

IBM Rational Rhapsody Developer for Ada Reverse Engineering Add On



Notices

© Copyright IBM Corporation 1997, 2009.

US Government Users Restricted Rights—Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send written license inquiries to:

IBM Director of Licensing IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

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

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome
Minato-ku Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions. Therefore, this statement may not apply to you. ii This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

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

Intellectual Property Dept. for Rational Software
IBM Corporation
1 Rogers Street
Cambridge, Massachusetts 02142 U.S.A.

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

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

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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

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

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

Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1997, 2009.

IBM, the IBM logo, ibm.com, Rhapsody, and Statemate are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.html.

Table of Contents

NOTICES	2
INTRODUCTION.....	6
OVERVIEW	6
THE USER INTERFACE	6
CHOOSING THE FILES TO REVERSE ENGINEER.....	6
<i>Selecting Directories</i>	7
<i>Selecting Files</i>	7
OPTIONS	7
CUSTOMIZING REVERSE ENGINEERING RULESET	10
INSTALL “ADA TO UML 1.3” TRANSFORMATION RULESET.....	10
RUN OR DEBUG A RULESET IN RULESCOMPOSER	11
MANAGE ADA MODELS IN RULESCOMPOSER MODELS VIEW	13
<i>Read Ada source files or an Ada model XMI file</i>	13
<i>Save an Ada model as an XMI file</i>	14
APPENDIX A: ADA TO UML MAPPING	15

Introduction

The IBM Rational Rhapsody Developer for Ada Reverse Engineering Add On is a helper application for Rhapsody® that parses a collection of Ada files and generates a UML model from it.

Overview

Opens a project into IBM Rational Rhapsody Developer for Ada. From the "Tools" menu, a choice is available labeled "Reverse Engineer Ada Source Files". Selecting this menu item opens a window that allows you to select the files or directories to reverse engineer. After the selection is made, the files are parsed, and new UML objects are created in the current Rhapsody project.

The User Interface

The process of selecting files to reverse engineer is done using the Reverse Engineering Add On window. The window is shown below:

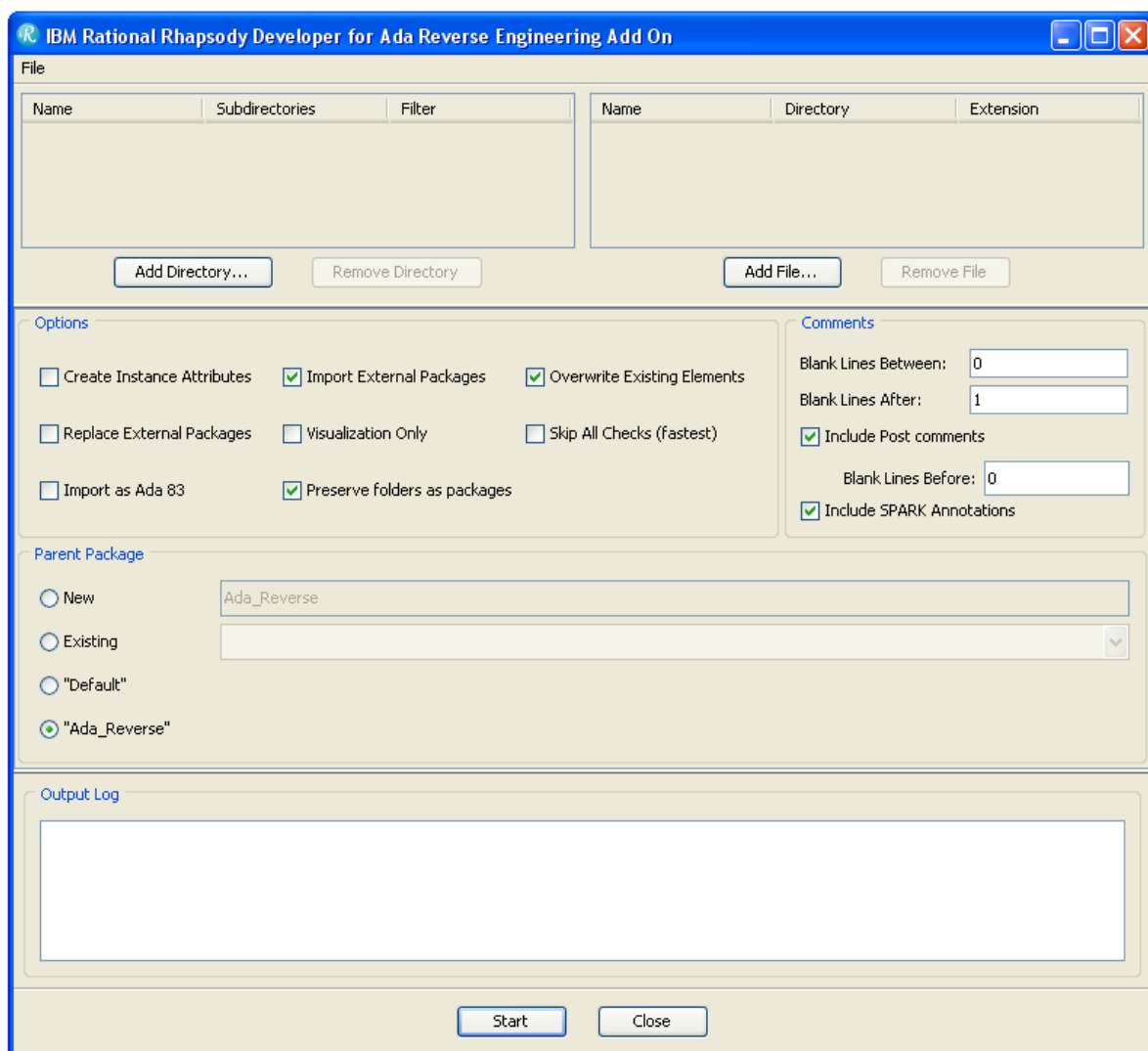


Figure 1: The Reverse Engineering Application.

Choosing the Files to Reverse Engineer

You can choose directories or individual files to reverse engineer from any location accessible from your machine.

Selecting Directories

To select a directory, push the "Add Directory" button in the upper left portion of the window.

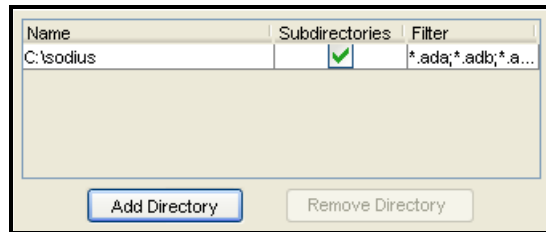


Figure 2: Selecting a Directory.

You are then presented with a directory chooser dialog box. After selecting a directory, the "Subdirectories" and "Filter" columns are editable. Checking the "Subdirectories" checkbox will include all files in all subdirectories of the selected directory. The text entered in the "Filter" column will be used to filter the set of files found in the directory (and subdirectories). The filter should be of the form "<pattern>;<pattern>;..." where each pattern can contain any number of "*" (to match any number of characters) or "?" (to match any single character).

Selecting Files

To select an individual file, push the "Add File" button in the upper right portion of the window

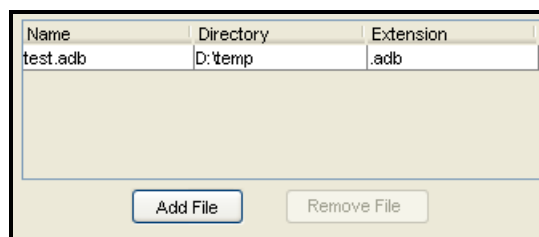


Figure 3: Selecting a File.

You are then presented with a multi-selection file chooser dialog box. After making a selection, the table will show the file name, the parent directory, and the extension.

Options

There are several options that control the operation of the reverse engineering engine.

Create Instance Attributes : In the process of creating UML types, the algorithm searches for a public or private type with the same name as the Ada Package and a "_t" suffix. If a type is not found, it takes the first record type definition encountered and assigns it as the record type that should represent the UML Class. As such, a UML type is not created for it, but instead the record elements are taken as instance variables of the class. If this behavior is not desired, the checkbox should be unselected.

Import External Packages : If one of the source files read has a "With" clause to an Ada package that is not part of the sources being reverse engineered, that Ada package is considered to be external, and has its "CG:Class:UseAsExternal" property set to true. This will prevent any code from being generated for this Ada package or any of its contained elements when doing forward generation. If this check box is not checked, these external packages will not be brought into the Rhapsody model.

Overwrite Existing Elements : If the model already contains a model element which is the same as the one being imported, checking this option will overwrite the version in the model.

Replace External Packages : If the model contains a model element which is the same as the one being imported, and the model element has the "CG:Class:UseAsExternal" property set to true, checking this option will replace the external element with the one being imported. However, any dependencies to the external element will be moved to the new element.

Visualization Only : This option is used if source code needs to be imported into Rhapsody, but the model must not be regenerated, in order to keep source unchanged. The property "CG:Class:UseAsExternal" is set to true. By this way, user will be able to use sources in its models, without regenerate them.

Skip All Checks : If this option is checked, the checks for existing elements in Rhapsody are skipped. This can be a noticeable performance enhancement for large models.

Preserve folders as packages : if this option is checked, folders of file system will be represented as packages into Rhapsody model. Ada packages are represented by rhapsody classes. Classes are located in the package which represents its folder's specifications. The first package in Rhapsody model is the folder selected in the "selecting directories" window".

Blank Lines Between : When looking for comments associated to an Ada construct (type, variable, package, etc...), this option determines how many blank lines are allowed in the comment block to still have it considered as part of the comment for element. In the example below, if this value is set to 0, typeB will only have a 1 line comment ("typeB comment line 2"), while if the value is set to 1, both comment lines will be taken. TypeA will always have 2 lines in its comment.

```
-- typeA comment1
-- typeA comment2
type typeA;
-- typeB comment line 1

-- typeB comment line 2
type typeB;
```

Figure 4: Sample Comment.

Blank Lines After : This option determines how many blank lines can appear before an Ada construct (type, variable, package, etc...) and still have the comment lines above be associated to the element. In the following example, if the value is set to 0, only typeA will have a comment. If the value is set to 1, typeB will also have a comment because there is only 1 blank line between the comment and the declaration.

```
-- typeA comment1
-- typeA comment2
type typeA;
-- typeB comment line 1
-- typeB comment line 2

type typeB;
```

Figure 5: Sample Comment.

Include Post Comments : This option determines if comment lines that appear after an Ada construct (type, variable, package, etc...) are associated to the element. The "*Blank Lines Before*" option indicates how many blank lines are permitted to appear between the element and its post comment. In the above example, if "Include Post Comments" is set to true, typeA will have the "—type B comment line 1" and "—type B comment line 2" comments associated with it. In the case where "*Blank Lines After*" is set to 2, those two comment lines will be associated to typeB instead. This is because the pre-comments take precedence in the case when the post-comments rule and the pre-comments rule are valid.

Reverse SPARK Annotations : This option is used to reverse engineer SPARK annotations. SPARK annotations are reverse engineered into the appropriate tags on the model elements. In order to have SPARK annotations reversed, the "*Include Post Comments*" option must be enabled.

Forward Generation Style : This option indicates the style that will be used for the Ada files that will be generated from the resulting Rhapsody model. The setting of this option is used to set the "Ada_CG.Component.AdaVersion" property on the active component in Rhapsody to the same value.

Select Parent Package : This option determines the package in Rhapsody where the new elements will be placed. If the selected package does not exist, it will be created. To specify a subpackage of a parent package, use the Rhapsody notation; e.g. "ParentPackage::ChildPackage".

Hidden properties

Hidden properties can be set in some specific use cases to create a model with a different pattern. In order to activate those properties, it must be set manually in the file AdaRevEng/RiARReverse.ini. If not set, the default value of this property is false.

renamedPackageInDependency : If this property is set to "true", then the renamed package will not create nested package with a renamed dependency. The renamed package will be modeled into a usage dependency by setting the property Ada_CG.dependency.GenerateRenamesPackage to "true". If the property is not present in the file or set to "false", then the renamed package will create a nested package (old implementation).

constantAsType : this property enables reversing Ada constants into Rhapsody types. In this case the type has the stereotype "constant". This enables code generator to generate constants in more appropriate location. This property is active if it is present in the file AdaRevEng/RiARReverse.ini and is equal to "true".

typeAsClass : this property enables reversing Ada types into Rhapsody class with a specific stereotype. The types which can be reversed in a class are subtypes, range types, array types, and variant record types. (enum and record types are still reversed into Rhapsody types.)

Customizing Reverse Engineering Ruleset

The same rules-based technology used for Rhapsody in Ada is available for Reverse Engineering. With this feature, the user has full control over his Ada profile used for modeling. The reverse engineering engine can be modified to represent Ada source code in a format best-suited for the user, and the code generation can be adapted to generate the appropriate Ada source code from this representation.

Install “Ada to Uml 1.3” transformation ruleset

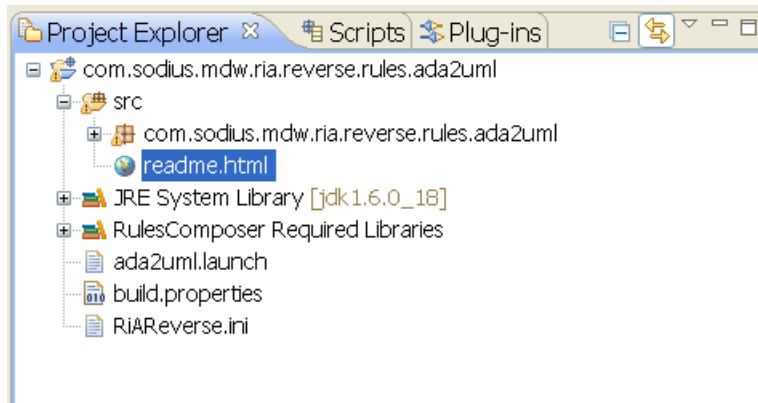
New rules can be developed by user, but the best way is to customize official **Ada Reverse Engineering** tool.

One sample available in **RulesComposer** contains exactly the same functions as those performed by the official tool.

Proceed as follow:

- Select in **RulesComposer** the menu item: **New > Example**,
- In the **New Example** window, choose **RulesComposer Sample** and press **Next**,
- In the **Import RulesComposer Sample** window, expand **Rhapsody Rulesets** node and choose **Ada Reverse Engineering** and press **Next**,
- In the next window, only one sample is proposed, press **Finish**.

In the **Project Explorer** you will find the project the *com.sodius.ria.reverse.rules.ada2uml*:



File *./src/readme.html* introduces this sample and explains how to deploy the ruleset in the official **Ada Reverse Engineering** tool.

You will find main entry point *Ada2UML.mqr* in package:
com.sodius.ria.reverse.rules.ada2uml

Physical path is:

./src/com/sodius/mdw/ria/reverse/rules/ada2uml/Ada2UML.mqr

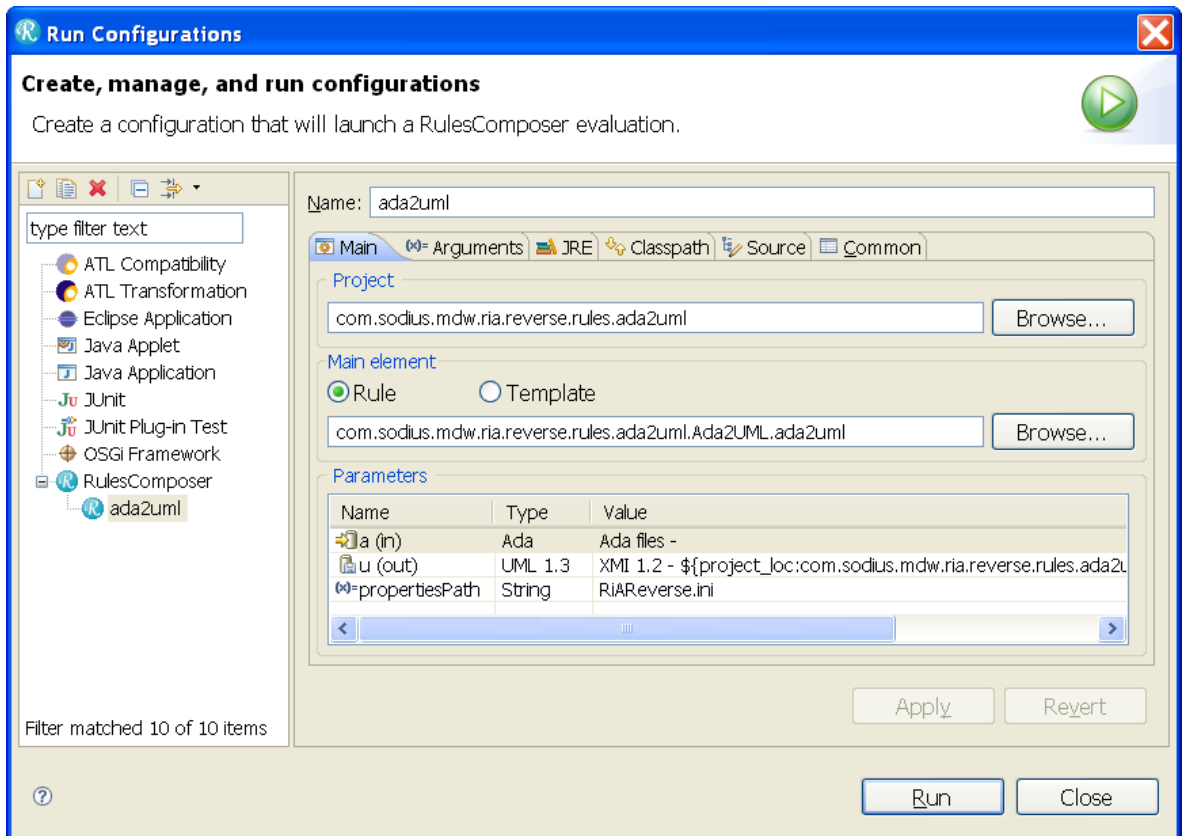
This module will help you understand how works the transformation and the reversion of Ada sources. I invite you to launch this ruleset under **RulesComposer debugger** to understand details of this transformation.

Run or debug a ruleset in RulesComposer

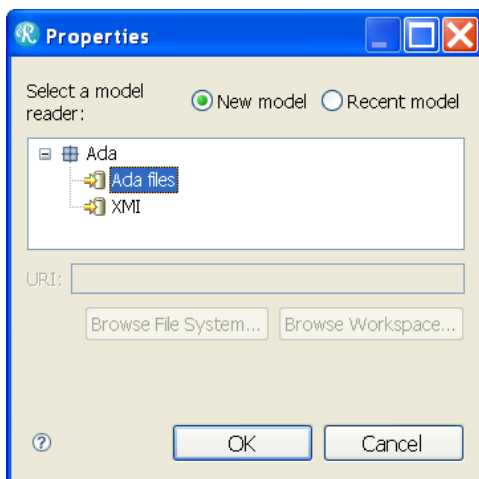
You will find in online Help more details how to run and debug ruleset (Help > Help Contents), here I will precise only parameters required for this transformation.

Proceed as follow:

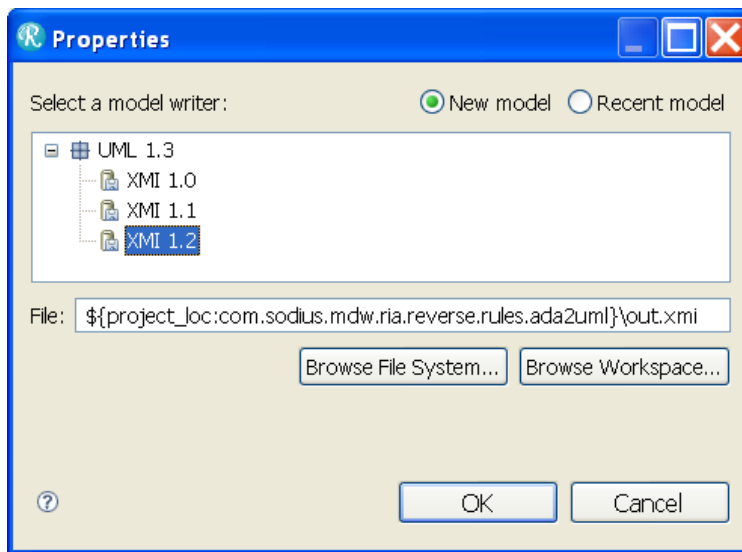
- Select in **RulesComposer** the menu item: **Run > Run Configuration...**
- In window **Run Configurations**, expand **RulesComposer** node and choose configuration *ada2uml*:



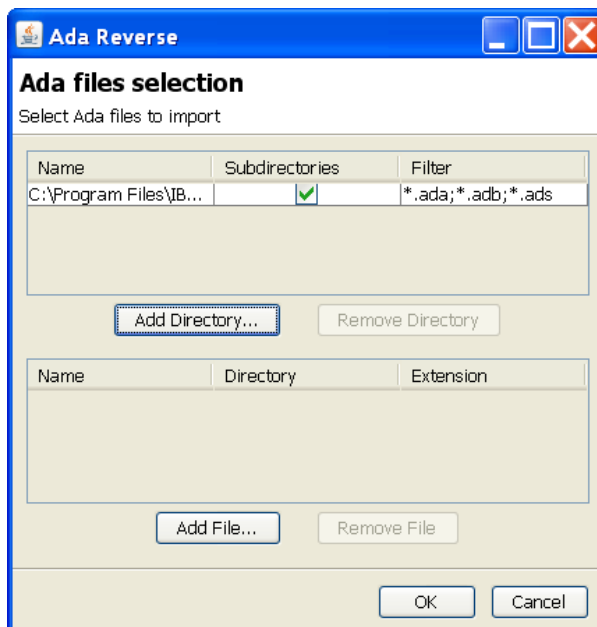
- On the right in **Main** tab, in **Parameters** grid, select input parameter "a (in)" (first line), this opens a new window: **Properties**:



- Choose the model reader **Ada file** if you want to read some Ada code source files or an **XMI** file if you have saved an Ada model in this format.
- Select second output parameter “*u (out)*” and choose the model writer XMI 1.2:



- Don't change anything for the last third input parameter “*propertyPath*”. This parameter defines the **Ada Reverse Engineering** configuration use by this transformation.
- This configuration *RiARReverse.ini* is the same provided in **Ada Reverse Engineering** tool.
- Press **Run** to save parameters and to start process of transformation.
- The launch configuration will automatically display the window **Ada Reverse** to select source files:



- The RulesComposer **Console** view will display the result of process:

```
[progress] Evaluation of ada2uml
[progress] Reading Ada files
[progress] Reading Multiple Ada files
...
[progress] Writing XMI 1.2 -\com.sodius.mdw.ria.reverse.rules.ada2uml\out.xmi
[progress] Done.
```

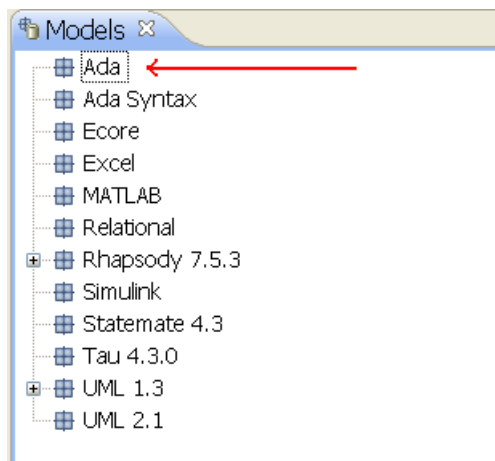
Debug process is similar:

- Select in **RulesComposer** the menu item: **Run > Debug Configuration...**
- In window **Debug Configurations**, expand **RulesComposer** node and choose configuration *ada2uml*.
- After, proceed as above for **Run Configurations**.

Manage Ada models in RulesComposer Models view

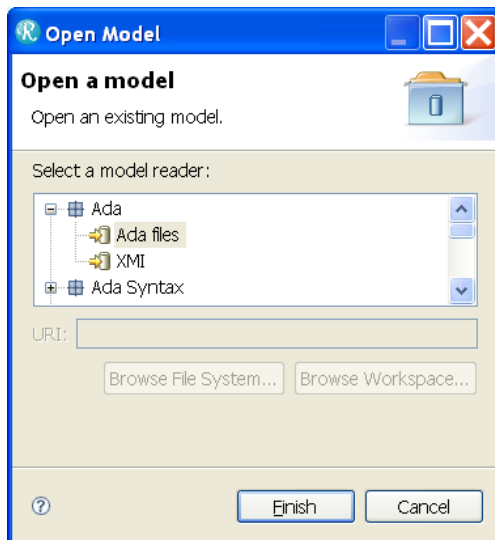
Read Ada source files or an Ada model XMI file

In Models view, a metamodel model **Ada** is provided:



Right click on this metamodel and select command **Open Model...**

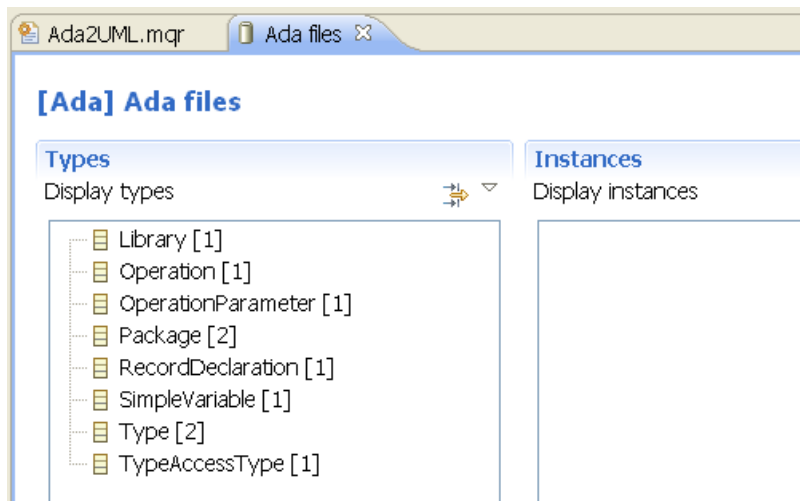
This action opens window **Open Model** similar to the **Properties** window that proposes to select a **Ada** reader:



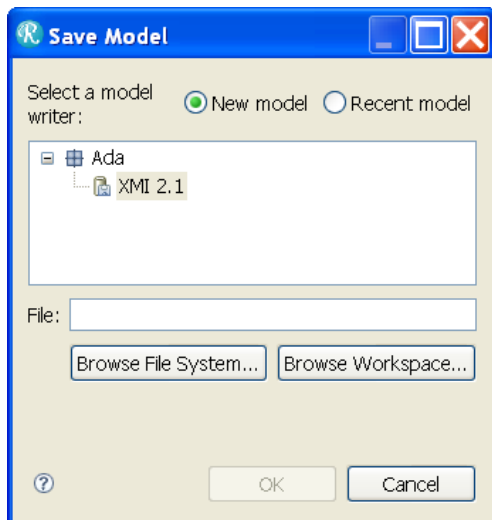
Choose **Ada files** or an Ada model **XMI** file.

Save an Ada model as an XMI file

Select the Ada model in RulesComposer:



In main menu, select command **File > Save as...**



Choose an XMI file in file system or in the workspace.

Appendix A: Ada to UML Mapping

All Ada constructs get reverse engineered into UML model elements, except for pragmas, which are skipped.

The following rules detail the mapping from Ada to UML.

Ada		UML	
Ada Package		<p>A UML Class is created. It's "Ada_CG.Class.GenerateAccessType" property is set to "None", and the "Ada_CG.Class.GenerateRecordType" property set to "False".</p> <p>If the "Create Instance Variables" option is selected, and a record type is found, the "Ada_CG.Class.RecrodTypeName" property is set to the name of the record type, and the "Ada_CG.Class.Visibility" property is set to the visibility of the record type.</p> <p>If no record type is created, then a static class is created by setting "Ada_CG.Class.IsStatic" to true.</p> <p>If the Ada package is not being reverse engineered, but is only referred to by the source files, the "CG.Class.UseAsExternal" property will be set to "True", if and only if all contained packages are also external.</p>	
	Name		Name
	Nesting		Sets the "Ada_CG.Class.isNested" property to "True" if the package is nested.
	Visibility		Sets the "Ada_CG.Class.NestingVisibility" property. (Only if the package is a nested package).
	Elaboration Code		Sets the "Ada_CG.Class.InitializationCode" property.
	Renames		A <<Renames>> dependency will be created to the Class being renamed.
	Specification Comment		Sets the "Ada_CG.File.SpecificationHeader" property.
	Implementation Comment		Sets the "Ada_CG.File.ImplementationHeader" property.
Function or Procedure		<p>All functions and procedures create static operations in Rhapsody.</p> <p>If the subprogram is "separate", the operation will be given the <<separate>> stereotype.</p>	
	Name		Name
	Function return type		The return type will result in the creation of a new locally-defined anonymous type in Rhapsody.
	Local declarations		Sets the "Ada_CG.Operation.LocalVariablesDeclaration" property.
	Renames		A <<Renames>> dependency will be created to the operation being renamed.

Ada		UML	
	Body		Sets the implementation field.
	Visibility		Ada_CG.Class.ImplementationEpilog
Argument		Argument	
	Name		Name
	Type		The type will result in the creation of a new locally-defined anonymous type in Rhapsody.
	Passing mode		The passing mode will be set on the argument if it is “in”, “out”, or “inout”. If the mode is “access”, the passing mode will be set to “in”, and the “Ada_CG.Argument.AsAccess” to “True”.
Context Statement		Dependency	
	“With” clause		A <<Usage>> dependency is created with the “CG.Dependency.UsageType” property set to Specification or Implmentation based on where the “With” clause appears.
	“Use” clause		Sets the “Ada_CG.Dependency.CreateUseStatement” property to “Use”.
	“Use type” clause		Creates a <<Usage>> dependency to the target type, and sets the “Ada_CG.Dependency.CreateUseStatement” to “UseType”.
Types and Subtype		Type	
	Name		Name
	Declaration		Declaration
Representation Clause		<p>Every representation clause (for example “for Byte’Size use 8;”) becomes a type in UML. The name of the type is the same as its represented type with a “RepresentationClause” suffix. If the representation clause has a qualifying attribute (in this case “Size”), it is appended as well. In this example, the new type is named “Byte_Size_RepresentationClause”. This new type will have a <<represents>> dependency to the underlying type.</p>	
Protected Object and Type		<p>Every protected object and protected type will become a type in UML. The name of the type will be the name of the protected object. The declaration will be the entire declaration of the protected object. This type will be stereotyped “RE_Protected_Object”.</p> <p>The protected object body will also become a separate type with the name equal to the name of the protected object with a “_body” appended. The entire body declaration will be used in the UML type declaration as well. This type will be stereotyped “RE_Protected_Object_Body”.</p> <p>The declaration type will have a <<body>> dependency to the body type.</p>	
Task and Task Type		<p>Every task and task type will become a type in UML. The name of the type will be the name of the task. The declaration will be the entire declaration of the task. This type will be stereotyped “RE_Task”.</p> <p>The task body will also become a separate type with the name equal to the name of the task with a “_body” appended. The entire body</p>	

Ada		UML	
		<p>declaration will be used in the UML type declaration as well. This type will be stereotyped "RE_Task_Body".</p> <p>The declaration type will have a <<body>> dependency to the body type.</p>	
Variable		<p>A variable in an Ada package will create a static attribute in UML. The attribute will have both the "CG.Attribute.AccessorGenerate" and "CG.Attribute.MutatorGenerate" properties set to false.</p>	
	Name		Name
	Type		The type will result in the creation of a new locally-defined anonymous type in Rhapsody.
	Renames		A <<Renames>> dependency will be created to the variable being renamed.
	Initial value		Initial value
	Visibility		The "Ada_CG.Attribute.Visibility" property is set to public, private or body.
Generic Package		Template Class	
	Generic formal parameters		Each generic formal parameter becomes an argument of the template class. The name of the UML argument will become the name of Ada formal. The declaration of the argument will be the entire formal parameter declaration.
Package Instantiation		Class Instantiation with a <<binds>> dependency to the generic class	
	Instantiation parameters		Class instantiation parameters
Generic Operation		Template Operation	
	Generic formal parameters		Each generic formal parameter becomes an argument of the template operation. The name of the UML argument will become the name of Ada formal. The declaration of the argument will be the entire formal parameter declaration.
Operation Instantiation		<p>An operation instantiation will create an Operation in Rhapsody, and this operation will have a <<binds>> dependency to the generic operation that it instantiates.</p>	