Vercors Component Environment

Mikolaj Baranowski
email: mikolaj.baranowski@gmail.com

October 23, 2008

Contents

1 Introduction

This document explains usage of Vercors Component Environment and also implementation of its import/export feature.

2 Installation

2.1 Dependencies

2.1.1 Environment.

To run VCE on your machine, you need JAVA 1.5 or higher and Eclipse 3.3 (minimum).

2.1.2 Plugins.

Uncompress fallowing archiwes to your eclipse directory.

- emf-sdo-xsd-SDK-2.3.1.zip
- mdt-ocl-SDK-1.1.1.zip
- emf-query-SDK-1.1.zip
- emf-transaction-SDK-1.1.1.zip
- emf-validation-SDK-1.1.1.zip
- GEF-ALL-3.3.1.zip
- GMF-sdk-2.0.1.zip
- mdt-uml2-SDK-2.1.1.zip
- org.topcased.sdk-R-1.2.0-200712131010.zip

2.1.3 Vercors Component Environment.

Download fallowing jars to your plugin directory.

- fr.inria.oasis.vercors.vce_2.0.1.0631.jar
- fr.inria.oasis.vercors.vce.adl_2.0.1.0631.jar
- fr.inria.oasis.vercors.vce.diagrams_2.0.1.0631.jar
- fr.inria.oasis.vercors.vce.model_2.0.1.0631.jar
- fr.inria.oasis.vercors.vce.model.edit_2.0.1.0631.jar
- fr.inria.oasis.vercors.vce.model.editor_2.0.1.0631.jar

3 Creating diagrams

Fallowing figure sequences present typical model/diagram creations.

3.1 Creation of diagram.

Figure 1: To create new, empty diagram, select **New** → **Other**

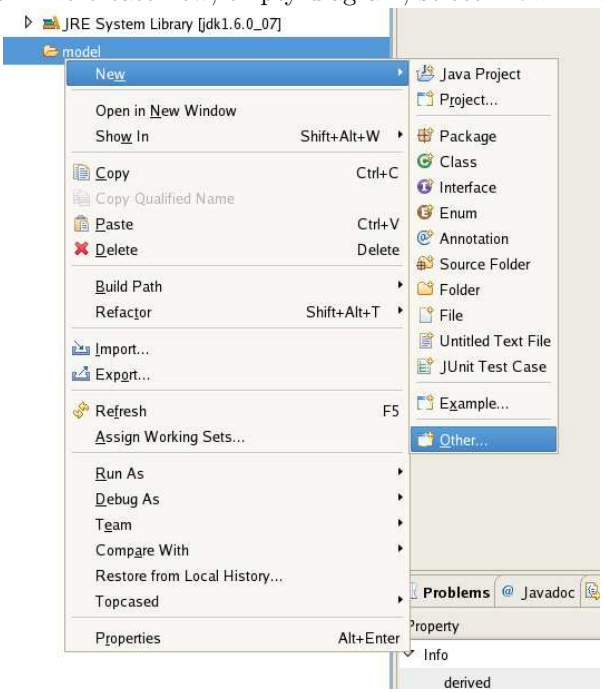


Figure 2: **Vercors Component Environment → Component Diagram.**

Figure 3: Then you can specify directory, model name and template.

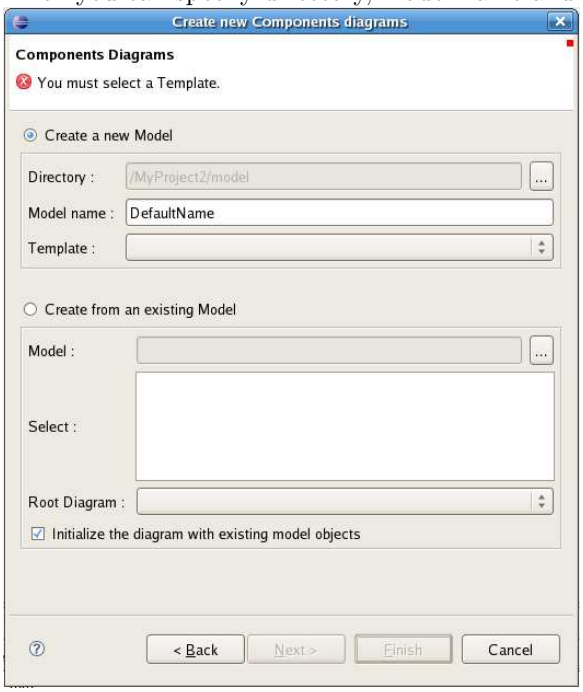


Figure 4: Click **Finish** to create diagram.

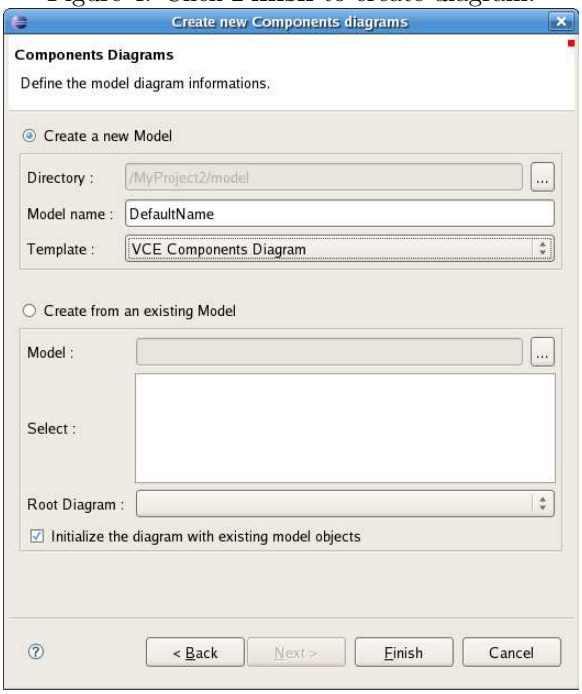


Figure 5: New diagram.

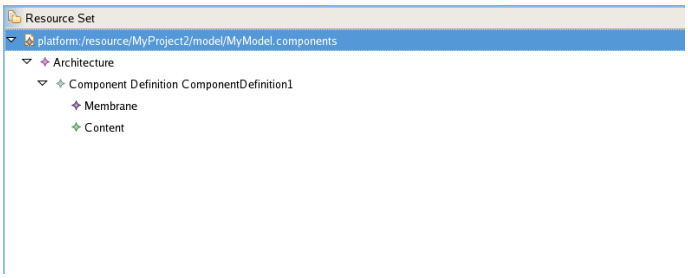


Figure 6: New model.

3.2 Creation of model.

VCE gives you a possibility to create empty model without associated diagram.

Figure 7: Select: **New** → **Other** → **Vercors Component Environment** → **Component Model**.

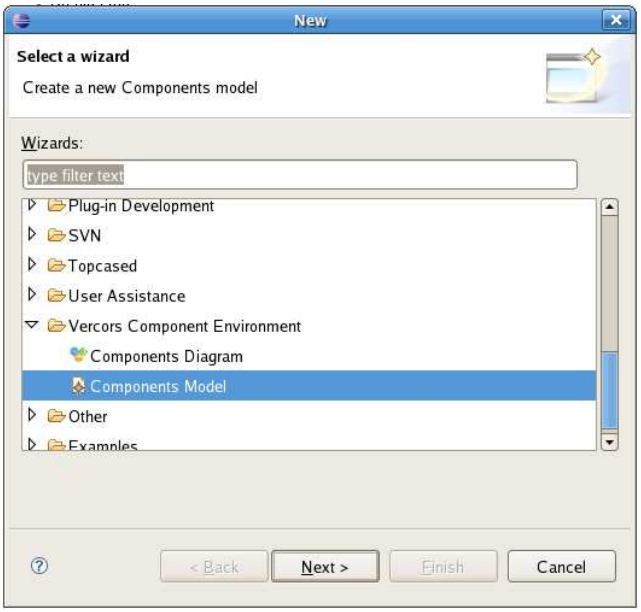


Figure 8: Then, specify model name.

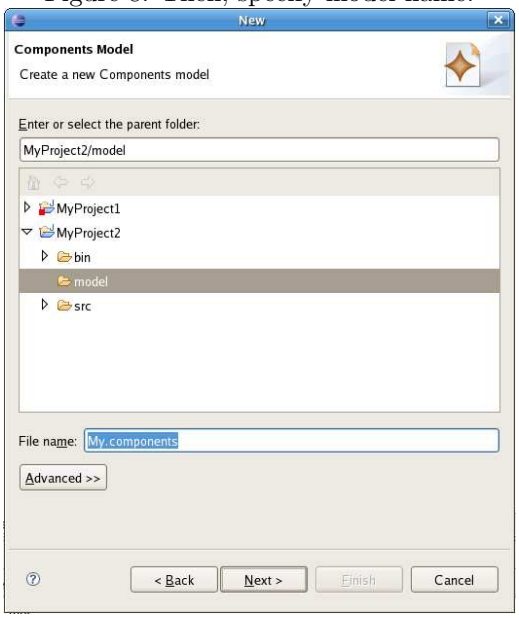
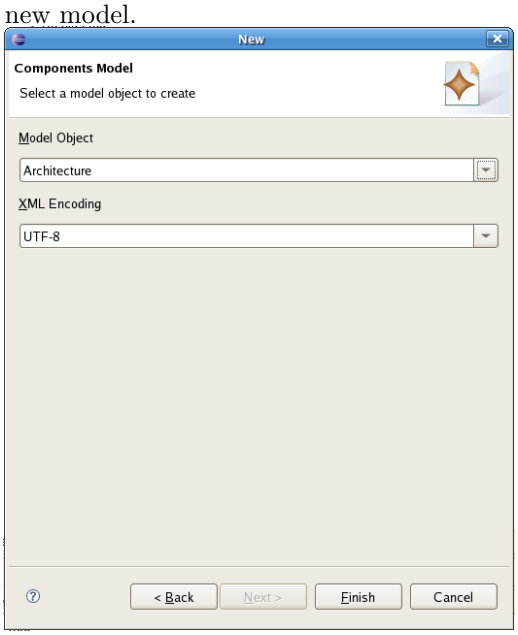


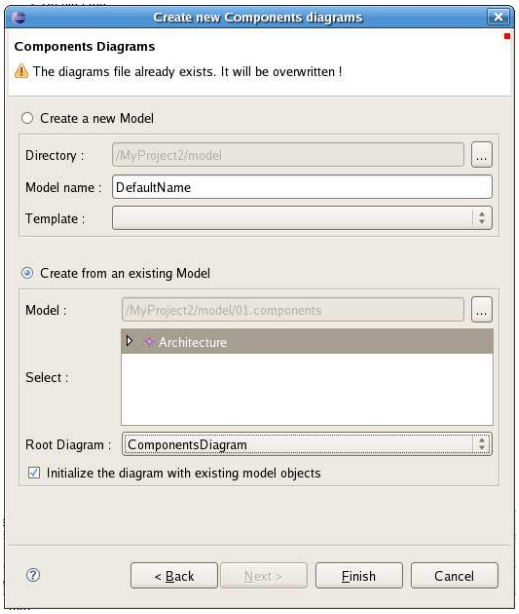
Figure 9: Select model root object (by default it is **Architecture**) and click finish to create a new model.



3.3 Creation of diagram on existing model.

To create diagram file based on existing model, select: **New → Other → Vercors Component Environment → Component Diagram.**

Figure 10: Then you need to select *Create from existing Model*, path to Model and *Root Diagram*.



3.4 Creation of more than one diagram for the same model.

Unfortunately, for now, you can not specify diagram file name in diagram creation wizard and if there is already one it will be overwritten. But, you can go around this problem by changing file name before doing steps from section 3.3.

3.5 Changing signatures and content classes.

Figure 11: You want to create your own interface ...

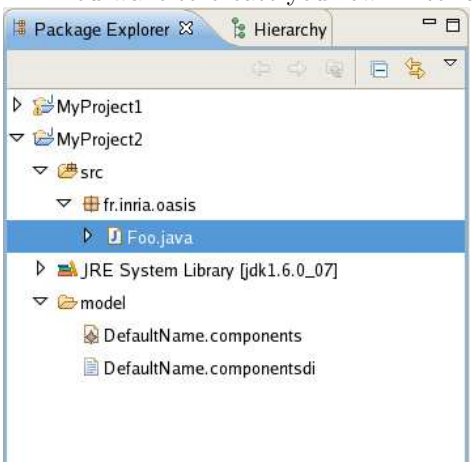


Figure 12: ... and set primitive content class, in context menu select **Change Model Properties → Change Primitive Content Class**

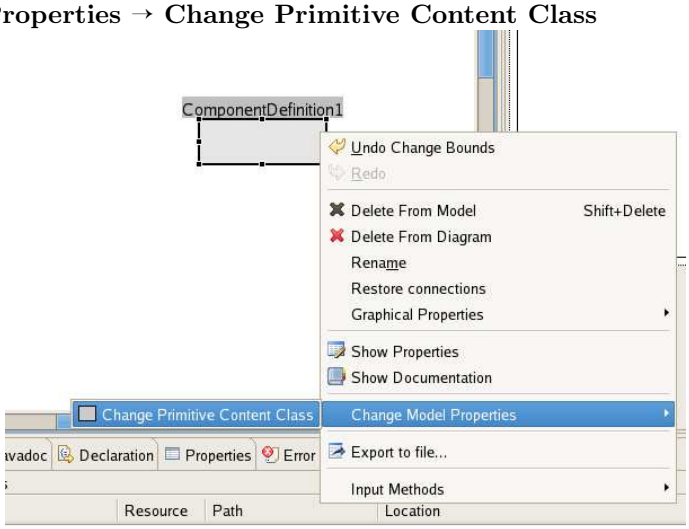


Figure 13: Or, to set interface signature, select **Change Model Properties → Change Interface Signature**.

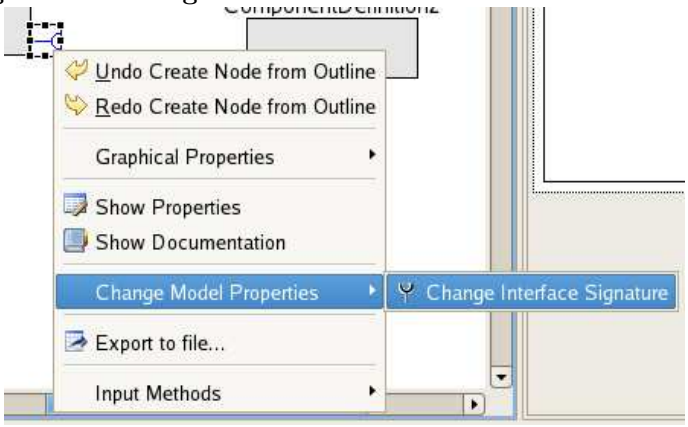
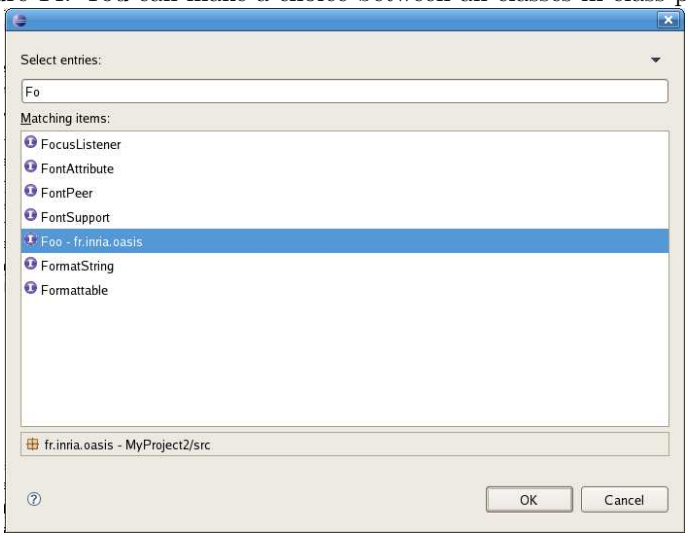


Figure 14: You can make a choice between all classes in class path.



4 ADL.

The Fractal Architecture Description Language[2] is a XML-based language used to define component architectures. VCE allow us to translate models between ADL and VCE internal representation.

4.1 Importing

Figure 15: From context menu select **Import**

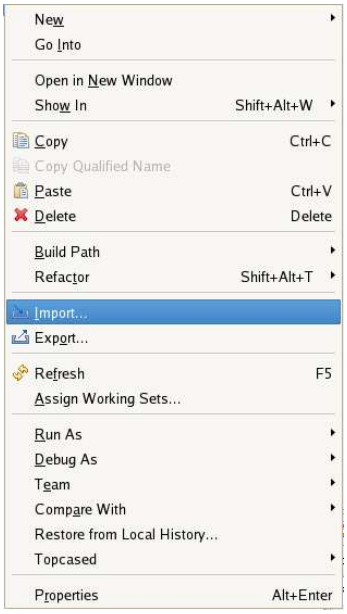


Figure 16: From **VCE Import Wizards** group select **ADL Import Wizard**

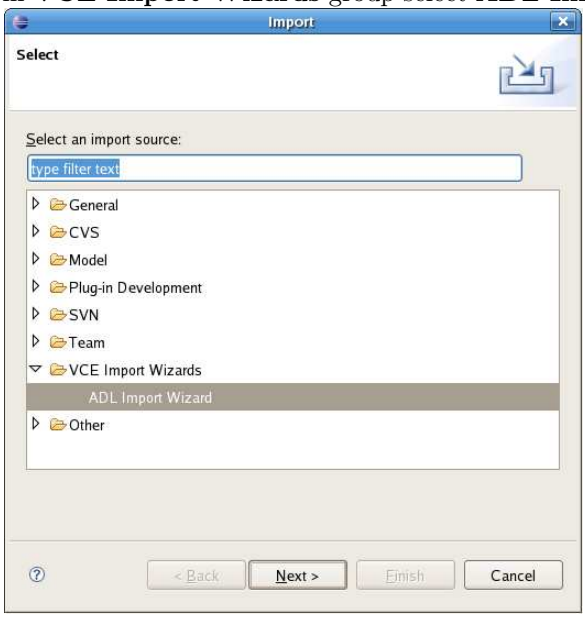


Figure 17: Select ADL files to import and specify target directory. For now, you can not set model file name explicate. It is composed from ADL file name (sequence *.fractal* is changed for *.components*).

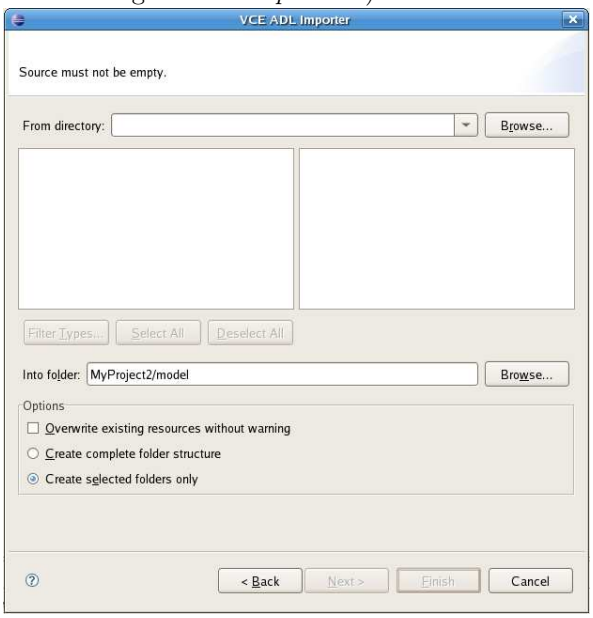


Figure 18: Click finish to create imported models and diagrams.

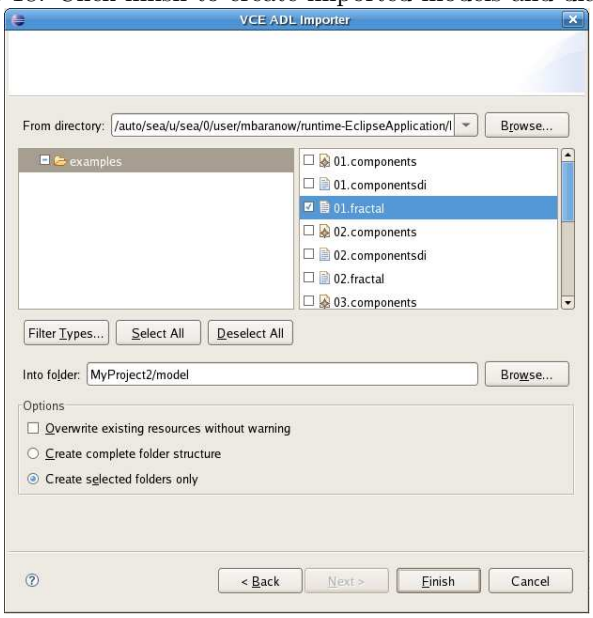


Figure 19: New files.

4.1.1 Importing with coordinates

By default, diagram elements are initialized with coordinates from ADL file.

4.2 Exporting

Figure 20: From context menu select **Export**

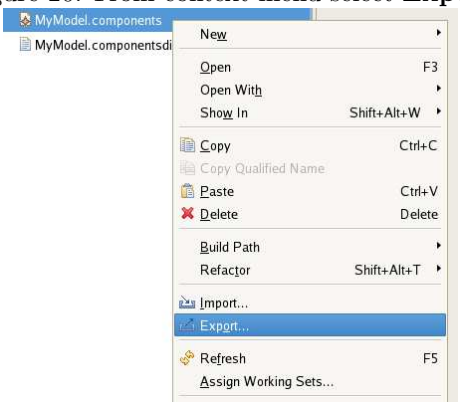


Figure 21: Select item **ADL Export Wizard** from **ADL Exports Wizards**

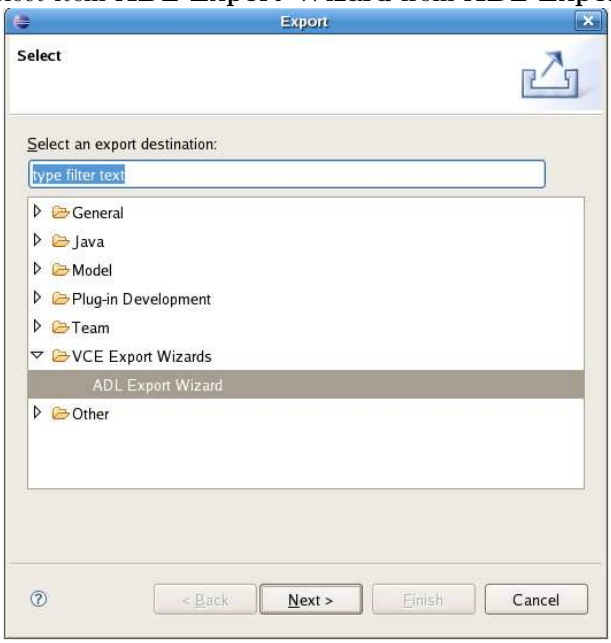


Figure 22: Select models which you want to export to ADL.

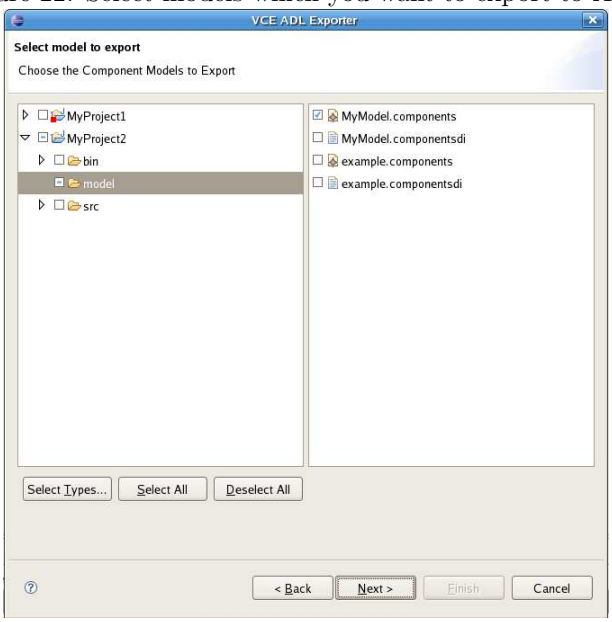


Figure 23: Specify ADL file name.

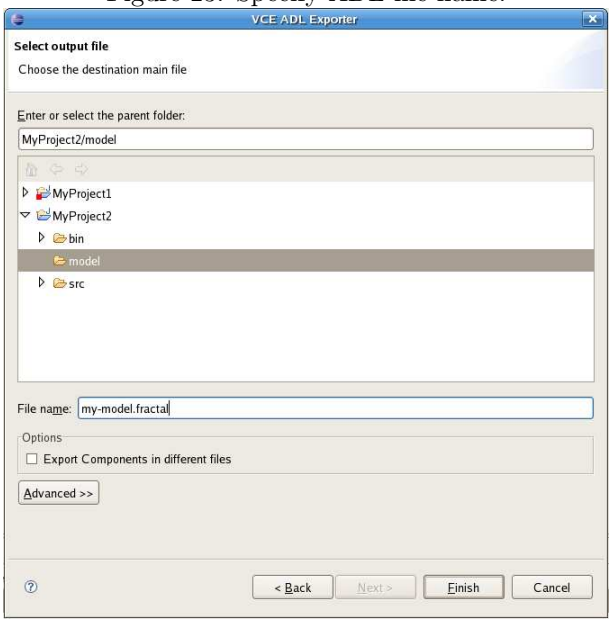
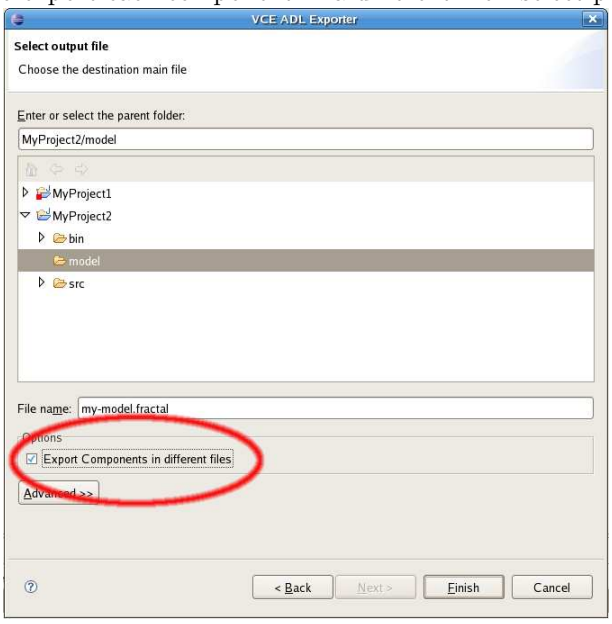


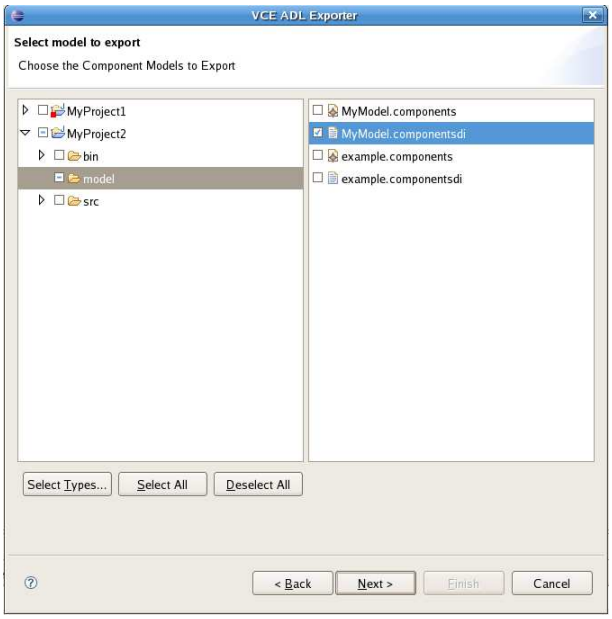
Figure 24: To export each component in a different file - select proper option.



4.2.1 Exporting with coordinates

VCE provides functionality which lets you to export model with diagram coordinates. It means that you can keep sizes and positions of diagram figures in ADL file and restore them in different tool or with import feature.

Figure 25: To export diagram coordinates you need to make export from diagram – not like before – from model file.



4.3 Translation of internal interfaces

ADL definition doesn't keep information about internal interfaces. It means, with exporting you are loosing all information: name, signature and cardinality of internal interface.

Figure 26: Exporting collective internal interface. From left to right: before exporting, visualization of ADL representation, after importing from ADL. Because internal interface has only one binding, it is restored as singleton interface.

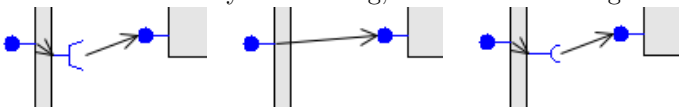


Figure 27: From left to right: before exporting, visualization of ADL representation, after importing from ADL. In that case, internal interface has two bindings and it is restored as collective interface then.

4.4 Implementation

4.4.1 Importing from ADL

The main import class is `fr.inria.oasis.vercors.vce.adl.wizards.ADL-ImportWizard`.

`ADLImportWizardSelectionPage` (figure 18) is used to provide basic graphical interface which lets user to select ADL files to import and specify target directory.

Translating process VCE import feature is developed using objectweb fractal loader[1]. The key to understand translating process is method `performFinish` in class `ADLImportWizard`. It uses component loader defined in `fr.inria.-oasis.vercors.vce.adl.VCELoader`.

`VCELoader` extends default loader provided from objectweb `org.objectweb.-fractal.adl.BasicLoader` which uses classloader to load every signature and content class used in diagram. The alternative to changing classloader is to remove it from loading process.

After all, we don't want to instantiate these classes but only get their names. `VCELoader` limits this functionality by using classes `XMLLoader`, `VCETypeLoader`, `VCETypeBindingLoader`, `VCEImplementationLoader` from `fr.inria.oasis.-vercors.vce.adl` package.

Each ADL module (such as component, interface, coordinate) has proper `analyze*` function. For example: method `analyzeComponent` which has two arguments: `component` (instance of `org.objectweb.fractal.adl.components.-Component` and `componentDefinition` (instance of `fr.inria.oasis.vercors.-vce.model.components.ComponentDefinition`) gets information from `component` (which contains data from ADL file) and puts them to `componentDefinition` (which represents component in VCE model).

Importing internal interfaces This process starts in `analyzeBinding` method in `ADLImportWizard` class. When function finds binding between interfaces of the same type, method `makeServerClient` or `makeClientServer` is invoked (they are named after palette item in diagram editor which creates these figures).

In the easiest case, mediator interface is created and two bindings, one to each external interface.

Situation is more complicated for collective internal interfaces. This case is recognized in condition: `if (clientServerMap.containsKey(sourceInterface))` or its equivalent in other method. It means that there was already one connection from `sourceInterface`. Mediator interface which already exists is changed for new collective interface.

4.4.2 Exporting to ADL

VCE export feature is implemented using JAXB. In `performFinish` method, after validation, model is passed to `ADLModelTranslator`. There, every part of model has a proper method.

Translating process TODO

Exporting internal interfaces Actually, I should name this process "binding translation" – I need to keep connections between external interfaces (look at figures 26 and 27) without using internal interfaces.

Binding translating process is based in `caseInterface` and `caseBinding` methods in `ADLModelTranslator` class. In first one, all processed interfaces are stored in `interfaceSet` variable.

In fact, this method analyzes only external interfaces and this is my intention. `caseBinding` uses this variable to distinguish between internal and external interfaces.

In second method, every interfaces of binding from server interface to client interface are stored in `interfaceInterfaceTranslator` map. Internal interface becomes a key and external interface - value. This information lets me to make a binding between external interfaces in the next step.

4.4.3 Coordinates

Coordinates are represented by 6 attributes:

- `x0` which keeps horizontal coordinate of left top corner of figure
- `x1` which keeps horizontal coordinate of right bottom corner of figure
- `y0` which keeps vertical coordinate of left top corner
- `y1` which keeps vertical coordinate of right bottom corner
- `name` which keeps name of figure (coordinates are distinguished by names)
- `color` – not obligatory attribute

Coordinates are normalized. For example, if `x0` and `y0` equal `0`, left top corner of diagram element is based in left top corner of available space and if `x1` and `y1` equal `1`, right bottom corner of diagram element is based in right bottom corner of available space.

Importing coordinates Coordinates importing process is placed in `ADLImportWizard` class. Coordinates from ADL file are stored in `coordinetesContainer` field and analyzing process begins in `saveModel` method where diagram file is initialized. After initialization, `initializeContent` method is invoked. Arguments are:

1. `Elist<org.topcased.modeler.di.model.DiagramElement> elements`
   this variable keeps diagram elements from same layer
2. `EList<fr.inria.oasis.vercors.vce.model.components.Component> components`
   this variable keeps components from same layer
3. `Coordinates[] coordinatesContainer`
   this variable keeps coordinates from same layer
4. `double xSize`
5. `double ySize`

Variables `elements`, `components`, `coordinatesContainer` represent trees. They are parsing at the same time. What is most important is that each layers of each tree has a correspondent layer in other trees. I mean - one layer of `DiagramElement` tree corresponds only to one layer of `Component` tree and one layer of Coordinates tree and vice versa.

Variables `xSize` and `ySize` keep height and width of parent diagram element (basically, it's size of parent component content). VCE keeps sizes explicate in pixels but, as I mentioned before, sizes in ADL are normalized. Then, position of every element is calculated by multiplying coordinates from ADL by one of these variables.

Condition `if(! matched)` is explained in a 4.5.

Exporting coordinates Coordinates exporting process takes a place when user is exporting model from diagram file. `''If''` statement with this condition is placed in `performFinish` method in `ADLExportWizard` class. If exporting element is identified as `DiagramsImpl`, `analyzeDiagramCoorinates` method is invoked.

`analyzeDiagramCoorinates` analyzes diagram element (there should be only one) and invokes `analyzeGraphNodeCoordinates` with diagram element as argument.

In `analyzeGraphNodeCoordinates`, if `GraphNode` represents `Component-Definition`, coordinates from VCE diagram are translated to ADL format and stored in variable `parentCoordinates`. Then, `analyzeGraphNodeCoordinates` is invoked for each content of diagram element.

4.5 Problems

There is a problem when component extends other component. Loader looks for parent component in classpath and, if classloader can not find it, it rises exception.

```
if(! matched)
```

5 TODO

- Importing – setting size of diagram.
- Exporting – coorinates relative to non-relative
- Importing – internal interfaces - remove this mapx

References

[1] *Fractal ADL Documentation*. http://fractal.objectweb.org/current/doc/javadoc/fractal-adl.

[2] *Fractal ADL Tutorial*. http://fractal.objectweb.org/tutorials/adl/index.html.