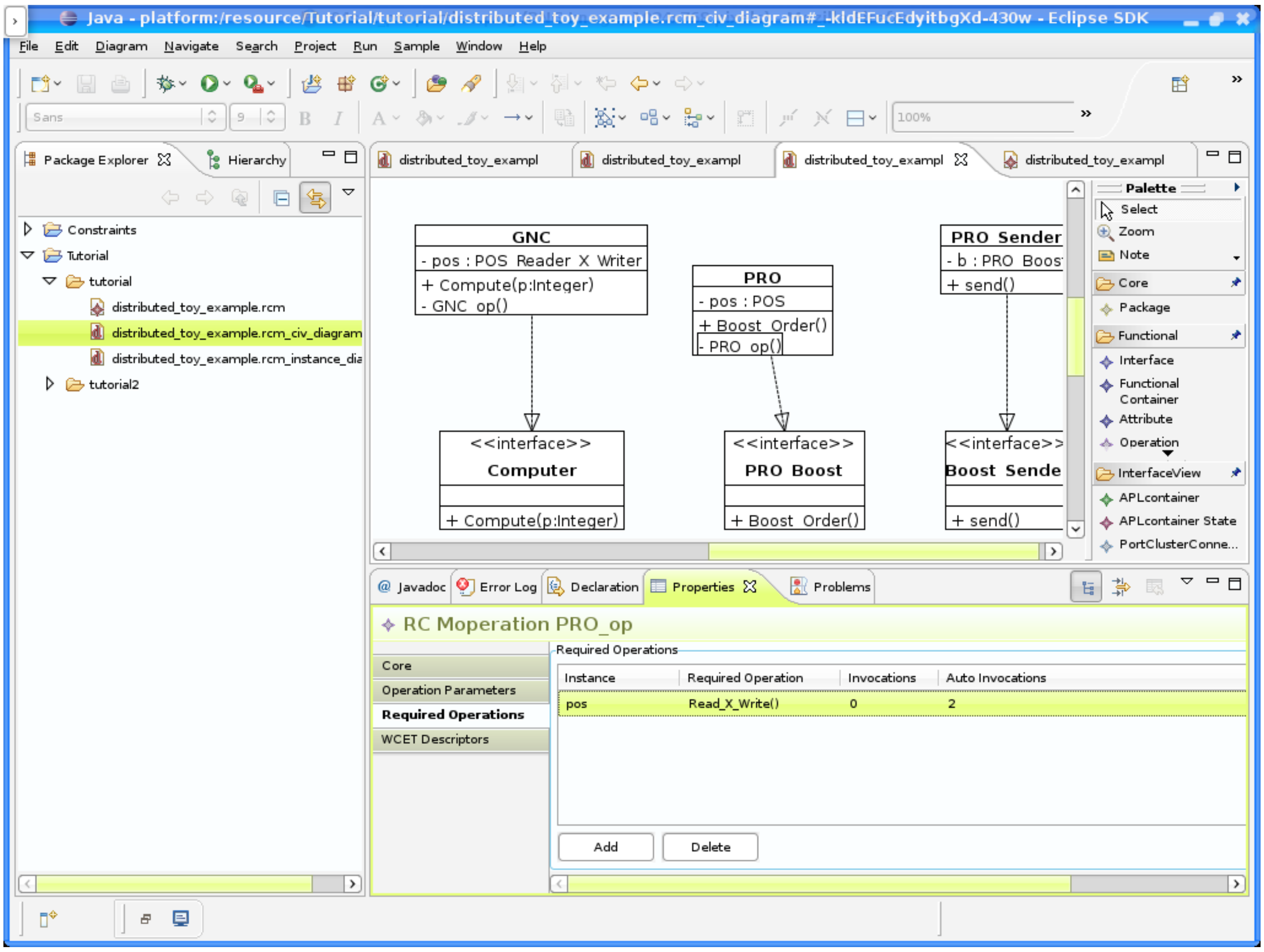


Items

Offset	Relative Rate
60	1
90	1

Figure 44: reactivity links, step 6

As an alternative to reactivity links defined on APLC, the methodology permits the user to define reactivity links at functional level too. This is done by selecting the operation whose execution has to trigger the reaction, and adding a new required operation descriptor targeting the desidered operation. The number of times the reaction operation is executed has to be specified in the "auto invocations" column. Instead of creating a dedicated required operation descriptor, it is possible to simply change the number of auto invocations of an already existing required operation targeting the desidered reaction. All restrictions on required operation descriptor hold even for required operation descriptors with autoinvocations: the only operations which may be required (i.e. target actions) are those operations which are accessible through a property of the classifier. In our example, we added an attribute typed POS to our PRO class, in order to access the Read_X_Write operation.



RC Moperation PRO_op

Required Operations

Instance	Required Operation	Invocations	Auto Invocations
pos	Read_X_Write()	0	2

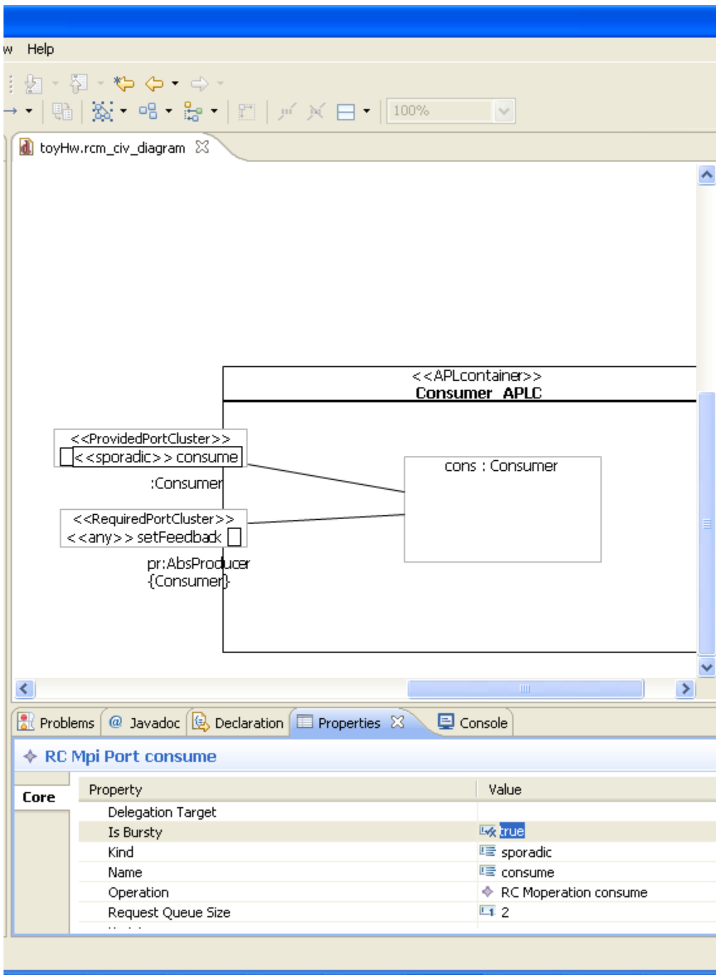
Figure 45: reactivity links, step 7

A.5.4 Sporadic operations with bursty activation

It is possible to allow sporadic operations to tolerate **bursty activations** without causing excessive pessimism in the feasibility analysis. A bursty activation occurs when for bounded time durations, the inter-arrival time between successive sporadic activations must be minimized in order for the service to cope with the inbound flow of invocations. After that time duration however, invocations become sparse and more spaced in time and therefore allow for a larger minimum inter-arrival time. In order that this type of sporadic operations do not cause excessive pessimism in the feasibility analysis, the need to cater for bursty activations must be specified in the model to be recognized in the analysis.

In contract with normal sporadic operations, a bursty sporadic operation permits the executing task to be released **a given number of times within a given time span** (called *burst interval*): within a burst interval the allowable sporadic activations may be so frequent and dense that the actual inter-arrival time between any two of them may be virtually null, which is an obviously illegal value for normal sporadic operations. A bursty sporadic operation may instead be executed, for a certain number of times at most, *without* enforcing its stipulated minimum inter-arrival time between successive releases. The analysis tools takes the particular behaviour of bursty activations into account and provides a precise evaluation of their lesser interference effect on the system.

In the Class Diagram, select a provided port marked as 'sporadic' and set 'isBursty' to true. The maximum number of consecutive releases allowed in the burst is specified in OBCSqueueSize.

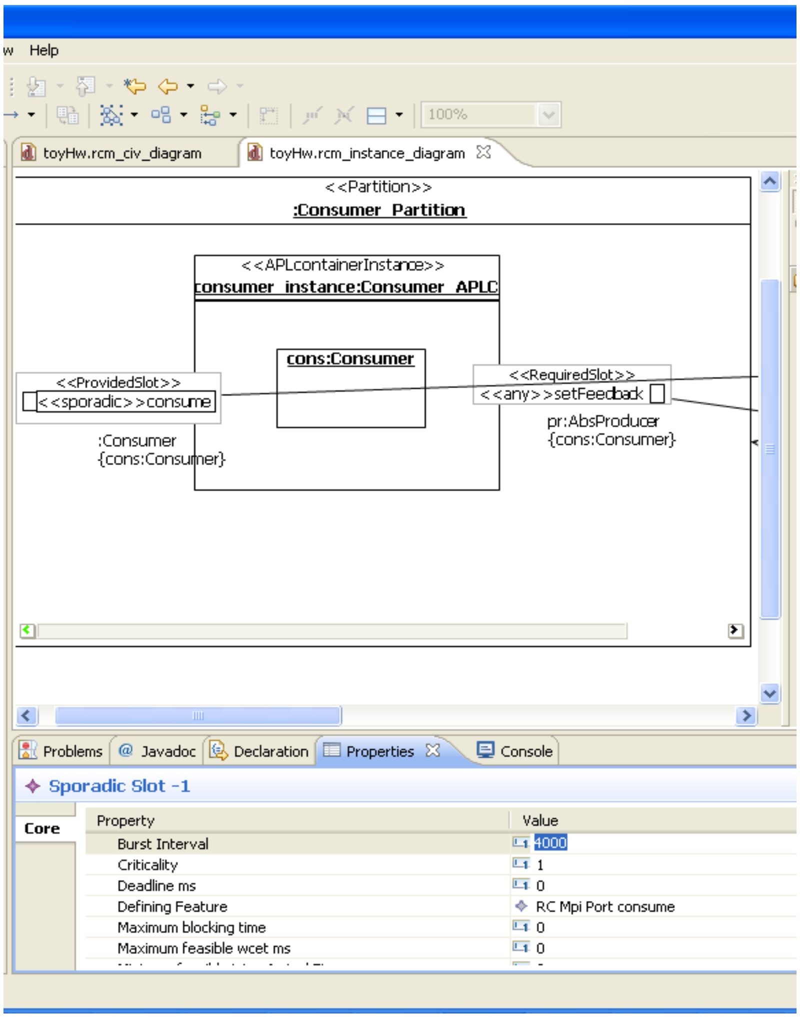


RC Mpi Port consume

Property	Value
Delegation Target	
Is Bursty	true
Kind	sporadic
Name	consume
Operation	RC Moperation consume
Request Queue Size	2

Figure 46: Setting sporadic as bursty

In the Instance Diagram, specify the burst interval in a sporadic slot. The minimum inter-arrival time can be set to zero.



Property	Value
Burst Interval	4000
Criticality	1
Deadline ms	0
Defining Feature	RC Mpi Port consume
Maximum blocking time	0
Maximum feasible wcet ms	0

Figure 47: Setting the burst interval

A.6 Deployment view Modelling

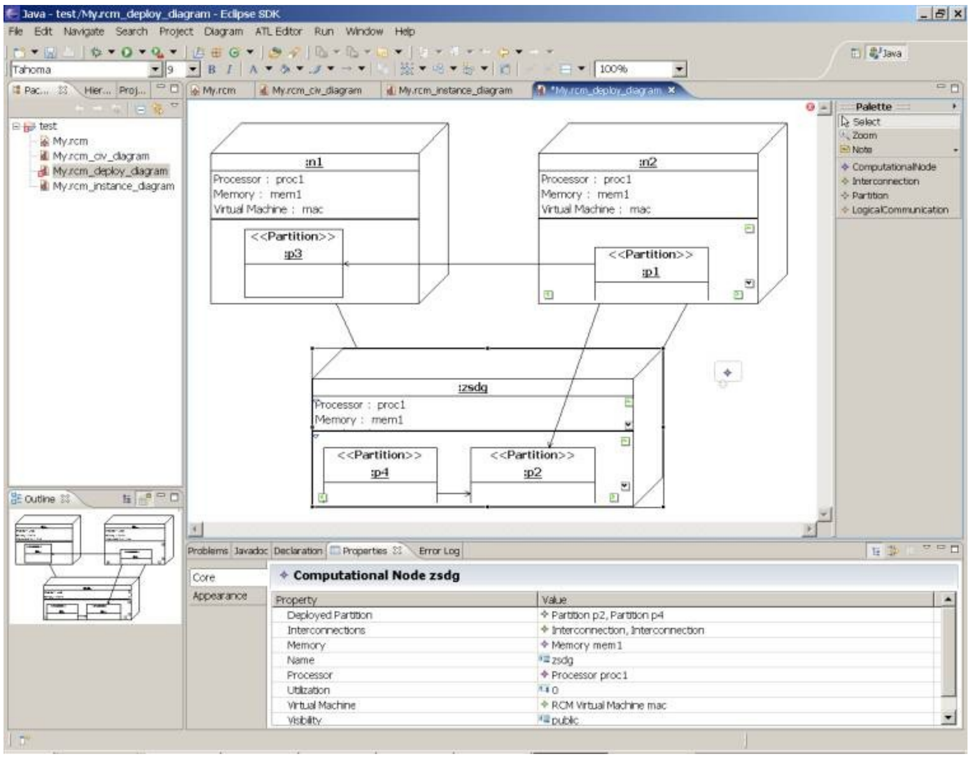
The HRT-UML plug-in allows to model the Deployment View. This view provides a description of: (1) the logical partitions and their mutual relations; (2) computational nodes and their physical interconnections; and (3) the intended mapping between both the logical communications specified in (1) and the physical interconnections specified in (2).

A.6.1 Deployment Diagram

In order to create a new Deployment Diagram :

– right-click on the .rcm model file and select *Initialize rcm_deploy_diagram diagram file* command;

– or go through File->New...->Other... and choose to create a new RCM Deployment Diagram from the HRT-UML2 folder. In this case a new model is also created by default.



Computational Node zsdg

Property	Value
Deployed Partition	Partition p2, Partition p4
Interconnections	Interconnection, Interconnection
Memory	Memory mem1
Name	zsdg
Processor	Processor proc1
Utilization	0
Virtual Machine	RCM Virtual Machine mac
Visibility	public

Figure 48: The deployment view editor

The Palette buttons are used to create:

- the Computational Nodes;
- the Interconnections between them;
- the Partitions that belong to the nodes and
- the Logical Communications that link them
- the different attributes of the node, to be create in the attribute compartment of a node (see figure 6.3) : Processor, Memory, RCMVirtualMachine.

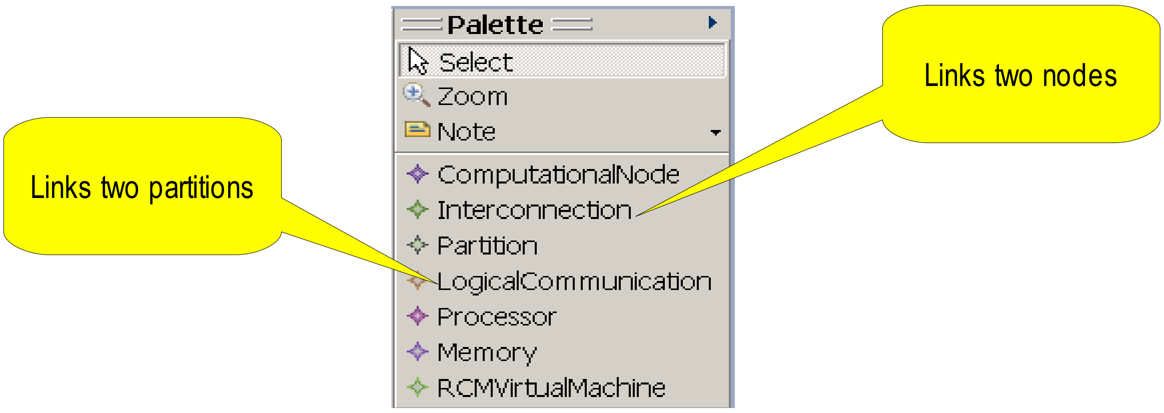


Figure 49: Deployment View Palette

In the following paragraphs, a description of the deployment diagram entities is given.

A.6.1.1 Computational Node

The computational node is represented by a cube with name, attribute and partition compartments.

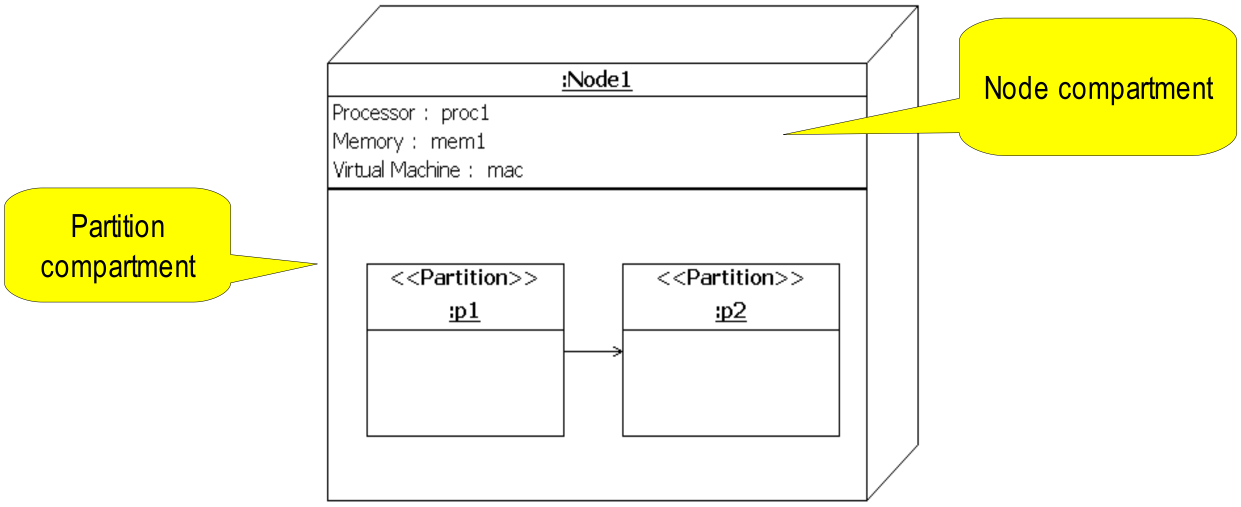


Figure 50: Computational Node representation

A.6.1.2 Partitions

Partitions may be created inside the Partition Compartment of a Node using the “Partition” button of the Palette. A new partition is then created in the .rcm model associated to the deployment diagram.

If a partition was previously created in the model, it can be allocated to a node when in the Deployment View. To do this select the node you desire to assign the partition to and in the Properties view of this node, open the Deployed Partition list.

To a partition can also be allocated an APLContainer Instance. The latter however must be previously created in the Instance view, and it's type (classifier), created in the Class Diagram (civ).

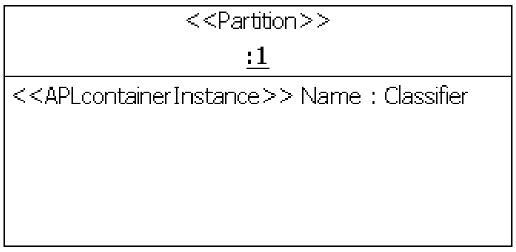
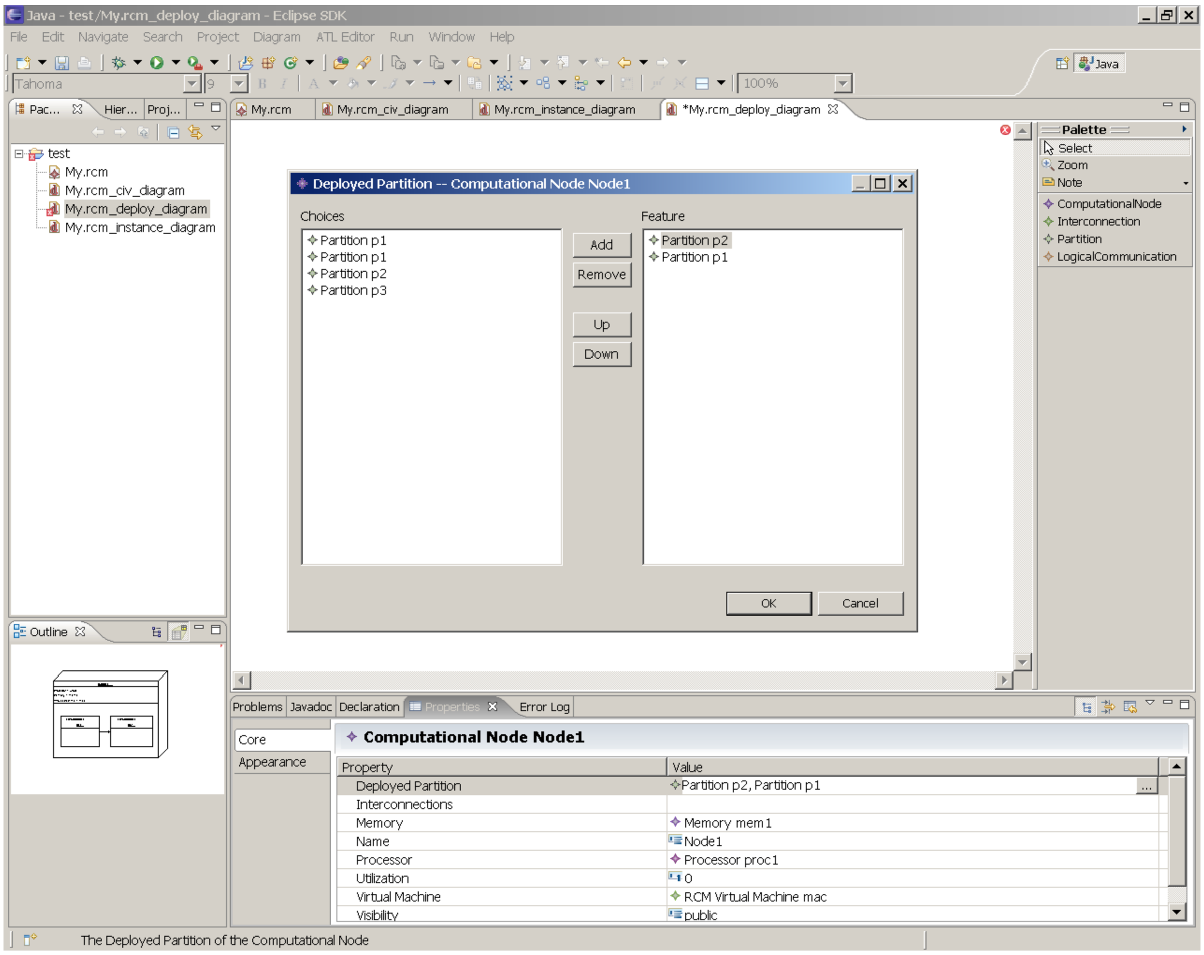


Figure 51: Partition with APLContainer Instance



Deployed Partition -- Computational Node Node1

Choices		Feature
Partition p1	Add	Partition p2
Partition p1	Remove	Partition p1
Partition p2	Up	
Partition p3	Down	

OK Cancel

Computational Node Node1

Property	Value
Deployed Partition	Partition p2, Partition p1
Interconnections	
Memory	Memory mem1
Name	Node1
Processor	Processor proc1
Utilization	0
Virtual Machine	RCM Virtual Machine mac
Visibility	public

The Deployed Partition of the Computational Node

Figure 52: Selecting a partition for a node

The list of partitions in the left compartment of the window that appears on the screen shows the partitions previously created in the model. Selected the partitions you wish to assign to your node and click on “add” then “Ok”. Use this same method to take away a partition from a node. Selecting a partition on the editor and hitting “Del” on the keyboard deletes it from the editor as well as from the model.

A.6.1.3 Logical Communication

Logical Communications between partitions can be created using the appropriate button from the Palette, as well as through the .rcm model view. Constraints for this element have yet to be implemented, and their behaviour to be adjusted.

A.6.1.4 Logical Communication constraints

Between two partitions can be created at the most two Logical Communications, each of them having opposite directions. Moreover a partition cannot have a Logical Communication starting and finishing on itself. However a partition may communicate with as many other partition as desired.

The implemented constraints will prevent the user from creating Logical Communications failing these rules.

A.6.1.5 Visible attributes of a Node

The Attribute compartment of a Computational Node may show the Processor, Memory and RCM Virtual Machine associated to that node. To do this, to possibilities are offered:

- previously creating them in the .rcm tree model under Deployment Information-> Physical Architecture->Physical Types (right click on Physical Types, then New Child, and choosing between Processor, Memory and RCMVirtualMachine). Once created, they can be chosen from a list in the Deployment view editor as for partitions, going through the Properties view of a Node and selecting the desired attribute in the list available;
- creating them directly in the Deployment View : buttons from the Palette are provided for creation of the attributes. After having selected the attribute, click on the Attribute compartment of the node you desire allocating the attribute to.

Once the attributes created, you can select them from the node to change their name, but also to set all other parameters of the attribute in the Properties view.

A.6.1.6 Interconnection constraints

A series of constraints have been implemented for the interconnections between nodes.

An interconnection cannot be created starting and ending on the same node (a node cannot be directly connected to itself). This constraint is dynamic, i.e. the editor does not let you create such a link.

The second constraint will appear as an error upon validation of the diagram, if not complied with : there must exist a path linking all nodes of the deployment graph together. To validate, simply click on Diagram->Validate in the Eclipse menu, after saving. If an error occurs, it is signalled by a white cross circled in red on the top right corner of the diagram as well as in the Problems view of the Eclipse workspace. The red cross is not always reliable, whereas the Problems view is, so be careful to control this view for a trustworthy diagnosis.

A.7 Model validation

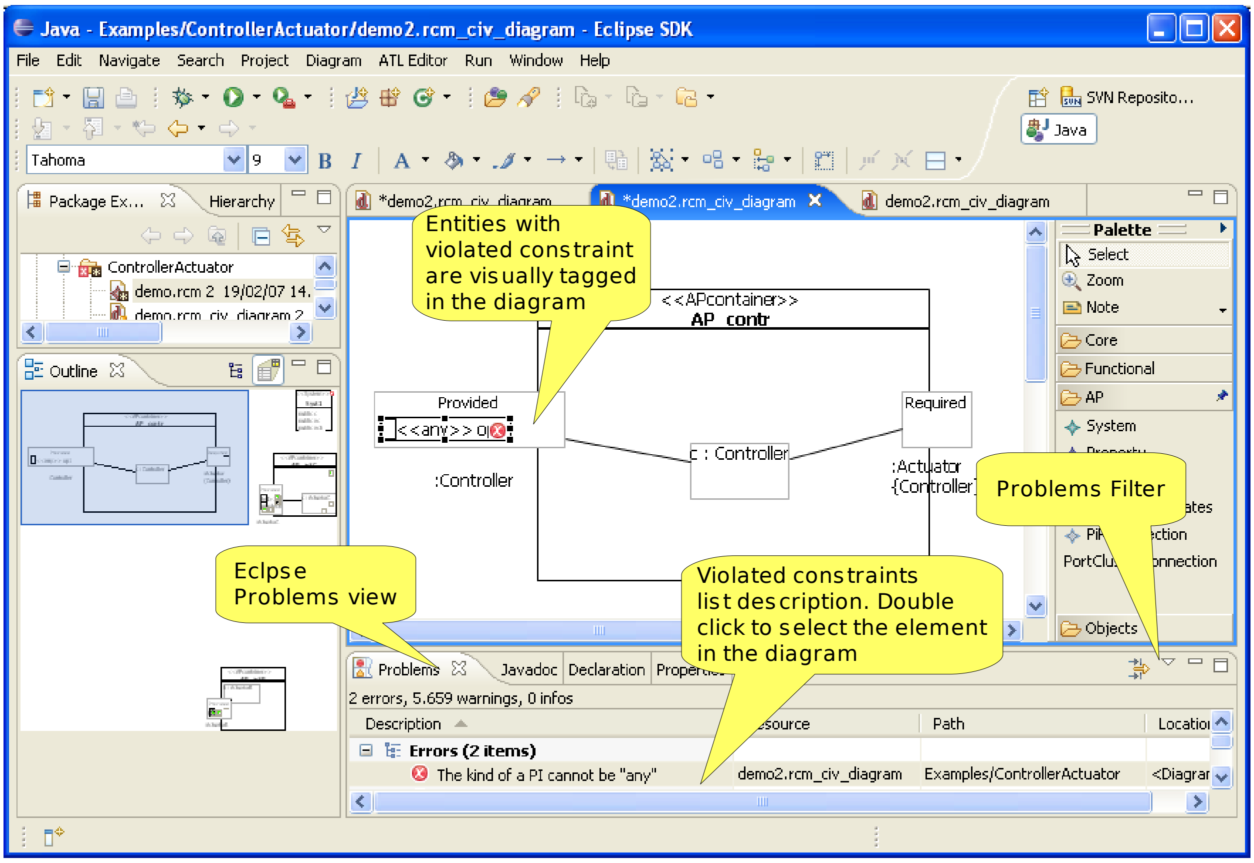
HRT-UML2 plug-in allows creating RCM models accordingly to the **correctness by construction** development; in this way it is guaranteed by construction that many RCM constraints [see Appendix A] cannot be violated.

Moreover several constraints are implemented as “live constraints”, i.e. they are automatically checked by the tool before committing a particular modification on the model. Actions that would invalidate these constraints are rolled back.

However not all the constraints defined by the RCM methodology can be imposed at model construction time, so a check to be invoked upon a populated model is required to validate the model.

To invoke the validation on a model you can open a diagram and select the command *Diagram -> Validate* from the Eclipse menu. So doing the violated constraints appears in the Eclipse Problems view; moreover all the entities having some problem with the validation are graphically marked with a red icon in the diagram.

To make the navigation of these entities easier it is possible to click a raw in the Problems View table to automatically retrieve and select the associated invalidated element in the diagram.


Java - Examples/ControllerActuator/demo2.rcm_civ_diagram - Eclipse SDK
Entities with violated constraint are visually tagged in the diagram
Eclpse Problems view
Violated constraints list description. Double click to select the element in the diagram
Problems Filter
<<APcontainer>> AP_contr
Provided
Required
:Controller
c : Controller
:Actuator {Controller}
Palette
Select
Zoom
Note
Core
Functional
AP
System
Objects
Problems
Javadoc
Declaration

2 errors, 5.659 warnings, 0 infos

Description	Resource	Path	Location
Errors (2 items)			
The kind of a PI cannot be "any"	demo2.rcm_civ_diagram	Examples/ControllerActuator	<Diagrar

Figure 53: validating the class diagram

Using the Problem Filter view (see figure 53) you can select the filter to apply to the current validation, e.g. to show problems related to the current diagram only or for the entire model.

A.8 Model transformation

The tool integrates the model transformations developed in the context of the DDHRT cluster: the round-trip analysis, the code generation and the interface view to concurrent view transformation.

A.8.1 Round-trip analysis

The round-trip analysis is started by the user from the interface view and it is performed on the concurrency view. The concurrency view model is automatically obtained from the interface view through a model to model transformation.

The goal of the round-trip analysis is to execute the feasibility and sensibility analysis on the concurrency view and then return the results back to the interface view entities for the user to inspect them. For an exhaustive explanation of this process please refer to [DDHRT3-1.TN.10].

In the current version of the HRT-UML2 tool the round-trip analysis is performed in three separated steps:

1. the concurrency view is obtained and the model to be given in input to MAST+ tool is generated
2. the analysis is performed using MAST+ tool
3. the analysis results are loaded back into the initial RCM model

To start the process, right click on the .rcm model file owning the interface view and select one of the available analysis listed in *Round-trip Analysis -> Generate MAST+* input file menu:

1. Classical Response Time Analysis
2. Feasibility Analysis
3. Sensitivity Analysis
4. Holistic Analysis

The following constraint has to be satisfied to perform the Classical Response Time Analysis, Feasibility Analysis and Sensitivity Analysis: all instances are deployed on partitions which are associated within a single computational node.

The tool generates the concurrency model and than performs other transformations.

The result of this first step is a MAST+ input file stored in the temporary folder named *RoundTripAnalysisTemp* created in the current Eclipse resource: this file has "*.mastInput*" extension.

The .mastInput file obtained is then used to perform the selected analysis with MAST+. When this analysis is executed an ouput .xml file is obtained. This file is then used to load back the results in the starting RCM model that has been used to generate the MAST+ input file.

When the loading is terminated a report is showed telling if the system is feasible or not and listing the founded problems. Then you can decide to save the RCM model updated with the analysis results in the same .rcm file or save it in a different one.

A.8.2 Generating Ada2005 code

A model to code transformation generates Ada2005 code from an RCM model.

Given an RCM model it is possible to generate code for the functional view only or generate code for the entire model.

To start the generation right click on the *.rcm* model file and from the *Code Generation* menu select:

1. *Generate Ada 2005 for Functional View* command to generate code from functional view only, or
2. *Generate Ada 2005 code* command, or
3. *Generate Ada 2005 Heterogeneous Functional Models Code* command (see next chapter),

Currently the code is generated in a default directory created at the same level of the .rcm file. The name of the directory is *<name of the .rcm file>_ADA2005*.

⛔ In order to safely generate code a WCETdescriptor has to be created for every operation; it is possible to leave the WCETdescriptor.WCET attribute to its default (i.e. maximum) value.

A.8.3 Code Generation for modelling tools integration (SCADE, SDL..)

HRT-UML/RCM toolset allows to generate code for VMLC's using heterogeneous functional models as their OPCS: this can be achieved using the *Generate Ada 2005 Heterogeneous Functional Models Code* generation command. The functional models may be designed with SCADE or SDL and their code-level implementation generated by the code generation engines provided with the design tool, or they can be manually written in C/C++ or Ada: this information has to be specified for each RCMfunctionalContainer using the *implementation language* attribute (see A.2.1.1).

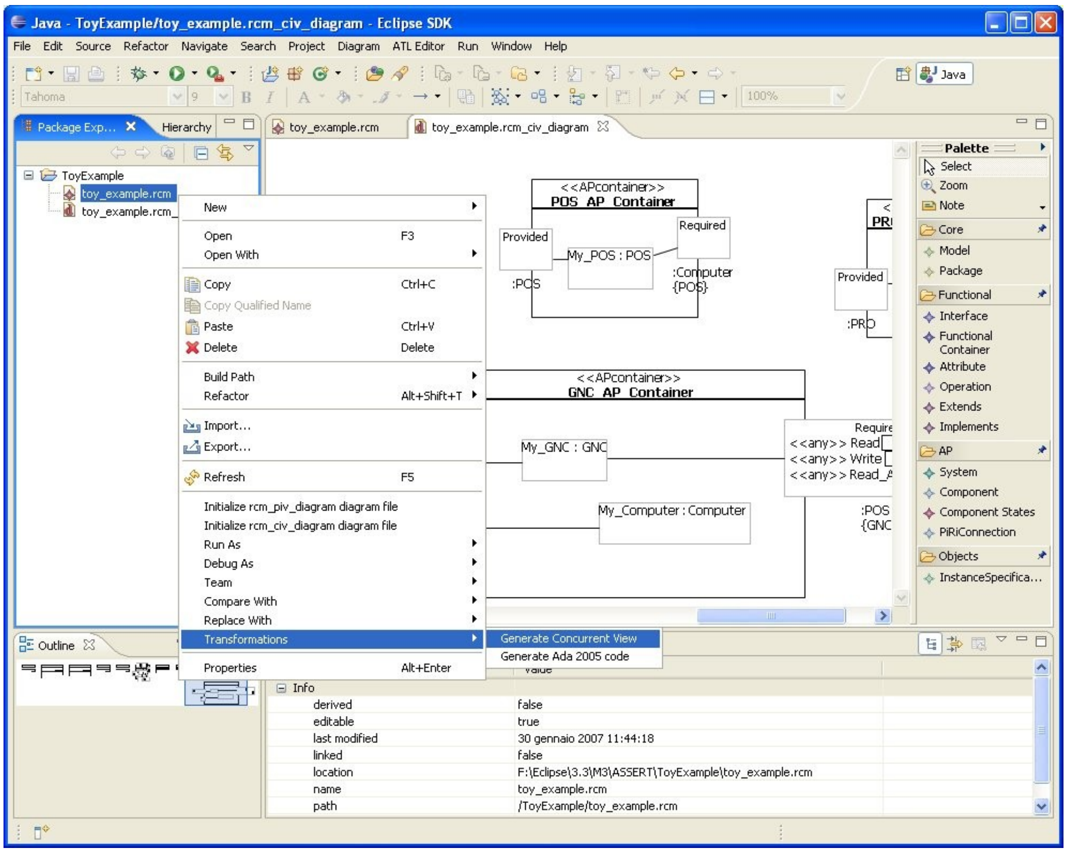
The communication between heterogeneous functional models requires the definition of a platform independent Data View and the generation of ASN.1 (un)marshallers and "glue code" with the tools developed in ASSERT [SEM07]. See also Appendix D for related information.

In case of heterogeneous code generation an *.aadl* file is also generated under the default directory *<name of the .rcm file>_ADA2005*: this file gives a description of the RCMoperations using the AADL language, it has to be used to drives the "glue" generation performed by the ASSERT builder tool developed by Semantix (Appendix D).

A.8.4 Interface View to Concurrent View

This transformation refines the RCM model by the generation of the concurrent view starting from the interface view:

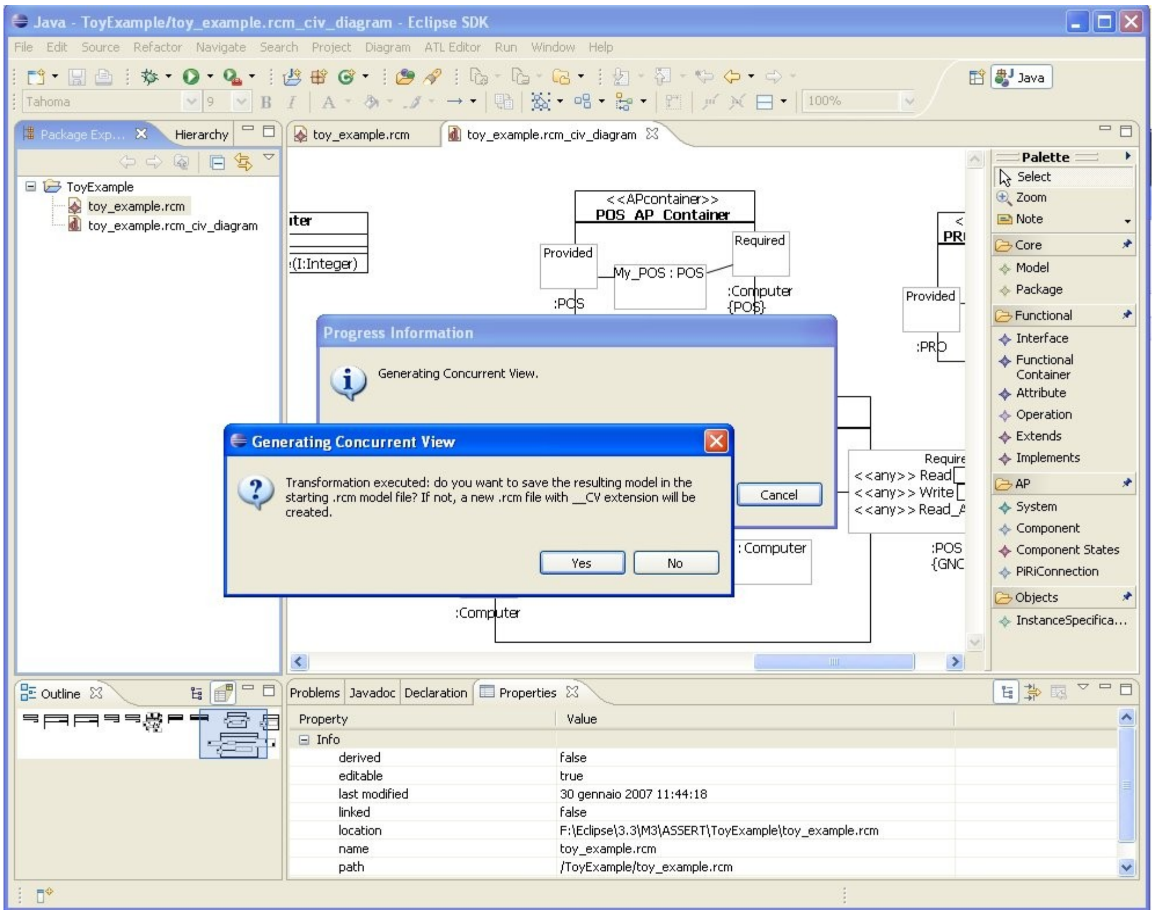
- right click on the .rcm model file and select *Generate Concurrency View* command



Property	Value
Info	
derived	false
editable	true
last modified	30 gennaio 2007 11:44:18
linked	false
location	F:\Eclipse\3.3\M3\ASSERT\ToyExample\toy_example.rcm
name	toy_example.rcm
path	/ToyExample/toy_example.rcm

Figure 54: generating Concurrent View

When the transformation terminates the following dialogue window appears:



Generating Concurrent View.

Generating Concurrent View

Transformation executed: do you want to save the resulting model in the starting .rcm model file? If not, a new .rcm file with __CV extension will be created.

Yes No

Property	Value
Info	
derived	false
editable	true
last modified	30 gennaio 2007 11:44:18
linked	false
location	F:\Eclipse\3.3\M3\ASSERT\ToyExample\toy_example.rcm
name	toy_example.rcm
path	/ToyExample/toy_example.rcm

Figure 55: Generating Concurrent View dialogue window

It is preferable to answer "**No**" and let the tool write the transformed/extended model into a new .rcm file. The new model file is created under the current Eclipse resource and its name it is tagged with the "__CV" extension.

Appendix B Current Version Limitations

In the following a list of other relevant current tool limitations and unsupported features is given.

Functional View

- Checks on the namespace visibility rules are partially implemented by the tool. For instance drag and drop of entities between packages can invalidate visibility rules.
- To be able to delete an item located inside a compartment (e.g. an operation or a property of a class) be sure to select the item just with one mouse click; in fact the delete command is not available when the item is clicked two or more times and a rectangle appears around it.

Interface View:

- A port cluster connection made at class level is not automatically reflected at instance level.

Functional View and Interface View Synchronization

Most of the modification actions on the functional view are automatically reflected by the tool in the interface class and instance view. For instance it is possible to add/remove an operation on an *RCMfunctionalContainer* referenced by an *APLcontainer* and, for the latter, have the corresponding elementary provided port added/removed in the proper provided port cluster; the same for *RequiredOperation* (in the functional view, the cause) and required ports (in the interface view, the effect).

PortCluster figures for an APLC can disappear during the synchronization phase: in this case you can perform a "*Show all port cluster*" action on the APLC to make them visible again.

There are some kind of modifications on the functional view that can resolve into an invalid state for existing *RequiredOperations* (and so required port in the Interface view): these problems about invalid *RequiredOperation* have to be fixed by the user in the functional view in order to let the tool to synchronize the interface view accordingly. More properly, for a *RCMfunctionalContainer*, the following actions:

- altering the inheritance hierarchy of a required type, i.e a type referred by an owned *RequiredOperation* entity
- modifying the visibility of an operation referred by an owned *RequiredOperation* entity

can invalidate the *RequiredOperation* of the *RCMfunctionalContainer*. After performing one of the previously action it is safe to:

- run a validation of the model

- search for *RequiredOperation* problems in the *Problems View* and fix them (e.g. deleting the *RequiredOperation* or setting a new valid Operation for it). By fixing all of these problem the interface view will be correctly updated and synchronized.

Regarding the interface instance view: problems with missing connection for an *RiSlot* (i.e instance of required elementary port) can appear after a model validation (for instance this happens if a new required elementary port is added to a required port cluster that, at instance level, is already linked and satisfied); in this case the problem can be rapidly solved by deleting and then creating the outgoing link for the port cluster owning the invalid *RiSlot.*

Instance View:

- Editing commands on the structure are available but they don't have to be used, given that an *APLcontainer* instance structure is automatically managed by the tool according to its typing classifier.

XPath/UUID in model and diagram XML files

The tool (more properly the EMF plug-in), in its default configuration, use the XPath language to trace references between elements in the XML files. The tool also supports the UUID mechanism as alternative: to set(unset) the usage of UUID open the Eclipse window

Window->Preferences->Run/Debug->StringSubstitution

and set the *UseUUID* variable to 'true'('false'). You can switch the usage of UUID on and off for the same model with no limits. So for instance you can have XPath, then UUID and then XPath again.

Currently the round-trip analysis process (i.e. the ATL transformation) is not able to read the UUID informations on the .rcm model given in input to the process. In this way if the starting .rcm file is overwritten at the end of the round-trip process, the diagram files having UUID as model elements references result damaged; so, you must use diagrams (and so model) with the XPath system if you want to avoid this situation.

General

- **Time** unit for HRT attributes is *ms* and it is not configurable.
- **Validation**: see Appendix A for the current status of the RCM constraints definition and implementation

Appendix C RCM Constraints

Core

NamedElement

Name	Mode	Severity	Description	Status	Comment
NonEmptyNames	Live	Error	Name length must be at least 1.	Proposed	
SimpleNames	Live	Error	Names may only contain letters, digits, or underscores [a-z] or [A-Z] or [_] or [0-9]	Proposed	
NonDigitNames	Live	Error	First character of the name must be a letter [a-z] or [A-Z]	Proposed	
NonAdaKeywordNames	Live	Error	Name must not be one of the following: abort, abs, abstract, accept, access, aliased, all, and, array, at, begin, body, case, constant, declare, delay, delta, digits, do, else, elsif, end, entry, exception, exit, for, function, generic, goto, if, in, interface, is, limited, loop, mod, new, not, null, of, or, other, out, overriding, package, pragma, private, procedure, protected, raise, range, record, rem, renames, requeue, return, reverse, select, separate, subtype, synchronized, tagged, task, terminate, then, type, until, use, when, while, with, xor. All keywords are case insensitive.	Proposed	
UniqueCaseInsensitiveNames	Live	Error	No two elements in the set may have the same name, or names whose difference is only a matter of uppercase/lowercase.	Proposed	
UniqueCaseSensitiveNames	Live	Error	No two elements in the set may have the same name.	Proposed	
CatchAllNames	Live	Error	The name of entities at the same level has to be unique, not empty and not null.	Building	Not implemented for Interconnection! Currently implemented for: Packageable entity, Computational

Name	Mode	Severity	Description	Status	Comment
					Node, DeployableInstance, Partition, Computational Node, PhysicalTypes entities.

Interface View

APLC

Name	Mode	Severity	Description	Status	Comment
realizingComponent	CbyC		Not user-accessible	Completed	
orderedAttribute	CbyC		Not user-accessible	Completed	
RiPortClusters	CbyC		For every statereference S owned by this APLC A, A owns a dedicated* RCMriPortCluster for every property P owned by the type T of S, typed to the type K of the property, as long as the P is required by any of the methods implemented by T or any of its superclasses. Moreover, if any of those require an invocation on "this", A owns a dedicated* RCMriPortCluster for "this". [*: every RCMriPortCluster is associated to one and only one statereference and to one and only one property owned by the type of that statereference, except the cluster for "this", which is not associated to any property]	Completed	
PiPortClusters	CbyC		For every statereference S owned by this APLC A, A owns a dedicated* RCMpiPortCluster for every type T implemented by S, typed to T. [*: every RCMriPortCluster is associated to one and only one statereference]	Completed	
AutoAssemblies	CbyC		If this APLC owns a RCMriPortCluster with a statereference S of type T but not for any particular property of T, and therefore associated with S itself as "this", then there exists an RCMassembly starting from the RCMriPortCluster and ending to the only RCMpiPortClusterof type T associated with S.	Completed	

Name	Mode	Severity	Description	Status	Comment
ClonedPorts	CbyC		Whithin every single APLC, all the RCMpiPorts which refer the same* operation, which are owned by portclusters of the APLC which refer the same statereference, must have the same values for the following attributes: kind, variates, variators. [*: the two operations must be the same operation, or one of them must override/implement the other one]	Building	Variators are not cloned yet
isEventManager	CbyC		Not user-accessible	Completed	
isTimerManager	CbyC		Not user-accessible	Completed	

Component

Name	Mode	Severity	Description	Status	Comment
requiredTypes	CbyC		Not user-accessible	Completed	
providedTypes	CbyC		Not user-accessible	Completed	
publicTypes	CbyC		Not user-accessible	Completed	
distributionRole	CbyC		Not user-accessible	Completed	
ownedConnection	CbyC		Not user-accessible	Completed	

Structured Classifier

Name	Mode	Severity	Description	Status	Comment
ownedConnector	CbyC		Not user-accessible	Completed	
ownedPortCluster	CbyC		Not user-accessible	Completed	

RCMpiPortCluster

Name	Mode	Severity	Description	Status	Comment
PPCtype	CbyC		For every operation owned by the type of this portcluster PC, PC owns a dedicated RCMpiPort associated with that operation.	Completed	

RCMriPortCluster

Name	Mode	Severity	Description	Status	Comment
RPCtype	CbyC		For every operation owned by the type of this portcluster PC, if the operation is required by an operation provided by the type of the statereference associated with	Completed	

Name	Mode	Severity	Description	Status	Comment
			PC, PC owns a dedicated RCMriPort associated with that operation.		

ReactivityLink

Name	Mode	Severity	Description	Status	Comment
callsNr	Live	Error	value must be strictly positive	Confirmed	
action	Live	Error	Action must be a nominal PI provided by the owner of this ReactivityLink, and it must be a "concrete" port	Confirmed	
reaction	Live	Error	Reaction must be an operation with public visibility, or a different visibility but in that case the target of the associated ReactivityLinkSlot is subject to further restrictions.	Proposed	

RCMpiPort

Name	Mode	Severity	Description	Status	Comment
ConcreteConcurrentTypes	Batch	Error	RCMpiPort kind must be one of the following {cyclic, passive, sporadic, modifier, protected}	Completed	
VariatorVariates	Batch	Error	A variator must refer a nominal operation	Completed	
CouplesOfPorts	Batch	Warning	If two ports "share" the same variable on the same state of the same APLC, then they can only be: {cyclic modifier} {sporadic modifier} {modifier modifier} {protected protected} {unprotected unprotected}	Completed	
QueueSize	Live	Error	RCMpiPort.requestQueueSize: its value must be greater or equal to 2.	Completed	

FunctionalView

Required Operation

Name	Mode	Severity	Description	Status	Comment
autoInvocations	Live	Error	value must be non-negative	Confirmed	

Classifier

Name	Mode	Severity	Description	Status	Comment

ownedInstantiationParameter	CbyC		Not user-accessible.	Completed	

Interface

Name	Mode	Severity	Description	Status	Comment
PropertiesInInterfaces	CbyC		An Interface may not have any property	Proposed	
NoFwNominalOperation	CbyC		An interface may not contain FwNominalOperation	Proposed	

RCMoperation

Name	Mode	Severity	Description	Status	Comment
callbackCaller	CbyC		Must be a sister non-abstract RCMoperation (either inherited or locally defined)	Completed	
callbackReceiver	CbyC		Must be a RequiredOperation owned by the callbackCaller operation.	Building	
callback	Live	Error	Either both callbackCaller and callbackReceiver are null, or none of them is null	Building	Provided in Batch mode.
NoRecursion			no operation x can feature a required operation which contains x in the closure of its required operations	Proposed	
PositiveWcet	Batch	Error	A concrete operation of a RCMfunctionalContainer must have a WcetDescriptor for every VM in the system, and its wcet must be > 0	Completed	.
actionOf	CbyC		Not user-accessible	Proposed	
reactionOf	CbyC		Not user-accessible	Proposed	
isAbstract	Live		From false to true is denied if an RCMpiPort referrring the operation exists.	Proposed	

FunctionalContainer

Name	Mode	Severity	Description	Status	Comment
InterfaceMethodsCbyC	CbyC		For every method of every interface a class implements, the class must have a method which implements that method.	Building	this is implemented just when "adding" an implemented interface
InterfaceMethodsBatch	Batch		For every method of every interface a class implements, the class must have a method which implements that method.	Completed	

AbstractMethods	CbyC		For every abstract method of every abstract class a class directly* extends, the class must have a non-abstract method which implements that method. [*: not when superclass relation with abstract class is due to the extension of a concrete class]	Proposed	
isAbstract	Live		Changing to "true" is denied if an APLcontainerState which type is the RCMfunctionalContainer exists.	Completed	
isLeaf	Live		Changing to "true" is denied if a derived class exists.	Proposed	

Operation

Name	**Mode**	**Severity**	**Description**	**Status**	**Comment**
redefinedOperation	CbyC		Whenever an operation x has the same signature of an operation y inherited by the classifier, x.redefinedOperation == y. (not neatly defined, see comments)	Proposed	If there are more than one operation with the same signature, redefinedOperation should be the lowest in the hierarchy

Instance View

InstanceSpecification

Name	**Mode**	**Severity**	**Description**	**Status**	**Comment**
slot	CbyC		Not user-editable. The actual list of slots is read-only, and is always derived automatically.	Completed	
specification	CbyC		Not user-accessible.	Completed	

APLcontainerInstance

Name	**Mode**	**Severity**	**Description**	**Status**	**Comment**
RiSlots	CbyC		If its classifier is not null, the APLcontainerInstance must contain exactly one RiSlot for each ownedElementaryPort owned by every RCMriPortCluster owned by the classifier, each of the slot having that port	Completed	

			as definingFeature.		
PiSlots	CbyC		If its classifier is not null, the APLcontainerInstance must contain exactly one PiSlot for each ownedElementaryPort of every RCMpiPort owned by the classifier, each of the slot having that port as definingFeature. If the kind of the port is sporadic, the slot must be a SporadicSlot. If the kind of the port is cyclic, the slot must be a CyclicSlot. If the kind of the port is modifier, the slot must be a ModifierSlot. In the remaining cases, its metaclass must not be a sub-metaclass of PiSlot different from PiSlot.	Completed	
ClusterSlots	CbyC		If its classifier is not null, the APLcontainerInstance must contain exactly one Slot for each ownedPortCluster owned by the classifier, each of the slots having that portcluster as definingFeature. If the portcluster contains a port whose kind is protected, the slot must be a ProtectedSlot, otherwise, its metaclass must not be a sub-metaclass of Slot different from Slot.	Completed	
StateReferenceSlots	CbyC		If its classifier is not null, it must contain exactly one Slot for every ownedStateReference of the classifier, and the definingFeature of that slot must be that ownedStateReference.	Completed	
InternalLinks	CbyC		If its classifier is not null, it must contain exactly one APInstanceLink for every assembly contained into the classifier, linking the slot whose definingFeature is the source of the assembly to the slot whose definingFeature is the target of the assembly	Building	
ClonedSlots	CbyC		APLcontainerInstance.slots: all those whose defining feature is a port implemented by the same port must have the same values for all attributes.	Completed	
AplcAsClassifier	CbyC		An APLcontainerInstance can have only an APLcontainer as classifier.	Completed	
realizingVMLCinstance	CbyC		Not user-accessible.	Completed	
utilization	CbyC		Not user-editable, Read-only, and is always derived automatically.	Completed	

reactivitySlots	CbyC		For every ReactivityLink owned by the classifier of this instance, this instance owns a ReactivityLinkSlot whose definingFeature is that ReactivityLink	Confirmed	
Classified	Batch	Error	An APLcontainerInstance must have a classifier.	Completed	

APInstanceLink

Name	Mode	Severity	Description	Status	Comment
ClusterLinks	CbyC		Intuitively, whenever an APinstanceLink exists between two slots whose defining features are RCMportCluster(s), then there exist also many APinstanceLink which connect the slots whose defining features are children of the two RCMportCluster(s), and viceversa.	Completed	
TimeCompatibleLinks	Live	Error	If riSlot is a RiSlot and piSlot is a PiSlot, then riSlot.maximum_allowed_execution_time must be not greater than piSlot.wcet_ri_closure	Building	
TypeCompatibleLinks	CbyC	Error	Intuitively, the two ends of an APinstanceLink must be somewhat type-compatible	Completed	
NoCycles		Error	Fulfillment and usage links cannot form a cycle.	Proposed	
PortSlotsOnly	CbyC	Error	An APInstanceLink cannot be connected to a Slot unless the definingFeature of that slot is a Port.	Confirmed	
NoDeferredWithOutParameters	Batch	Error	No link may exist between a PI and a RI when the PI is deferred and features an operation with "out" parameters, and the RI is generated for a Required Operation of an operation which has no associated callback operations.	Confirmed	Deferred with "out" may be used only with callback operations.

Slot

Name	Mode	Severity	Description	Status	Comment
providingLink	CbyC		Not user-editable, read-only.	Completed	
requiringLink	CbyC		Not user-editable, read-only.	Completed	
definingFeature	CbyC		Not user-editable, read-only.	Completed	
owningInstance	CbyC		Not user-accessible	Completed	
value	CbyC		Not user-accessible	Completed	

ProtectedSlot

Name	Mode	Severity	Description	Status	Comment
ceiling	CbyC		Not user-editable, read-only.	Completed	
protocol	CbyC		not user-editable, read-only, (currently is not used)	Completed	
flows	Batch		A protection protocol is needed only if at least 2 threads use the service	Confirmed	

PiSlot

Name	Mode	Severity	Description	Status	Comment
WcetRiClosure	CbyC		Intuitively, the wcet_ri_closure of a pislot is the cost of the concrete operation provided plus the wcet_ri_closure of the immediate services connected to the rislots related to the pislot, if any. If no wcet_ri_closure is computable, then its value is set to -1.	Building	
NoCyclicWithParameters	CbyC		kind may not be cyclic if the referred operation has parameters	Proposed	
NoDeferredWithReturn	CbyC		kind may not be deferred {cyclic, sporadic, variator} if the referred operation has non-null return type	Proposed	

RiSlot

Name	Mode	Severity	Description	Status	Comment
FulfilledRiSlots	Batch	Error	Every RiSlot must be connected to a PiSlot via an APInstanceLink	Completed	
maet	Live	Error	maximum_allowed_execution_time must be greater or equal to 0	Proposed	

CyclicSlot

Name	Mode	Severity	Description	Status	Comment
priority			Not user-editable, read-only.	Completed	
period_ms	Batch	Error	must be greater than 0	Completed	Before launching analysis
criticality	Batch	Error	must be greater than 0	Completed	Before launching analysis
OBCS_ceiling			Not user-editable, read-only.	Completed	
task_ceiling			Not user-editable, read-only.	Completed	

phase_ms	Batch	Error	must be greater then or equal to 0	Completed	Before launching analysis
worst_case_response_time_ms			Not user-editable, read-only.	Completed	
deadline_ms	Batch	Error	must be greater than 0	Completed	Before launching analysis
minimum_feasible_period_ms			Not user-editable, read-only.	Completed	
maximum_feasible_wcet_ms			Not user-editable, read-only.	Completed	
maximum_blocking_time_ms			Not user-editable, read-only.	Completed	
number_of_preemptions			Not user-editable, read-only.	Completed	
worst_case_running_time_ms			Not user-editable, read-only.	Completed	
slack_time_ms			Not user-editable, read-only.	Completed	
utilization			Not user-editable, read-only.	Completed	

SporadicSlot

Name	**Mode**	**Severity**	**Description**	**Status**	**Comment**
priority			Not user-editable, read-only.	Completed	
minimum_interArrivalTime_ms	Batch	Error	must be greater than 0	Completed	Before launching analysis
criticality	Batch	Error	must be greater than 0	Completed	Before launching analysis
OBCS_ceiling			Not user-editable, read-only.	Completed	
task_ceiling			Not user-editable, read-only.	Completed	
worst_case_response_time_ms			Not user-editable, read-only.	Completed	
deadline_ms	Batch	Error	must be greater than 0	Completed	Before launching analysis
minimum_feasible_interArrivalTime_ms			Not user-editable, read-only.	Completed	
maximum_feasible_wcet_ms			Not user-editable, read-only.	Completed	
maximum_blocking_ti			Not user-editable, read-only.	Completed	

me_ms					
number_of_preemptions			Not user-editable, read-only.	Completed	
worst_case_running_time_ms			Not user-editable, read-only.	Completed	
slack_time_ms			Not user-editable, read-only.	Completed	
utilization			Not user-editable, read-only.	Completed	
burstInterval	Live	Error	it must be equal or greater than 0.	Completed	

ReactivityLinkSlotItem

Name	Mode	Severity	Description	Status	Comment
offset_ms	Live	Error	value must be non-negative	Confirmed	
relativeRate	Live	Error	value must be strictly positive	Confirmed	

ReactivityLinkSlot

Name	Mode	Severity	Description	Status	Comment
items	CbyC		Every ReactivitySlot must have as many item as the value of callsNr of its definingFeature (see constraint definingFeature).	Confirmed	
definingFeature	CbyC		The definingFeature of every ReactivityLinkSlot must be a ReactivityLink.	Confirmed	
reactionSlot	CbyC		Must be a slot whose definingFeature is an immediate {protected, unprotected} RCMpiPort whose operation is the same* specified as "reaction" by the definingFeature of this ReactivityLinkSlot. [*: the two operations must be the same operation, or one of them must override/implement the other one]	Confirmed	

UnprotectedSlot

Name	Mode	Severity	Description	Status	Comment
flows		Warning	A unprotected protocol cannot be safe if more than one thread use the service	Confirmed	

ModifierSlot

Name	Mode	Severity	Description	Status	Comment
number_of_preemptio			Not user-editable, read-only.	Completed	

ns					
worst_case_running_time_ms			Not user-editable, read-only.	Completed	
slack_time_ms			Not user-editable, read-only.	Completed	
utilization			Not user-editable, read-only.	Completed	

Deployment View

Name	Mode	Severity	Description	Status	Comment
OneThreadPerPartition	Batch	Error	every partition containts at least an APLcontainerInstance containing a SporadicSlot or a CyclicSlot, or more of them.	Completed	
OnePartitionPerNode	Batch	Error	Every computational node is associated to a partition	Completed	
OneNodePerPartition	Batch	Error	Every partition is associated to a computational node	Completed	
WorkingNodes	Batch	Error	Every computational node has a processor, a virtual machine and a memory.	Completed	
PartitionCriticality	Batch	Error	Given a ComputationalNode, its deployed partitions may not have the same criticality.	Completed	
PriorityRangeWellFormedness	Batch	Error	RCMVirtualMachine.maxInterruptPriority > RCMVirtualMachine.minInterruptPriority > RCMVirtualMachine.maxPriority > RCMVirtualMachine.minPriority > 0	Completed	
JointNodes	Batch	Error	ComputationNodes have to form a joint graph.	Completed	
LogicalConnections	CbyC		a logical connection between two partitions exists iff a link between two aplc instances deployed in the two partitions exists	Completed	

DeployableInstance

Name	Mode	Severity	Description	Status	Comment
originalInstance	CbyC		Not user-accessible	Completed	
instanceSet	CbyC		Not user-accessible	Completed	

Interconnection

Name	Mode	Severity	Description	Status	Comment

bandwidth	Batch	Error	has to be greater than 0	Confirmed	
maxPacketSize	Batch	Error	it has be greater than 0	Confirmed	
minPacketSize	Batch	Error	it has to be greater than 0	Confirmed	
maxPacketPropagation	Batch	Error	it has to be greater than 0	Confirmed	

Analysis View

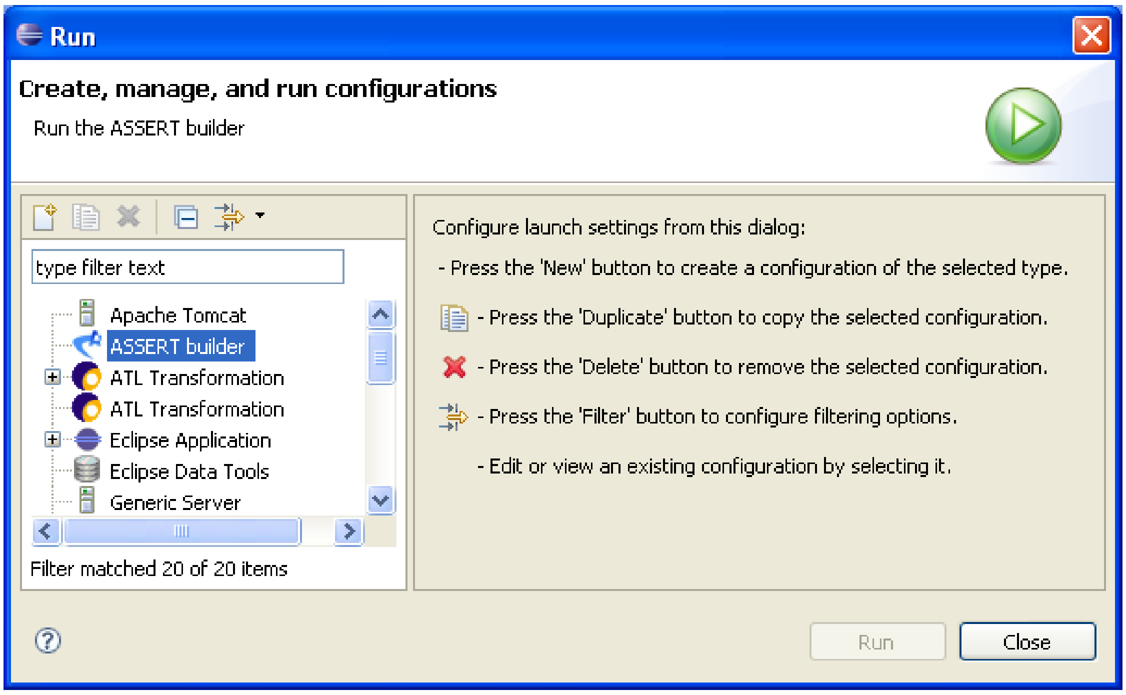
Name	**Mode**	**Severity**	**Description**	**Status**	**Comment**
NonDistributed	Batch	Cancel	All DeployableInstances are deployed on partitions which are associated within a single computational node.	Completed	This constraint is required to perform the following analyses: Classical Reponse Time Analysis, Feasibility Analysis, Sensitivity Analysis
feasibility	Batch	Error	For every sporadic and cyclic slot, worst_case_response_time <= deadline.	Completed	

Appendix D The ASSERT Builder Launcher Plug-in

HRT-UML2 comes with a dedicated plug-in named ***hrtruml2.assert.builder.launcher*** that allows to invoke the ASSERT Builder tool (developed by Semantix) from the Eclipse environment; please refer to [SEM07]for an exhaustive explanation of the ASSERT Builder.

Information about the environment and settings required by the ASSERT builder tool are not given here.

To invoke the launcher from the Eclipse select *Run->Open Run Dialog...*



Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.

- Press the 'Duplicate' button to copy the selected configuration.

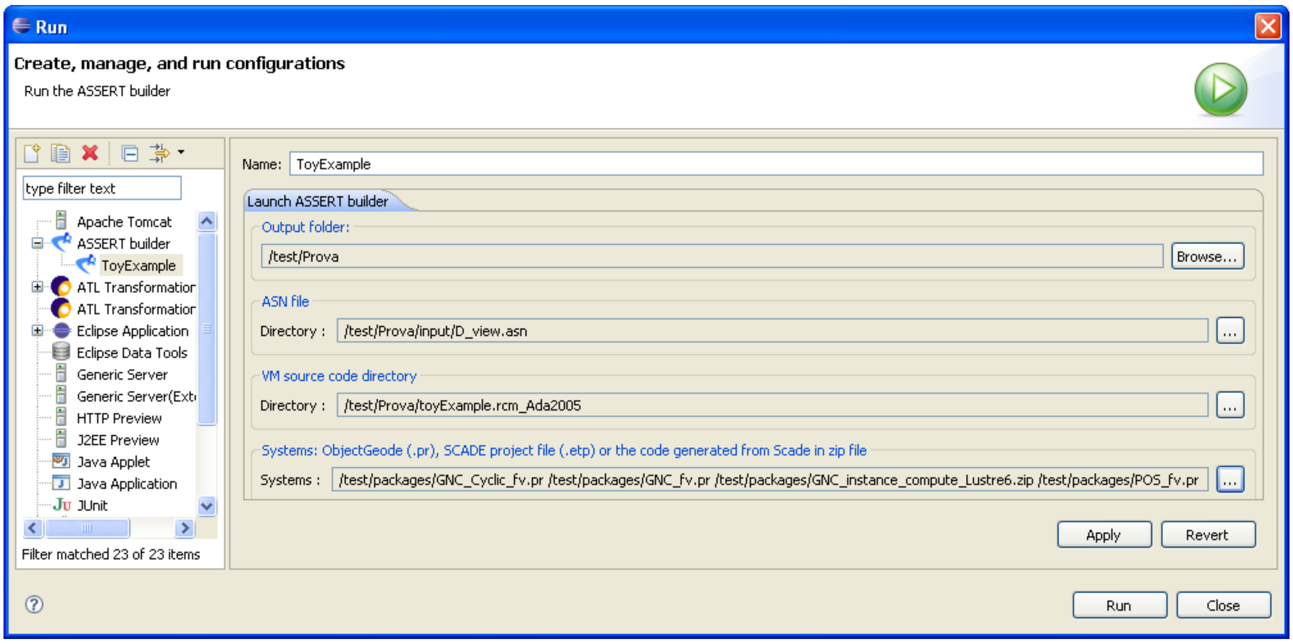
- Press the 'Delete' button to remove the selected configuration.

- Press the 'Filter' button to configure filtering options.

- Edit or view an existing configuration by selecting it.

Figure 56: Run dialog

Right click on the *ASSERT builder* type and select *New* to create a new run configuration.


Run

Create, manage, and run configurations

Run the ASSERT builder

type filter text

- Apache Tomcat
- ASSERT builder
  - ToyExample
- ATL Transformation
- ATL Transformation
- Eclipse Application
- Eclipse Data Tools
- Generic Server
- Generic Server(Ext
- HTTP Preview
- J2EE Preview
- Java Applet
- Java Application
- JUnit

Filter matched 23 of 23 items

Name: ToyExample

Launch ASSERT builder

Output folder: /test/Prova

ASN file
Directory : /test/Prova/input/D_view.asn

VM source code directory
Directory : /test/Prova/toyExample.rcm_Ada2005

Systems: ObjectGeode (.pr), SCADE project file (.etp) or the code generated from Scade in zip file
Systems : /test/packages/GNC_Cyclic_fv.pr /test/packages/GNC_fv.pr /test/packages/GNC_instance_compute_Lustre6.zip /test/packages/POS_fv.pr

Figure 57: ASSERT Builder run configuration

Then you have to specify:

- the folder where the output of the ASSERT Builder tool will be generated
- the ASN file
- the folder with the VMLC code generated through the HRT-UML2 tool (10.2)
- the files representing the systems generated with the other ASSERT technologies

When all the information above have been specified you can press *Run* to launch the builder.

A launch configuration, together with its specified information, is persistent, so it can be re-used.

Bibliography

[BT07] Matteo Bordin, Marco Trevisan The HRT-UML/RCM metamodel 004033.DDHRT_UPD.TN.04

[CP05] V. Cechticky, A. Pasetti, O. Rohlik Software Building Block Adaption Techniques 004033.DVT_ETH.DRVB.2

[CP07] Daniela Cancila, Marco Panunzio Deployment Attribute004033.DDHRT3-1.TN.2

[CV06] Daniela Cancila, Tullio Vardanega AP-level Containers: A Survival Kit 004033.DDHRT3-1.TN.1

[CVHN06] D. Cancila, T. Vardanega, I. Hamid, E. Najm An HRT-UML/RCM Interface Grammar for AP-level Modeling 004033.DDHRT_UPD_DVRB.03

[PV07] Marco Panunzio, Tullio Vardanega Model-based round-trip timing analysis: status report 004033.DDHRT3-1.TN.10

[SEM07] Semantix Description and design of the code generating tool 004033.DVT SEMANTIX.DVRB.1.I1.R1.4.4-3

[UEE07] UPD,ENST,Ellidiss Full definition and behavioural model of AP-level containers 004033.DDHRT UPD.DVRB.03