THALES

Research & Technology

Report / Technical Document

VSL EDITOR: USER AND DEVELOPER GUIDE

DEPARTMENT / SERVICE	DOCUMENT NUMBER	PAGE
STI/LSE	61565273 305 5	1/35
	-	REVISION

DOCUMENT CONTROL

	- the : 10/30/2007	A the:	B the:	C the:	D the:
Written by Signature	F. Nizou				
Approved par Signature					

Revision index	Modifications
A	
B	
C	
D	

CONTENTS

1. PURPOSE

The VSL Editor, named in first place the NFP Editor is an Eclipse plug-in implementing the VSL Language that proposes a GUI for editing VSL expression within Eclipse. The Original version V0 was provided by the CEA LIST, this documentation is the V1, completed and extended to support Papyrus and RSA documentation. This document presents the VSL editor for Eclipse, RSA and Papyrus.

In the first part, I will present global use cases and the specifics of each modeler.

In the second part I will present the global architecture of the editor.

Finally I'll focus on the realization for each component.

Because RSA, Papyrus and Eclipse versions share features, for each part of the document I will presents general information and will then detail modeler specific information.

2. DOCUMENTS

2.1. MANDATORY

NA

2.2. REFERENCE

[MARTE]	UML profile for MARTE, Beta 1 (http://www.omg.org/cgi-bin/doc?ptc/2007-08-04)
[PAPYRUS]	Papyrus UML modelling tool (http://www.papyrusuml.org)
[RSA]	Rational Software Architect (http://www-306.ibm.com/software/awdtools/architect/swarchitect/)
[Eclipse]	http://www.eclipse.org/

3. VSL EDITOR USE CASES

3.1. Generality

The VSL Editor is a text box, which can be opened on many UML Elements. It includes content assist, syntax checking and semantic checking (type check and model reference check mostly). Errors are displayed at the time expressions are edited.

In this document, I will call a « VSL Editable » element, element which type is a TupleType, a CollectionType, an IntervalType, a ChoiceType, an Integer, a Boolean, a Real, a String an UnlimitedNatural.

The Figure 1 shows the VSL Editor:

1. The UML element on which the editor has been open

2. The expected type of the vsl expression
3. The text box
4. The error message

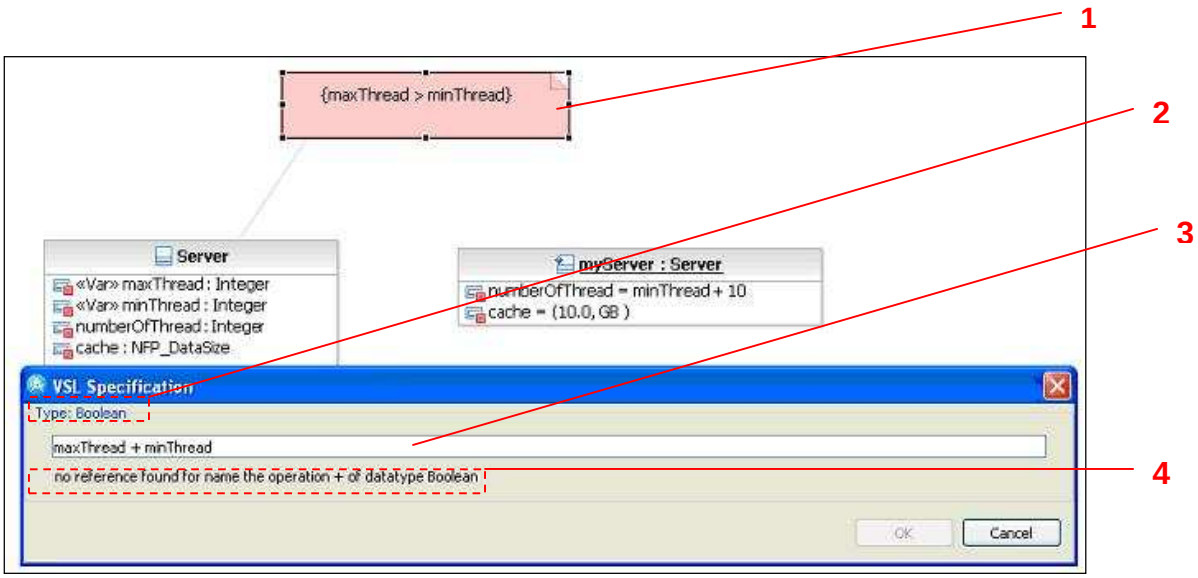

Figure 1: VSL editor

The Figure 2 presents the content assist, which proposes help for model references, datatypes, operators and some grammar snippets.

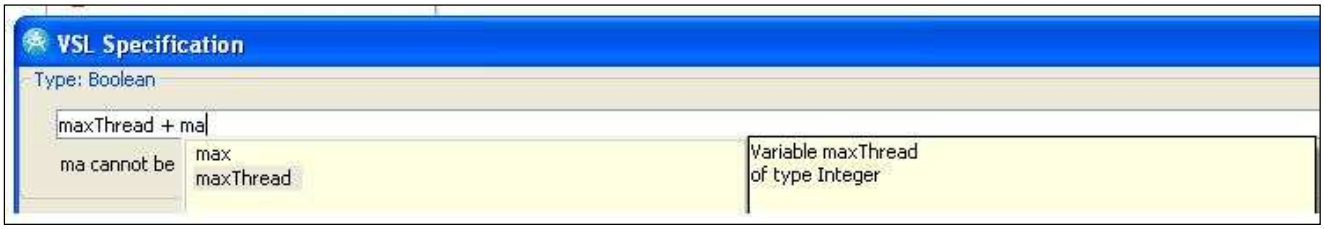

Figure 2: Content Assist

Although VSL language can be used for any ValueSpecification, the VSL editor only provide editing support for the following elements:

- Default value of property, if the property is VSL editable
- Slot's value if the defining property is VSL Editable
- Constraint. In this case, expected DataType is Boolean
- Stereotype property, if the property is VSL Editable

For technical reasons, the following syntax rules are not supported:

- In ChoiceSpecification, the name of the choice is not optional
- In TupleSpecification, the field name is not optional but when the tuple as an <<unit>> attribute. In that case, the tuplespecification can be written using the “(value, unit)” form (e.g. (1.1, s) for NFP_Duration).
- In Variable: the direction is not optional
- In the general case, any ambiguous reference to the model will provoke an error. Full Qualified Names shall be used.

For the consistency of the solution, the following decision have been made:

- Instant value are typed by NFP_DateTime
- Duration Value are typed by NFP_Duration
- Jitter value are typed by NFP_Duration
- Variable declaration are typed like the variable type
- VSL_Expression is the super type of any VSL Expression.

3.2. Eclipse specifics

In Eclipse, stereotype property editing feature is not available.

The editor can be opened from the uml treeview, in the menu CT-RTE>Edit Value Specification:

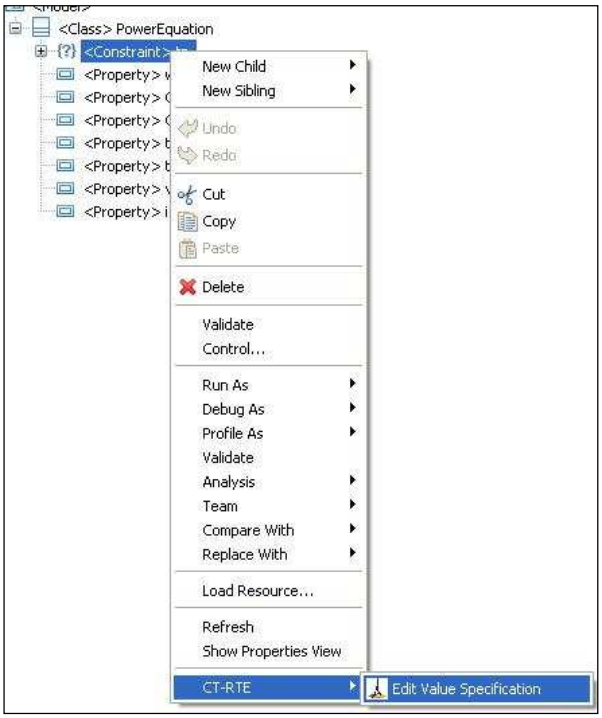

Figure 3: VSL Editor for Eclipse

Note: The VSL Editor needs the imported library to be loaded to inspect them. Eclipse using lazy loading (see Figure 4). The VSL Editor needs those to be loaded (see Figure 5.) to work.

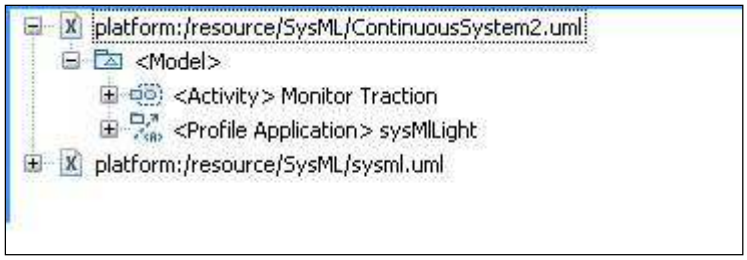

Figure 4: Eclipse UML model when imports are not loaded

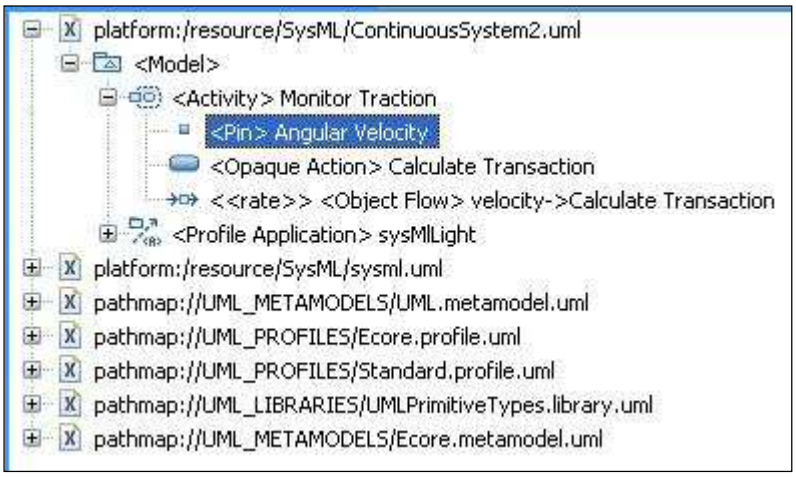

Figure 5: Eclipse UML model when imports are loaded

3.3. RSA specifics

In RSA, the VSL editor is associated with the MARTE profile provided by Thales.

The following elements are editable from diagrams (see Figure 6.) or RSA the treeview (see Figure 7): slots, default value of properties and constraint.

The stereotype property value can be edited with the specific property tab (see Figure 8.).

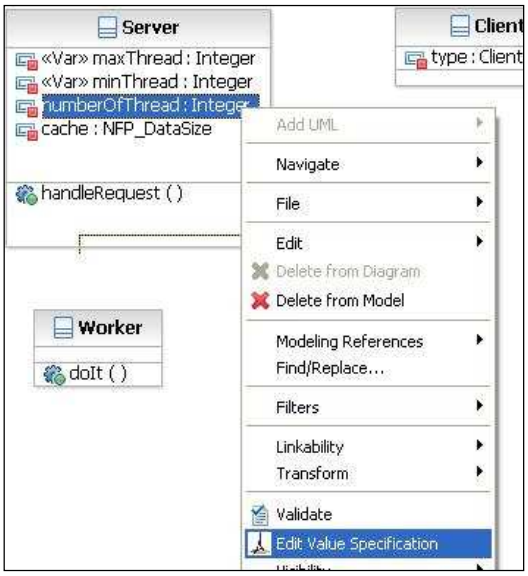

Figure 6: VSL editing from graphics editor in RSA

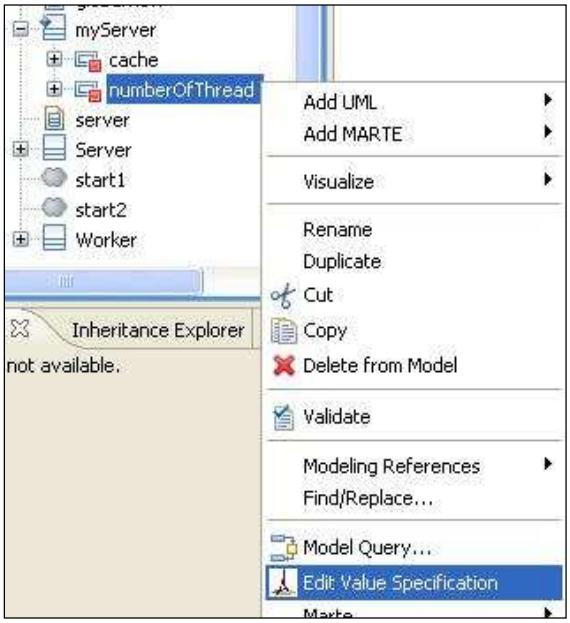

Figure 7: VSL editing from RSA treeview

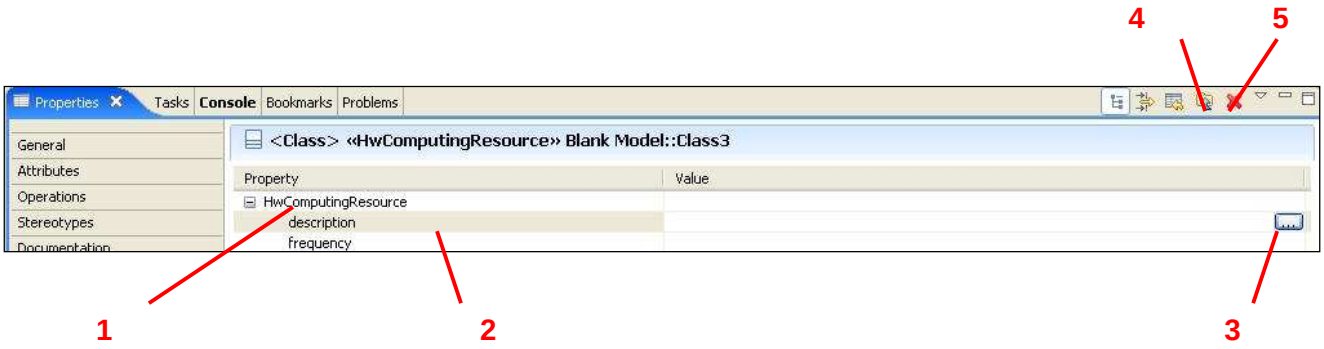

Figure 8: VSL editing for stereotype property in RSA

1. The applied stereotype
2. The stereotype attributes (only if its type is a Datatype)
3. The button for opening the VSL Editor
4. Add a new entry for multivalued property
5. Remove the selected entry for multivalued property

3.4. Papyrus specifics

The Papyrus version is linked to the MARTE profile released by the CEA in Papyrus.

The Editing is available for the following graphics elements: slots, property and constraint (the tree view is not supported) by the menu “Edit Value Specification” (see Figure 9). Editing for stereotype property values are made from the specific “VSL editor” property tab (see Figure 10).

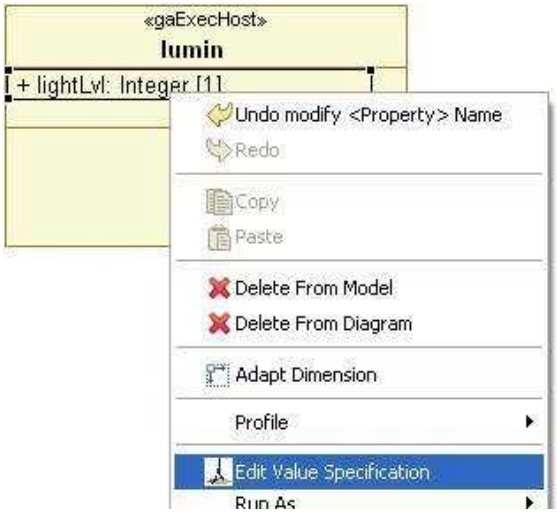

Figure 9: VSL editing in Papyrus

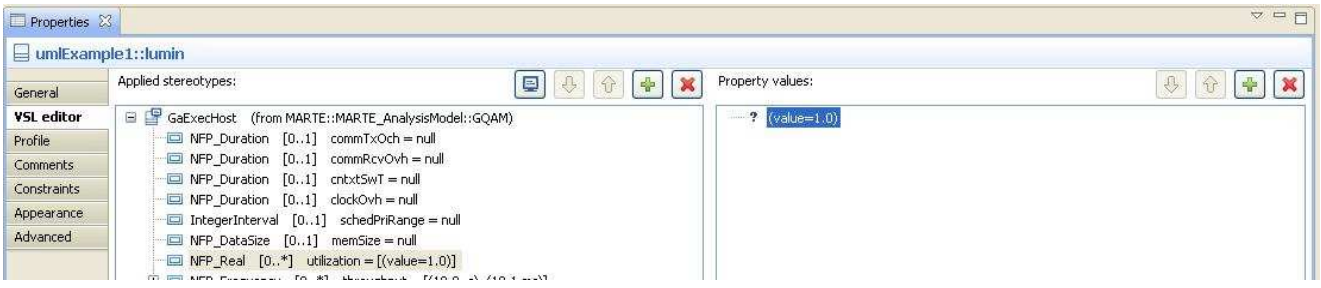

Figure 10: VSL editing for stereotype property in papyrus

1. Applied stereotype name

2. Attributes of the stereotypes (only if the type is a datatype).
3. Add a value for the selected property (open the editor)
4. Remove the selected value
5. The VSL Expression, double click to open the editor on it.

Note: This property tab is a copy of the Papyrus profile property tab.

3.5. The VSL View

The VSL editor includes a view (Eclipse View) to visualize, as an AST, a VSL expression. This view is only available for RSA and Papyrus.

To open the view: windows > show view > Others …

Figure 10.b: Open VSL view

and select « VSL View » in the « VSL » category.

The VSL view display as an AST, the VSL expression of the selected element. It's available anywhere the "Edit VSL" pop up menu is available. For technical reason, stereotype properties are not handled.

Figure 10. shows the vsl view we one selects the constraint «maxThread > minThread ».

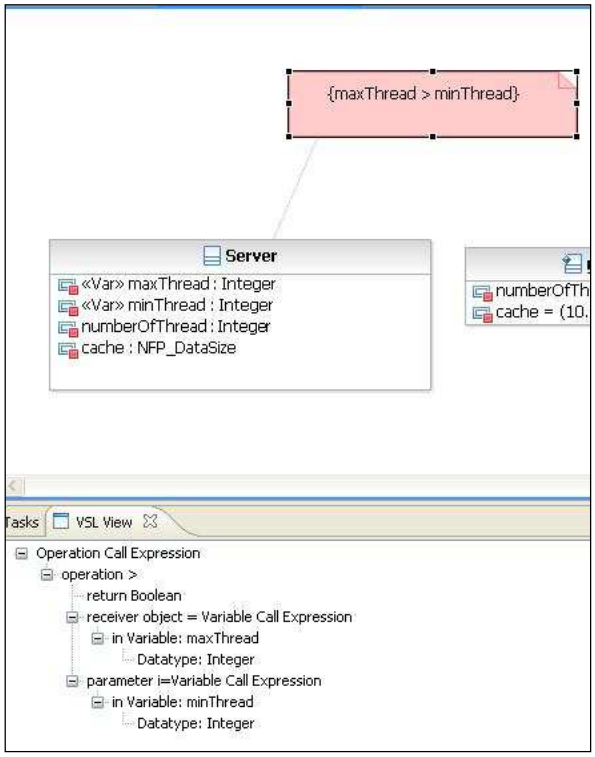

Figure 10.c: VSL View

4. GLOBAL ARCHITECTURE OF VSL EDITOR

This part presents the global architecture of the solution. First, I'll present the logical view of the architecture, then physical architecture. Finally features and platform version will be presented.

4.1. Logical architecture

The figure 11 shows the logical architecture of the solution:

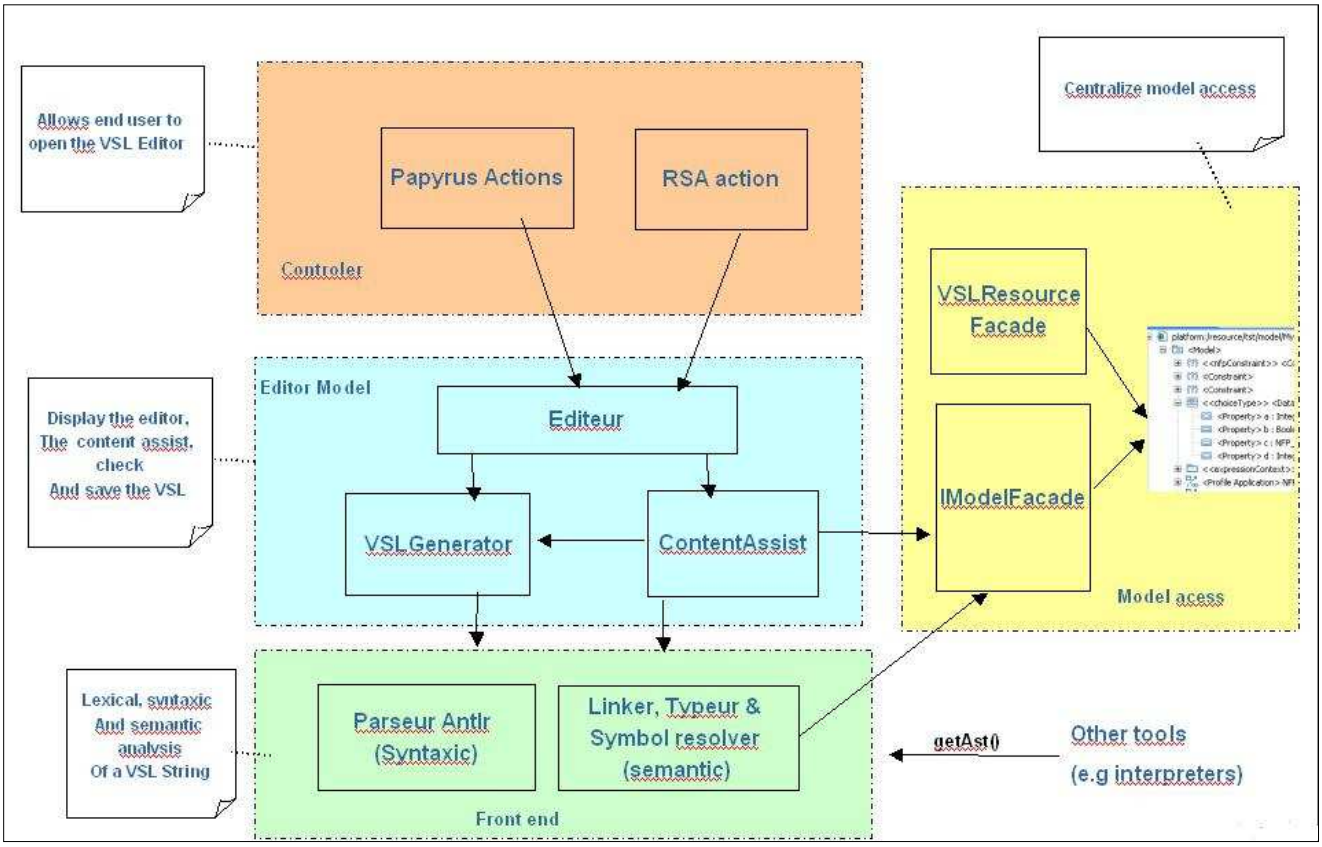

Figure 11: Logical architecture of the solution

4.1.1. Controller

The controller is a part, which depends on each modeler. This part define UI classes (Actions, Menus etc.)

4.1.2. Editor model

This part is the model of the editor. It contains the editor classes, the content assist generator and the global mechanism linking the text box with the parser.

This part is used by the controller, requires the model access and uses the Frond end.

4.1.3. Front End

This part is alike a classic compiler front end. Its role is to build an AST (Abstract Syntax Tree) from a string, checking syntaxic and semantic correction.

4.1.4. Model Access

This part manages accesses to the UML model. The parsers use it to resolve datatypes and model identifiers. This part also contains strategy of how to read and write in the model. This part is strongly dependant on a modeler (actually, strongly dependant of the profile).

Its role is not to abstract the UML2 API (of Eclipse) but to externalize at maximum its access.

4.1.5. VSL Meta model

This part is the implementation of the VSL Meta model.

4.2. Physical architecture

This part presents the real plug-ins and their roles, in regard of the logic architecture.

4.2.1. Eclipse

The com.odf.ctrte.nfp plug-in contains Actions for the Eclipse treeview and the UI classes of the editor. The com.cea.parser plug-in contains the classes of the parser, the Model access for Eclipse UML (default implementation) and the VSL editor model (content assist etc.).

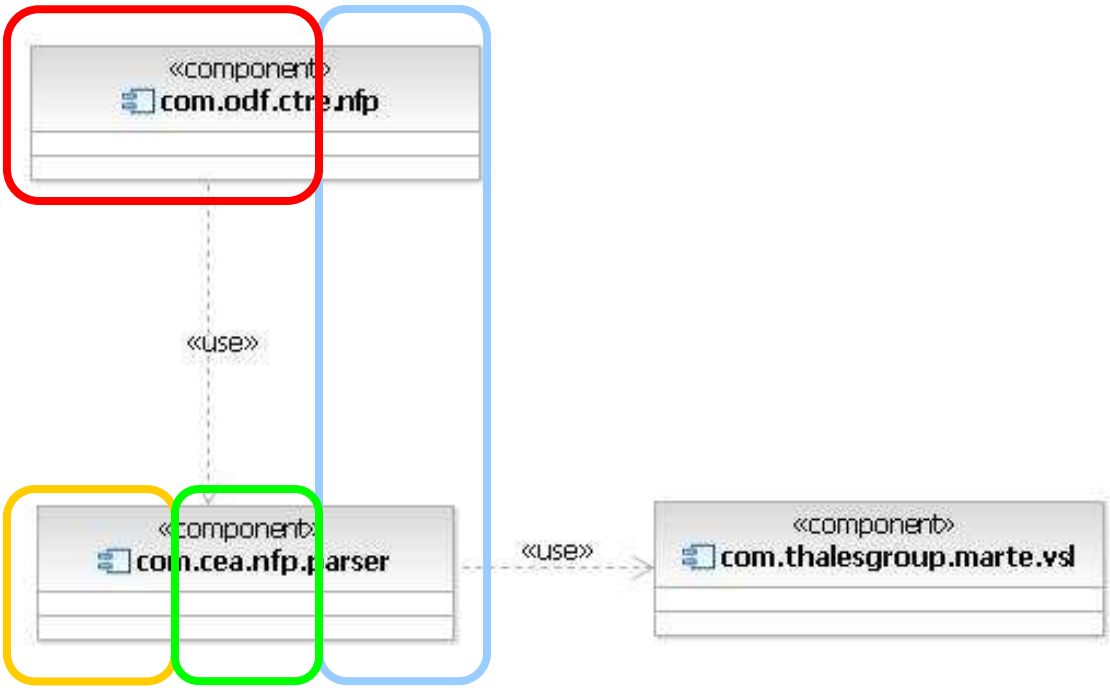

Figure 12: Eclipse plug-in and their roles

The colors used in figure 12 relate to the figure 11 colors.

4.2.2. RSA

The com.thalesgroup.rsa.nfp plug-in contains Actions, View and model access for RSA. Although there is no *java* dependency in the java code, the behavior of this plug-in is dependant of the com.thalesgroup.marte.rsa.profile (RSA implementation of MARTE by Thales).

The plug-in com.odf.ctrte.nfp is present but only the UI classes of the editor are used.

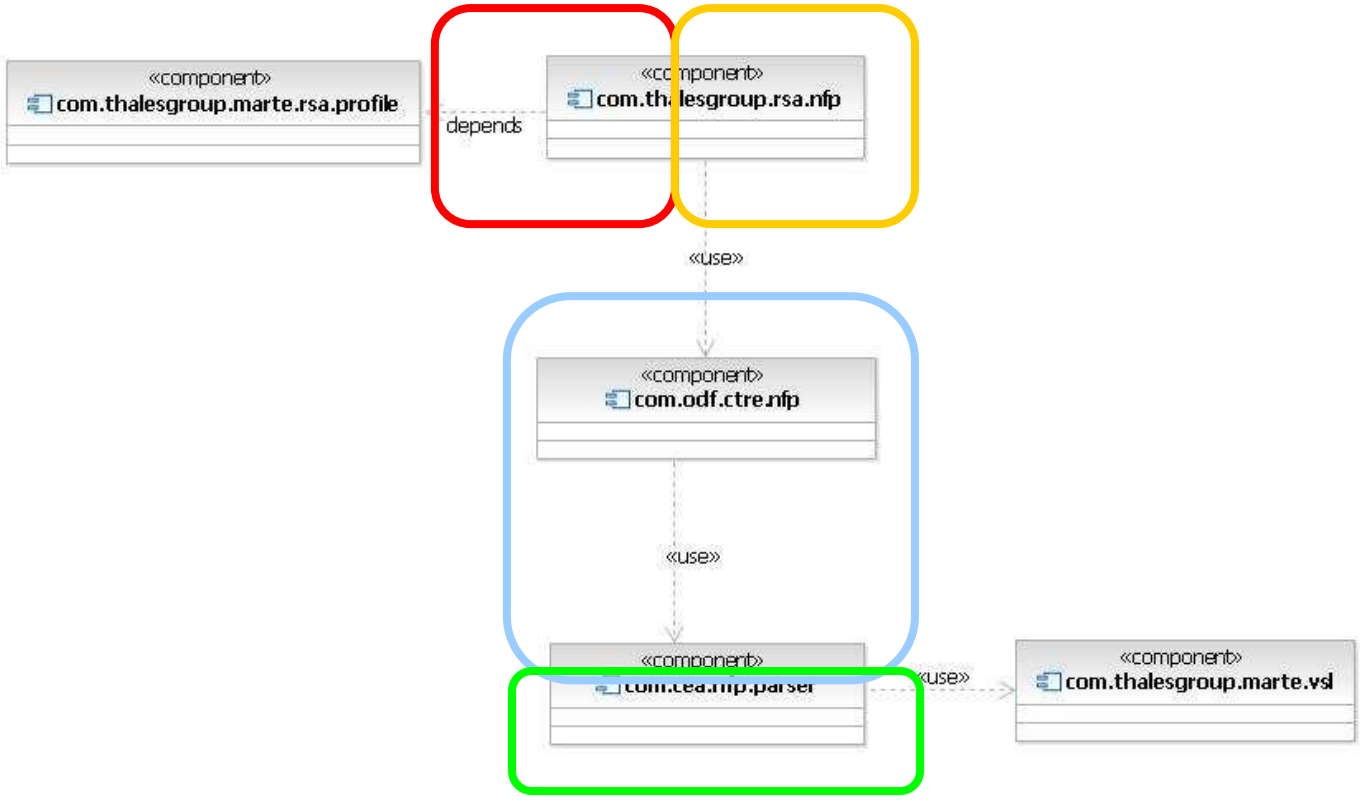

Figure 13: RSA Plug-ins and their roles

4.2.3. Papyrus

The plug-in com.thalesgroup.marte.papyrus.nfp.ui contains Actions, Views and Model Access for Papyrus. Although there is no *java* dependency in the code, the behavior of this plug-in is dependent of the com.cea.papyrus.marte.profile plug-in (Papyrus implementation of MARTE by the CEA).

The plug-in com.odf.ctrte.nfp is present but only the GUI classes of the editor are used.

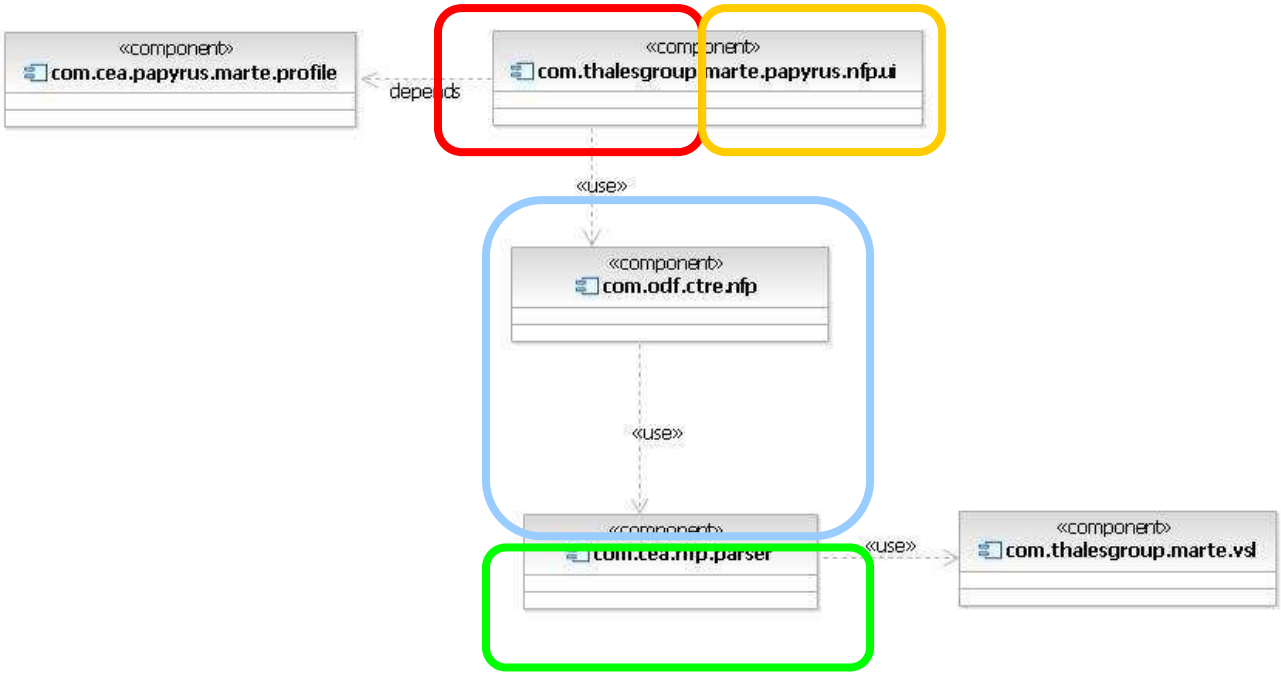

Figure 14 Papyrus plug-ins and their roles

5. FEATURES, VERSIONS AND DEPLOYMENT

5.1. Eclipse

There is no feature for Eclipse, still, the VSL editor can be used and only require the following plug-in to be deployed:

- com.cea.nfp.parsers
- com.thalesgroup.vsl
- org.odf.ctrte.nfp

This configuration has been tested for Eclipse 3.2

5.2. RSA

The feature for RSA is com.thalesgroup.marte.rsa.nfp

It contains:

- The com.thalesgroup.marte.rsa.nfp.ui plug-in
- The com.cea.nfp.parsers plug-in
- The com.thalesgroup.marte.rsa.nfp.ui plug-in
- The com.thalesgroup.vsl plug-in

- The org.odf.ctrte.nfp plug-in
- The com.cea.nfp.parsers plug-in
- The com.thalesgroup.vslview plug-in
- The com.thalesgroup.vslview.rsa plug-in

This configuration has been tested for RSA 7.0.0

5.3. Papyrus

The Papyrus feature is com.marte.papyrus.nfp

This feature contains:

- com.cea.nfp.parsers plug-in
- com.thalesgroup.marte.papyrus.nfp.ui plug-in
- com.thalesgroup.vsl plug-in
- org.odf.ctrte.nfp plug-in
- com.cea.nfp.parsers plug-in
- com.thalesgroup.vsl plug-in
- com.thalesgroup.vslview plug-in
- com.thalesgroup.vslview.papyrus plug-in

This configuration has been tested for Papyrus 7.2

6. DETAILED ARCHITECTURE OF THE VSL EDITOR

This part presents each component of the VSL editor.

6.1. Controller

Controllers are responsible for opening the editor on specific modeler UI elements. Therefore this part is specific to each modeler, although, the implementation of each modeler follows the same design (see Figure 15).

Actions and specific views define buttons and menu entries for each VSL editable element. Their role is to compute the selection and they must immediately delegate the treatment to the model (MVC model).

The model is a singleton, its role is to instantiate modeler dependant Facade and open the editor with the correct parameters.

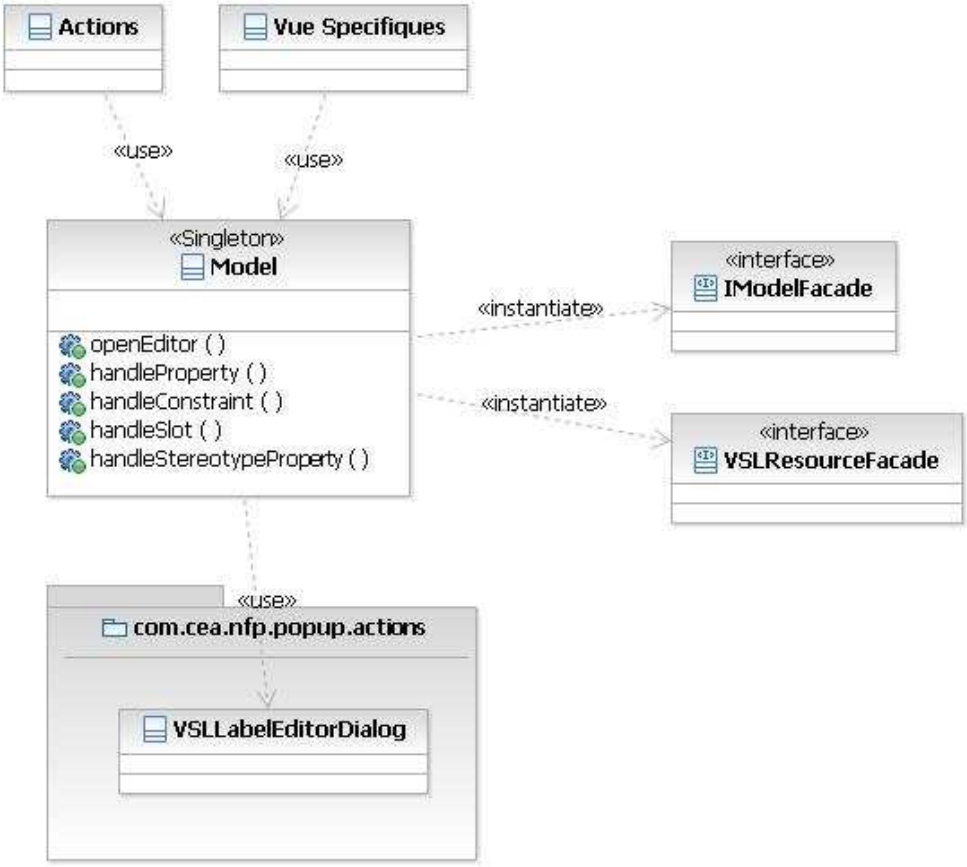

Figure 15: General view of the controllers

6.1.1. Eclipse specifics

The controller classes are within the org.odf.ctrte.nfp plug-in.

It is composed of 3 actions and a Model:

- **ConstraintEditionAction** : Define Action for the pop up menu on Constraint
- **SlotEditionAction**: Define Action for the pop up menu on Slot
- **PropertyEditionAction**: Define Action for the pop up menu on property
- **UMLModel**: Use the EclipseUMLModelFacade

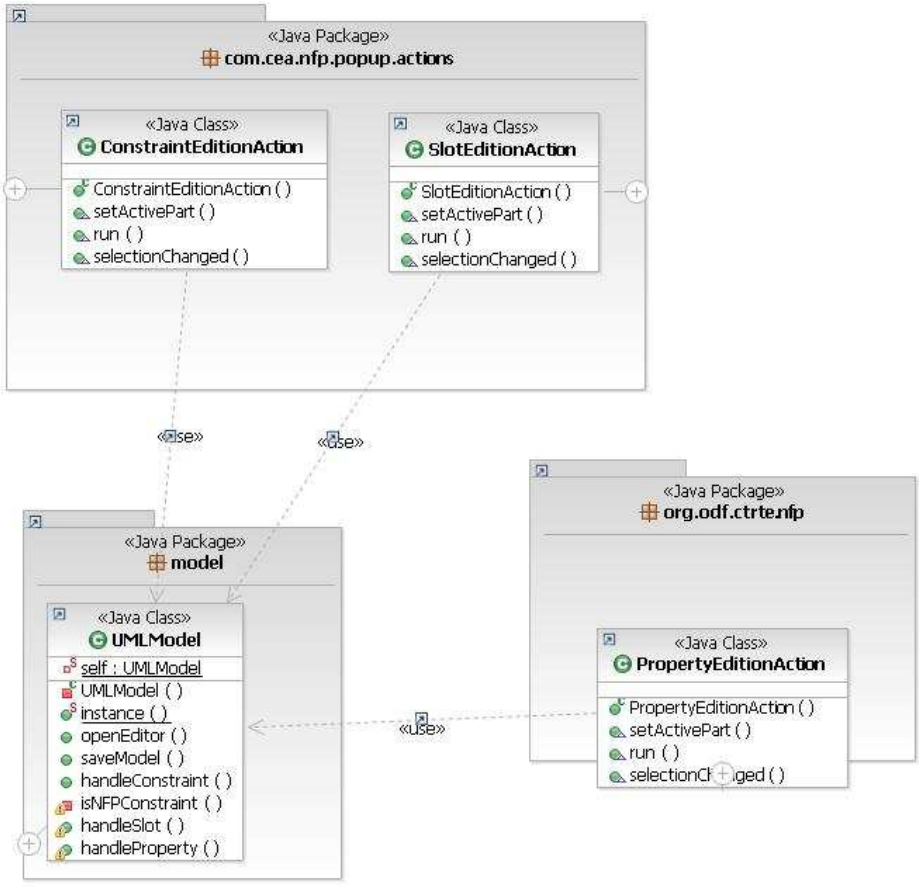

Figure 16: The Eclipse controller

6.1.2. RSA specifics

The figure 17 shows actions for RSA.

- **ConstraintEditionAction**: Actions for constraints (graphics)
- **ConstraintTreeViewEditionAction**: Actions for constraints (treeview)
- **SlotEditionAction**: Actions for slots (graphics)
- **SlotTreeViewEditionAction**: Actions for slots (treeview)
- **PropertyEditionAction**: Actions for properties (graphics)
- **PropertyTreeViewEditionAction**: Actions for properties (treeview)
- **RemoveArrayEntryAction**: Action for the property tab (see figure 18), remove an entry for multivalued stereotype property.
- **AddArrayEntryEditionAction**: Action for the property tab (see figure 18), add an entry for multivalued stereotype property.
- **VSLCellEditor:** Define button for VSL Editing of stereotype property value (see figure 18).

- **RSAModel:** The model for RSA. RSA use the EMF transaction mechanism, each method called handleXXX (XXX being an UML element) is only a delegate call of internalHandlerXXX within a transaction. The RSAModel use the RSAModelFacade.

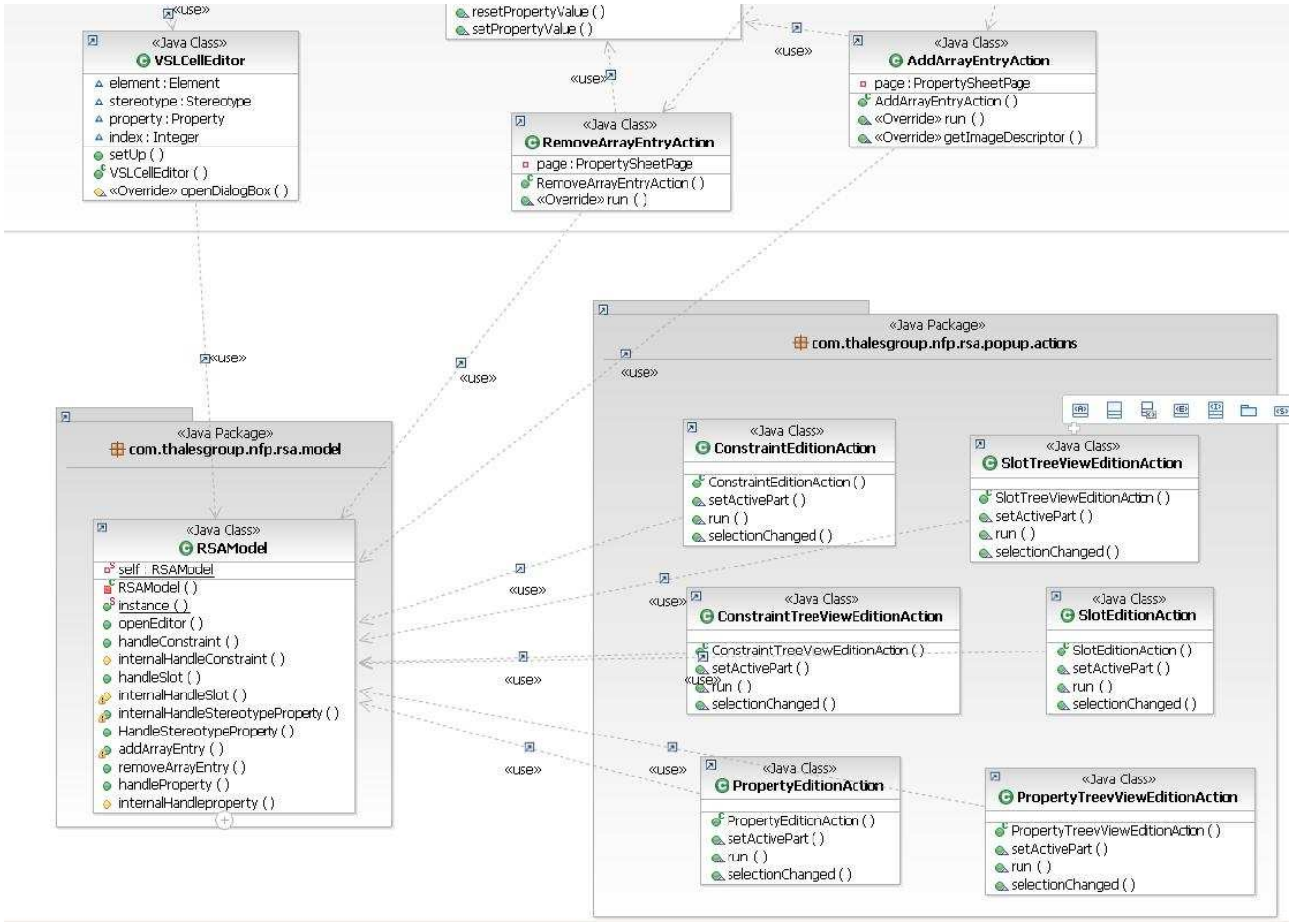

Figure 17: Actions and Models for RSA

The figure 18 shows the architectures of the classes involved in the property tab.

We made this tab because there is now way to add buttons for property stereotype. This tab reprensents, with a treeview, the applied stereotypes, stereotypes attributes and VSL value of the selected element (see figure 19).

- **CustomStereotypePropertySection**: This part define the Section (see Eclipse property Tab tutorials for more details)
- **UMLElementSourceProvider:** The source provider. Computes the correct ISource
- **ElementPropertySource**: The source provider
- **MultivaluedPropertyPropertySource**: The source provider for multivalued stereotype property.
- **MultiValuedPropertyLabelProvider**: The label provider for multivalued stereotype property.
- **VSLFilter**: Filter the selection to only show VSL editable stereotype property.

- **VSLDescriptor**: Define the way a VSL expression is displayed and edited.
- **VSLCellEditor**: Define the button opening the VSL editor.

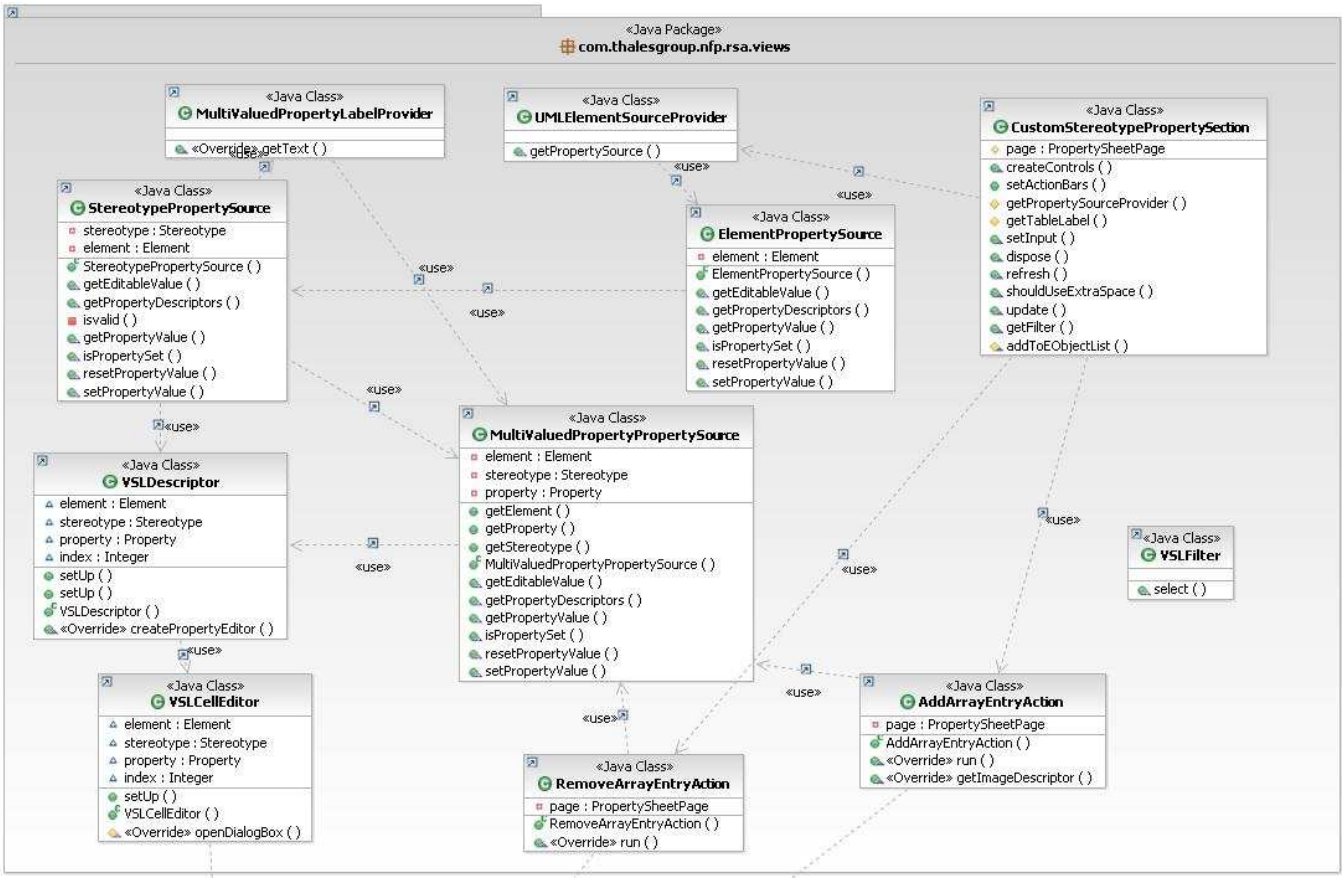

Figure 18: Property section classes

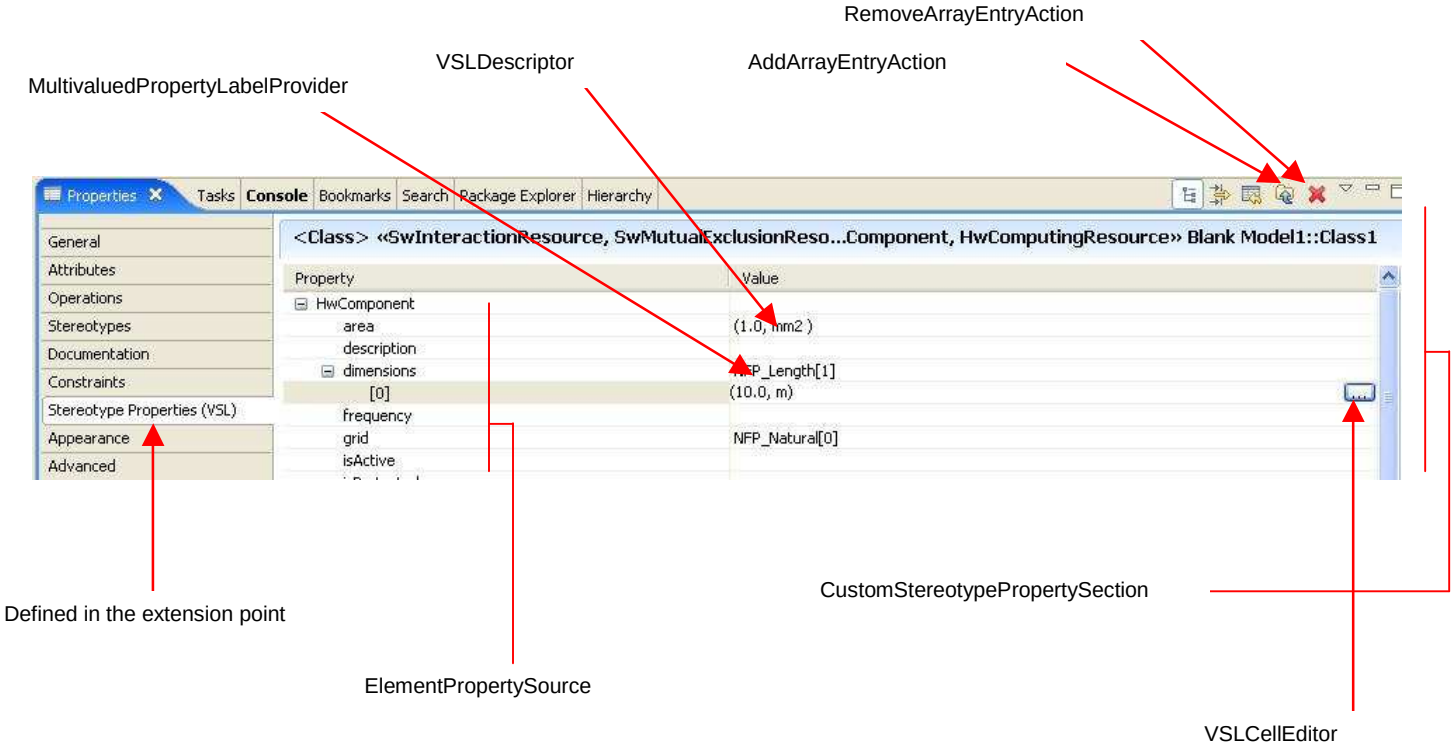

Figure 19 : the property section and the classes involved

The figure 19 shows the property section and annotates each visual part with the responsible class. See http://www.eclipse.org/articles/Article-Tabbed-Properties/tabbed_properties_view.html for more details on Eclipse Property Sheet mechanism.

6.1.3. Papyrus specific

The figure 20 shows the controller architecture for Papyrus.

- **PapyrusModel**: The model for papyrus (MVC model). Build the Model facades and contains treatments for each editable UML element.
- **ConstraintEditionAction**: Action for Constraints (graphic)
- **SlotAditionAction**: Action for Constraints (graphic). Require papyrus modification to work, see 7.1.
- **PropertyEditionAction**: Action for Properties (graphic)
- **VSLDatatypeValueTreeObject**: Editing action in the propertySheet (see Figure 21), to modify a VSL.
- **VSLPropertyComposite**: Editing action in the propertySheet (see figure 21) to add a new VSL.

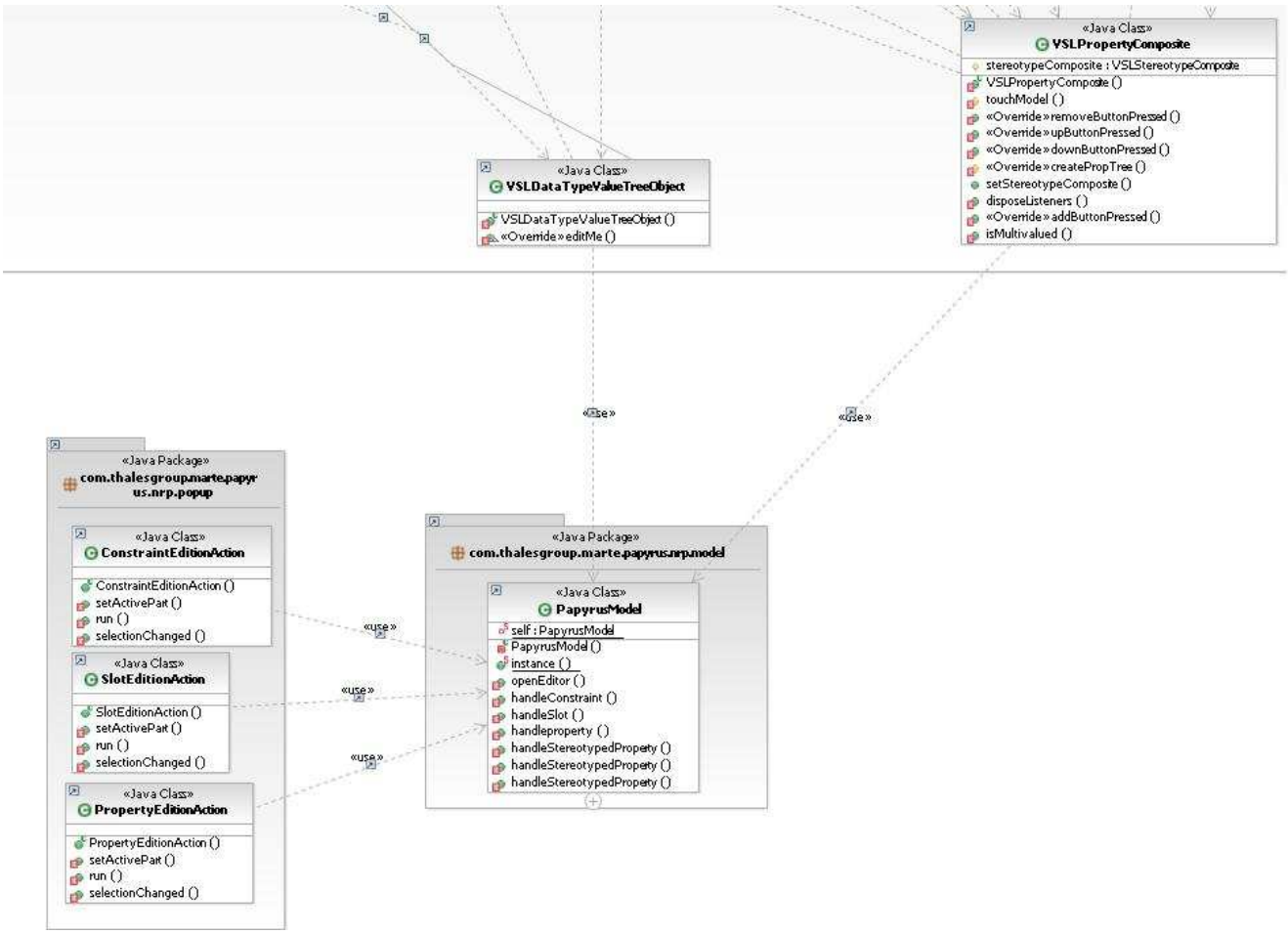

Figure 20: Papyrus controller

Like for RSA, we had to add a new PropertyTab in the papyrus PropertySheet to add actions for stereotype property value Editing. This part is strongly inspired (i.e. copy/paste) of the property tab for profile defined in the plug-in com.cea.papyrus.ui (class com.cea.papyrus.ui. properties.tabbed.PropertyViewSection) available in EPL.

Note: Classes named VSLX are copy of the class X.

- **VSLSection**: Define the section in the propertyTab (see Figure 22)
- **VSLStereotypeComposite:** Define the displayer for applied stereotypes and buttons.
- **VSLStereotypeTreeObject**: Display a stereotype in the treeview.
- **VSLRootElementTreeObject** The root of the treeview. Create a VSLStereotypeTreeObject for each applied stereotype of the selected element.
- **VSLValueTreeObject**: The super type of graphical elements displayed in the right part of the section.
- **VSLPropertyTreeObject**: Display a property in the treeview.
- **VSLDatatypeValueTreeObject**: Display the VSL expression
- **VSLPropertyComposite:** Define the right part of the section.
- **VSLStereotypePropertiesDoubleCLickListener**: Double click listener for VSL value Editing.

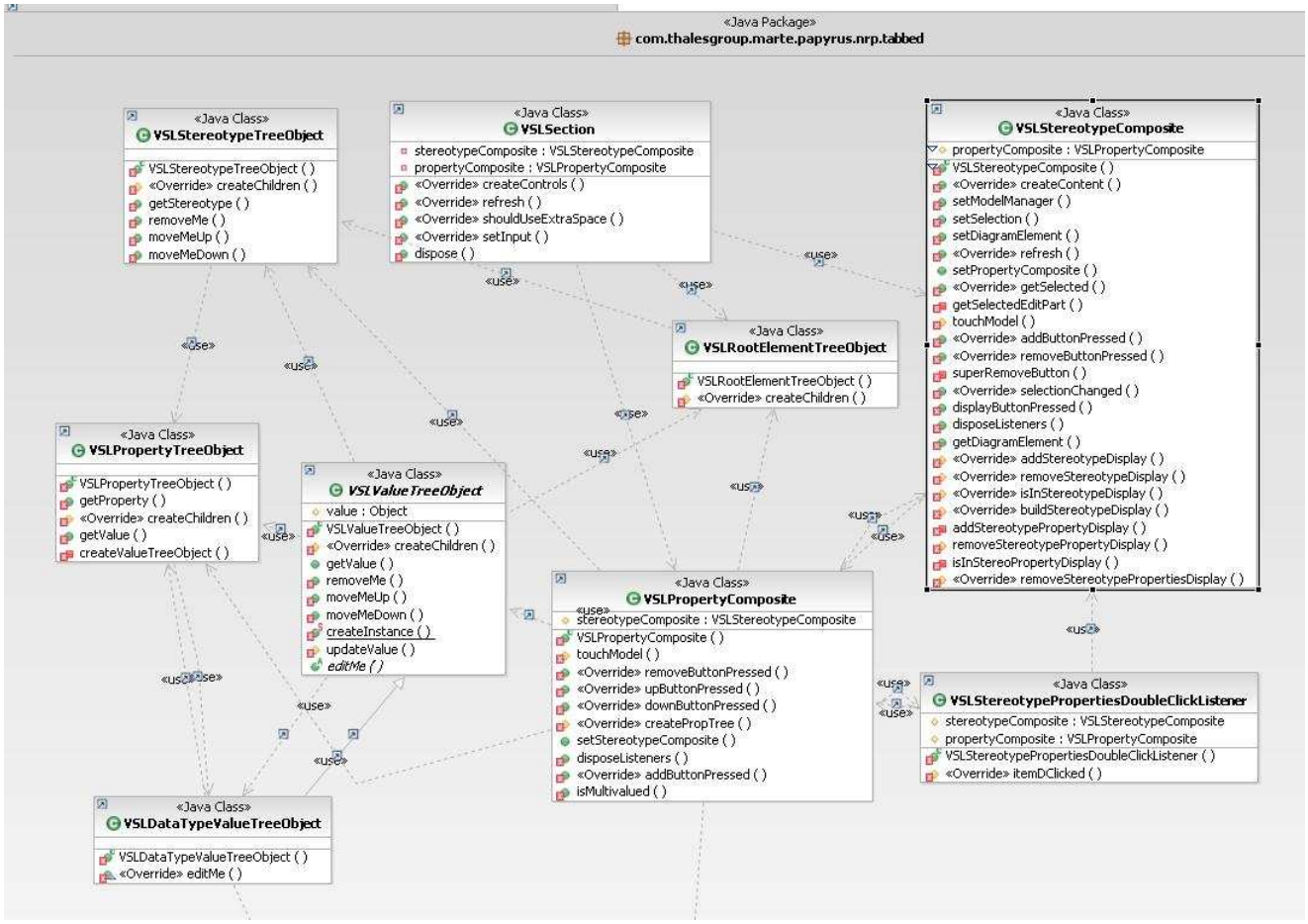

Figure 21: Property section classes for Papyrus

La figure 22 shows the property section and annotates each visual part with the responsible class. See http://www.eclipse.org/articles/Article-Tabbed-Properties/tabbed_properties_view.html for more details on Eclipse property sheet mechanism.

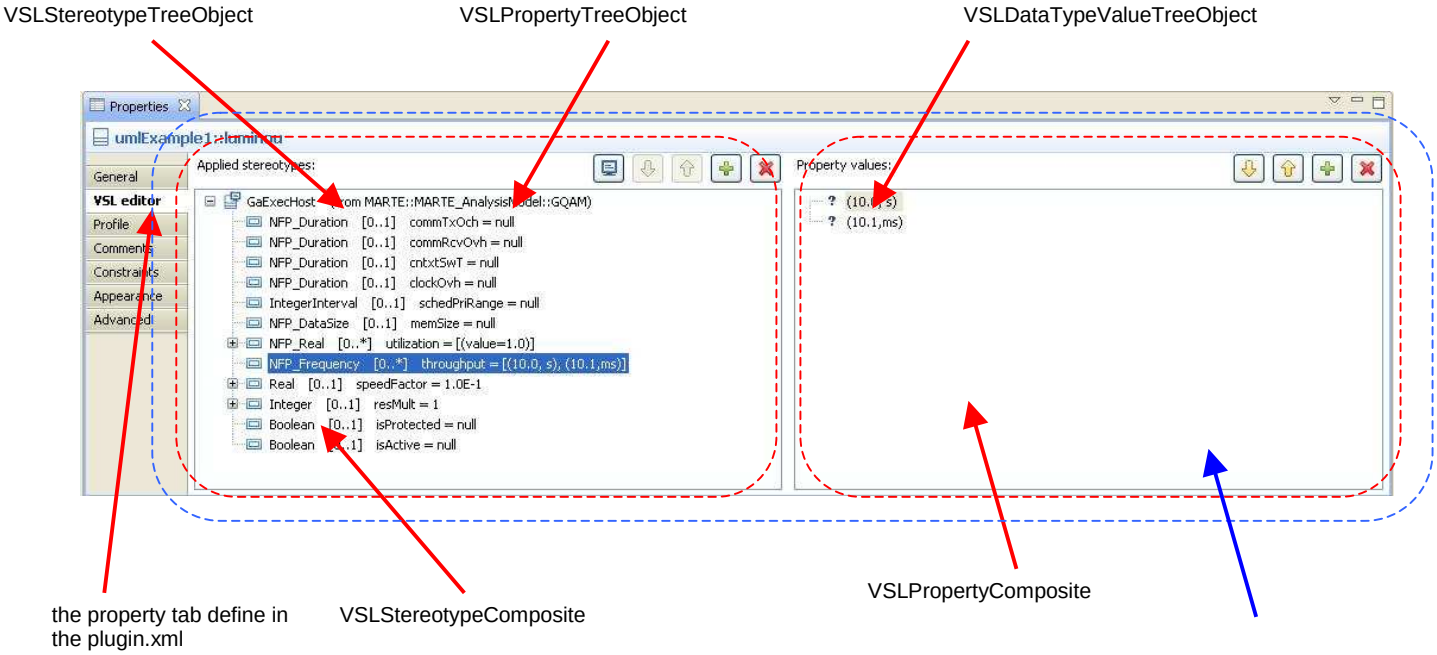

Figure 22: The property Section.

6.2. Facade

Facades hides model access from the rest of the code, like datatypes fetching or how to read and write a VSL in the model.

There are two types of facade: IModelFacade and VSLResourceFacade, both interfaces are defined in com.cea.nfp.parsers.

- **IModelFacade**: Contains model access methods, to get all the datatypes, properties, or elements by their names. See Javadoc for more details.
- **VSLResourceFacade**: Represent a VSL expression for a given element (it is close to the strategic pattern). This interface hide the way the VSL string is read and written in the model. It also hides how the expected datatype is computed.

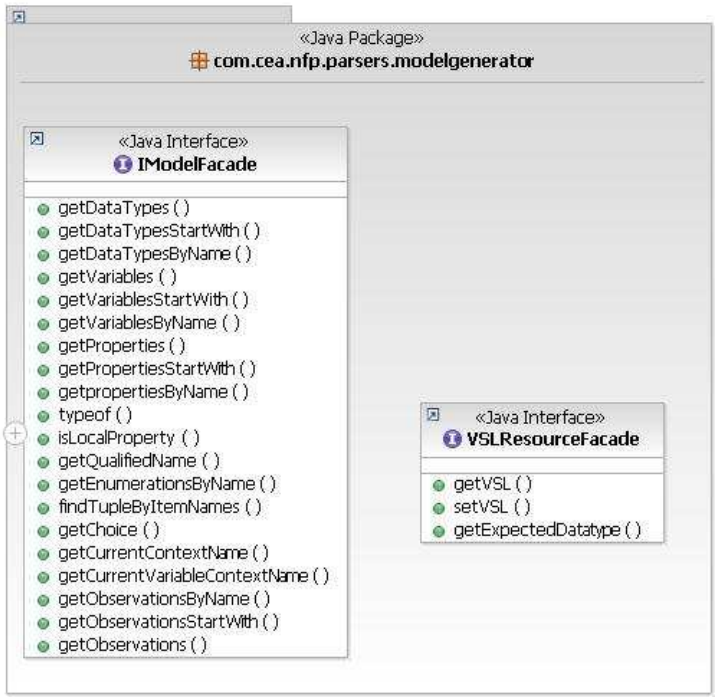

Figure 23: Facade interfaces

6.2.1. Eclipse specifics

The Eclipse version owns an implementation of IModelFacade, which is the default implementation. This implementation is com.cea.parsers.nfp.uml.UMLModelFacade in the plug-in com.cea.nfp.parsers.

In Papyrus and RSA, architectures are similar. Therefore this description is to be considered as the references, only variation points will be detailed.

UMLModelFacade delegates its treatments to 4 classes: one for properties, one for variables, one for observations and one for Datatypes.

Models references are stocked in a Hashtable, for performance reason.

For more details about Facade instantiation, refer to part 7 of this document.

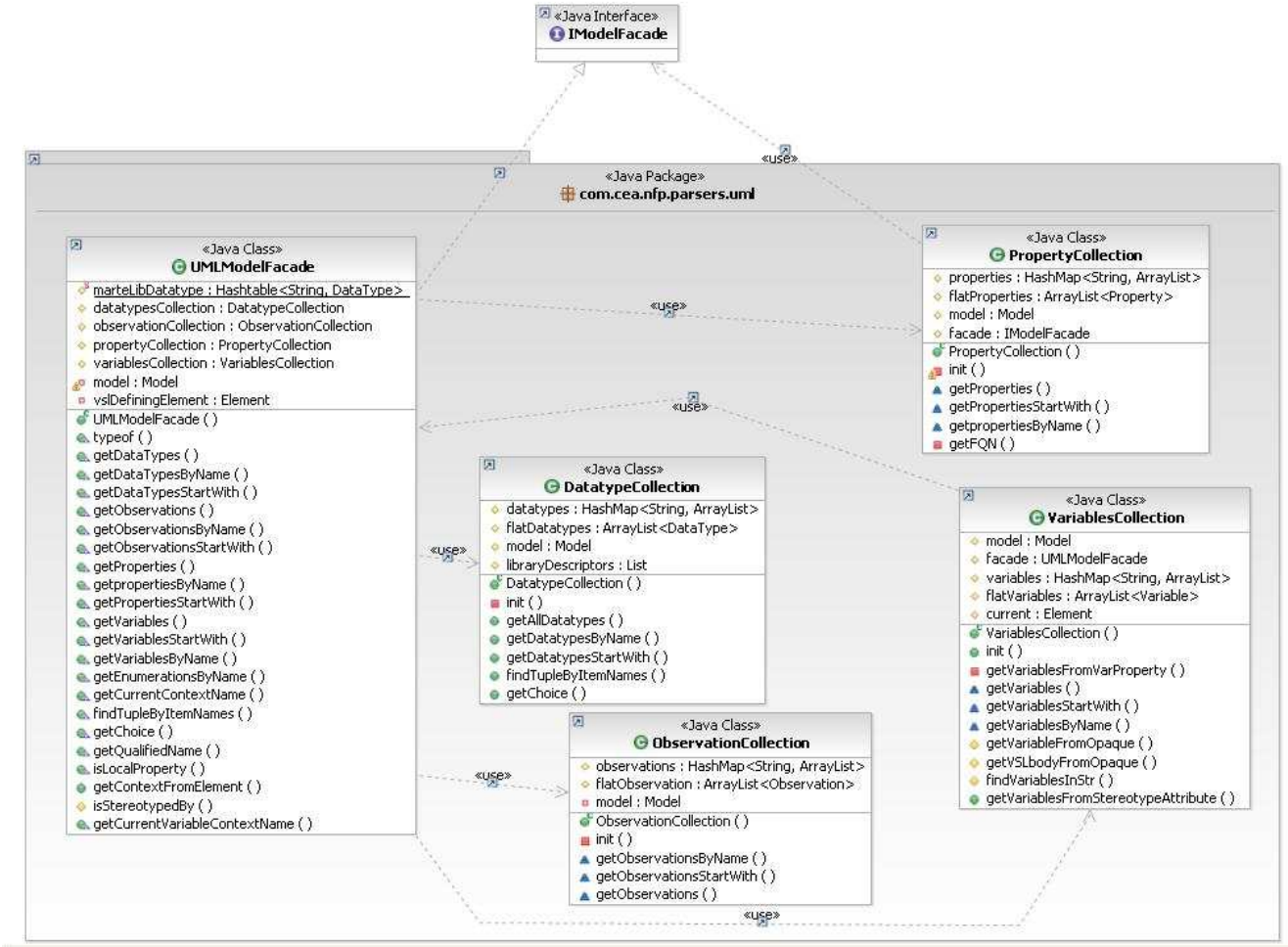

Figure 24: UML2 Facades

For UML, there are two implementations of VSLResourcefacade: EclipseUMLResourceFacade that is a facade for opaqueExpression (default facade) and PropertyResourceFacade that is a façade for property (read/write in defaultvalue feature).

These two facades are in the plug-in plug-in org.odf.ctrte.nfp, package com.cea.nfp.popup.actions.

6.2.2. Papyrus specific

Facades are in the package com.thalesgroup.marte.papyrus.nfp.modelAcess, in the plug-in com.thalesgroup.marte.papyrus.nfp.ui

The implementation for papyrus of the IModelFacade is PapyrusModelFacade.

There are two VSLResourceFacade specifics for papyrus: StereotypeResourceFacade and StereotypePropertyResourceFacade. Both are used for stereotype property.

6.2.3. RSA specifics

Facades are in the package com.thalesgroup.nfp.rsa.facade of the plug-in com.thalesgroup.marte.rsa.nfp.ui.

The implementation for RSA of the IModelFacade is RSAModelFacade. We used the model indexation mechanism of RSA for model search.

There are 3 VSLResourceFacade, specific for RSA: PropertyResourceFacade, StereotypeAttributeFacade and StereotypePrimitiveTypeFacade.

In RSA, the treatment for Datatype resolution and VSL in stereotype property is special, due to the implementation of MARTE for RSA. Because of the limitation of RSA for profile definition, the types in MARTE that shall be datatypes are Class (bearing the same name) in the internal marte lib. Those class contains a value feature (type string) to stock VSL Expression.

Therefore, RSAModelFacade.typeOf method is special, as the VSLResourceFacade implementation for Stereotype property.

6.3. The Editor

The view and controller class of the editor (referring to MVC pattern) are located in the plug-in org.odf.ctrte.nfp, package com.cea.popup.action, and follows Eclipse Editor pattern. The model is located of the plug-in com.cea.nfp.parser, package com.cea.nfp.parsers.modelgenerator.

- **VSLLavelEditorDialog:** The view, this class builds the graphic part of the editor and configures the listeners and validators. The buttonPressed method is the method that saves the VSL String in the model, delegating to the VSLResourceFacade.
- **VSLLabelValidator**: This class informs the editor about the validity of the text. This is used to disable or enable the OK button and display error message.
- **DocumentListener**: This class is in charge of calling the validator for each input change.
- **TVLLabelKeyListener**: The ContentAssist controller (Triggered on Ctrl-Space).
- **EnterKeyListener**: The Save & Close Controller (Triggered by Entry).
- **VSLGenerator:** The Model. Called by VSLLabelValidator.isValid().

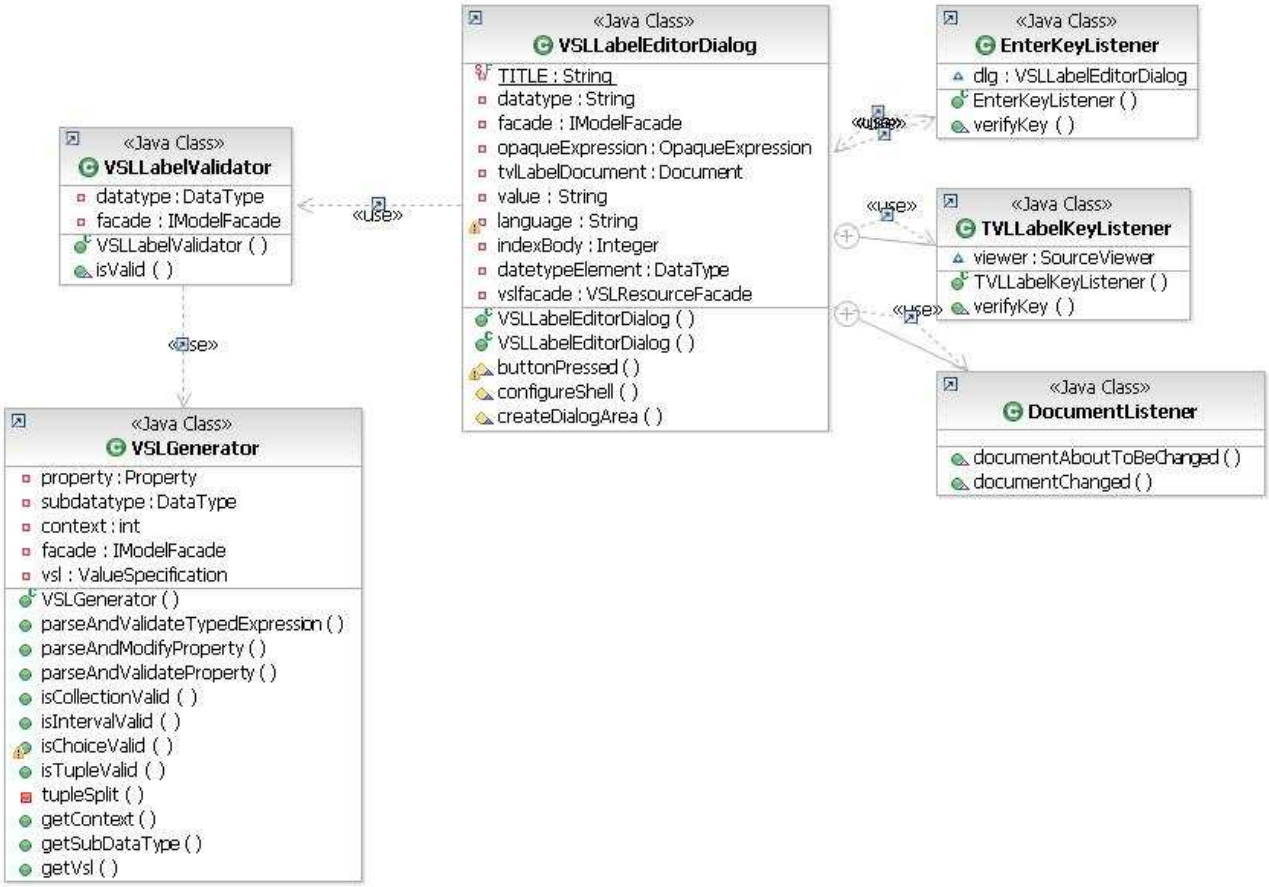

Figure 24: Editors classes

6.4. The Content Assist

The classes of the content Assist are located in the com.cea.parsers plug-in.

- **VSLLabelCOmpletionPrecessor:** this class is called when the user press « Ctrl-space » by the Eclipse completion system. The called method is computeCompletionProposals that use the expected type, the cursor position and the context (computed by the VSLModelGenerator) to call a specific ICompletionProposalComputer (or several).
- **IcompletionProposalComputer:** The generateCompletionProposal method is in charge of computing a list of IcompletionProposal to be displayed. The package com.cea.nfp.parsers.texteditor.completionproposals contains an implementation of the interface for each VSL language construction.

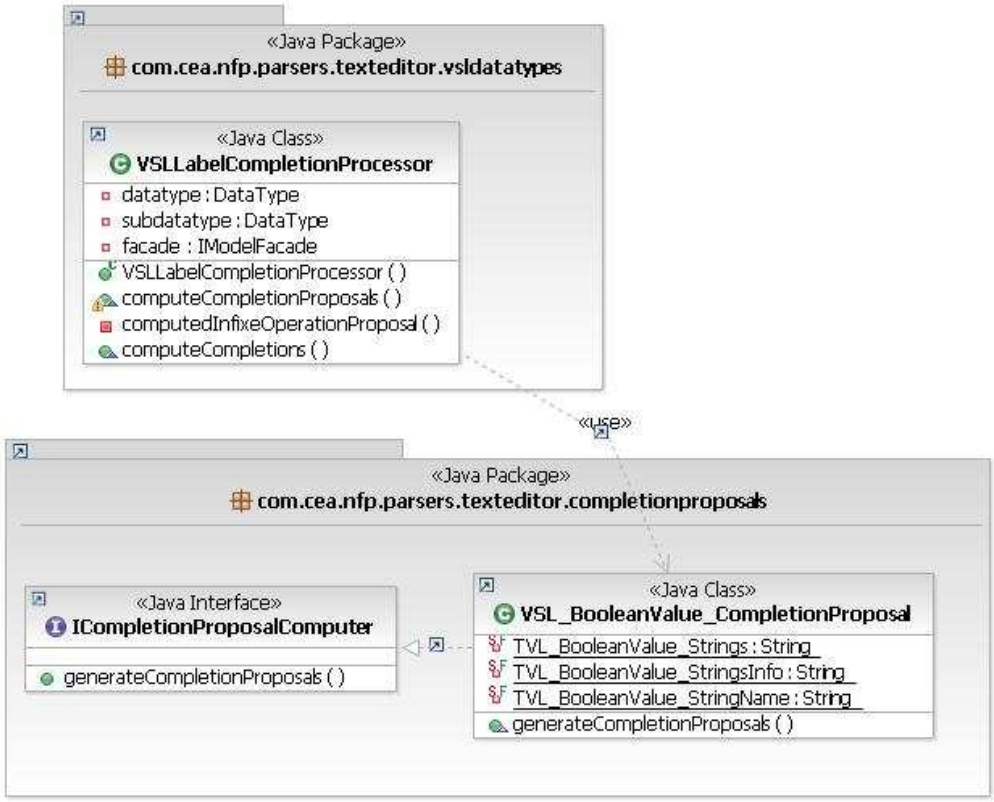

Figure 26 : Content Assist Classes

6.5. The Parser

The parser has been generated by Antlr, a LR parser (see http://www.antlr.org/).

Antlr files (.g extension) are located in the plug-in com.cea.nfp.parsers plug-in, folder resources/grammars. VSL_Lexer.g defines lexemes and VSL_Parsers.g defines the grammar. The generated classes are located in the antlrsrc folder.

The 2 important classes are VSLLexer and VSLParser. Those are used for the parsing. The generation of the parser is made with the antlr.jar using the ant file resources/build.xml, target *generateGrammar*.

The parser builds an Abstract Syntax Tree (AST) corresponding to the VSL String. This AST is an object extending org.uml2.uml.ValueSpecification, mainly from the classes defined in the com.thalesgroup.marte.vsl. plug-ins.

The AST generated by VSLParser is not complete and the class Link is to be used to complete it. This mechanism is described by the figure 27 and an example is presented in the figure 28.

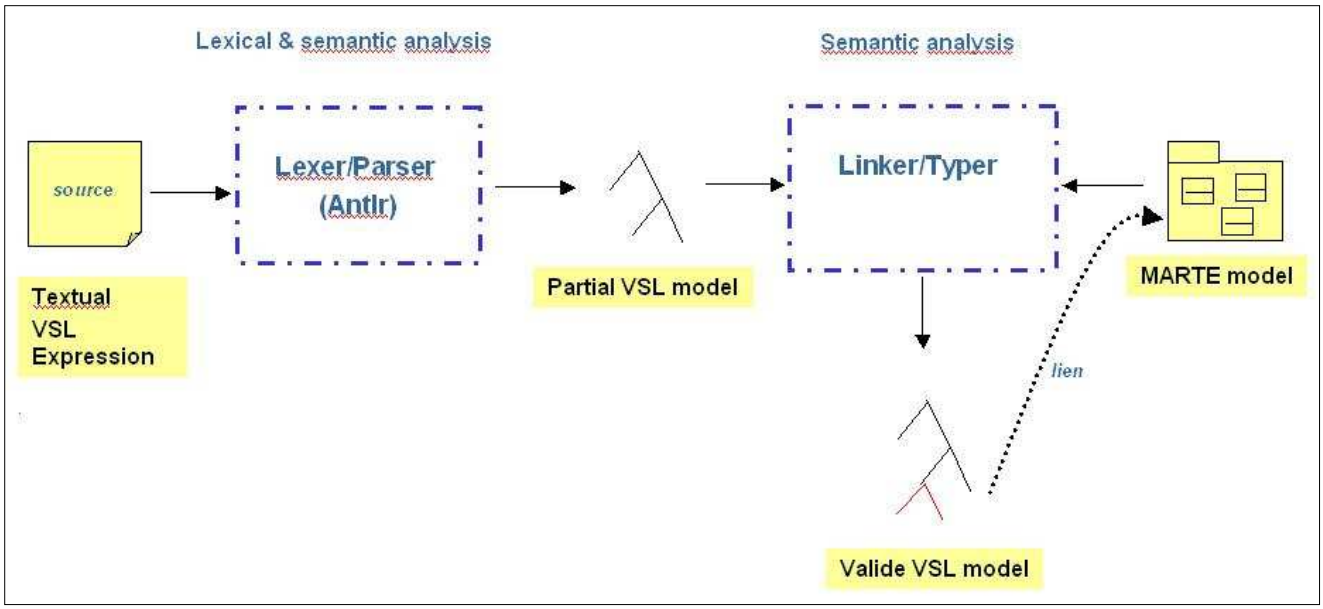

Figure 27: Parser functioning (component)

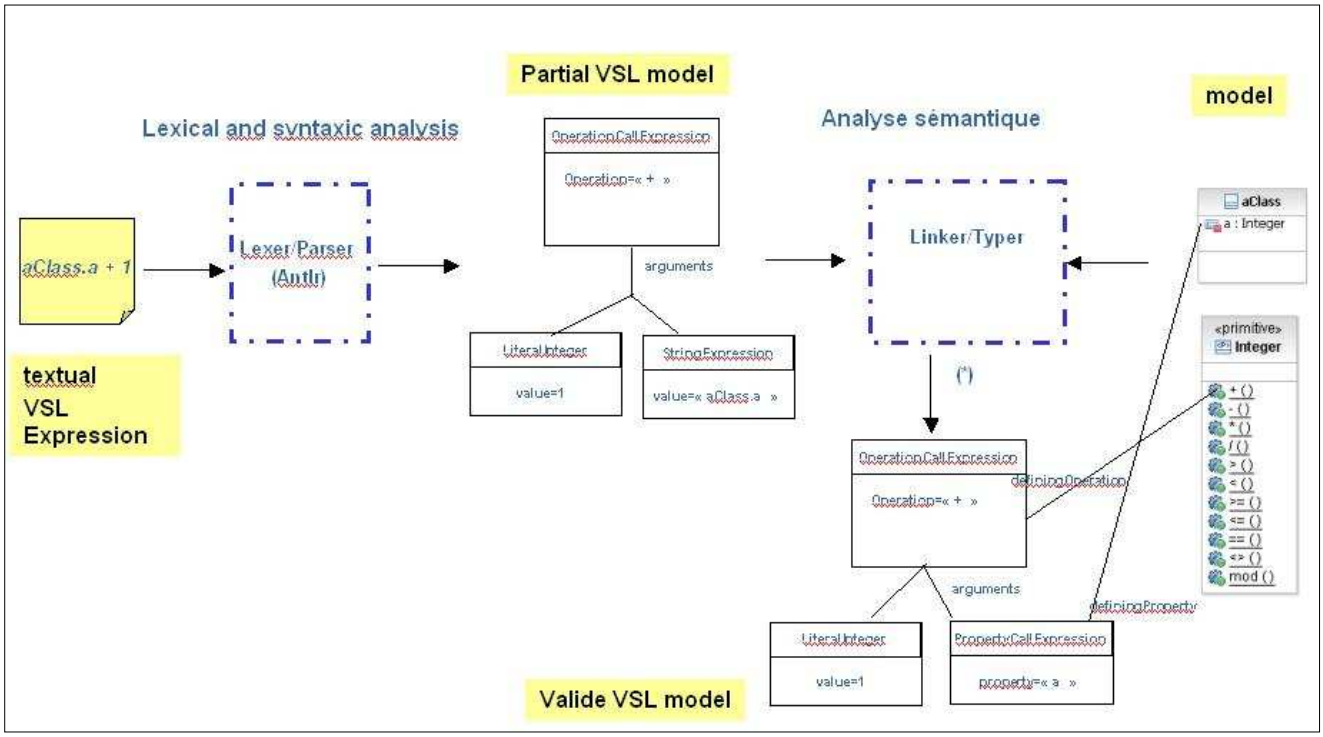

Figure 28: example of the parser functioning

The VSL complete construction is done by:

- Building an OperationCallExpression, which operation is not resolved and one of the arguments is an unresolved identifier.

- Resolving “a.Class.a” the model as a property typed by Integer and resolving the operation “+” in this datatype.
- Building the complete VSL Expression.

6.6. The Typer/Linker

The Typer and the Linker are located in the com.cea.nfp.parsers plug-in, package com.cea.parsers.modelGenerator.

Linker: This class is in charge of building a complete VSL model. Its first task is to link the VSL with the model (by resolving references) and check its correctness. For the tupleSpecification resolution, a special class TupleLinker is in charge.

TupleLinker: This class resolves tuples, linking them with their matching datatype.

Typer: This class is used to type any VSL Expression and to check type correctness. It cannot be used for a non-linked VSL.

Those classes are the principal users of the IModelFacade.

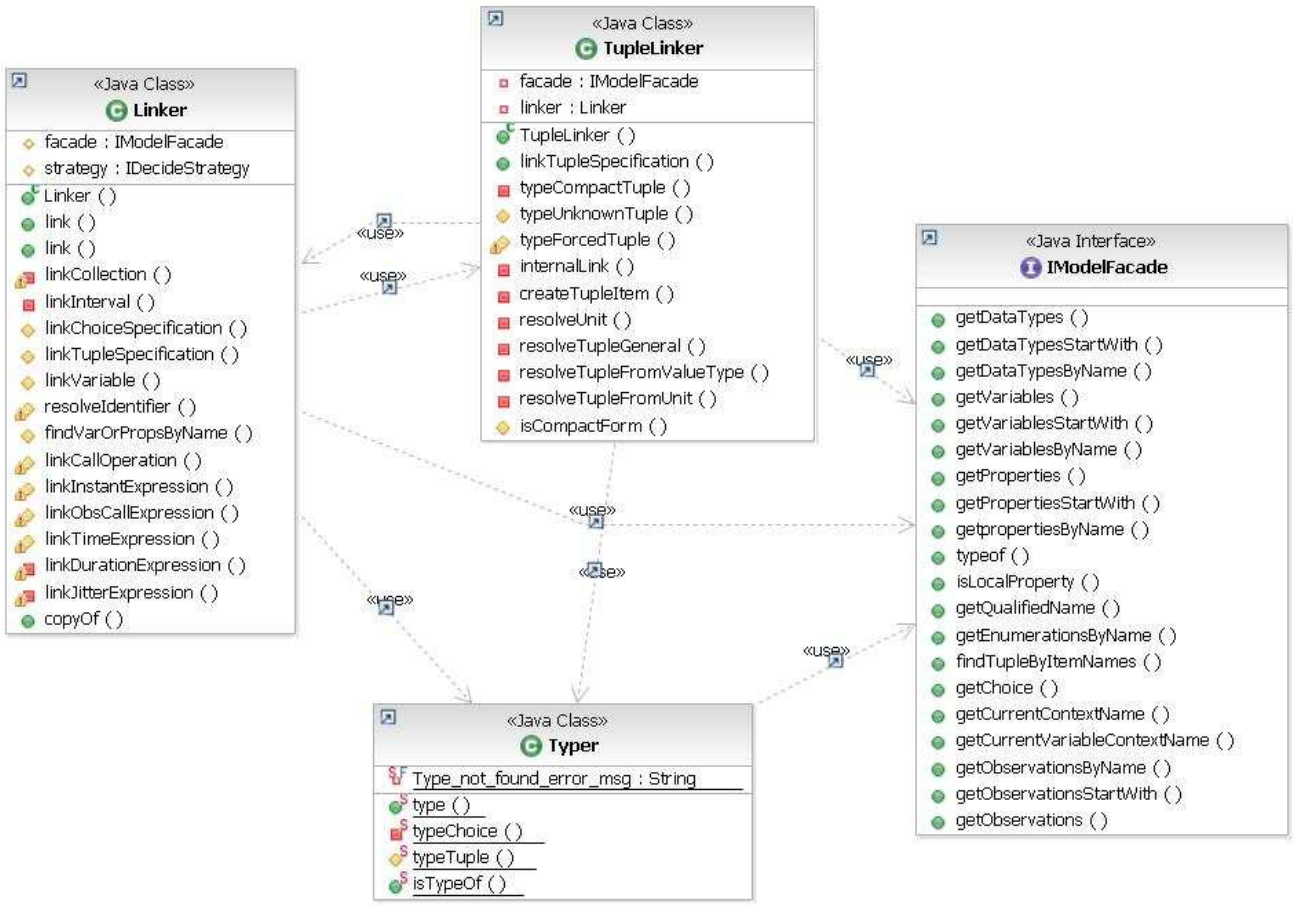

Figure 29: Linker and Typer classes

7. USING VSL PARSER THROUGH API

To fully parse a VSL String, one can use the class com.cea.nfp.parsers.modelgenerator.VSLFullParser, located in the com.cea.nfp.parsers. plug-in.

Two methods are available:

```
public static ValueSpecification parse(String text, DataType expected,
                IModelFacade facade) throws Exception
public static ValueSpecification parse(String text, IModelFacade facade)
                throws Exception
```

The typical invoking code is:

```
NamedElement element ;
 String vslStr ;
IModelFacade facade = new RSAModelFacade(element.getModel(), element);
DataType type = facade.typeOf(element);
Try {
 ValueSpecification vsl = VSLFullParser.parse(vslStr, type, facade);
 …
} catch (Exception e){
      …
}
```

element is the element « bearing » the ValueSpecification in the moder. It is used to compute the naming context (ContextExpression).

The ImodelFacade to used depends of the modeler. RSAModelFacade for RSA, PapyrusModelFacade for Papyrus and UMLModelFacade for Eclipse UML (or any of your own implementation).

The following Table shows which datatype to use for which elements.

VSL	Element	Datatype Expected
A slot value	The slot	facade.typeof(slot.getDefiningProperty())
The default value of a property	the property	facade.typeof(property)
The body of a constraint	the Constraint	facade.getDatatypeByName(« Boolean »)
The value of a stereotype property	The class on which the stereotype is applied	Facade.typeof(property)

This list is not complete considering all the capacities of the VSL Language. Futures developers may add new rules.

The ecore VSL Meta model and the emf generated classes are located in the com.thalesgroup.marte.vsl plug-in.

8. MODIFICATION OF MODELING TOOLS

8.1. Required modifications in papyrus

For VSL to be saved in the model for slot value, 2 modifications are to be done in the classes com.cea.papyrus.ui.composites.SlotComposite (method getLabel (ValueSpecification)) and com.cea.papyrus.classdiagram.editpart.SlotEditPart (method getLabel (ValueSpecification)) of the plug-in com.cea.papyrus.

In both case, code have to support to possibility of having a ValueSpecification (right now, a ClassCastException is thrown).

9. FAQ

The VSL editor display an error message saying « + » is not define in PrimitiveType (e.g. Integer)

A common error is to use Primitive type from Standard UML lib instead of Marte lib. Only PrimitiveType from marte lib define those operations.

How a VSL expression typed VSL_Expression is handled?

VSL_Expression (from Marte library) is considered as the super type of all Datatype, hence when the expected type is VSL_Expression, any VSL expression is accepted.

10. KNOWN ISSUES

The following grammar rules are not correctly handled:

- TimeInterval are parsed as IntervalSpecification

- ChoiceSpecification with the chosenAlternative not provided are not handled (e.g. (period=1.0, jitter=1.0) instead of periodic(period=1.0, jitter=1.0)
- Variable with VariableDirectionKind not provided are not handled (e.g. $myVar:Integer instead of in$myVar:Integer)
- Variable with datatypeName not provided are not handled (e.g. in$myVar instead of in:myVariable:Integer)
- OperationCallExpression with no arguments and parenthesis not provided are not handled
- Infix operator cannot be written normal (e.g. 1.+(1))
- DurationExpression in the (obs when expr – obs) form are not correctly handled. Due to the lack of parenthesis.
- InstantExpression in the (obs when expr + obs) form are not correctly handled. Due to the lack of parenthesis.
- Expressions in interval like [a+1..a+2] are not correctly handled. Due to the lack of parenthesis.
- TupleSpecification when tuple item names are not correctly handled (e.g (1, 2) instead of (a=1, b=2)). The only exception is when one of the tuple item is an unit.

More details about known issues and limitation can be found in the following two document “Test VSL for Papyrus.doc” and “Test VSL for RSA.doc”.