Towards Integrated Tool Support for the User Requirements Notation

Jean-François Roy, Jason Kealey, and Daniel Amyot

SITE, University of Ottawa, Canada
{jroy, jkeal036, damyot}@site.uottawa.ca

Abstract. The User Requirements Notation (URN) combines the Goal-oriented Requirement Language (GRL) with the Use Case Map (UCM) scenario notation. Although tools exist in isolation for both views, they are currently not meant to work together, hence preventing one to exploit URN to its fullest extent. This paper presents *jUCMNav*, a new Eclipse-based tool that supports both UCM and GRL in an integrated way. jUCMNav supports links between the two languages that can be exploited during analysis. An overview of the current editing and analysis capabilities is given, with a particular emphasis on the new concept of GRL *strategies*, which simplify the evaluation of GRL models. The extensibility of the tool is also discussed.

1 Introduction

The User Requirement Notation (URN) [1, 8] enables the modeling and analysis of user and system requirements at a high level of abstraction. It combines two complementary views: the Goal-oriented Requirement Language (GRL) for modeling goals, (non-functional) requirements, alternatives, and rationales [16], and the Use Case Map (UCM) notation for operational scenarios superimposed onto architectural components [17]. An overview of URN's concrete syntax is given in Appendix A, and a simple URN model is introduced in Section 2.

Tools exist in isolation for each individual view. The UCMNav tool [12] supports the various applications of the UCM notation via an X11-based graphical editor and transformation procedures to various target languages (including Message Sequence Charts). UCMNav however suffers from usability and maintainability issues and only the scenario-oriented view of URN is supported, not GRL. For creating and analysing GRL models, the best solution currently available is OpenOME [18]. This visual editor supports multiple goal and agent languages (including GRL, the NFR framework, and *i**) and can be integrated to different development environments (Protégé and Eclipse). However, as it does not cover scenario languages, URN is again only partially supported.

This paper introduces *jUCMNav*, a new open-source tool for editing and analysing URN models. This tool is a plug-in for Eclipse, an extensible Java-based development platform. jUCMNav was first developed to support the UCM notation [9], but GRL was recently added to achieve complete coverage of URN.

This tool enables the creation of links between elements of both views, hence producing an original and highly desirable integration. A particular emphasis was put on producing a usable and maintainable tool to support transformations and explore extensions to the notation.

In this paper, our goal is to provide an overview of jUCMNav and of its capabilities, as this is the first tool that supports the URN notation in its entirety. A simple URN model is first introduced in Section 2. Section 3 gives an overview of jUCMNav's architecture and metamodel while Section 4 presents the editing capabilities of the tool. Section 5 describes analysis capabilities, with a particular emphasis on the new concept of *strategies*, which support multiple evaluations of GRL models in a simple way. In Section 7, we give an overview of the extensibility of the tool, and then we conclude with a discussion of ongoing development work.

2 A Simple URN Model

This section includes a brief example that illustrates part of the URN notation and some of its typical uses. The interested reader can access more comprehensive tutorial material online[1].

The context is the following. Since security has become an important objective in a company that develops Web-based applications, the company is considering improving how to access these applications securely. Different stakeholders may have different concerns related to that new feature. For instance, management is interested in minimal costs, users desire a system that is easy to use, and company shareholders want to see a good return on their investments. Also, alternative means of authentication (e.g., passwords, cardkeys, or biometric information) can lead to different impacts on how well security is achieved, and at what cost.

In GRL, *softgoals* (clouds) are used to express qualitative and non-functional concerns such as security and performance, whereas *goals* (ellipses) are used to denote functional concerns. *Tasks* (hexagons) usually represent element of solutions used to achieve goals. All these types of intentional elements can be decomposed as AND/OR graphs, and they can also *contribute* to each other at various degrees, positively or negatively. An example GRL model capturing some of the aspects of our example is shown in Figure 1.

Stakeholders can be captured as *actors* (dashed circles), which can include intentional elements of interest (see Figure 2). Actors may depend on each other to achieve goals or tasks, or for resources to be produced. One such *dependency* is depicted in our example: shareholders depend on users for a high utilization of the system. The ease of use on one side can hence influence the return on the investment on the other side.

Some aspects of requirements are more operational or architectural in nature and are better represented as scenarios. The UCM view models scenarios as causal sequences of *responsibilities* (crosses on a path). Scenarios evolve from

[1] Please see http://jucmnav.softwareengineering.ca/twiki/bin/view/ProjetSEG/ and http://www.UseCaseMaps.org for tutorials, tools, and demonstrations.

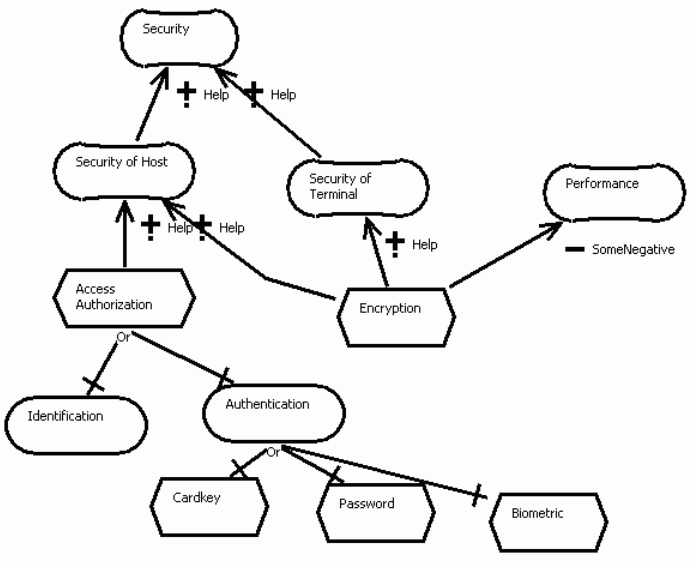

Fig. 1. Simple GRL diagram with decompositions and contributions.

start points (filled circles), representing pre-conditions or triggering events, to *end points* (bars), representing post-conditions or resulting events. The various scenario elements can be bound to actors and architectural *components* (rectangles). Paths can be forked and joined using alternatives and concurrency. An AND-Fork is used in Figure 3 to split the path into two concurrent paths. Complex maps can be decomposed in sub-maps. *Stubs* (diamonds) are containers for such sub-maps, called *plug-ins*. Start/end points in the plug-in can be bound to input/output segments of the stubs, hence ensuring continuity of the scenarios across multiple map levels.

URN models can help answer many analysis questions at that level, such as:

- How are the top-level goals affected by a given selection of alternatives? For instance, each of the alternative authentication task could have side-effects (called *correlations* in GRL) on other goals in the system, e.g. cost. The best trade-off can hence be searched by studying multiple combinations, which we will call strategies in Section 5.1.
- How best can we satisfy the goals of the various stakeholders?
- What is the most suitable component architecture to support the scenarios while achieving a good global trade-off?
- If some selected GRL tasks and goals describe operations or activities, are they supported by scenarios in the UCM model?
- Are the scenarios documented the ones stakeholders really want?
- What happens to the scenarios when objectives change, and vice-versa?

However, to answer such questions and help automating the analysis process, the elements of the two views need to be linked explicitly. This is one of the

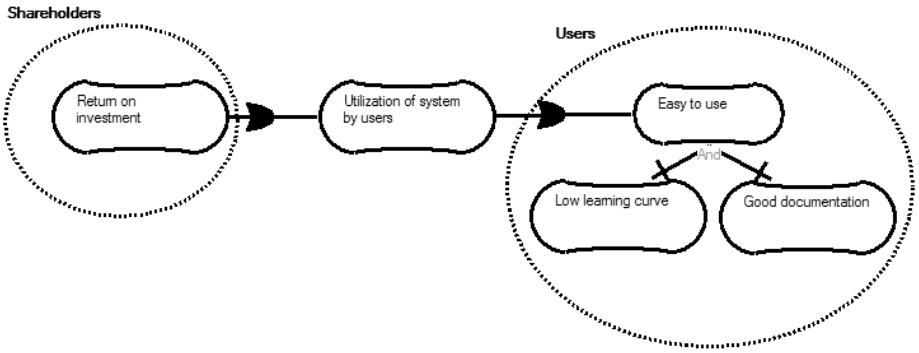

Fig. 2. Simple GRL diagram with actors and dependencies.

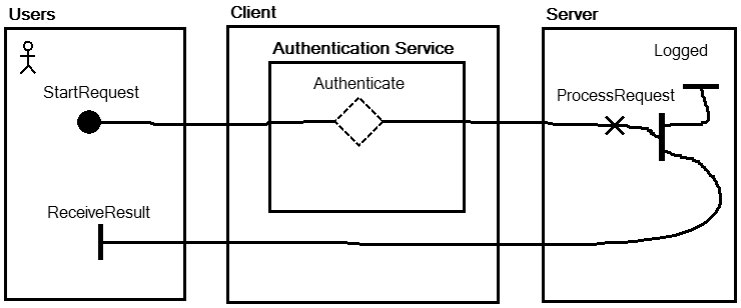

Fig. 3. Simple UCM diagram (map).

main motivations behind the creation of the jUCMNav tool. To enable this support, we created a metamodel for GRL and linked it to the UCM part. We also created implementation metamodels to generalize and reuse the implementation mechanisms already present in jUCMNav's UCM editor. These are explained in the next section.

3 jUCMNav Architecture and Metamodels

Based on a Model-View-Controller (MVC) architecture, jUCMNav makes extensive use of two complementary Eclipse plug-ins: the Graphical Editing Framework (GEF) [5] and the Eclipse Modeling Framework (EMF) [4]. GEF provides rich reusable components and a flexible infrastructure for creating graphical editors (MVC's view and controller). EMF handles the model part of MVC with a set of Java classes generated automatically from a metamodel (e.g. URN's) commonly expressed with UML class diagrams. EMF also provides the serialization of models in XMI, hence automating the saving/loading of models. Changes to the metamodel are automatically replicated in the implementation with minimal coding effort. However, we observed that several types of changes (e.g., delet-

ing/renaming an attribute or a class) can break backward compatibility of the XMI files produced [3].

We have developed two distinct metamodels in order to split the core URN concepts from the additions required to capture graphical layout information as well as elements and attributes that have no semantic impact. Thus, we separated the abstract syntax from the internal representation of the concrete graphical syntax.

The abstract metamodel defines the concepts of both URN views. For the GRL sub-notation, the abstract syntax metamodel in Figure 4 defines basic GRLGraphs, which contain intentional elements (softgoal, goal, resource and task), beliefs, actors, and links (contribution, decomposition and dependency). For the UCM sub-notation (not shown here), the metamodel defines concepts such as UCMmaps, which contain component references, path nodes, and node connections. Different sub-types of path nodes exist, such as start and end points, responsibility references, AND/OR forks and joins, waiting places, and timers. The complete metamodel also includes classes and associations describing component and responsibility definitions, performance annotations, and scenario definitions.

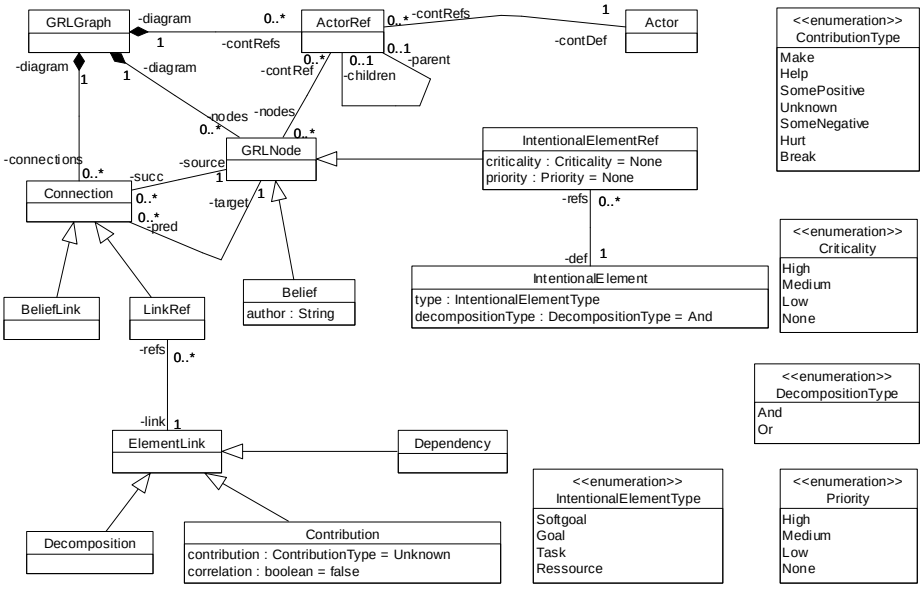

Fig. 4. Main elements of the abstract URN/GRL metamodel.

From this abstract syntax metamodel, we developed an *implementation metamodel* and used it to generate Java code via EMF. In jUCMNav, URN's implementation metamodel is composed of nearly 100 classes.

We transformed the abstract syntax metamodel to an implementation metamodel in two steps. First, we created packages for GRL and UCM, and added a URNcore package that defines concepts common to GRL and UCM, including a generic URNmodelElement class, which is a superclass of most of the URN conceptual classes. Also, amongst the most important elements in this package are the *interfaces* that define the common traits between both URN sub-languages, such as diagrams, nodes, connections, containers, and container references. A container is an element that can contain nodes whereas references allow for multiple instances of a container in the same URN model. These generalizations enable the simplification and standardization of the editors for both notations.

The second step of the metamodel refactoring was the addition of (visual) attributes and classes for the implementation of our concrete syntax. Attributes are elements such as position (x, y), size (height, width), color, and informal descriptions. These changes are mainly located in the interfaces of the URNcore package. We also added classes in both of the notations to support link routing in URN diagrams.

Figure 5 shows how the basic GRL notation implements the URN abstract interfaces. For instance, the ActorRef class implements the IURNContainerRef interface. Note that all the classes, attributes, and associations from the abstract syntax metamodel are preserved in this implementation metamodel.

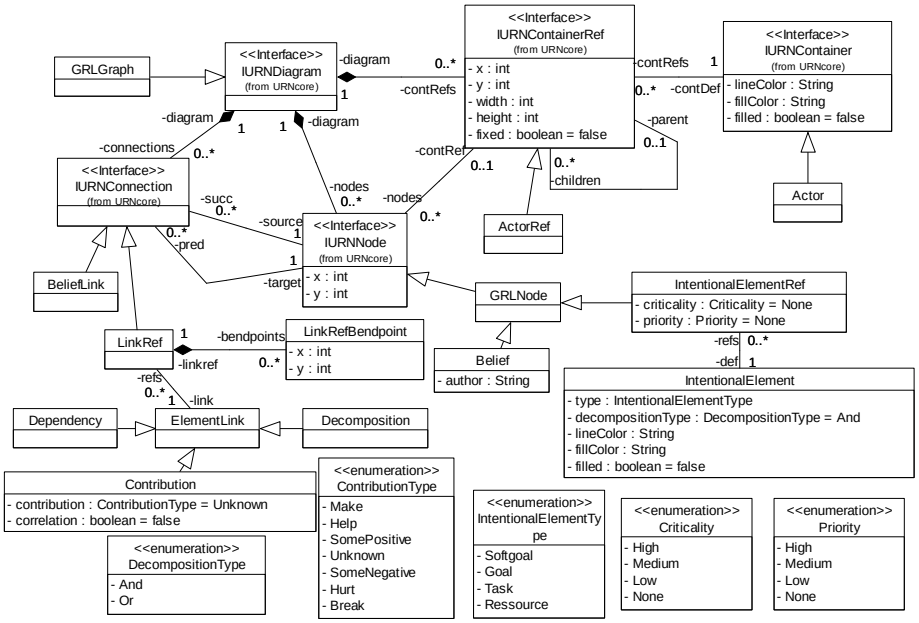

Fig. 5. Main elements of the implementation URN/GRL metamodel.

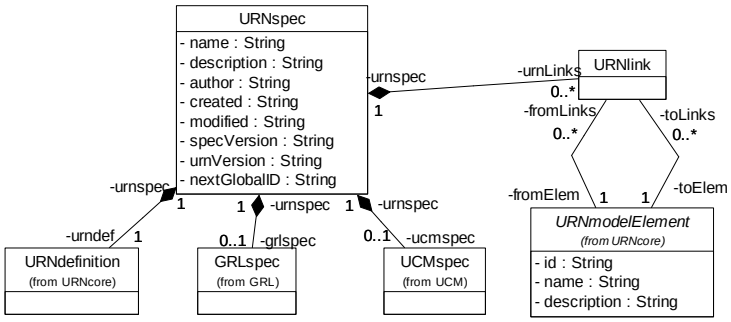

Fig. 6. Links in the URN package of the implementation metamodel.

The new LinkRefBendpoint class has been added to support link routing. This class defines the position in a graph where its associated link should be routed. This package also includes the analysis attributes of the GRL model, i.e. evaluations and strategies, which will be further explored in Section 5. The same interfaces are reused in GRL and UCM. For instance, GRL nodes and UCM path nodes both implement the IURNNode interface, as they both have a location and can be moved, connected together, and bound to a IURNContainerRef container (i.e., an ActorRef in GRL and a ComponentRef in UCM). Most of the editing operations performed on the nodes, links, and components hence become common to GRL models and UCM models.

To complete the integration of the two notations, we also added a top-level package named URN (Figure 6) that includes URN definitions, GRL specifications, and UCM specifications. In addition, the URNlink class (also part of the abstract syntax) allows one to define relationships between any pair of URN model elements. This important capability will be explored in greater detail in Section 5.3.

4 jUCMNav Editor Capabilities

Our new URN tool supports editing both the Use Case Map notation (Figure 7) and the Goal-Oriented Requirements Language (Figure 8).

The core path elements are supported: start points, end points, responsibilities, stubs, waiting places, timers, and forks/joins (both alternative and concurrent paths). Furthermore, various component types (actor, agent, process, and team) are available, as is binding a component or path element to a parent component. The more unconventional elements, such as timestamps, dynamic responsibilities, and dynamic components have not yet been integrated, but their addition should be straightforward. jUCMNav only allows the creation of syntactically valid UCM models, even taking into consideration implicit loops. Not only is the creation and manipulation more intuitive than other UCM/GRL tools, but the deletion mechanisms are richer, more robust, and less restrictive.

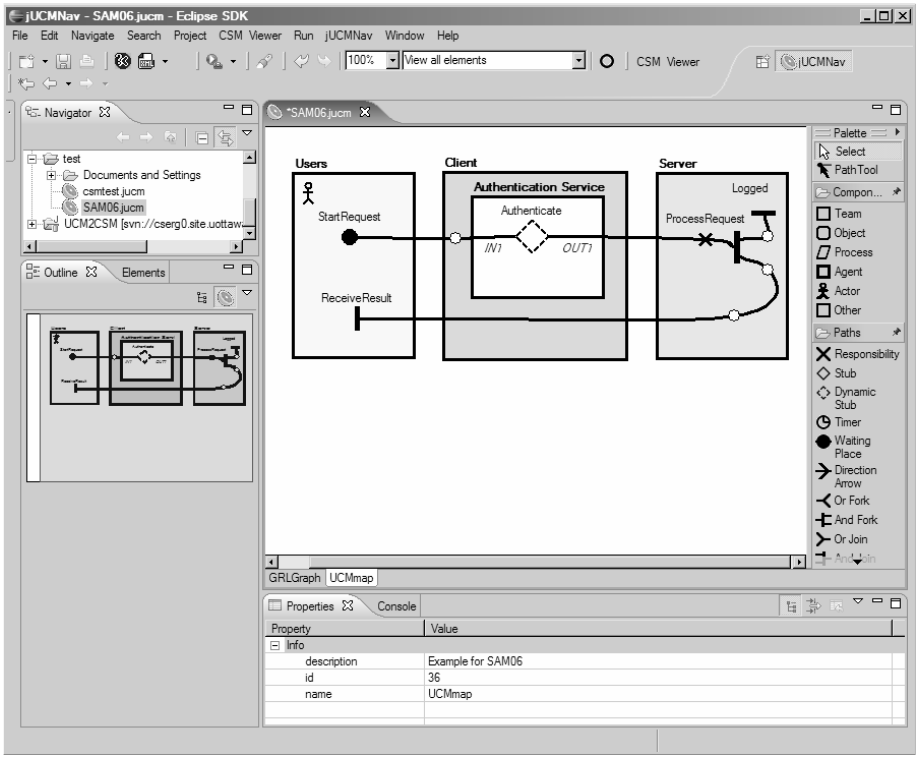

Fig. 7. UCM view in jUCMNav.

The GRL editor supports most of the constructs defined in the draft standard [16]. The intentional elements supported are goals, softgoals, tasks, and resources. These elements can have multiple references to simplify the creation and visualization of complex model via multiple diagrams. These references can be bound to actors, influencing the result of some analysis features offered in the tool.

In contrast with previous GRL tools and for a better integration with its UCM counterpart, the actor's boundary (dashed circle) is not optional and has many commonalities with UCM components in its implementation and behaviour. Beliefs are also available in the application; however they are used mainly to document rationales in the graphical view of the model when linked to intentional elements (without affecting analysis). Finally, the links supported include AND/OR decompositions, contributions, correlations, and dependencies, with their respective attributes, annotations, and graphical representations.

In addition to conventional dropdown and contextual menus, the new editor infrastructure offers a good user experience thanks to drag and drop editing, group manipulation and especially unlimited undos and redos. Furthermore, taking advantage of the standard Eclipse views, jUCMNav features an outline

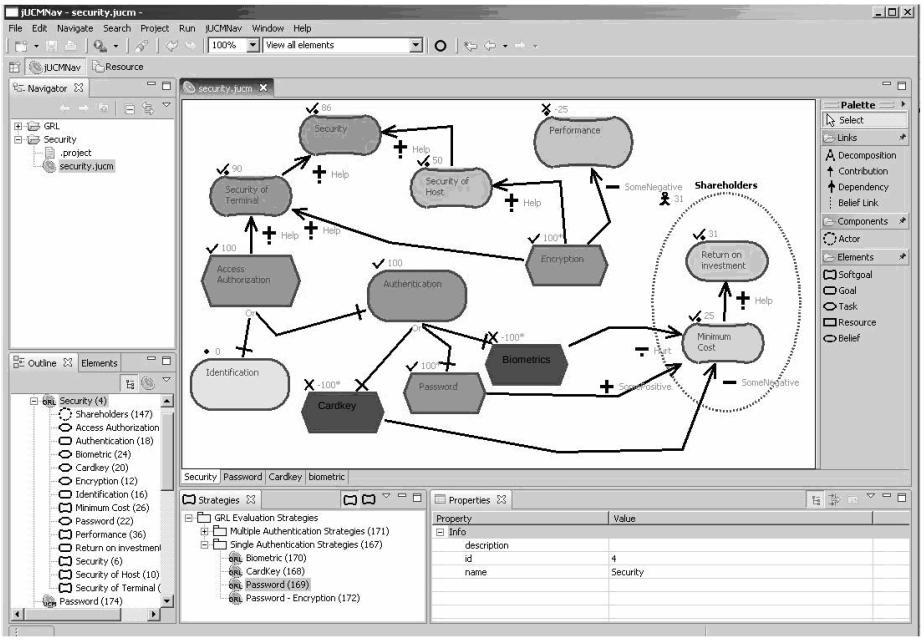

Fig. 8. GRL view with strategy analysis in jUCMNav.

(hierarchical and graphical), a properties view, and a resource view. These views can be moved, closed, or maximized. Both GRL and UCM diagram editors use the same Eclipse-based user interface metaphors. Images can also be exported in various formats.

A new feature in jUCMNav that is available to both notations is an optional auto-layout mechanism, which relies on Graphviz [13] to position the diagram elements. Although imperfect, the presence of this feature is necessary in the context of automated reverse/round-trip requirements engineering. A tool that generates UCM/GRL models from design artifacts such as code, execution traces, requirements, or textual use cases hence does not require manual positioning of the elements.

The auto-layout mechanism is also used in jUCMNav's *catalogues*, which are repositories of reusable GRL models or patterns often used to describe common model elements and relationships related to security, performance, and other non-functional aspects. Using the import/export facilities integrated in Eclipse, this feature allows one to export a model's intentional element definitions and links to an XML file. Modellers can then reuse patterns from such catalogues to kick-start new URN models or add elements to existing ones. The import creates the GRL definitions and links in the new model and builds a new GRL diagram representing the pattern.

5 New Analysis Capabilities for URN Models

5.1 GRL Strategies

By providing access to a complete URN model, jUCMNav can offer novel analysis mechanisms. In order to more easily analyze GRL models and find what selection of alternatives can lead to the best trade-off amongst the often conflicting goals of the stakeholders, we developed the concept of GRL *strategies*, which are user-defined sets of initial evaluations on a GRL graph (Figure 9). These evaluations are satisfaction levels initially assigned to some of the intentional elements in the model (often the leaves of the graph), which are then propagated to the top-level intentional elements through the various links. Evaluations are used to determine how well goals in a model are achieved in a given context.

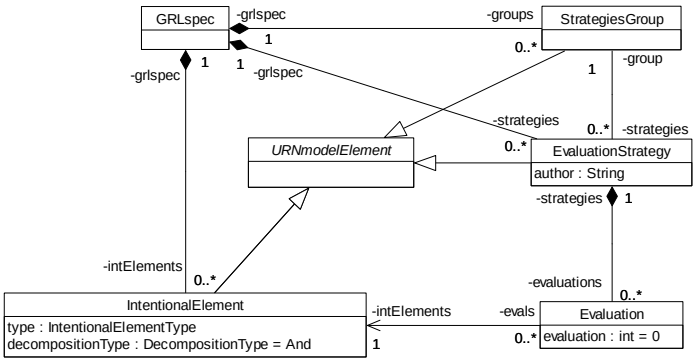

Fig. 9. Evaluation strategies metamodel.

In jUCMNav, strategies can be created, grouped, modified, evaluated, and deleted through the Strategies View. Once a strategy is selected (e.g., Password strategy in Figure 8), the user can access and modify the initial satisfaction level of an intentional element by using the Properties View.

In GRL, satisfaction levels for intentional elements are shown graphically using a qualitative scale (satisfied, weakly satisfied, weakly denied, and denied). During jUCMNav's requirement elicitation phase, we realized that some users were interested in having a quantitative interpretation of satisfaction levels in a strategy. We have hence implemented an equivalent but more granular representation using numerical values between -100 (denied) and +100 (satisfied). These values are used to display feedback on the affected intentional elements. Both the numerical value and the corresponding qualitative symbol can be used. In addition, element references are color-coded with shades varying from red (-100) to green (+100).

Once a value is entered in a strategy, the propagation algorithm is applied immediately and the user can see the result on the fly. Users can also change the evaluation value of any node in the model, not only the leaf nodes.

The evaluation algorithm, inspired from [1, 7], has been implemented with an automatic conflict resolution mechanism that does not require user involvement. Evaluations depend on the various links (decomposition, contribution, and dependency) between the intentional elements. An evaluation is first calculated from the Decomposition links, as a standard AND/OR graph. For AND and OR decompositions, the results correspond respectively to the minimal and maximal evaluations of the source nodes. In our metamodel (Figure 4), the decomposition type is an attribute of the target IntentionalElement node, which causes a node to be decomposed by only one type of decomposition.

The propagation algorithm then evaluates the Contribution links. For each contribution x of a target element with N input contributions, the satisfaction level of the source element and the contribution level are used as described in Algorithm 1. The contribution level, LEV_x, is given a numerical value between -1 and 1 according to the contribution type on the link (1 for make, 0.5 for help, -1 for break, etc.). The satisfaction level, $NEVAL_x$, is normalized to a value between 0 (denied) and 100 (satisfied). The normalized evaluation is multiplied by the contribution level. The results of each of the contributions are added and normalized to provide the total contribution, $TCON$, between -100 and 100.

The normalized evaluation is calculated using the $Tolerance$ attribute, which is set to 0 by default but can be modified by the jUCMNav user. It defines the range of values that are considered satisfied (or denied). For example, with a tolerance of 10, evaluations between 90 and 100 are considered fully satisfied and evaluations between -90 to -100 are considered fully denied. If there are no make/break contributions, then the result is normalized to weakly satisfied or weakly denied ($100 \pm (1 + Tolerance)$) and is added to the decomposition value.

Algorithm 1: Contribution evaluation

$$TCON = \sum_{x=1}^{N} NEVAL_x \times LEV_x$$

if $((TCON \geq (100 - Tolerance))$ **and** $(LEV_{x=1..n} \neq 1))$
then
  $TCON = 100 - (1 + Tolerance)$
else
  if $((TCON \leq (-100 + Tolerance))$ **and** $(LEV_{x=1..n} \neq -1))$
  then
    $TCON = -100 + (1 + Tolerance)$
  endif
endif

When jUCMNav's strategy view is used (see Figure 8), elements with an initial value in the selected strategy are indicated with the * annotation. Figure 10 shows the evaluation of a given strategy on the GRL diagram of Figure 1, and its impact on Security and Performance.

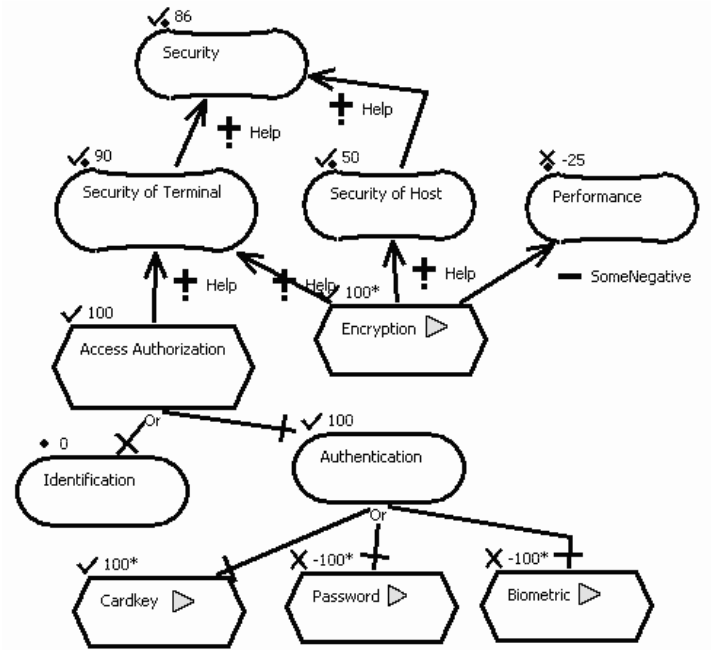

Fig. 10. Evaluation of a GRL model.

Finally, the Dependency links are evaluated. The minimal value among the dependees is compared with the current evaluation of the source node. The resulting evaluation corresponds to the minimum value of those two evaluations. The rationale is that an intentional element cannot have a higher value than those it depends on. Figure 11 shows a case where an element A depends on two other elements, B and D, which depend on elements C and E respectively. By default, evaluations are set to 0. Element C does not influence the evaluation of B because it is greater than the default evaluation. However, element E is less than the default evaluation of element D, which causes D's evaluation to become -30. This is in turn propagated to element A.

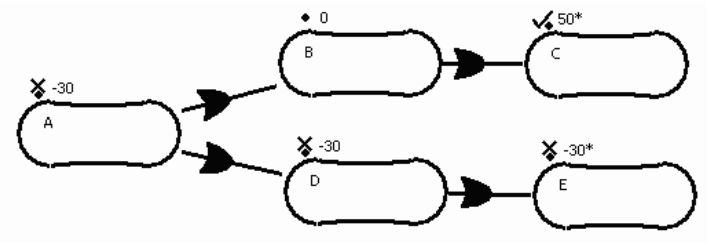

Fig. 11. Dependencies evaluations.

The implementation of this algorithm has been done in a generic and open way using the *strategy design pattern* [6] (not to be confused with GRL strategies), which offers the possibility to easily implement other propagation and evaluation algorithms. To implement such an extension, the developer makes use of the provided Eclipse extension point, which includes methods to calculate the evaluations of one node based on its decomposition, dependency, and contribution links, as well as methods to specify how the evaluations should be propagated in the model. This means that several variants of this algorithm, with different tolerances and logic, could be supported by jUCMNav.

5.2 Actor Evaluation

Our tool also offers a novel analysis label for actors in order to help visualize negotiations between stakeholders and assess the global satisfaction level of actors for a given strategy. This actor label is a value between -100 and 100 computed from the *criticality* and *priority* attributes of its intentional elements references. For a given actor, the evaluation algorithm iterates through its list of bound intentional elements. For both priority and criticality, it multiplies the evaluation of each element by the corresponding factor (by default, 1.5 for high, 1.0 for medium, 0.5 for low and 0 for none), and computes the average per bound intentional element. Finally, it sums up both evaluations and normalizes the result between -100 and 100. A simple example is shown in Figure 12, which illustrates part of a more complex model that includes Figure 10 and the strategy discussed in the previous section. The selection of the CardKey alternative that led to a good security now also leads to high costs that will dissatisfy management.

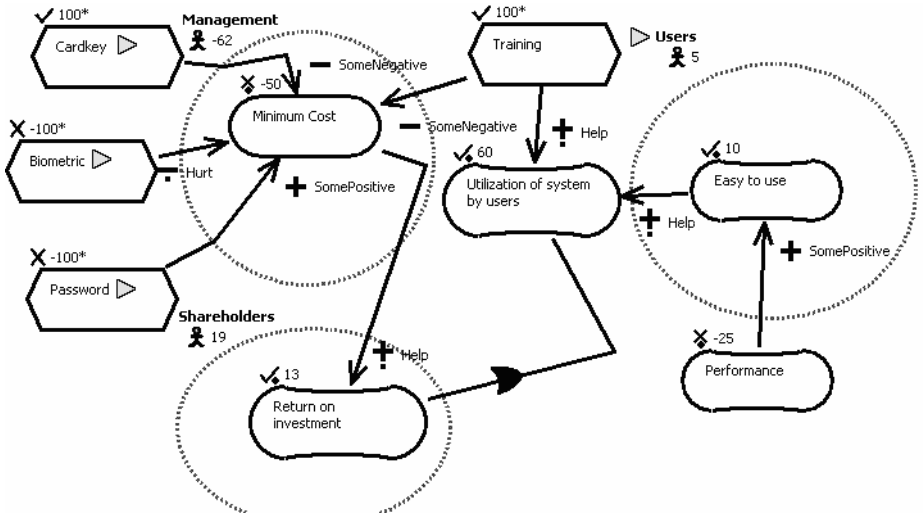

Fig. 12. GRL diagram annotated with links and actor evaluations.

5.3 URN Links

The integration of UCM and GRL views in the same tool allows for the creation of various types of traceability links between elements of both notations, as shown in Figure 6. These links can be used to measure the impact of a modification to any evolving GRL/UCM diagram on the other aspects of the model. They can also improve consistency between the URN views. For instance, links can be defined between GRL intentional elements or actors as source, and UCM responsibilities, components, or maps as target. In this case, when the user selects a strategy, the satisfaction level of the source GRL element is *also* displayed on the target UCM element at the other end of the link (if any). Using this approach, one can evaluate the impact of a goal strategy on the operational and architectural aspects of the model.

The partial URN model in Figure 13 extends the simple scenario of Figure 3 to one that authenticates the user and then processes the request over encrypted channels if the request and the user are valid. This diagram is part of the same URN model as Figure 10 and Figure 12. In this model, URN links were created from the GRL Encryption task to the UCM Encryption and Decryption components, as well as to the UCM Encrypt and Decrypt responsibilities. Other URN links are set between the User actor in GRL and the User component in UCM, as well as between the Authentication tasks (Cardkey, Password, Biometric) and each of the corresponding UCM plug-in maps (bound to the Authenticate stub but not shown here).

The triangles in this figure and the previous ones are not part of the URN notation. They indicate the presence of URN links, and the evaluation results from the corresponding GRL elements are displayed between curly brackets. For instance, the Encryption UCM component shows the degree of satisfaction of the linked Encryption GRL task. Feedback is updated automatically as other strategies are selected.

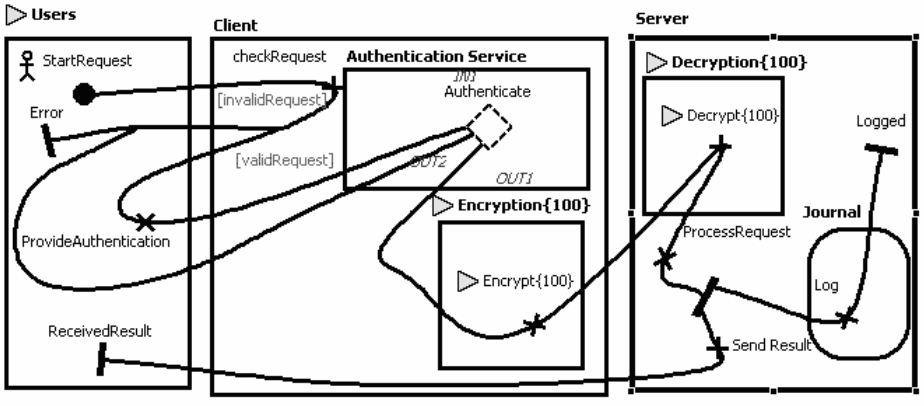

Fig. 13. UCM diagram annotated with links and actor evaluations.

6 Extensibility of the Tool

jUCMNav can be extended with new algorithms for evaluating strategies in a GRL graph, as discussed in Section 5.1. Taking advantage of Eclipse's component model, the tool also offers other extension opportunities.

As suggested previously, URN models can be generated from other artefacts. A specific example is the automatic generation of UCM models from textual use cases defined in a structured natural language. Textual use cases are inherently ambiguous, and completeness and consistency are often hard to analyze. Tools already exist to extract domain models, scenarios, and finite state machines from textual use cases to help facilitate this analysis (e.g., UCEd, a use case editor [15]). We recently demonstrated the usefulness of jUCMNav extension points with a complementary plug-in that generates graphical UCM models (with automatic layout) from validated UCEd project files [11].

jUCMNav's extension points, which provide access to the URN model under design, were also used in a plug-in that enables the import and synchronization of URN models in a requirements management system, namely Telelogic DOORS. jUCMNav can export URN models (i.e., UCM and GRL views) via script files in the DOORS eXtensible Language (DXL). URN elements can be linked to other requirements in DOORS and both views can be kept synchronized as they evolve (e.g., by re-importing the modified URN model) [10]. Figure 14 shows one of the views, corresponding to a UCM diagram, as seen from DOORS.

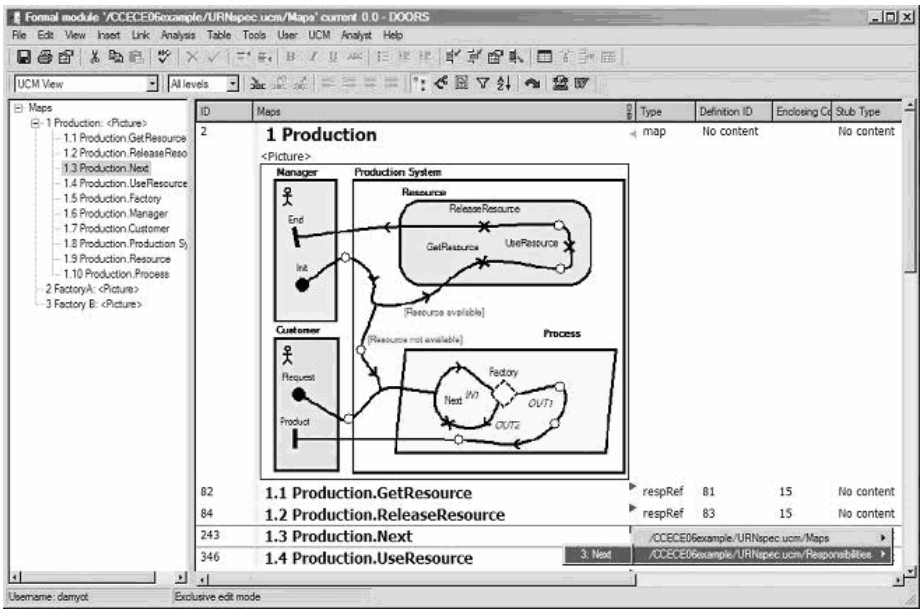

Fig. 14. URN model in Telelogic DOORS.

7 Conclusions and Future Work

The development of jUCMNav allowed us to validate many of the existing URN concepts recently expressed with a metamodel. This is the first tool that supports both URN views in a uniform and unified way, thanks in part to the generalization done at the level of the implementation metamodel, which eased the addition of the GRL editor by reusing much of the code developed for the original UCM editor. This open platform also allowed us to prototype and to explore new URN concepts related to GRL strategies, propagation algorithms, and catalogues, as well as various useful links and connections between GRL and UCM. Strategies and links for URN models contribute greatly to answering the types of questions mentioned in Section 2. Extension points were added and exercised for the creation of various functionalities such as use case import, integration with a requirements management system, image export, catalogue export, and support for multiple GRL evaluation algorithms.

In the near future, the missing UCM notation elements will be added. Also, jUCMNav will be extended to support *scenario definitions* enabling dynamic analysis and transformations to MSCs, UML, and test goals [2], as well as an export mechanism to the Core Scenario Model for *performance modelling* [19]. A simple data model compatible with SDL is being added, and UCM scenarios will be defined with a user interface similarly to GRL strategies. We will also add an import filter for the old UCMNav file format, for backward compatibility.

The integration of GRL and UCM in one tool opens the door to many new possibilities. We plan to add better analysis capabilities in jUCMNav that will measure the impact of strategic decisions on the scenario and architectural aspects of the model. This will be possible by building dynamic views of the UCMs. For example, operational choices for goals, realized through tasks, have an influence on the system architecture. We will modify the UCM views depending of the operational choices made in the GRL strategy. The main contribution of this feature would be to visualize impact of goals and non-functional choices through scenarios. We will also work on further URNlink types and on improving the modeling and analysis process with such links.

Acknowledgments

This research was supported by NSERC, through its programs of Strategic Grants, Discovery Grants, and Postgraduate Scholarships. We are grateful to E. Tremblay, J.-P. Daigle, J. McManus, Y. Kim, J. Sincennes, and G. Mussbacher for various contributions to the tool. We also thank A. Prinz and the anonymous reviewers for their comments on the workshop version of this paper.

References

1. Amyot, D. and Mussbacher, G: URN: Towards a New Standard for the Visual Description of Requirements. In E. Sherratt (Ed.): Telecommunications and beyond: The Broader Applicability of SDL and MSC (SAM 2002). Lecture Notes in Computer Science 2599, Springer 2003, 21–37.

2. Amyot, D., Cho, D.Y., He X., and He, Y.: Generating Scenarios from Use Case Map Specifications. Third International Conference on Quality Software (QSIC'03), Dallas, USA, November 2003, 108-115.
3. Amyot, D., Farah, H., and Roy, J.-F.: Evaluation of Development Tools for Domain-Specific Modeling Languages. Fifth Workshop on System Analysis and Modelling (SAM06), Kaiserslautern, Germany, May 2006. This issue.
4. Eclipse: Eclipse Modeling Framework (EMF), `http://www.eclipse.org/emf/`
5. Eclipse: Graphical Editing Framework (GEF), `http://www.eclipse.org/gmf/`
6. Gamma, E., Helm, R., Johnson, R., and Vlissides, J.M.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, USA, 1995.
7. Giorgini, O., Mylopoulos, J. and Sebastiani, R.: Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology. Engineering Applications of Artificial Intelligence, 18(2):159–171, March 2005.
8. ITU-T: Recommendation Z.150, User Requirements Notation (URN) – Language Requirements and Framework. Geneva, Switzerland, 2003.
9. Kealey, J., Tremblay, E., Daigle, J.-P., McManus, J., Clift-Noël, O., and Amyot, D.: jUCMNav: une nouvelle plateforme ouverte pour l'édition et l'analyse de modèles UCM. 5ième colloque sur les Nouvelles TEchnnologies de la RÉpartition (NOTERE'05), Gatineau, Canada, August 2005, 215–222. `http://jucmnav.softwareengineering.ca/twiki/bin/view/ProjetSEG/WebHome`
10. Kealey, J., Kim, Y., Amyot, D., and Mussbacher, G.: Integrating an Eclipse-Based Scenario Modeling Environment with a Requirements Management System. 2006 IEEE Canadian Conf. on Electrical and Computer Engineering (CCECE'06), Ottawa, Canada.
11. Kealey, J. and Amyot, D.: Towards the Automated Conversion of Natural-Language Use Cases to Graphical Use Case Maps. 2006 IEEE Canadian Conf. on Electrical and Computer Engineering (CCECE'06), Ottawa, Canada.
12. Miga, A.: 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/
13. North, S., *et al.*: Graphviz, 2005. `http://www.graphviz.org/`
14. OMG: Unified Modeling Language (UML), version 2.0, October 2004. `http://www.uml.org/#UML2.0`
15. Somé, S.: An Environment for Use Cases based Requirements Engineering. Formal demonstration. 12th IEEE Int. Requirements Engineering Conf. (RE04), Japan, September 2004. `http://sourceforge.net/projects/uced/`
16. URN Focus Group: Draft Rec. Z.151 – Goal-oriented Requirement Language (GRL). Geneva, Switzerland, Sept. 2003.
17. URN Focus Group: Draft Rec. Z.152 – Use Case Map Notation (UCM). Geneva, Switzerland, Sept. 2003.
18. Yu, E.: OpenOME, an open-source requirements engineering tool, 2005. `http://www.cs.toronto.edu/km/openome`
19. Zeng, Y.X.: Transforming Use Case Maps to the Core Scenario Model Representation. M.Sc. thesis, SITE, University of Ottawa, Canada, June 2005.

Annex A: Overview of the User Requirements Notation

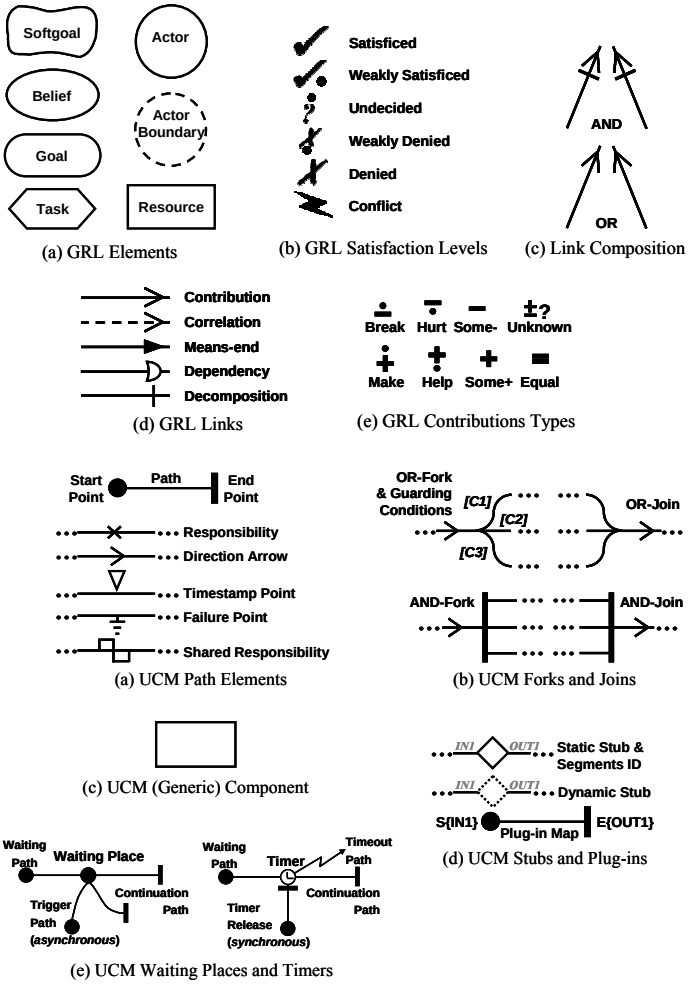

Fig. 15. Summary of the GRL and of (a subset of) the UCM concrete notations.