INTEGRATING AN ECLIPSE-BASED SCENARIO MODELING ENVIRONMENT WITH A REQUIREMENTS MANAGEMENT SYSTEM

Jason Kealey, Yongdae Kim, Daniel Amyot, Gunter Mussbacher
SITE, University of Ottawa
800 King Edward
Ottawa, ON, K1N 6N5
email: jkealey@shade.ca, eric17k@hotmail.com, {damyot | gunterm}@site.uottawa.ca

Abstract

Scenarios are useful for the elicitation, description, and analysis of software requirements. However, they need to be used in cooperation with complementary general requirements, and both views must be linked in a way that supports traceability, navigation, and analysis. Jiang proposed to introduce Use Case Map (UCM) scenario models into the Telelogic DOORS requirements management system and to maintain relationships as both views evolve over time. This paper describes new tool support based on an open platform, the jUCMNav Eclipse plug-in, which was extended to integrate with DOORS. This novel integration provides a more reliable and usable front-end to UCM modeling in collaboration with general requirements management. Additional capabilities for creating links automatically and improving this process are also discussed.

Keywords*: Requirements management; scenarios; Use Case Maps; integration; Eclipse*

1. Introduction

Scenarios have gained in popularity for the elicitation, description, and analysis of system and software requirements. For instance, Use Case Map (UCM) models provide visual representations of use cases and scenarios superimposed on architectural entities, but also allow further evaluation and analysis of requirements [3][10]. However, scenarios cannot specify all types of requirements, and yet they are often expressed separately from other requirements. In order for scenarios to be used in cooperation with complementary general requirements, both views must be linked in a way that supports traceability, navigation, and analysis.

In his thesis [4], Jiang proposed to introduce graphical UCM scenarios into a leading requirements management system (Telelogic DOORS [8]) and to maintain relationships as both views evolve over time. This approach allows one to integrate many scenarios and use cases in an analyzable UCM model, simplifies the linking of requirements artifacts (requirements at several levels, test goals, and scenarios), and provides facilities for completeness and consistency analysis during evolution. Tool support was provided via an import API written in DOORS' extensible scripting language (DXL).

Whereas Jiang's approach is based on an old UCM editor (UCMNav [7], written in C++) that generates DXL scripts, this paper describes tool support based on a new open platform: the jUCMNav Eclipse plug-in [5][6]. We have created a new plug-in that extends jUCMNav to generate DXL scripts that allow, via our API, the importing and synchronization of UCM models in DOORS. Not only does this provide a much more usable, robust, and familiar design environment, this also demonstrates the generality of our API and its independence vis-à-vis the older UCMNav tool and its obsolete UCM metamodel representation, as well as the extensibility of jUCMNav. Along the way, the API was also enhanced to support automatic creation of links by taking into consideration the hierarchical structure of UCMs.

This paper gives an overview of the integration itself in section 2, and its usage is briefly illustrated in section 3. We present our conclusions and future work in section 4.

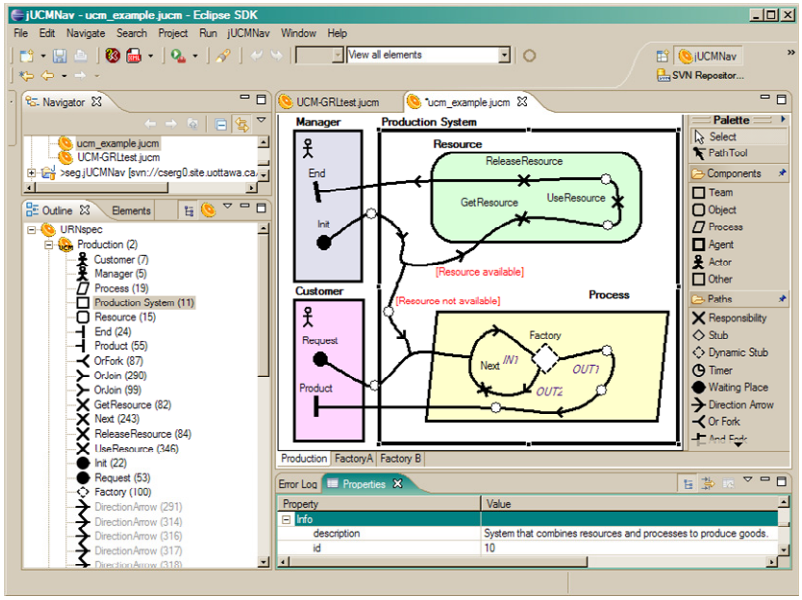
2. Integrating jUCMNav and DOORS

jUCMNav is a plug-in developed with and for the Eclipse framework [2] that supports the creation and maintenance of Use Case Map scenario models, which are composed of elements such as responsibilities, components, start and end points, forks and joins for alternatives and concurrent scenario paths, stubs (shown as diamonds) for model decomposition, etc. jUCMNav also supports goal models in the Goal-oriented Requirements Language (GRL) [9]. Both GRL and UCM are part of the User Requirements Notation (URN) proposed as an ITU-T standard [3].

Figure 1 shows the user interface that includes a tool palette, a property window, an outline window, and a sample UCM model. Multiple files can be edited, and each file contains a URN model and many UCM sub-models.

Following the Eclipse component model, jUCMNav supports extension points that can be used by other plug-ins. One such point is defined as an interface (`URNExport`) that provides access to the URN/UCM model currently edited:

```
public void export (URNspec urn, String Path)
```




```
#include "addins/UCM/lib/UCMUtilities.dxl"

beginImport( "URNspec" )
component( "4", "Manager", "Actor",
   "Role responsible for initiating the production", "" )
component( "6", "Customer", "Actor",
   "Person that request goods to be produced", "" )
…
responsibility( "81", "GetResource",
   "Acquire the resource needed", "0" )
responsibility( "83", "ReleaseResource",
   "Release the resource for future usage", "0" )
…

map( "2", "Production", "Production.bmp", ... )
  respRef( "82", 269,114,"15", "81", "GetResource", "" )
  stub( "100", 294,268,"Factory", "dynamic", "102;" )
  compRef( "5", 17,17,77,168,"no", "4", "Manager", "", "" )
  …
map( "102", "FactoryA", "FactoryA.bmp", … )
  …
map( "313", "Factory B", "Factory B.bmp", … )
  …
endImport
```

Figure 1. jUCMNav, with sample UCM model exported in DXL.

Telelogic DOORS is a requirements management system that supports various types of requirements objects, attribute types, and traceability links. DOORS can handle hierarchical projects and *folders*, and it uses *modules* to store requirements objects and link types. Like most such environment, DOORS allows links between objects and provides capabilities for analyzing and exploiting these links. This tool also offers a scripting language (DXL) which enables other applications to create and manipulate requirements documents.

A DXL API was created to import UCM models into DOORS. The elements imported represent a subset of the entire UCM model. Figure 2 shows a metamodel that describes the structure of the various modules created automatically during the import process, as well as the various elements transformed into DOORS objects. The containment relationships in this metamodel indicate belonging to modules or parent object, whereas the associations correspond to typed links automatically created between objects.

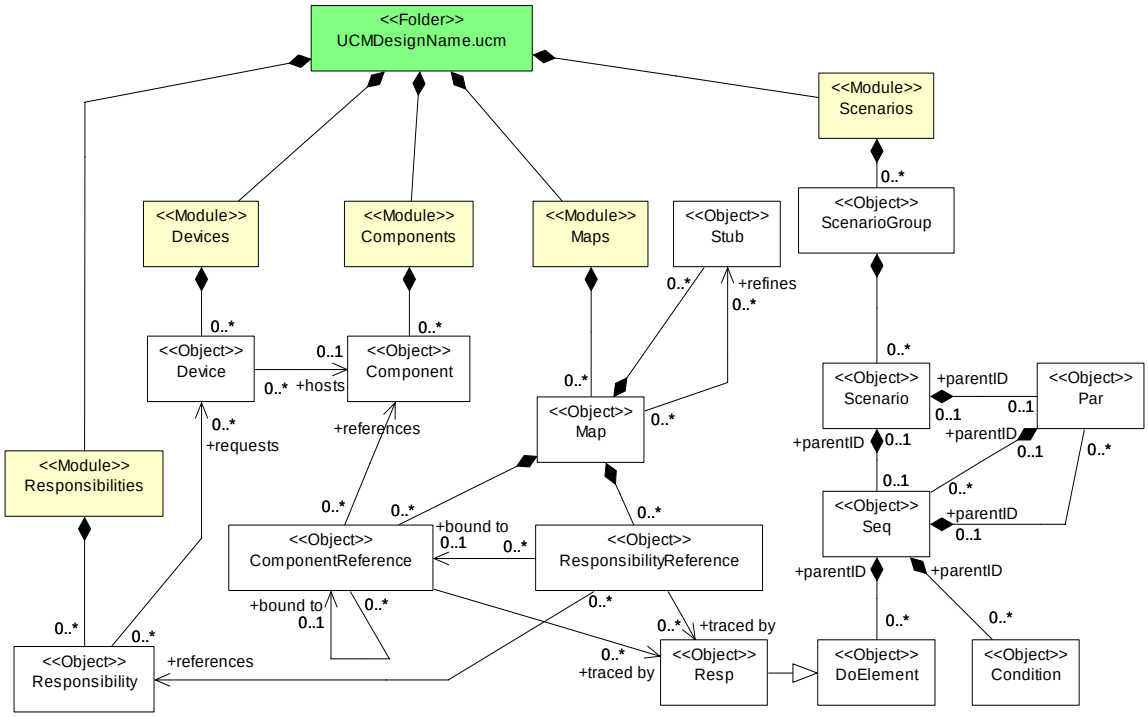


Figure 2. UCM metamodel representation in DOORS.

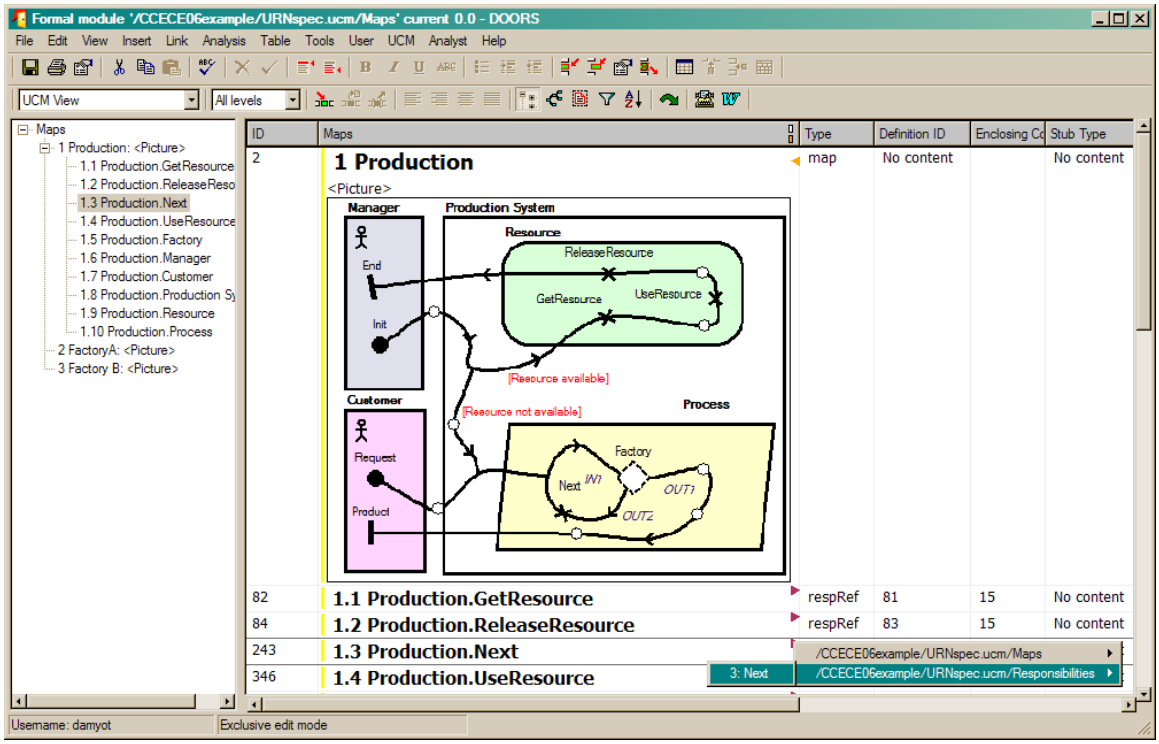


Figure 3. Sample UCM model imported in DOORS (extract).

A new plug-in was added to jUCMNav that makes use of the extension point discussed earlier in order to generate a DXL script describing the URN/UCM model edited. An extract of the script corresponding to our sample UCM model is shown on the right side of Figure 1. According to the DXL API defined in [4], the script describes the list of responsibility and component definitions (with their names, identifiers, and other attributes), and then it lists, for all UCM diagrams in the model, the responsibilities and components referenced by the map as well as the stubs. Each of these items is in fact a procedure call to a library (written in DXL) that we added to DOORS in order to create and update corresponding objects in the DOORS database. jUCMNav also exports bitmap files for each map, which are referenced in the DXL script.

On the DOORS side, the running DXL script will create the DOORS modules for the responsibilities, components, maps, and various types of links according to the schema in Figure 2. In Figure 3, we see the Maps module corresponding to the model in Figure 1. The graphical map is present, together with the objects (and their attributes) corresponding to the UCM elements imported. The triangle symbols in DOORS indicate the presence of incoming (◀) and outgoing (▶) links. As shown in Figure 3, these links are automatically generated and can be accessed via pop-up menus, sorted by link types.

3. Using the Integration

As illustrated in the previous section, when a UCM modeler has finished editing a UCM model, jUCMNAV exports the model as a DXL script which can be loaded and run within DOORS. Afterward, the requirements engineer can easily link external requirements from other modules to the UCM elements imported, or vice-versa, using the facilities that DOORS provides. These traceability links can be exploited for impact analysis, coverage and consistency analysis, and other such types of assessments. Two challenges however have to be addressed: how to minimize the number of links to be created manually from/to external requirements, and how to keep the UCM and DOORS views synchronized as they evolve.

3.1. Link Auto-Completion

We added to Jiang's DXL library a *link auto-completion* mechanism to minimize the possibly large number of links that have to be created manually by the DOORS user between external and UCM requirements. Automatically created links allow the DOORS user to become aware of and exploit links directly that would otherwise need to be discovered transitively via (potentially many) intermediate links.

The architecture of internal links between UCM elements and external links between UCM elements and external requirements assumes an overall bottom-up approach for links between DOORS modules. In other words, the direction of links flows from the lowest-level document (e.g. system requirements) to the top-level document (e.g. user requirements). Furthermore, our linking approach effectively hides all internal links from the requirements engineer. The UCM model appears as one link unit, even though it consists of several modules with many links. The auto-completion mechanism derives external links automatically from internal links and manually entered links. Figure 4 illustrates these types of links in a situation where the UCM model is found between user and system requirements.

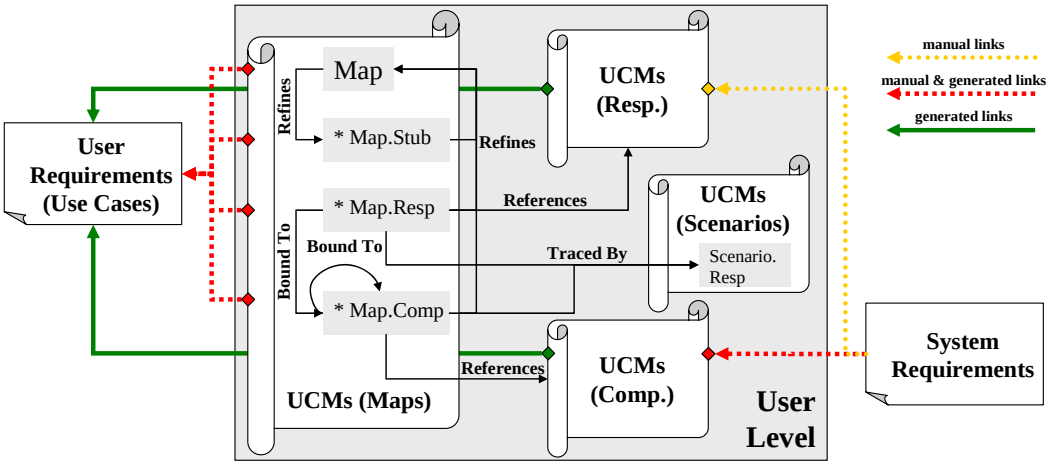


Figure 4. Illustration of the link auto-completion mechanism.

3.2. Evolution Management

Changes and modifications can be done to the UCM model in jUCMNav or to the external requirements linked to the UCM objects imported from jUCMNav.

In case the UCM model is modified, it simply needs to be re-exported by jUCMNav and re-imported by DOORS. The DXL library uses the identifiers of the UCM elements to compare what is currently in the DOORS database and what is in the new UCM model. Additions, deletions, and selected attribute modifications are automatically flagged. If modified objects are linked to external modules, these links are also flagged by DOORS as *suspect*, hence requiring further attention. Multiple *custom views* are provided to emphasize particular aspects of the modules updated (e.g., deleted, new, and redundant UCM objects) and to help the requirements engineer make rapid and informed decisions about the impact of each modification.

In case of a change to an external module, a report is generated detailing the impact of the changes to the UCM model. This report, available as a custom view in DOORS and exportable in text or HTML, is used to update the UCM model which eventually will be re-imported into DOORS.

In both cases, the link auto-completion mechanism is used to finalize the update process.

4. Conclusions

We have presented an extension of the new jUCMNav modeling environment that allows UCM scenario models to be imported in DOORS and linked to other types of requirements. This tool is more usable and robust than the older UCMNav, which was used by Jiang to produce DXL scripts. jUCMNav is also being extended to support GRL models, which opens new opportunities for linking complete URN models with external requirements and enabling new types of analyses. The link auto-completion mechanism introduced here is also available to the UCMNav-based solution. In the near future, jUCMNav will be extended to support *scenario definitions* [1], which enable individual scenarios to be extracted from complex UCM models and imported in DOORS (as supported by UCMNav).

Acknowledgements

This research was supported by NSERC, through its Canada Graduate Scholarship and Discovery programs. We are grateful to B. Jiang, J. Sincennes and Telelogic for their technical support.

References

[1] D. Amyot, D.Y. Cho, X. He, and Y. He, “Generating Scenarios from Use Case Map Specifications”, *Third International Conference on Quality Software (QSIC'03)*, Dallas, É-U, November 2003, 108-115.

[2] Eclipse Project, http://www.eclipse.org/

[3] ITU-T, *Recommendation Z.150, User Requirements Notation (URN) – Language Requirements and Framework*, Geneva, Switzerland, 2003.

[4] B. Jiang, *Combining UCM Scenarios with a Requirements Management System*. MCS thesis, SITE, University of Ottawa, Canada, 2005.

[5] jUCMNav Project (tool, documentation, and meta-model) http://jucmnav.softwareengineering.ca/twiki/bin/view/ProjetSEG/WebHome

[6] J. Kealey, E. Tremblay, J.-P. Daigle, J. McManus, O. Clift-Noël, and D. Amyot, “jUCMNav: une nouvelle plateforme ouverte pour l'édition et l'analyse de modèles UCM”, *Nouvelles TEchnnologies de la RÉpartition (NOTERE’05)*, Gatineau, Canada, August 2005, 215-222.

[7] A. Miga, *Application of Use Case Maps to System Design with Tool Support*, M.Eng. thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa. October 1998. http://www.UseCaseMaps.org/tools/ucmnav/

[8] Telelogic AB: *DOORS/ERS*. http://www.telelogic.com/products/doorsers/

[9] URN Focus Group, *Draft Rec. Z.151 – Goal-oriented Requirement Language (GRL)*, Geneva, Switzerland, Sept. 2003.

[10] URN Focus Group, *Draft Rec. Z.152 – Use Case Map Notation (UCM)*, Geneva, Switzerland, Sept. 2003.