

jUCMNav Modules Extension User Manual

Introduction

When building large-scale goal-oriented models using the *i** framework, the problem of scalability arises. One of the most important causes for this problem is the lack of modularity constructs in the language.

jUCMNav is a graphical editor and an analysis and transformation tool for the *User Requirements Notation* (URN). This tool supports the *Goal-oriented Requirement Language* (GRL) and the Modules Extension adds the notion of modularity to its basis.

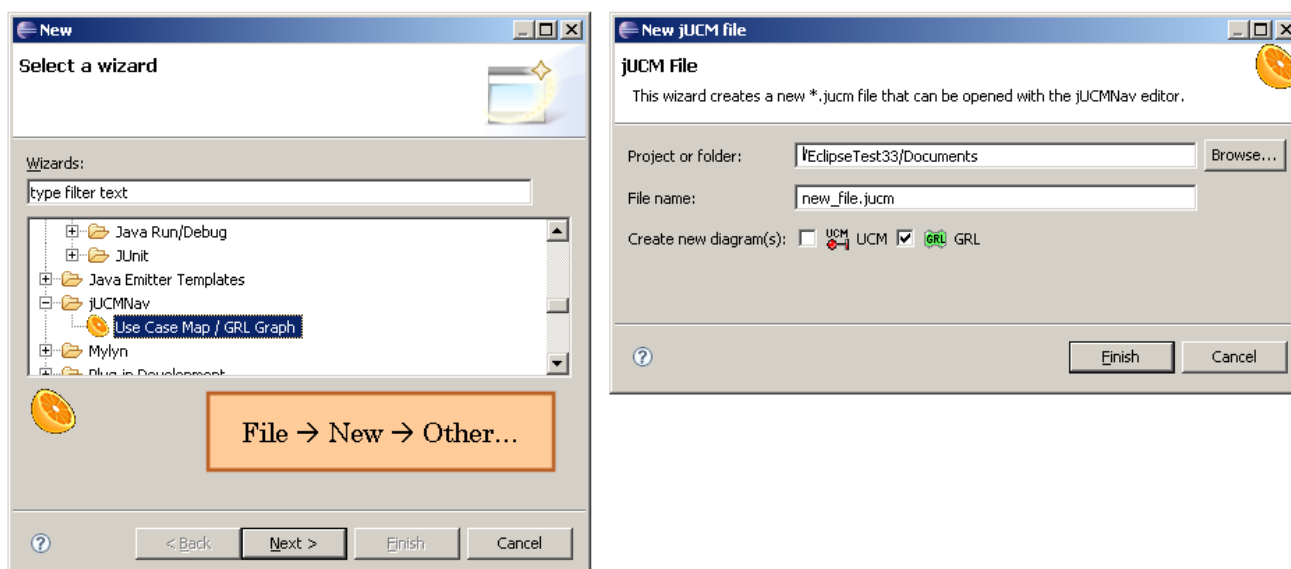
Module Definition

The extension supports SModule and SRModule definition. A UndefinedModule structure has also been included in order to represent the abstract notion of a generic *i**-Module and as a base to be used for Module Application purposes.

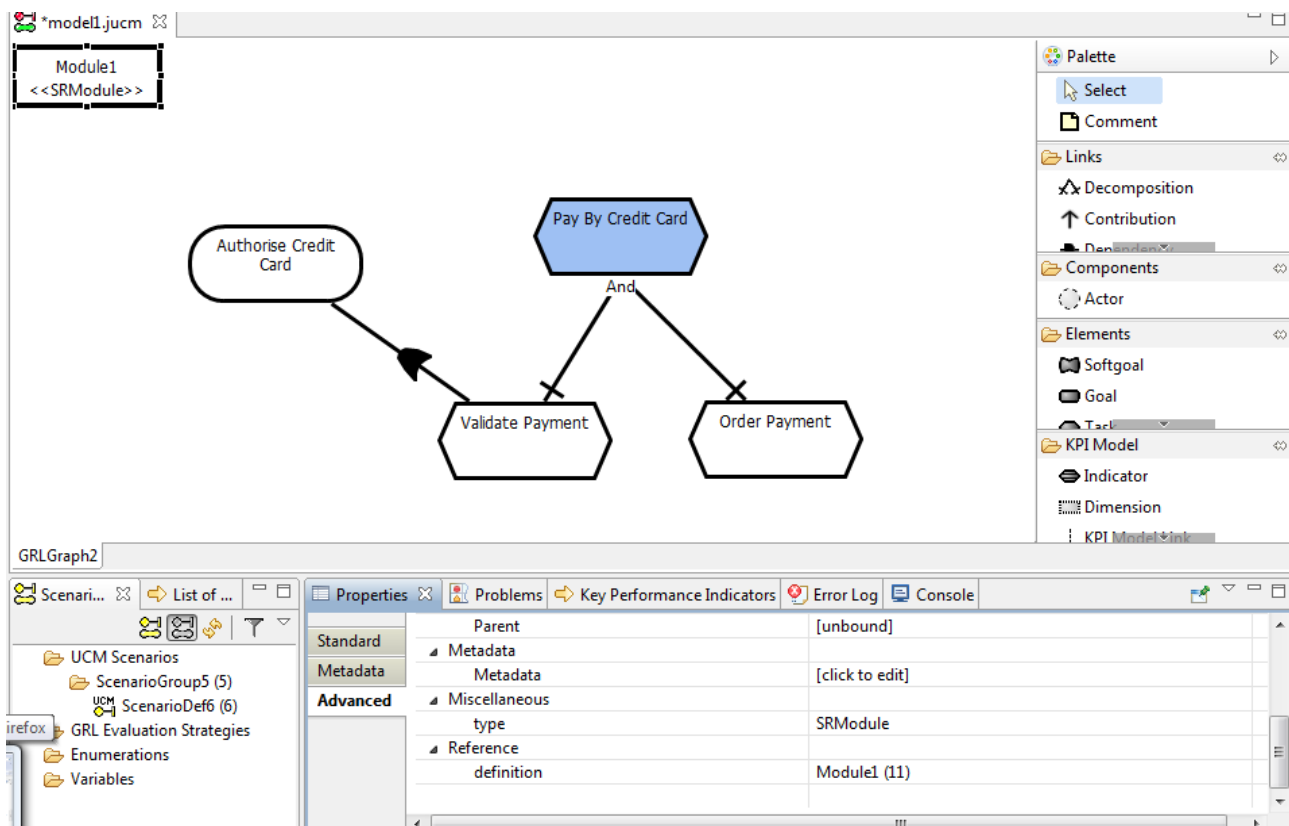
The basic GRL editing and GRL strategies management and application offered by jUCMNav remains the same.

Creating and Opening Modules

You will allways be working with Modules (even if it is with its most abstract notion) so the creation and definition of Modules is transparent to the creation of new GRL graphs.



Once created you will be able to edit and change the Module's nature through its reference label. A click on the Module label will open then its properties view. Through the advanced set of properties you will be able to change the Module's Type, name, etc.



- *UndefinedModules*

When creating a Module its default type will be set to *UndefinedModule*. This type of Module is the most generic of all and it is specialised by SRModules, SDModules, and other User-defined Modules. It has no extra integrity constraints.

- *SRModules*

When working with SRModules you will be able to set its root elements through the correspondent Intentional Element advanced properties view. Root Intentional Elements are represented by blue references (see picture above). You can set an Intentional Element as root through its advanced properties view.

- *SDModules*

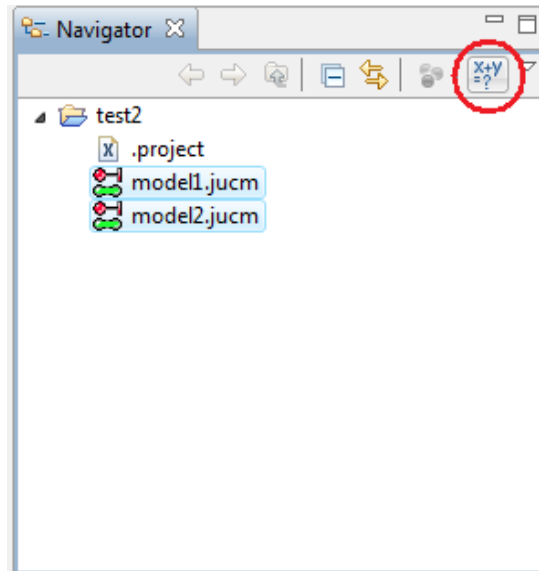
When dealing with this kind of Modules one will be able to work transparently with actors

Checking Module Integrity

I*-Modules have its own integrity constraints. JUCMNav already offers a set of predefined OCL-based constraints that are accessible through the preference menu. In this extension you will also find a set of predefined constraints for each Module type together with jUCMNav's default constraint sets. This catalogue can be easily extended by the user. *See jUCMNav's help content for detailed information.*

Module Operations

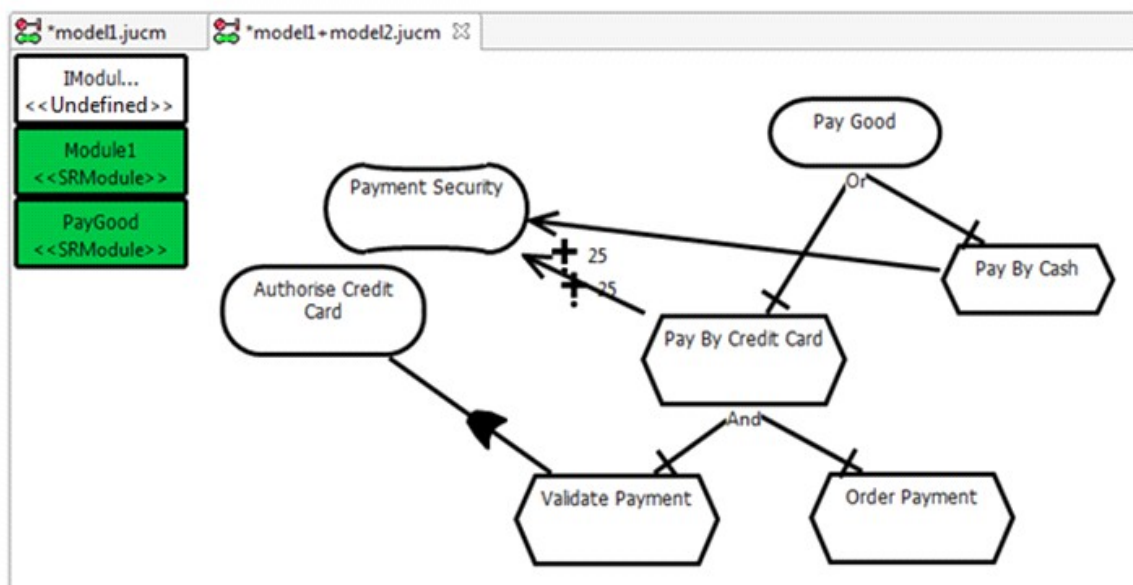
The extension also offers two Module Operations. One can apply the operations through Eclipse's Navigator View.



By using the above encircled button any of the defined operations can be executed. Depending on the selected modules's nature Combination or Application will be applied. Then, no matter what operation is used, a result Module will be created and saved in the same workspace (notice that you can only apply operations to Modules in the same workspace and once you execute the operation all other open modules will be save and closed). The result file will be named after its source Models (note that file and Module name need not be the same).

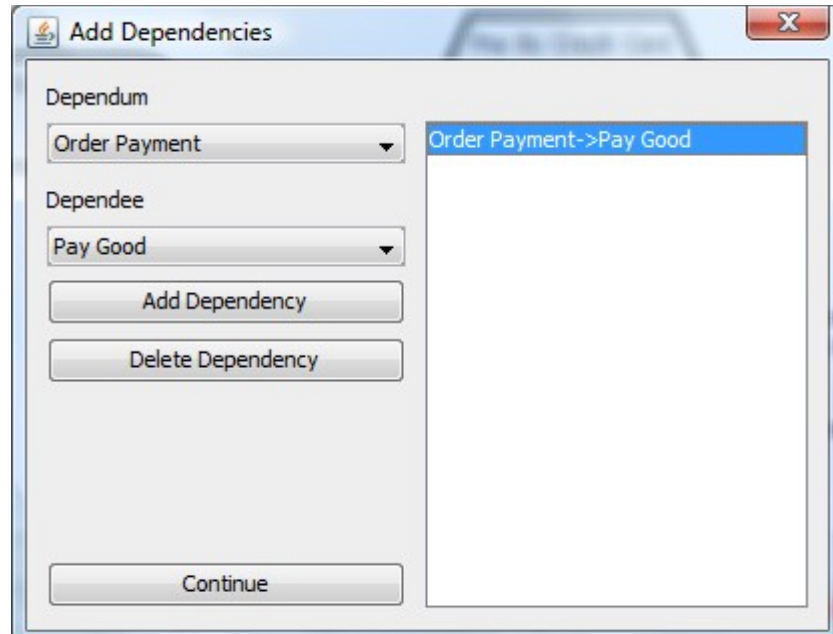
Module Combination

The result of this operation will be a default *UndefinedModule*. Below the default label you will be able to keep track of the used Modules (green labels) used to obtain the result and its types.



Module Application

If one of the input Modules is an *UndefinedModule* Module Application will be assumed. Before applying the operation you will be asked for a dependency match list (which you can leave empty).



Once the dependency match list is introduced a resulting *UndefinedModule* similar to the output of Module Combination will be obtained with the specified dependencies added. Note that the once applied, the result module might be inconsistent. In order to guarantee the consistency you can use the static constraints through the preference's menu. See *jUCMNav's help content* for detailed information.