Structure101g UML Flavor

Structure101g UML Flavor

Chapter 1. Description

This UML/XMI flavor extracts the nodes and dependencies in the UML Diagrams encoded in XMI format. It currently supports the following UML Diagrams in the input UML model:

1. Usecase Diagram
2. Package Diagram
3. Class Diagram
4. Activity Diagram
5. State-machine Diagram
6. Communication Diagram
7. Component Diagram
8. Deployment Diagram

This UML/XMI flavor currently works with both Headway Software Structure101g and also ReStructure101g products. *But we encourage you to use with the ReStructure101g product since it makes better use of the nested nature of the structure in the UML models.*

One can also use it with Structure101g product, but currently we have a limitation that one can not create Architecture view from any of the other views, for UML model inputs.

This flavor is dependent on the following libraries:

1. Apache Xalan-J for input XMI transformation.
2. Aoache Xerces for XML parsing.

Chapter 2. Installation Steps

This section describe in detail, the installation steps for installing the base Headway (Re)Structure101g product and also the Tejas Software UML/XMI on top of it.

Step 1: *Installation of the Base (Re)Structure101g Product*

1. First, install the Headyway (Re)Structure101g product from Headway Web Site [http://www.headwaysoftware.com/products]

2. Copy the (Re)Structure101g license that you got thru' e-mail to the Structure101g Install Directory, usually *C:\Program Files\Structure101\s101g*

3. Start Structure101g. While starting for first time, it will ask for *flavor home directory* which is by default: *C:\Documents and Settings\user\structure101g\flavors*. This file locations is referred to, in this document as $FLAVOR_INSTALLATION_HOME.

Step 2: *Installation of the UML/XMI*

1. Install the UML/XMI flavor from the download site. This can be done by following the UI options:

   a. Flavors -> Install... -> Install Flavors screen

   b. Select “http://www.headwaysoftware.com/structure101/g/flavors/pre-release” in “Flavor site” field, if flavor is released.

   c. Select “http://www.headwaysoftware.com/structure101/g/flavors” in “Flavor site” field, if it is _not in pre-release. The flavor is installed in $FLAVOR_INSTALLATION_HOME as a separate version. *There can be multiple versions of the same flavor installed.*

2. Get the flavor evaluation license from Headway support.

Note: For preview flavors, evaluation license with a reasonable duration is already bundled with the flavor.

Step 3: Start using the flavor by importing the sample models bundled with the flavor in the $FLAVOR_INSTALLATION_HOME/com.tejassoftware.uml_<ver no.>/examples directory, as explained in the next section. But you should try out with your own UML model files as input.

Do not forget to provide your feedback & suggestions for improvement at the Tejas Software Tech Support [http://tejassoftware.com/techsupport.html]

Chapter 3. Flavor Usage Instructions

It is easy to start using the UML/XMI flavor since you just need the UML models exported from your UML editor of choice into the XMI format. Only restriction we have is that the XMI version should be 2.1. The steps required to create the v2.1 XMI varies from tool to tool, hence it is *not described* here. You may refer to the documentation specific to each tool.

Step 1: *Prepare your UML Model in XMI v2.1*

Step 1a: If XMI v2.1 Export is Support in UML Tool

From the UML tool that you are using, **export the complete UML model as XMI v2.1 file** using the steps specific to each tool. Or, you can use one of the XMI files bundled with the flavor for initial evaluation.

> $FLAVOR_INSTALLATION_HOME/com.tejassoftware.uml_{ver}/examples directory.

Step 1b: If XMI v2.1 Export is NOT Supported in UML Tool

If XMI v2.1 is not supported in the UML tool, but a lower version of XMI v1.x is supported, use the nsUML .NET Converter [http://numl.sourceforge.net/index.php/Main_Page] for converting it into XMI v2.1.

Step 2: *Creating a New Project for UML/XMI Flavor*

Create a new project for UML/XMI flavor by selecting the UI options: `File -> New which pops up the *New Project Dialog* window.`

Step 3: *Providing the Necessary Input Parameters*

The following configuration parameters are expected by the UML/XMI flavor.

1. The input folder containing the XMI files. One can either select a directory or only one XMI file.

Currently we support processing of only one XMI file but in the v1.0 release version, we will support multiple files with cross-references across the files also being processed in the transformation.

Step 4:

Press *Finish* button to complete the dependency extraction. Depending on the number of input files & their sizes, the extraction time can vary.

Step 5:

Once completed, you can explore the dependencies using the features available in the Structure101g user interface [http://www.headwaysoftware.com/products/structure101/g/] or ReStructure101g UI [http://www.headwaysoftware.com/tour/restructure101/intro/]

3.1. Supported Input Types and UML Editors

Currently only the XMI 2.1 format is supported. But the XMI input file could be using any one of the following UML versions:

UML Version	Namespace URL

OMG UML v2.0	http://schema.omg.org/spec/UML/2.0
OMG UML v2.1	http://schema.omg.org/spec/UML/2.1
OMG UML v2.1.1	http://schema.omg.org/spec/UML/2.1.1
OMG UML v2.2	http://schema.omg.org/spec/UML/2.2
OMG UML v2.3	http://schema.omg.org/spec/UML/2.3
OMG UML 09/2009	http://www.omg.org/spec/UML/20090901
Eclipse UML 2	http://www.eclipse.org/uml2/2.0.0/UML

Also, only recently OMG has removed ambiguities in the OMG XMI specification and has started a Model Interchange Working Group [http://www.omgwiki.org/model-interchange/doku.php] for doing XMI interoperability testing.

Until, most tool vendors take part in certifying their products for conformance to the MIWG Test-suite [http://www.omgwiki.org/model-interchange/doku.php#test_suite] , we will make sure that the variations in XMI encoding are taken care of, in the XMI exported formats of the tools listed in the table below:

UML Editor	**Versions Tested With**
Sparx EA	8.0
NoMagic’s MagicDraw	16.9
Visual Paradigm for UML	7.0
Altova’s UModel	2011
IBM’s Rational Software Modeler	7.5.4
IBM’s Rational Software Architect	8.0
OMG’s MIWG Test Suite	Release 11

Chapter 4. Modules/Sub-modules Types & their Dependencies

This section describes the nodes and dependency types extracted from UML XMI inputs & represented in the (Re)Structure101g Dependency Model.

4.1. Node/Module Types

Each Node/Module type represent a specific entity in the domain of analysis. For the UML/XMI, we have defined a node or module type for the entities represented in table below. Each module type has its own icon which is used in the UI of the tool.

Node Type	Icon	Description
uml-model		Represents UML Model, root node of the structure
diag-class		Represents a class diagram, *NOT USED currently since difficult to identify in XMI*.
package		Package in a class diagram
class-abstract		Abstract class in a class diagram
interface		Interface in a class diagram
class-concrete		Instantiable class in a class diagram
class-association		Class which represents an association between 2 or more classes
component		A component in a component diagram
diag-uc		Usecase Diagra, *NOT USED currently since difficult to identify in XMI*.
actor		Actor in an usecase diagram
usecase		Usecase in an usecase diagram
diag-stmc		State-machine diagram
state-start		Start state
state-end		End State
state		Any other intermediate state

stmc-region		State machine region, used as nested node
diag-act		Activity diagram, *NOT USED currently since difficult to identify in XMI*.
join		Join pseudo state in an activity or state-machine diagram
junction		Junction pseudo state in an activity or state-chart diagram
fork		Fork pseudo state in an activity or state-machine diagram
entry-pt		Entry point to a compound/ complex/sub-state
exit-pt		Exit point from a sub-state.
conn-pt		Connection point in activity or state-chart diagram
action-state		Action node in an activity diagram
activity		Activity which is expanded in an activity diagram
swimlane		Swimlane representing a role in an activity diagram
action-snd-signal		Send signal action in an activity diagram
action-acc-evt		Receive event action in an activity diagram
region-interruptible		Interruptible region in an activity diagram, *NOT supported currently*.
region-expansion		Expansion region in an activity diagram
expansion-node		Expansion node in an activity diagram
diag-collab		Communication or Collaboration diagram
collaboration		Collaboration in an usecase or collaboration diagram
object		Object which is instance of class, used in collab. diagram
diag-deploy		Deployment diagram, *NOT USED currently since difficult to identify in XMI*.
node		A deployable node in a deployment diagram

deploy-spec		Deployment spec. for a deployable artifact
artifact		An artifact, mostly refers to one which is deployable
note		Comment for a node, *comment attached to an edge is NOT supported.*

4.2. Sub-module Types

Sub-modules are nested types that are nested within modules types. The table below shows the sub-module types that are defined. For each sub-module type, the valid parent module-type within which it can occur, is also indicated.

Sub-Module Type	Icon	Description
data-type		UML Data-type which is used in other diagram elements.
property		Property for a class or object.
method		Method in a class.
method-param		Method param in a method.
method-return		Method return type if there is any.
extn-point		Extension point in an usecase.
input-pin		Input pin for an action-state.
output-pin		Output pin for an action-state.
activity-param		Parameter to an action-state in activity diagram.

4.3. Dependency or Edge Types

Dependency or edge types are defined for connections or links between two modules or sub-modules. A dependency or edge is directed. The table below shows the types & their description. Each dependency type is valid only between certain types of modules or sub-modules. This is also indicated in the table below.

Dependency Type	Description
associated-with	Class is associated with a another class. Converted from uml:Association.
depends-on	Entity depends on another entity, converted from uml:Dependency.
aggregate-of	Class is an aggregation of set of another class.
composed-of	Class is composed of a set of another class.

inherits-from	Class inherits from another class.
is-of-type	Object is of type of a class
transition	State transitioning to another state
object-flow	Object flow from one action state to another in an activity diagram.
control-flow	Control flow from one action state to another in an activity diagram.
error-flow	Error path from one action to an exception handler node.
is-in-state	Class object instance is in a particular state. Usually occurs in Activity diagram.
calls-async	Object instance calls synchronously another object instance member function.
calls-sync	Object instance calls asynchronously another object instance member function.
realizes	Component realizes an interface.
uses	Component uses an interface
includes	Usecase includes another Usecase.
extends	Usecase extends another usecase via an extension point.
refines	Usecase refines another usecase.
connected-to	Node is connected-to another node in deployment diagram.
same-as	Node is same as another, used in obly entry & exit nodes to a compound state in State-Machine Diagram.
deployed-in	A component deployed in a node in deployment diagram.
note-link	Link from a note to the annotated element, which can be only another node, currently.

Chapter 5. UML Diagrams Supported and their Details

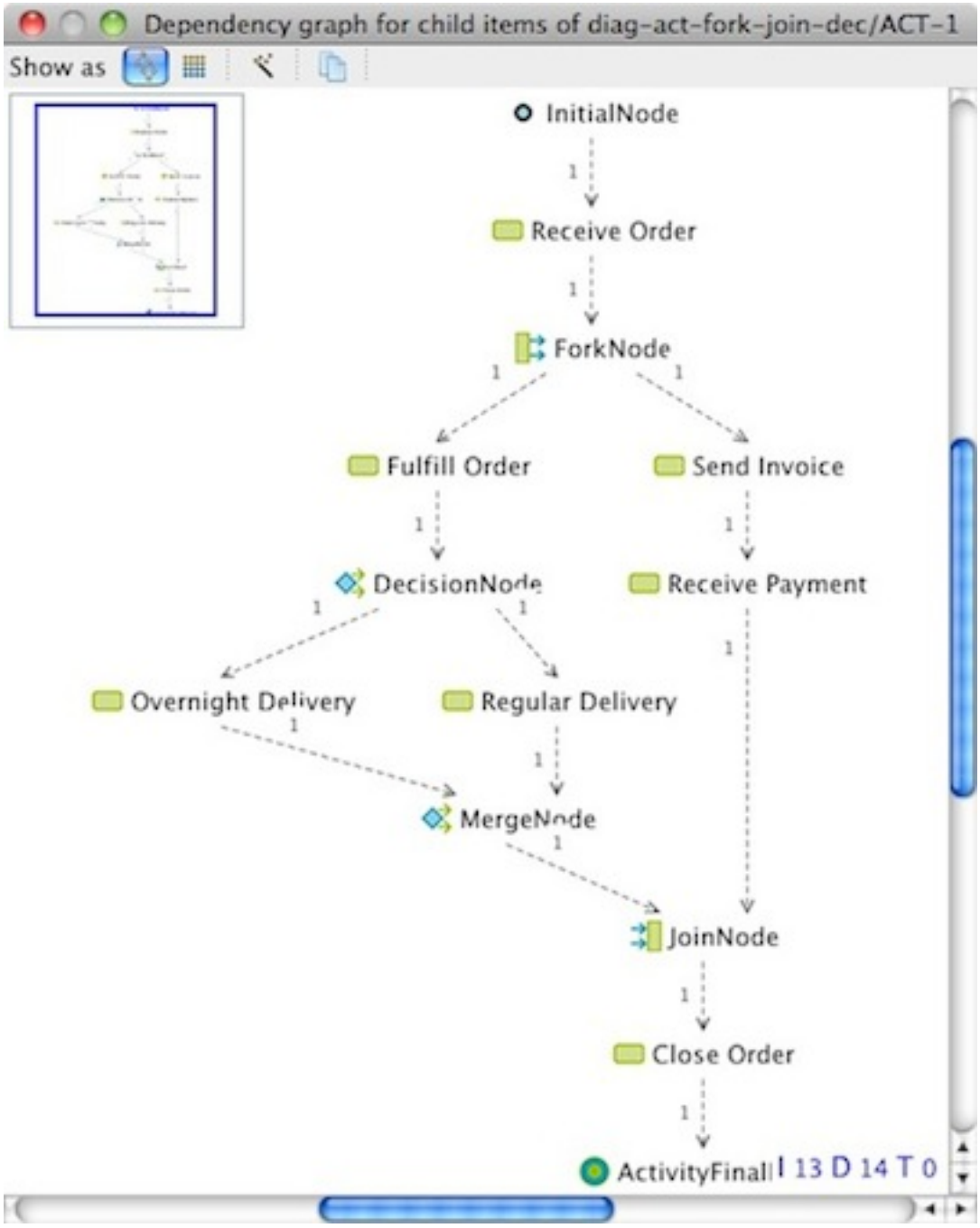
One of the advantages of using (Re)Structure101g for exploring a UML model is to get a overview of the complete model in one single view, which is *not provided* by most of the UML editors.

The other use is to refactor your model to make it better so that the changes can be imported back into the model, but this feature is not currently available for UML/XMI input. We will include it in a future release of the UML/XMI flavor.

5.1. Activity Diagram

Certain points to notice are:

1. The mode of the ExpansionRegion is mapped to the signature. Later this could be mapped to a module type for each mode if it makes sense.

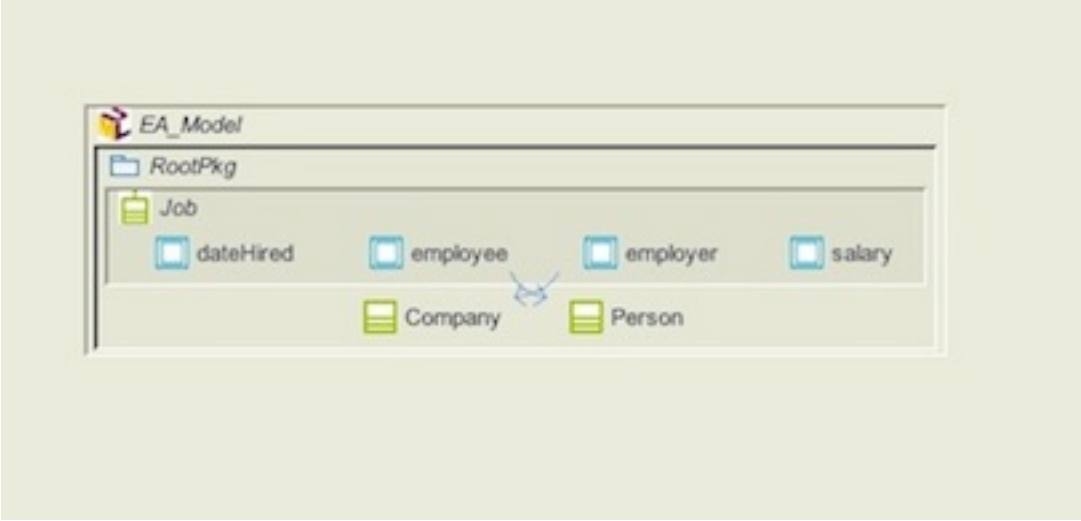


5.2. Class Diagram

Certain points to notice are:

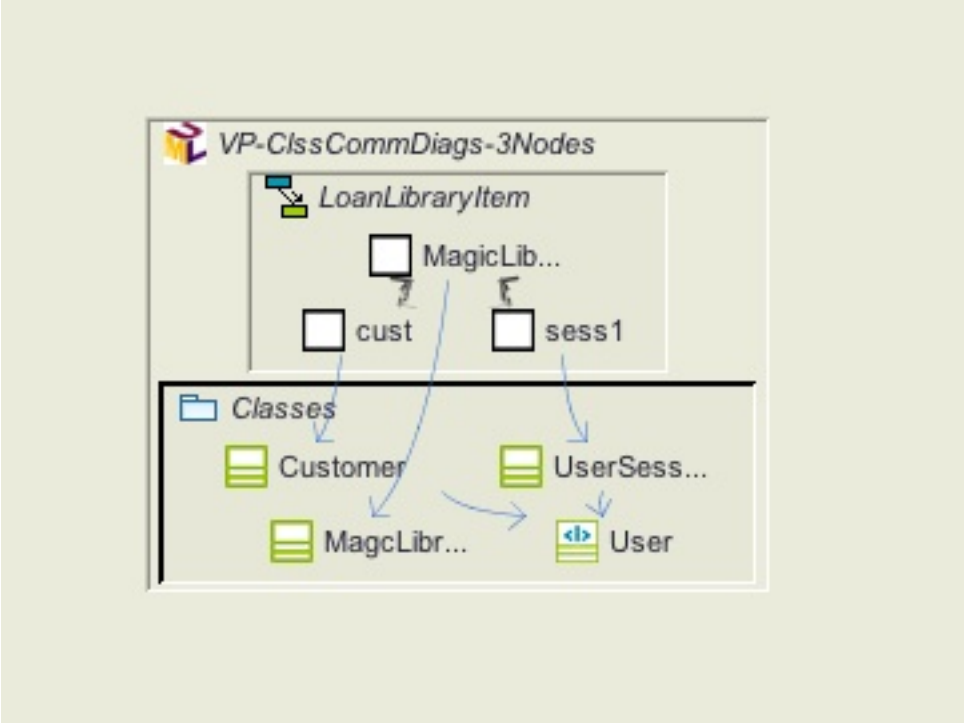
1. Association class is mapped to a separate module with type called class-association.

The following LSM shows two classes with an association class associating them using properties:



5.3. Communication/Collaboration Diagram

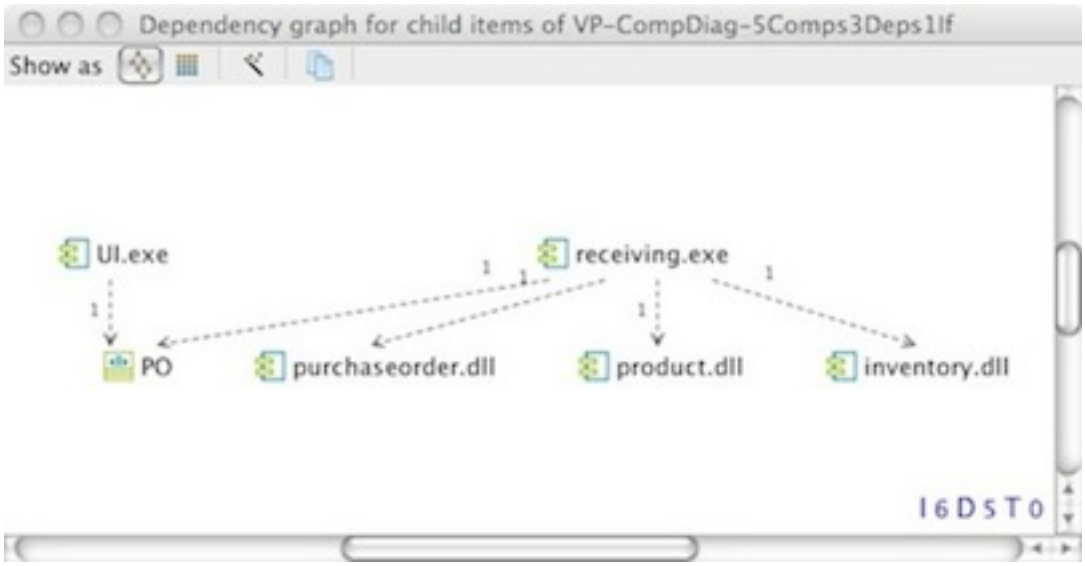
The following diagram shows a communication diagram between three class instances. The class diagram corresponding to the three class instances is also shown. The instances which are of module-type object, are linked to the classes in the class diagram using the is-of-type links.



5.4. Component Diagram

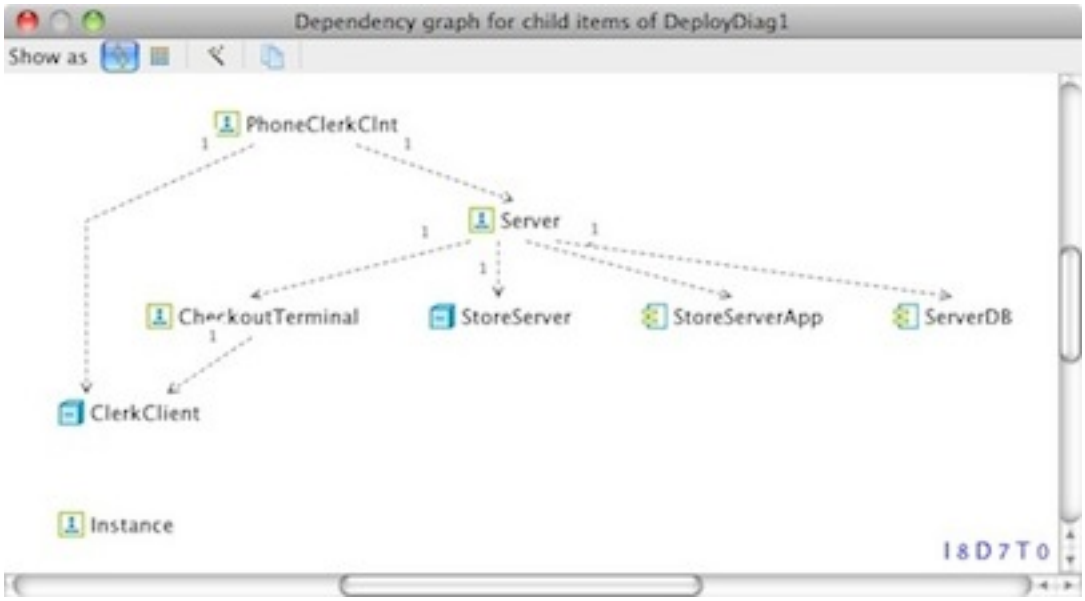
Component diagrams show components with the interfaces they are providing or using from other components. It could also include other artefacts which make up a component.

The following diagrams – as a dependency graph – shows two component with the PO interface being used by the UI.



5.5. Deployment Diagram

An example of a deployment diagram with components within nodes is shown below:

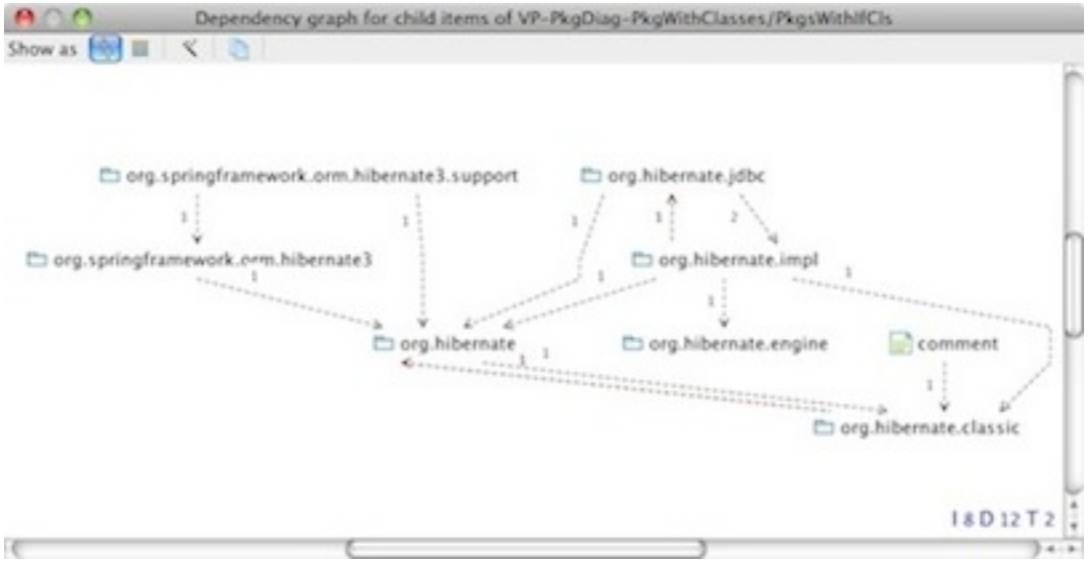


The contained components within nodes use the dependency-type ‘deployed-in’ and NOT node containment.

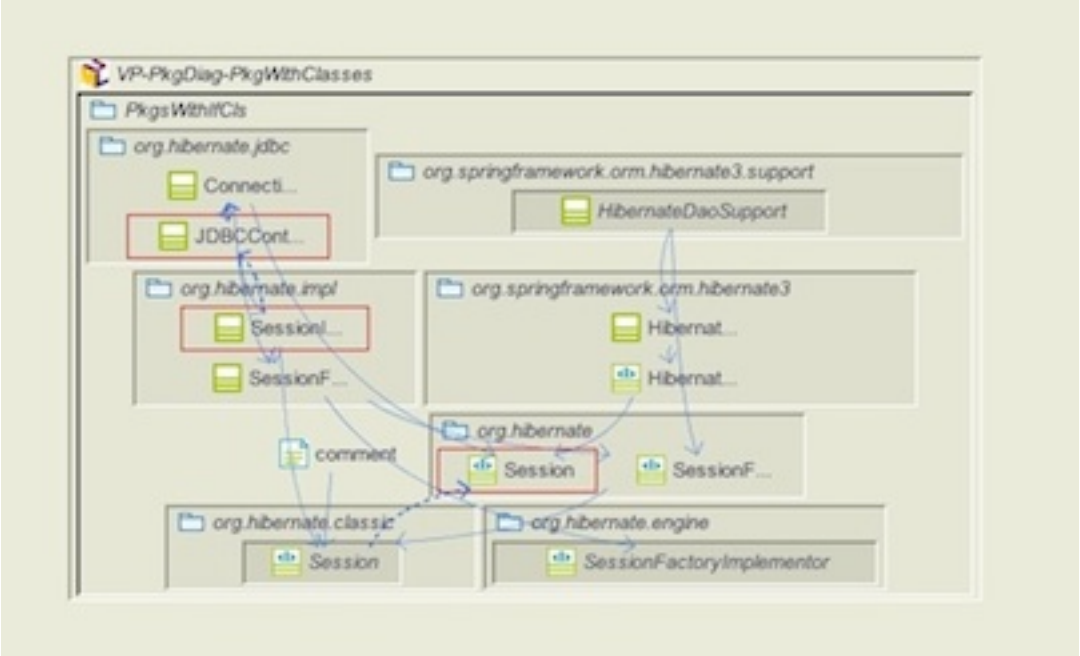
5.6. Package Diagram

Certain points to note are:

1. Package diagram can have interface classes



In the above dependency graph, not all level of nodes are shown, only one level is shown. This problem will be fixed in the core ReStructure101g product. But the architecture diagram included below, shows all the levels of the nested nodes, which includes interface classes of some of the packages.

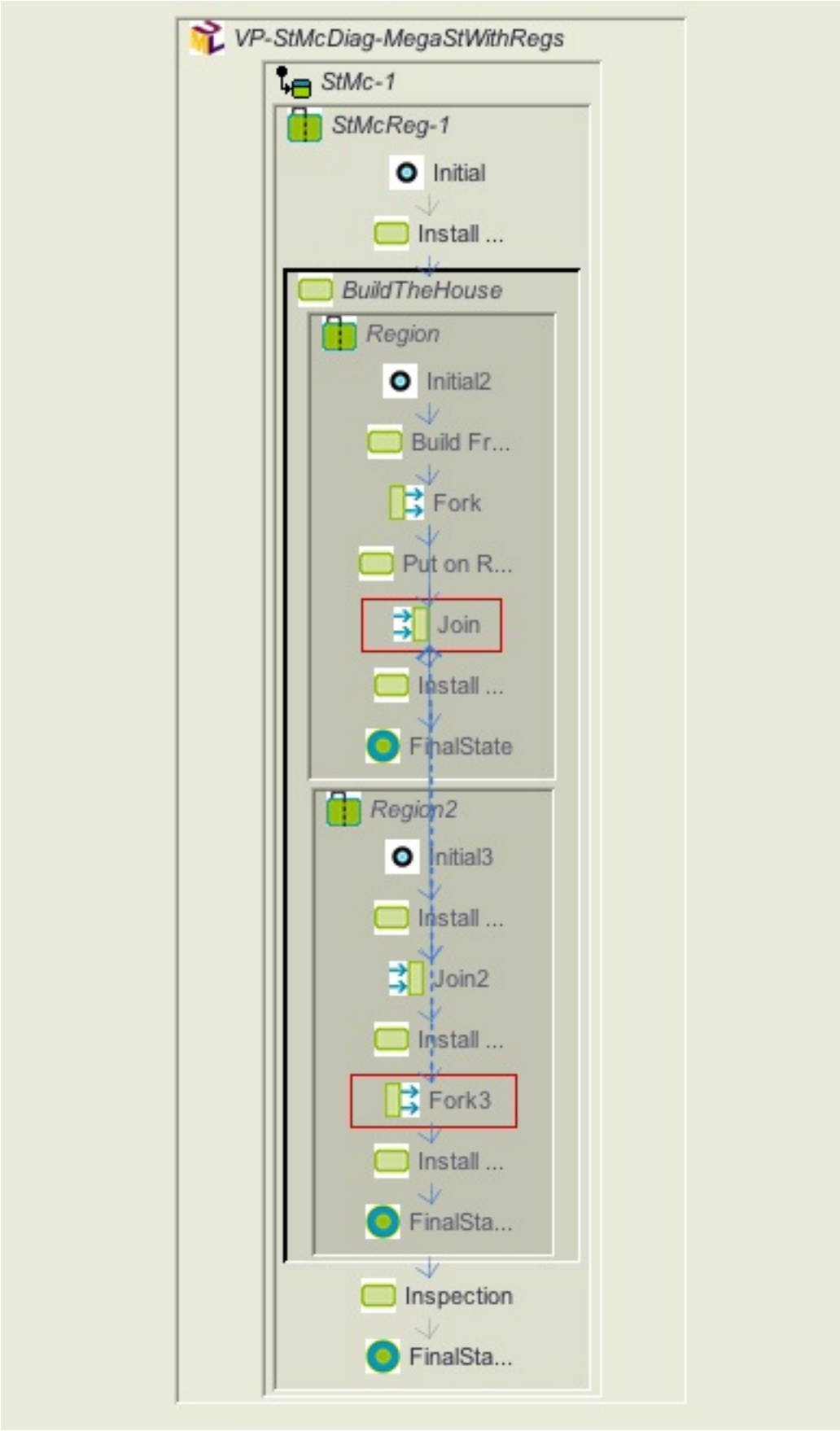


5.7. State-Machine/State-Chart Diagram

Certain points to note are:

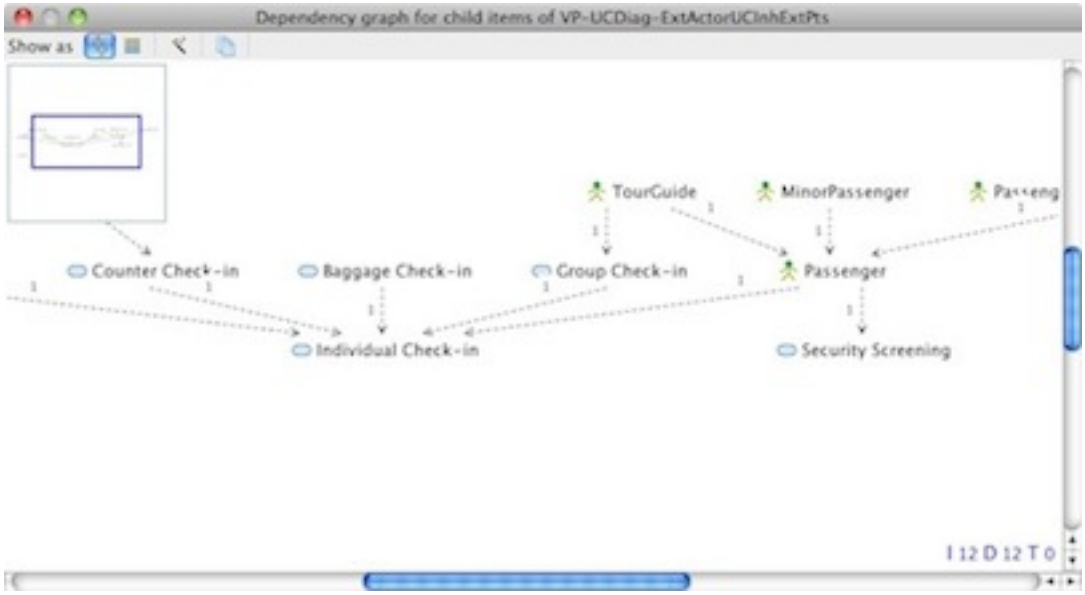
1. If there are entry and exit points to a compound state and that compound state is expanded later, then the entry & exit points are linked to the corresponding entry & exit points in the compound state using the same-as dependency type.

An example of a state-machine with multiple regions, in architecture diagram view is shwn below:


VP-StMcDiag-MegaStWithRegs
StMc-1
StMcReg-1
Initial
Install ...
BuildTheHouse
Region
Initial2
Build Fr...
Fork
Put on R...
Join
Install ...
FinalState
Region2
Initial3
Install ...
Join2
Install ...
Fork3
Install ...
FinalSta...
Inspection
FinalSta...

5.8. Usecase Diagram

An example usecase diagram with inheritance between actors and other dependency types between usecases, in dependency graph layout is shown below:



Chapter 6. Known Issues / Enhancements

1. Ticket #83: Implement processing of UML 1.x/2.x Object Diagram
2. Ticket #85: Implement processing of UML 2.x Timing Diagram
3. Ticket #86: Implement processing of UML 2.x Composite Structure Diagram
4. Ticket #90: Implement processing of UML 2.x Interaction Overview Diagram
5. Ticket #92: Implement processing of UML 2.x Sequence Diagram
6. Ticket #219: Handle interruptible activity region in UML Activity Diagram