Structure101g SysML Flavor

Structure101g SysML Flavor

Chapter 1. Description

OMG SysML is an extension of UML that is used mostly in the embedded systems modeling for Model-Driven Software Development (MDSD). Since systems models is one of the important artifacts from which code is generated, it becomes imperative to maintain good design models and hence tools for analyzing and re-architecting such models becomes important. Tejas Software's SysML flavor for Headyway Software's Structure101g and ReStructure101g is one such tool which helps in analyzing & re-architecting SysML models.

The SysML flavor extracts the nodes and dependencies in the SysML Diagrams encoded in XMI format. A listing of the UML stereotypes or concepts used in SysML is available as part of the NIST UML/SysML Class Browser [http://syseng.nist.gov/se-interop/sysml/class-list]. Even though, we do not process all of the UML/SysML Stereotypes and types listed here, it will help in understanding how some of these elements are mapped to the module, sub-module and dependency types that are defined.

As an extension of UML, OMG SysML has re-used some diagrams from UML, modified certain diagrams and introduced a few new diagrams. The following table lists the SysML diagrams, a short description and the category each diagram belongs to.

It currently supports the following SysML Diagrams in the input model:

Diagram Type	**Description**	**Category**	**Supported Or Not**
Package Diagram (PkgD)	Grouping of blocks & other entities into packages.	**Reused** from UML2	Supported
Sequence Diagram (SD)	Grouping of blocks & other entities into packages.	**Reused** from UML2	Supported (See Note [1])
State-machine Diagram (StMcD)	Model the state transitions of a block instance.	**Reused** from UML2	Supported
Usecase Diagram (UCD)	Capture the usecase scenarios of the system with actors.	**Reused** from UML2	Supported
Activity Diagram (ActD)	Captures the internal logic of how an activity is executed.	**Modified** from UML2	Supported
Block Definition Diagram (BDD)	Definitions of blocks (which are derived from Classes) in the design	**Modified** from UML2 Class Diagram	Supported
Internal Block Diagram	Design of how the internals of a block are structured. It is usually instantiated from the corresponding BDD.	**Modified** from UML2 Composite Structure Diagram (CSD)	Supported
Parametric Diagram (ParD)	It is used to support system analysis (perf., reliability,etc.)	**New**	Not supported

	by defining constraint blocks.		
Requirements Diagram (ReqD)	It is used to specify discrete requirements of the system & their relationships.	**New**	Supported

Note [1] **:** Only the communication or interaction diagram variant is supported. But numbering is used to show the nesting of the operation calls.

This SysML 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 SysML 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 SysML model inputs.

This flavor is dependent on the following libraries:

1. Apache Xalan-J for input XMI transformation.

2. Apache 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 SysML 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 SysML*

1. Install the SysML 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 SysML 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 SysML flavor since you just need the SysML models exported from your SysML 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 SysML Model in XMI v2.1*

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

From the SysML tool that you are using, **export the complete SysML 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 SysML Tool

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

Step 2: *Creating a New Project for SysML Flavor*

Create a new project for SysML 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 SysML flavor.

1. The input folder containing the SysML XMI files. One can either select a directory or only one XMI file.
2. Tick the check-box if the is-of-type links are to be generated for instances. If not selected, the type of the instance is encoded as part of the name as <inst. name>:<data type>.

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 SysML Editors

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

UML Version	Namespace URL
OMG SysML v1.0	http://schema.omg.org/spec/UML/2.0
OMG SysML v1.1	http://schema.omg.org/spec/UML/2.1
Sparx SysML v1.1 profile	http://www.sparxsystems.com/profiles/SysML1.1/1.1
OMG SysML v1.2	http://schema.omg.org/spec/UML/2.1.1

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. The interoperability demo was held in Dec 2009 [http://www.omg.org/news/meetings/tc/ca/special-events/MIWG.htm] and the following vendors were testing the interoperability among their tools:

1. Adaptive
2. Artisan
3. EmbeddedPlus
4. IBM
5. NoMagic
6. Sodius
7. SOFTEAM and
8. SPARX Systems
9. US National Institute of Standards and Technology (NIST) is providing an XMI validation tool that forms a key part of this process.

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 MagicDraw SysML Plugin	16.9
Altova UModel	2011
IBM Rational Rhapsody	8.0
OMG MIWG Test Suite	Release 11

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

This section describes the nodes and dependency types extracted from SysML 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 SysML, 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
sysml-model		Represents SysML Model, root node of the structure.
diag-pkg		Represents a package diagram
diag-seq		Sequence diagram
diag-stmc		State-machine diagram
diag-uc		Usecase Diagram
diag-act		Activity diagram
diag-blkdef		Block definition diagram
diag-intblk		Internal Block diagram
diag-param		Parametric diagram
diag-rqmts		Requirements diagram
actor		Actor in an usecase diagram
block		Block in a BDD
block-inst		Instantiable block in a IBD
connector		Connector in IBD.
constraint-block		Constraint block in a parametric diagram
interface		Interface in a class diagram

package	🗀	Package in a class diagram
flow-spec		Flow-spec in data-flow in IBD.
requirement	‹R›	A component in a component diagram
func-rqmt	‹R›	Functional Requirement in a requirement diagram
usability-rqmt	‹R›	Usability Requirement in a requirement diagram
extended-rqmt	‹R›	Extended Requirement in a requirement diagram
interface-rqmt	‹R›	Interface Requirement in a requirement diagram
perf-rqmt	‹R›	Performance Requirement in a requirement diagram
physical-rqmt	‹R›	Physical Requirement in a requirement diagram
usecase	⬭	Usecase in an usecase diagram
testcase	‹T›	Testcase in an usecase diagram
dsgn-constraint	‹R›	Design constraint in Parametric diagram
signal	‹S›	Signal in block definition diagram
view	🗀	in diagram
view-point	▤	View point in diagram
value-type	‹U›	Value type in diagram
state-start	●	Start state in a state-machine
state-end	◉	End State in a state-machine
state	▭	Any other intermediate state in a state-machine
stmc-region	▨	State machine region, used as nested node
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. _It is not currently implemented, see Issue #230 and #262.
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
region-expansion		Expansion region in an activity diagram
expansion-node		Expansion node in an activity diagram
reception		Reception of signal (action-snd-signal or action-acc-evt) in internal block diagram. Instance of a signal.
flow-end	⊗	Flow end node in activity diagram.
note		Comment for a node.

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
port	■	Port which is part of a block-inst in IBD.
flow-port	■	Flow port thru' which a flow flows.
block-prop		Property for a block or block-inst.

flow-prop	▭	Property for a flow.
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. Some of the other specific types are mapped to a generic type, since there is a limitation on the number of dependency types, this mapped generic types is also indicated.

Dependency Type	**Description**
associated-with	Block is associated with a another class. Converted from uml:Association.
depends-on	Entity depends on another entity, converted from uml:Dependency.
aggregate-of	Block is an aggregation of set of another block.
composed-of	Block is composed of a set of another block.
inherits-from	Block inherits from another block.
imports	Imports entities from another package.
allocates	A requirement is allocated to a design element such as a block, etc.
derived-from	Requirement is derived from another requirement. *This is mapped to generic type refines.*
refines	Requirement refines another requirement.
satisfies	Test case satisfies a requirement. *This is mapped to generic type traces.*
traces	A requirement traces to one or more design element.
verifies	A test case verifies a requirement. *This is mapped to generic type traces.*
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.
realizes	Component realizes an interface.
includes	Usecase includes another Usecase.
extends	Usecase extends another usecase via an extension point.
uses	Component uses an interface
source	Source block of the association block.
target	Target block of the association block .
carries	A flow carries information elements into another port of block.
note-link	Link from a note to the annotated element, which can be only another node, currently.
same-as	Node is same as another, used only for entry & exit nodes to a compound state in State-Machine Diagram.

Chapter 5. SysML Diagrams Supported and their Details

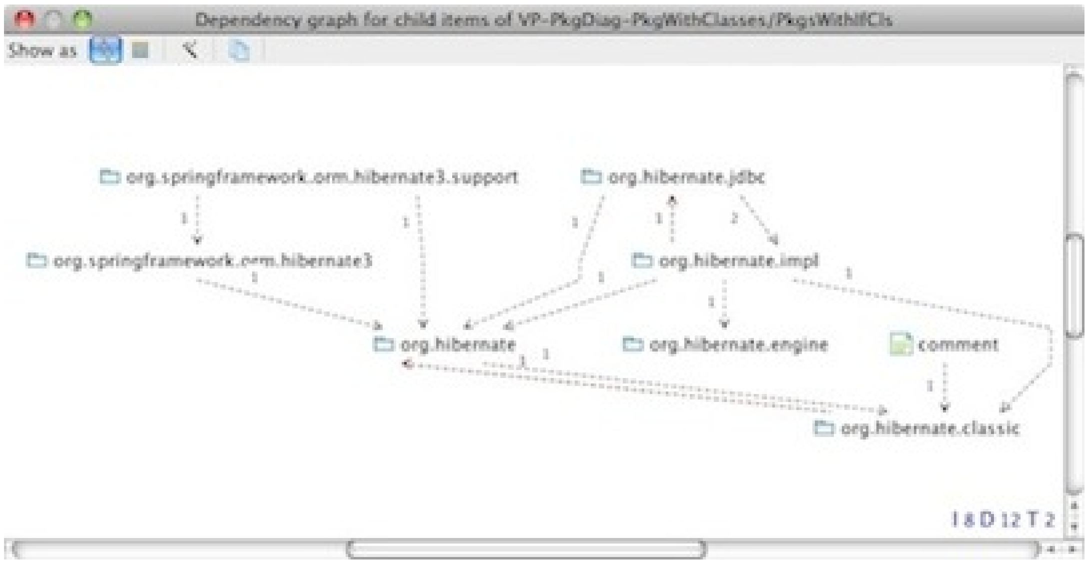
One of the advantages of using (Re)Structure101g for exploring a SysML model is to get a overview of the complete model in one single view, which is *not provided* by most of the SysML editors.

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 SysML input. We will include it in a future release of the SysML flavor.

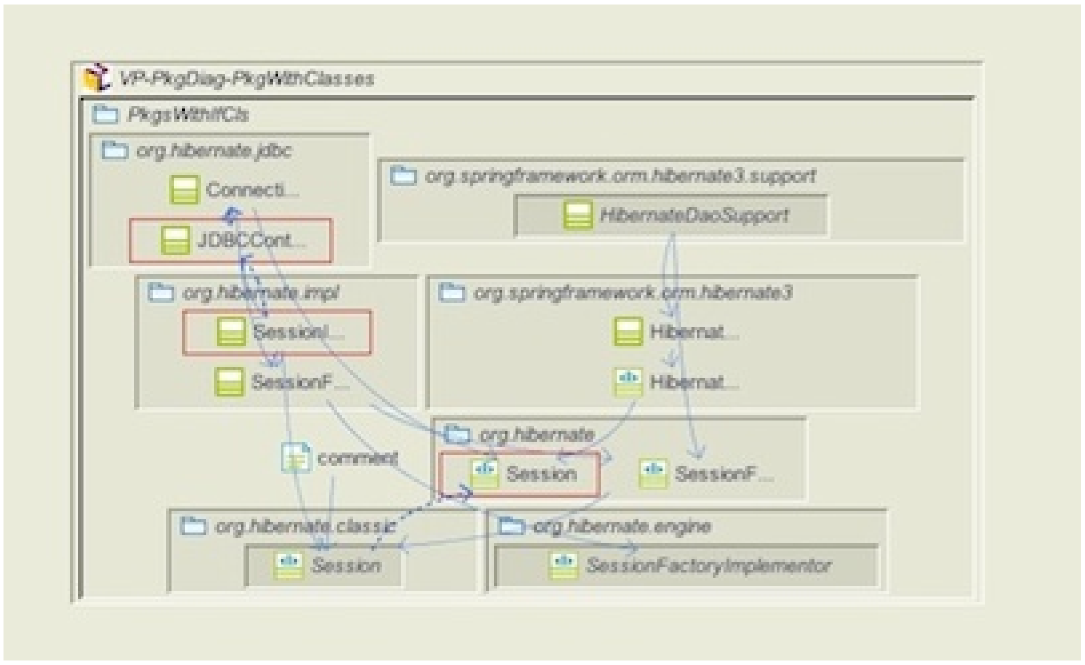
5.1. Package Diagram

Certain points to note are:

1. Package diagram can have interface classes

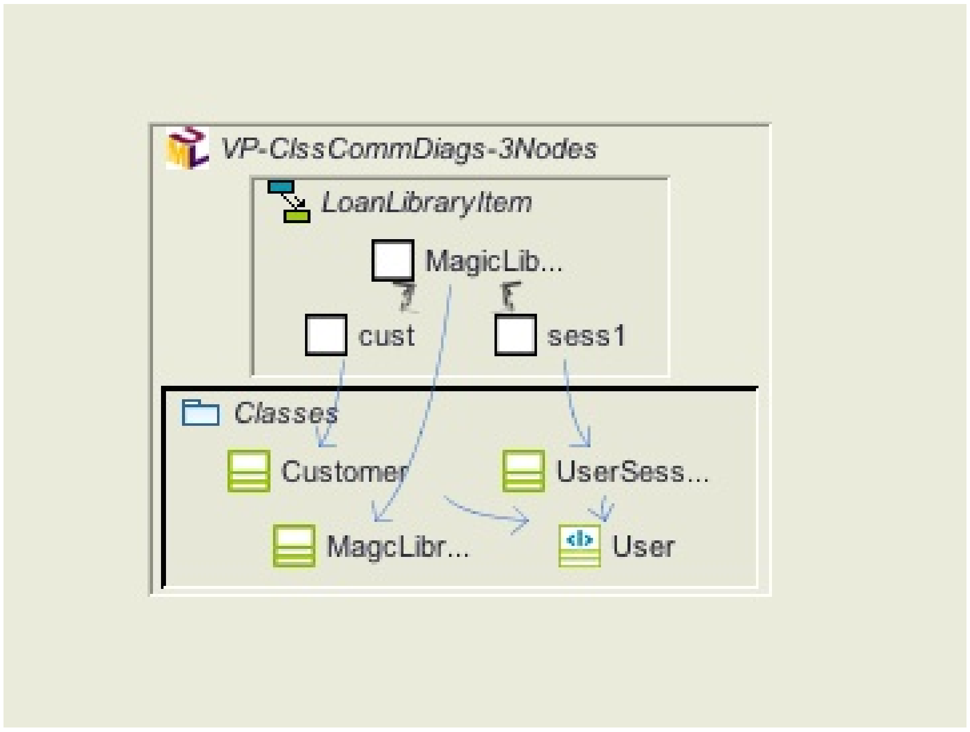

Dependency graph for child items of VP-PkgDiag-PkgWithClasses/PkgsWithIfCls
Show as
org.springframework.orm.hibernate3.support
org.hibernate.jdbc
org.springframework.orm.hibernate3
org.hibernate.impl
org.hibernate
org.hibernate.engine
comment
org.hibernate.classic
I 8 D 12 T 2

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.



5.2. Sequence Diagram

Even though the flavor supports sequence diagram, it is encoded in the same way as a communication diagram.

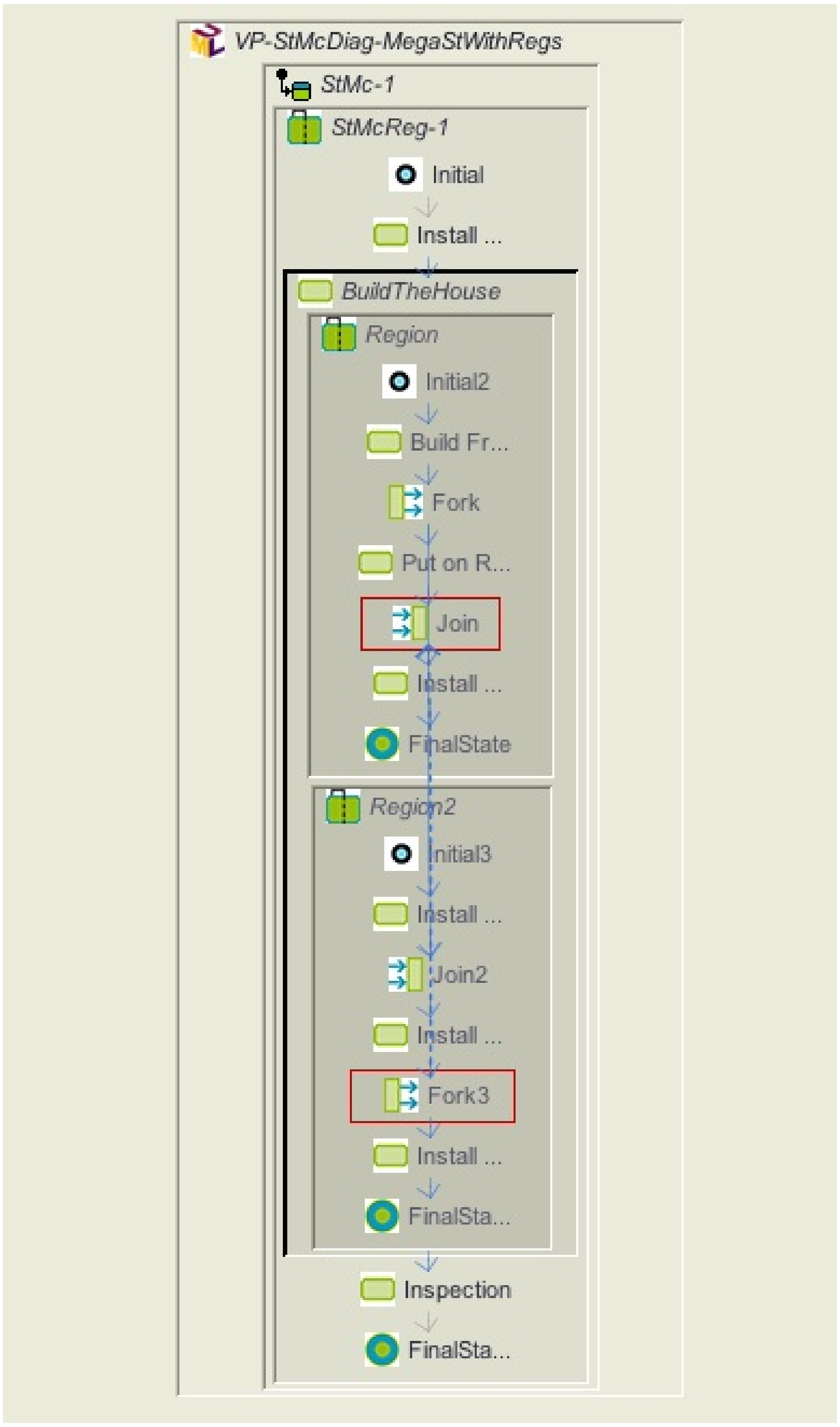


5.3. State-Machine/State-Chart Diagram

Certain points to note are:

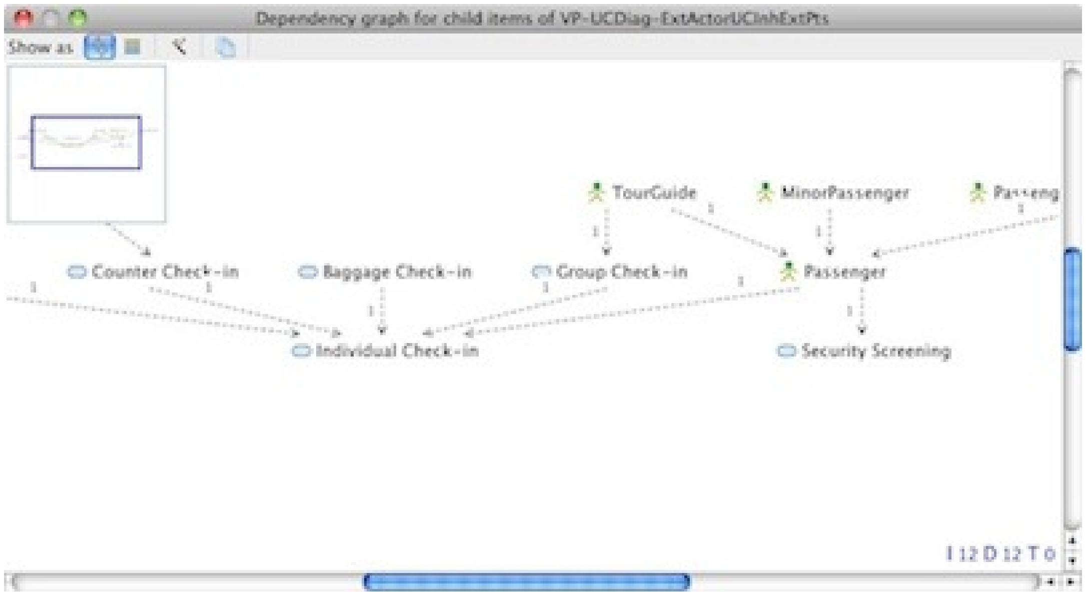
1. If there are entry and exit points to a compound state and that compound state is expanded later, then the entry & exit points are linked to the corresponding entry & exit points in the compound state using the same-as dependency type.

An example of a state-machine with multiple regions, in architecture diagram view is shown 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...

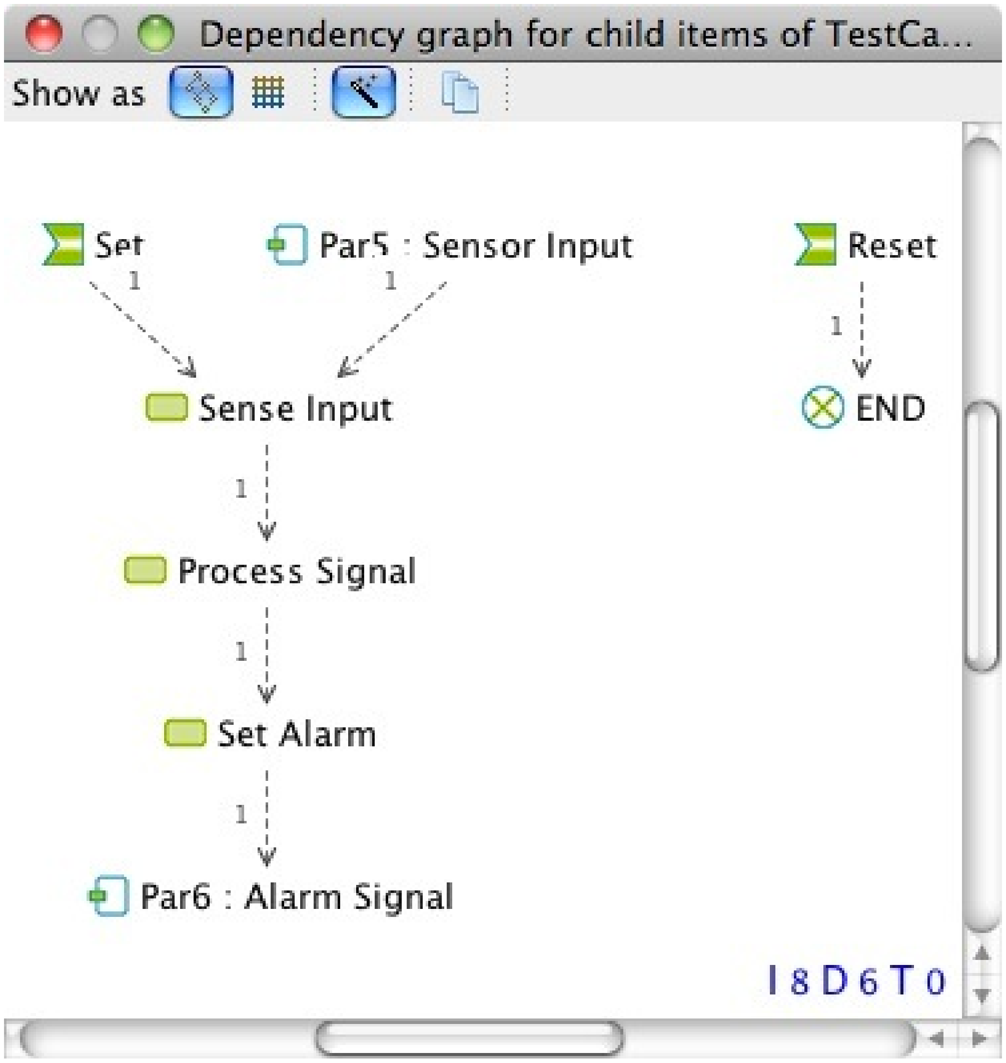
5.4. Use Case Diagram

An example use case diagram with inheritance between actors and other dependency types between use cases, in dependency graph layout is shown below:



5.5. Activity Diagram

The following screen shot shows the activity diagram in OMG MIWG Test Case 12 [http://www.omgwiki.org/model-interchange/doku.php?id=test_case_12]. This is shown as a dependency graph.

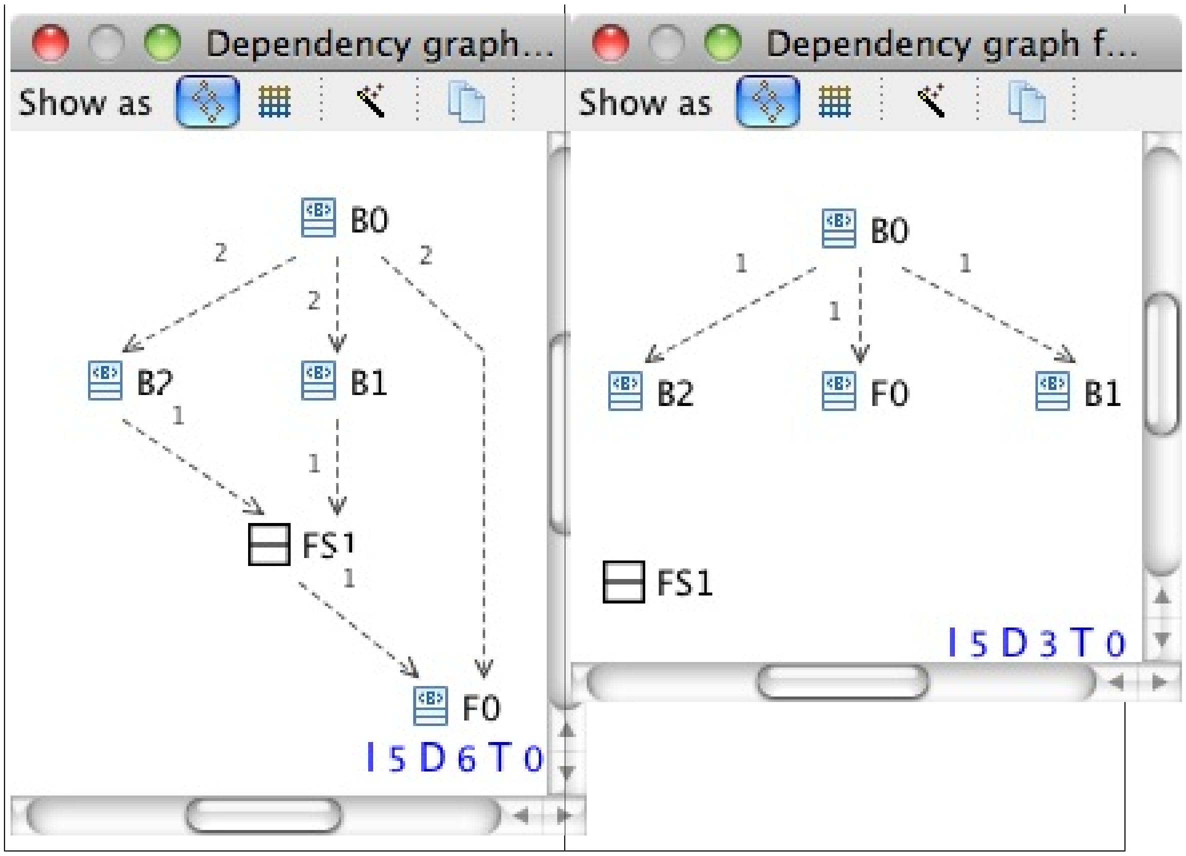


5.6. Block Definition Diagram

Similar to UML class diagram, the SysML Block Definition Diagram shows the blocks and their structural relationships.

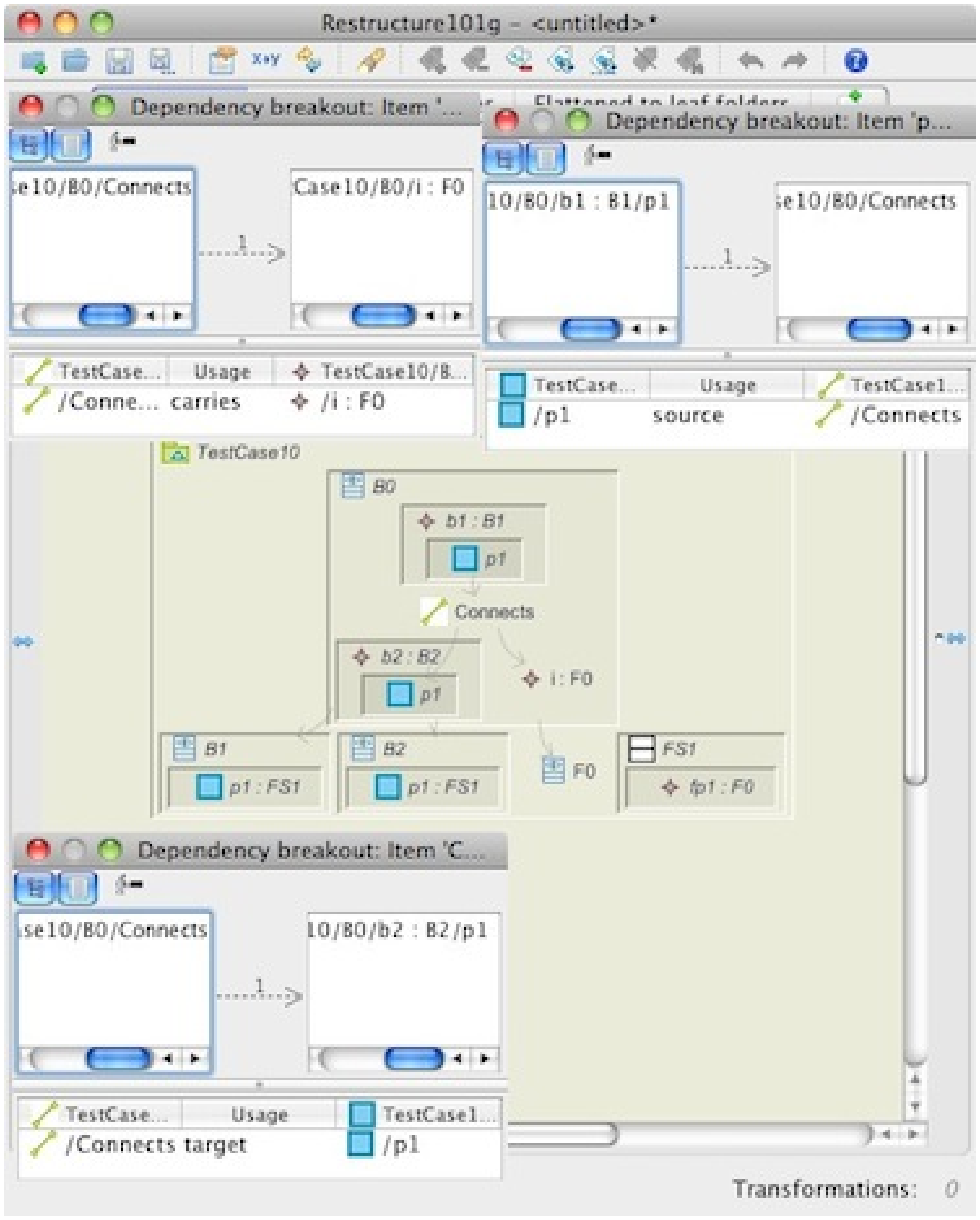
The following screen shot shows the Block definition diagram in OMG MIWG Test Case 10 [http://www.omgwiki.org/model-interchange/doku.php?id=test_case_10]. This is shown as a dependency graph for both the cases: left showing the case when is-of-type edge generation is enabled and the right showing when it is not enabled.

_. **With IOT Enabled**	_. **With IOT Disabled**



5.7. Internal Block Diagram

The following screen-shot shows the block definition diagram with the internal block diagram of Block B0. This corresponds to OMG MIWG Test Case 10 [http://www.omgwiki.org/model-interchange/doku.php?id=test_case_10]



TestCase...	Usage	TestCase10/B...
/Conne...	carries	/i : F0

TestCase...	Usage	TestCase1...
/p1	source	/Connects

TestCase...	Usage	TestCase1...
/Connects	target	/p1

The Levelized Structure Map (LSM) shows the two blocks B1 and B2 which are part of the Block B0 and the ports within each instance of the Blocks b1 and b2 with the connector between the blocks shown as ‘connects’ node which carries instance i of the flow F0.

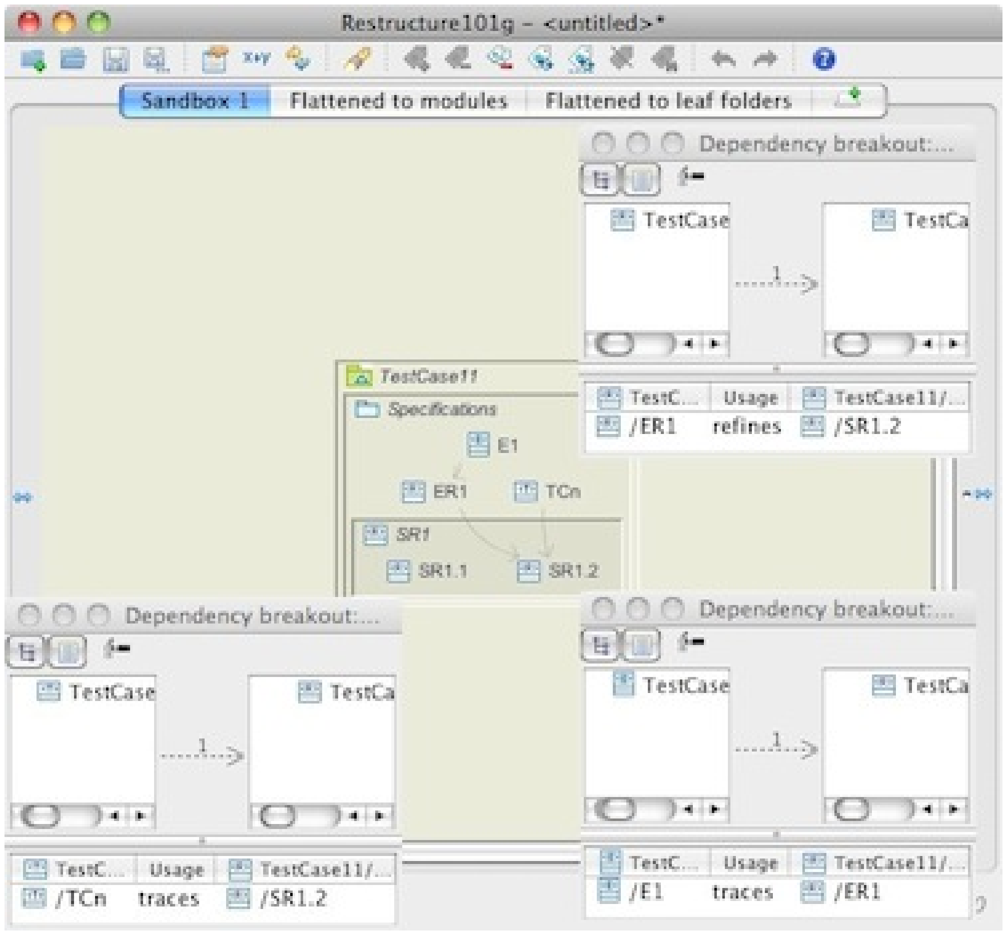
5.8. Parametric Diagram

Parametric diagram is currently *not supported*. It will be supported in the v1.0 release.

5.9. Requirements Diagram

Requirements Diagram depicts the requirements that the system has to implement. A Requirement diagram can also have test cases which verify it and also reference to blocks from block diagram which realize one or more requirements.

The following screen-shot shows the requirements diagram corresponding to OMG MIWG Test Case 11 [http://www.omgwiki.org/model-interchange/doku.php?id=test_case_11]



TestC...	Usage	TestCase11/...
/ER1	refines	/SR1.2

TestC...	Usage	TestCase11/...
/TCn	traces	/SR1.2

TestC...	Usage	TestCase11/...
/E1	traces	/ER1

Chapter 6. Known Issues / Enhancements

1. Ticket #100: Implement processing of SysML v1.2 New Parametric Diagram.
2. Ticket #205: Support for ISO Standard AP233 SysEngg input format.
3. Ticket #230: Handle variations in enc. of contained nodes in swimlanes/regions in ActDiag.
4. Ticket #231: Linking diagrams using element names and making it optional input parameter.
5. Ticket #262: Handle swimlanes in activity diagrams.