Traceability and Evaluation in Scenario Analysis by Use Case Maps

Dorin B. Petriu[1], Daniel Amyot[2], Murray Woodside[1], and Bo Jiang[2]

[1] Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6
{dorin | cmw}@sce.carleton.ca
[2] SITE, University of Ottawa
800 King Edward
Ottawa, Ontario, Canada, K1N 6N5
{damyot | bojiang}@site.uottawa.ca

Abstract. The Use Case Map (UCM) scenario notation has some strong features related to rapid capture and evaluation of requirements models. In this paper, we explain how a UCM model was developed from a requirements oracle case study: the Autonomous Shuttle Transport System. We further consider establishing links between scenario elements and other types of requirements. These links, which can be supported by requirements management tools, are useful to maintain both the scenarios and requirements during their evolution. We also demonstrate how simple performance models generated from UCMs may impact high-level requirements and architectures.

1 Introduction

Requirements, which are expressions of ideas to be embodied in the system or application under development and the conditions under which it will operate, are often collected in unconstrained forms including text, diagrams, tables, and equations or logical formulae. Requirements analysis then uses various techniques to investigate the consistency, completeness, feasibility, and consequences of the requirements. Nuseibeh and Easterbrook discuss integrated requirements engineering, combining a variety of techniques with automated tool support for effective requirements management [17]. They identify the need to move from contextual enquiry to elicit requirements, to more formal representations for analysis.

One form of requirements may be *scenarios*, which describe sequences of operations to be carried out in response to given events, requests, or interactions. Scenarios may be used to drive the elicitation and development of requirements, to refine requirements stated in other ways, and to connect other requirements whose relations would be otherwise unapparent. Lamsweerde gives a thorough discussion on the relationships between goals and scenarios, between informal and formal methods, and between scenarios and other requirements models [14].

Like many others, he noted that scenario specifications are incomplete and cannot be used as a substitute for all types of requirements. Various non-functional requirements, goals, quality attributes, and informal annotations are found in most requirements documents.

In order for scenarios to be used in cooperation with general requirements, they must be connected to external requirements in a way that supports traceability, navigation, and analysis. This paper presents an approach where Use Case Map (UCM) scenarios are constructed from an informal collection of requirements. UCM scenario elements are then imported into a popular *requirements management system* (RMS), namely Telelogic DOORS [21], and linked to other types of requirements. UCMs are abstract scenarios that are close to the requirements abstraction level, and they contain many types of elements that are potentially traceable to other types of requirements.

Scenario management and *scenario evolution*, which are discussed in their largest context by Jarke *et al.* [13], face the issue of maintaining traceability of scenarios that relate to each other and that evolve over time. To avoid an explosion in the number of individual scenarios describing a complex system, several approaches have been developed to capture common parts (often called episodes) and describe interdependencies through relationships such as precedence, alternatives, inclusion, extension, usage, etc., while at the same time improving consistency and maintainability. Breitman and Leite provided an extensive case study on scenario evolution based on such relationships, and they identified the need to develop suitable management systems that would take into consideration scenario relationships [8]. Interestingly, Use Case Maps contain many such relationships as first-class language constructs. Unfortunately, few substantial results are available for either the management of graphical scenarios like UCMs, or their integration to general requirements, with the noticeable exception of the work of Alexander [1] and a recent DOORS add-on called Analyst [22], which will both be discussed in section 6.

This paper introduces a scenario-oriented requirements engineering framework and focuses on three complementary contributions. First, sections 2 and 3 illustrate several steps used in the construction of a UCM model from informal requirements. The case study selected here is the Autonomous Shuttle Transport System (ASTS), presented as a requirements oracle at the *Scenarios: Models, Algorithms and Tools* Dagstuhl seminar [7]. ASTS is a rail-based transport system under development intended to enable individual traffic of people and goods, which today is mainly conducted by cars and trucks, by autonomously acting shuttles on rail [20]. The second contribution is a novel approach to the integration of UCM scenarios in a RMS. Section 4 presents how UCMs can be imported into DOORS, how they can be connected to external requirements, and how these links can be exploited for evolving scenarios, requirements, and designs. We demonstrate the feasibility of such an approach with a new UCMNav export filter, which generates documents that can be imported into a commercial requirements management system. A particular attention was paid to the unavoidable evolution of scenario models and other requirements. The third con-

tribution (section 5) builds on previous work to show that simple analysis and evaluation of performance models generated from UCMs can influence several requirements and architectural decisions early in the development process. Finally, our conclusions are discussed in section 6.

2 Requirements Capture using UCM

2.1 Basics of Use Case Maps

The Use Case Map notation was developed to capture scenario descriptions as causal flows of responsibilities for object-oriented design of real-time systems [9, 12]. In a requirements engineering context, UCMs also proved to have several benefits over many other scenario notations: they abstract from message exchanges, they support scenario integration and interaction detection, and they visually connect behaviour and architecture in a map view [3].

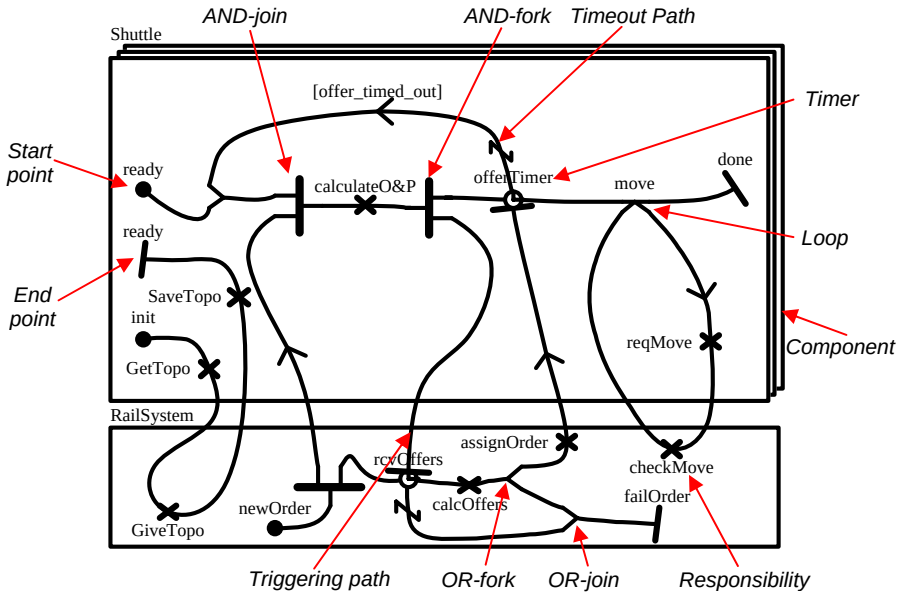

Fig. 1. Initial ASTS Use Case Map (version 1)

As shown in Fig. 1, the UCM notation uses filled circles for *start points* (triggering events and preconditions), bars for *end points* (resulting event and postconditions), crosses for *responsibilities* (abstract actions and activities), and rectangles for *components* (e.g., software module, hardware, actors). Components can contain responsibilities and sub-components. With *paths*, responsibilities can be causally linked in sequence, as alternatives, or in parallel. Maps can also be

decomposed hierarchically with *stubs* (shown as diamonds on a path, see Fig. 2) and *plug-ins* (sub-maps bound to stubs). UCMs are currently being standardized by the International Telecommunications Union as part of the User Requirements Notation (URN) [2, 11].

The UCM Navigator (UCMNav) is a multi-platform tool that supports the editing and analysis of UCM models [24], which can also be exported to various formats such as EPS (used for the figures in this paper), MIF, CGM, and SVG.

UCMs have been used as a basis for various kinds of model transformations. UCMNav can extract individual scenarios from complex UCM models and export them as XML files, which can further be transformed and refined (e.g., with the UCMExporter companion tool [4]) into Message Sequence Charts (MSC), UML 1.4 sequence diagrams, and TTCN-3 test case skeletons. We will take advantage of such transformations in section 4.3. UCM models have also proved to be a good basis for describing and synthesizing system component behaviour in LOTOS [5], SDL [10], and communicating state machines [6].

UCMs can be annotated with performance-oriented information, which enable UCMNav to export performance models in the form of Layered Queueing Networks (LQN) [18]. Enabling scenario-based performance analysis early in the design process and as close to the requirements specification phase as possible may influence several major decisions regarding the system architecture. This topic will be explored further in section 5, again using ASTS as an example.

2.2 Capturing ASTS Scenarios using UCM

The requirements for the ASTS were given to the workshop participants as handouts along with instructions to focus on the shuttle control [20]. One of the handouts provided a high-level overview of the system and described the railway network, the way in which customers place orders, the rail shuttles, and the way in which shuttle income and expenses are assessed. Another handout provided a more detailed description of the simulation environment in which the shuttle control software is evaluated as well as descriptions of typical Use Cases involving the shuttles.

UCMs capture the emerging behaviour of a system. This is done by tracing the behaviour and overlaying it on the system structure. The behaviour traces are called paths and the system structure is represented with components. Along the paths, responsibilities are identified and allocated to suitable components.

In this case, the first step towards creating a UCM for the ASTS involved identifying the system components. Initially the only components identified were the RailSystem and multiple Shuttles, as shown by the rectangles in the UCM in Fig. 1.

The second step was to identify the two main Use Cases from a shuttle's viewpoint which are initialization and serving customer orders. The initialization Use Case deals with the Shuttle acquiring the rail network topology from the RailSystem upon activation. The serving customer orders Use Case has the Shuttle waiting for a new order to arrive from the RailSystem and calculating and

submitting an offer. If the offer is accepted, then the Shuttle proceeds to move and serve the customer.

The initialization Use Case is shown in Fig. 1 as the UCM path that begins at the init start point inside the Shuttle component. The path is based on the Receiving Topology sequence diagram from [20], which simply describes a request from the Shuttle and the answer provided by the RailSystem (referred to as Kernel in the original document). The Shuttle requests the network topology by executing the GetTopo responsibility. The RailSystem records the topology as represented by the GiveTopo responsibility. Finally the Shuttle receives the topology and saves it as part of the SaveTopo responsibility. The Shuttle is now ready to serve customers.

The serving customers Use Case is synthesized from various sequence diagrams from the initial requirements [20]. The path begins with the Shuttle being ready and awaiting the arrival of a newOrder from the RailSystem. The RailSystem sets a timer for waiting on offers from different Shuttles, shown in Fig. 1 as the rcvOffers timer. In the UCM notation, timers are shown with a clock symbol and they are set when reached on a path. When the connected end point from a different scenario path (i.e., the triggering path) is reached in time, the timer is reset and the scenario can progress on the original path, otherwise the time-out path (shown with a zigzag symbol) is taken. When a new order arrives, the Shuttle calculates an offer and a path through the rail network (the calculateO&P responsibility) and sends it to the RailSystem while also setting an offerTimer to wait for a notification that it has been awarded the order. The RailSystem evaluates all the offers and chooses the best one (the calcOffers responsibility). It then notifies the winning Shuttle (the assignOrder responsibility).

The successful Shuttle receives the order assignment and proceeds to serve it. The move loop shows how the Shuttle traverses a track segment by first requesting permission to move onto a new segment (reqMove). The RailSystem checks whether the Shuttle can move safely to the new segment and then notifies it. Any Shuttle that does not get the order times out on the offerTimer timer and resumes waiting for another newOrder.

If the RailSystem does not receive any offers for a given order during the bidding period (rcvOffers times out) or none of the offers are acceptable (calcOffers does not have a winning bid) then it aborts the processing of that order. An order failure handling mechanism was not specified in the ASTS handouts, but such a mechanism can be added later.

This first UCM model shown in Fig. 1 was created in a little over an hour by a single person interpreting the ASTS documents and entering the UCM in the UCMNav tool. The advantage of UCMNav is that it provides a platform for quick editing of UCMs with facilities for exporting and importing models.

3 Scenario Evolution using UCM

The ASTS scenario was rapidly created and improved during the Dagstuhl seminar, which illustrates one of the strengths of the approach. During a group dis-

cussion of about an hour, the initial UCM shown in Fig. 1 was evolved through six steps. After specific feedback following a presentation to the other participants, version 7 (shown in Fig. 4) was created.

Scenarios evolve by the addition of functionality, steps to correct the logic of the path, encapsulation of detail, and restructuring of a set of scenarios (as described in [8]). For example, the first change was by addition, to extend the successful order completion path to incorporate payment to shuttles and to name the successOrder and failOrder end points in the RailSystem. The second change added a second optional shuttle movement in the scenario to get the shuttle to the pickup station. To simplify the map, it also encapsulated the shuttle movement behaviour into a plug-in map within the stub move. This gave version 3 as shown in Fig. 2 and 3. The move stub is used twice (shown as the diamond shapes labelled move), and in both cases, the plug-in map is bound to the stub according to this relationship: {<*IN1* → leave>, <arrive → *OUT1*>}, which ensures the continuation of the path accross connected maps.

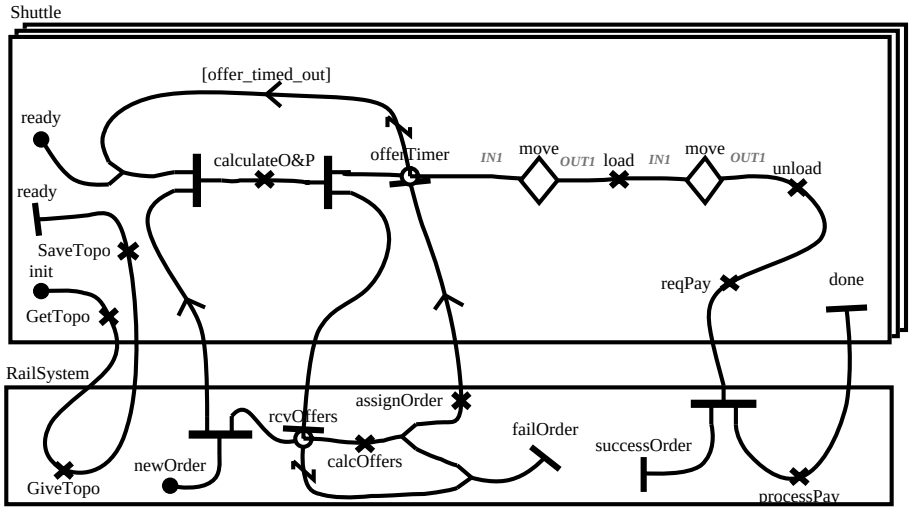

Fig. 2. ASTS UCM with move stubs (version 3)

The next steps are not shown by diagrams, but version 4 introduced an additional optional move of a shuttle for repositioning (as part of a global strategy to provide shuttles in all regions of the system), before a new order is received. Version 5 moved two responsibilities into two new components, a TopoAgent to create and maintain the system view of network topology, and a BankAgent to process payments. Initially these components were nested inside the RailSystem component, where the responsibilities were initially defined, but in version 6 they were made separate (as indicated in Fig. 4). Version 6 also introduced a CommunicationEnv component containing all the other components and representing

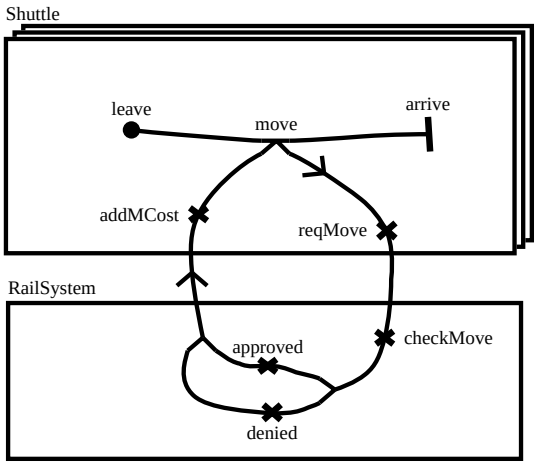

Fig. 3. Plug-in for the move stub in the ASTS UCM (versions 3 to 7)

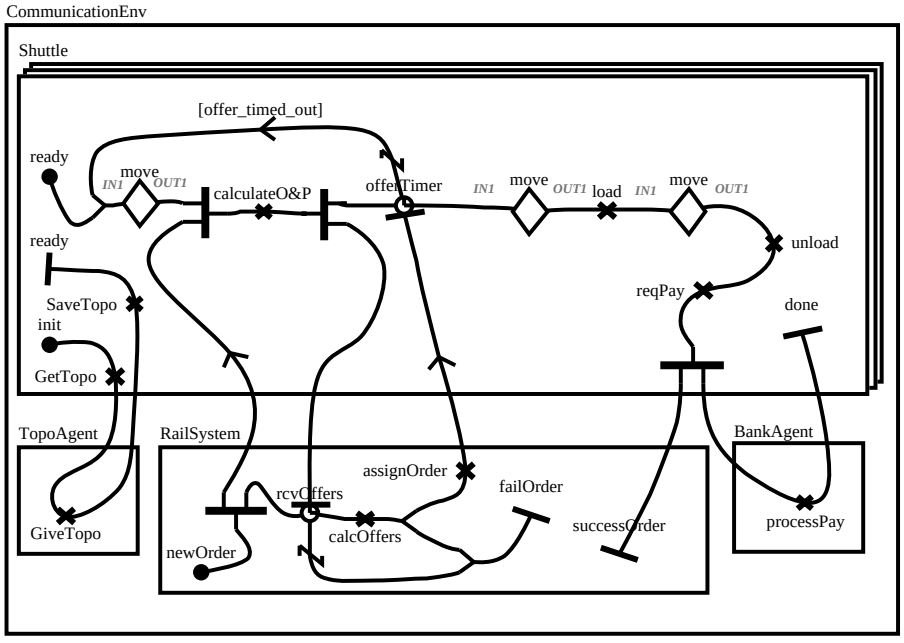

Fig. 4. ASTS UCM with additional move stub and three new components (version 7)

the simulation communication environment. This was done in order to align the UCM with the deployment diagram provided in the informal requirements [20]. Version 6 was presented to the other participants in the requirements oracle session.

Fig. 4 shows the final version (version 7) created in response to feedback received from other participants after the presentation. The only major change was made to the move plug-in where we added approved and denied alternatives to the RailSystem response when a Shuttle requests to move to a new track segment, as well as an addMCost responsibility to account for each track segment that a Shuttle travels on. These were not in the original loop of Fig. 1, nor in the original plug-in map.

4 Managing UCM Evolution in DOORS

The creation and evolution of scenarios and other requirements can be intertwined in many ways. Typically, scenarios will be used to discover requirements or to provide an operational view of existing requirements for understanding and validation. In turn, requirements can also trigger the discovery or evolution of scenarios. Such iterative process can be supported by requirements management systems (RMS), for example Telelogic DOORS [21]. Most RMS focus on structured textual requirements, with support for traceability, access control, and version control. Adding scenarios brings in a complementary view that can be beneficial to many stakeholders.

Many RMS can import requirements from various sources, including word processors. For instance, we can import the original ASTS informal requirements into DOORS, leading to an initial database of requirements objects, as shown by the document in Fig. 5. The nature of these requirements objects can vary from operational requirements to non-functional requirements and quality attributes. They can also be more or less structured, depending on the quality of the source document.

4.1 Combining UCMs with External Requirements

To combine scenario descriptions with other requirements, they should be linked using the facilities of the RMS. Links of this kind between scenarios and informal requirements were discussed also by Leite *et al.* [16], using an experimental RMS.

To use an RMS, the scenario elements must be imported into its data space. When this is done, the intrinsic links within the scenario can also be created as RMS links. These include predecessor/successor sequence links, linking responsibilities to the entity for the scenario, and linking components to scenarios and responsibilities. We have implemented this importation in DOORS using scripts native to the tool, and including facilities for incremental update from a modified scenario.

The process begins by representing the external requirements in the RMS. Fig. 5 shows the textual ASTS requirements in the DOORS tool. Then the

scenario is imported, and its elements are linked to other requirements. For example, a timing requirement for the scenario as a whole can be linked to the scenario entity, or a deployment requirement can be linked to the components it references. Fig. 5 shows an indication of a link from an ASTS requirements object to a UCM.

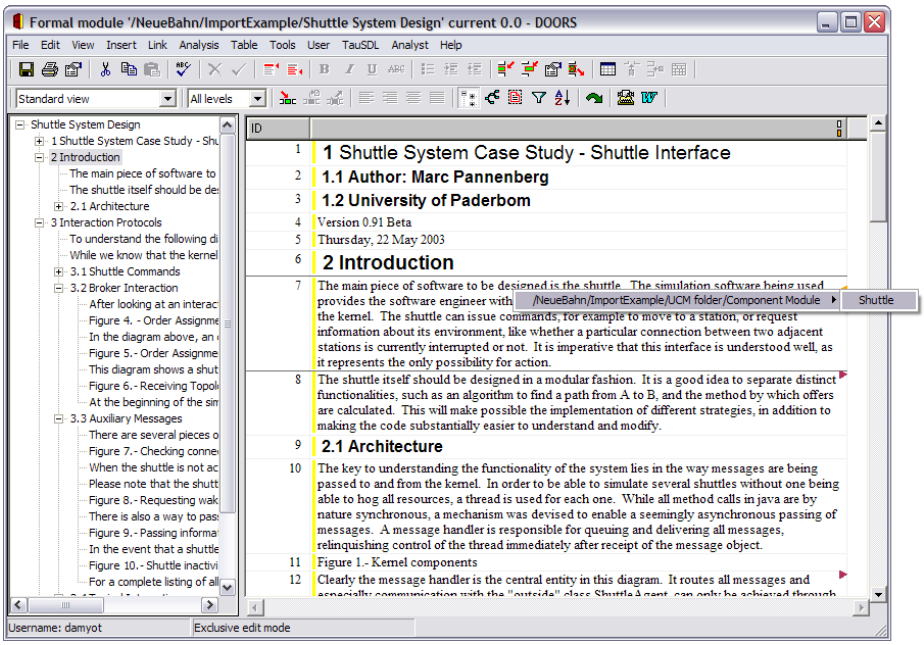

Fig. 5. Original ASTS description imported into DOORS, with links to/from UCMs

4.2 Exploiting Traceability Links: UCM Elements and Other Requirements

The links are used in reasoning about requirements and about changes to requirements. Objects have categories and links are typed. Links are also directional ("A depends on B"), and may be navigated in either direction (that is from a requirement object to those that depend on it, or to those it depends on). Fig. 6 shows a DOORS display of ASTS UCM components and a link from Shuttle to its responsibilities (above) and a display of UCM responsibilities linked with their components (below). Link direction is indicated by an arrowhead.

A "big picture" of relationships through links can help to identify clusters of dependencies, and missing information. Fig. 7 shows a traceability matrix indicating links between entities in the text document (represented by the bars at the top) and the UCM components (indicated by the bars below). The black

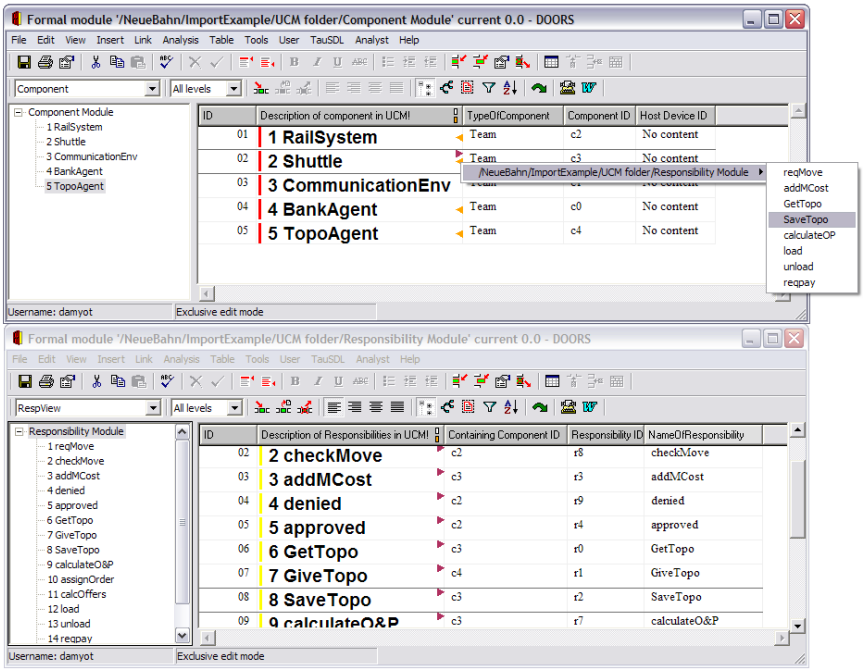

Fig. 6. UCM components and responsibilities in DOORS, with attributes and links

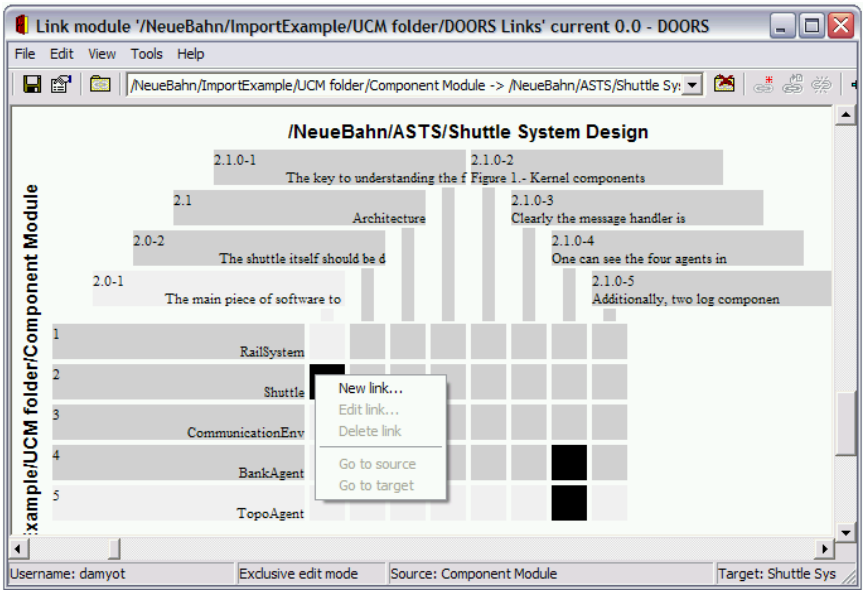

Fig. 7. Traceability matrix between UCM components and external requirements

spots in the matrix indicate the existence of links. If a UCM requirement object is not directly or indirectly linked to external requirements, then this might indicate that a link is missing or that this UCM element is not required. If a requirements change is resolved by a scenario change, the scenario can be updated in the UCM end and re-imported. As mentioned above, links to entities which have not changed are maintained when the map is re-imported.

4.3 Exploiting Traceability Links: UCM Scenarios and Other Requirements

A UCM scenario specification may imply many different paths, depending on the conditions that govern choices made during the execution. These choices can be specified as path preconditions, which are Boolean variables defining guarding conditions on OR-fork branches, timers, and dynamic stubs. The resulting scenario definition implies a corresponding sequential path or partial order. A UCM traversal mechanism [12], implemented in UCMNav, is used to extract the specific scenario (partial order) corresponding to a given definition, and stores the result in a XML file. Our DOORS import capability includes these specific scenario definitions. The XML file can also be converted to various forms [4], such as a Message Sequence Chart (MSC) or a UML sequence diagram.

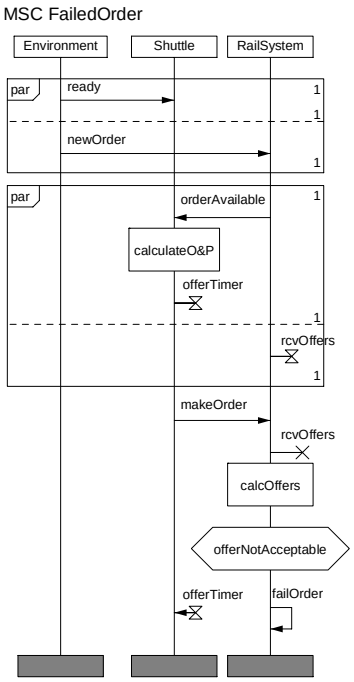

Fig. 8. Result of the *FailedOrder* scenario definition, converted to an MSC

The ASTS UCM in Fig. 4 was supplemented with such variables and conditions. One scenario was defined to describe what happens when a new order fails because the shuttle's offer is not acceptable. UCMNav can highlight the UCM paths traversed by this specific scenario. The resulting scenario was also converted to an MSC by UCMExporter, hence enabling a better visualization of the complete, end-to-end scenario (Fig. 8). Note that the move loop was not traversed in this scenario in order to keep the trace short. In general, UCM start/end points are converted to MSC messages and responsibilities to actions. MSCs were preferred to UML 1.x sequence diagrams here because they support explicit parallel inline statements as well as timers (as in UML 2.0). Additional messages are synthesized during the transformation to insure that inter-component causality is preserved. These synthetic messages have been renamed with more meaningful names here (e.g., orderAvailable and makeOrder).

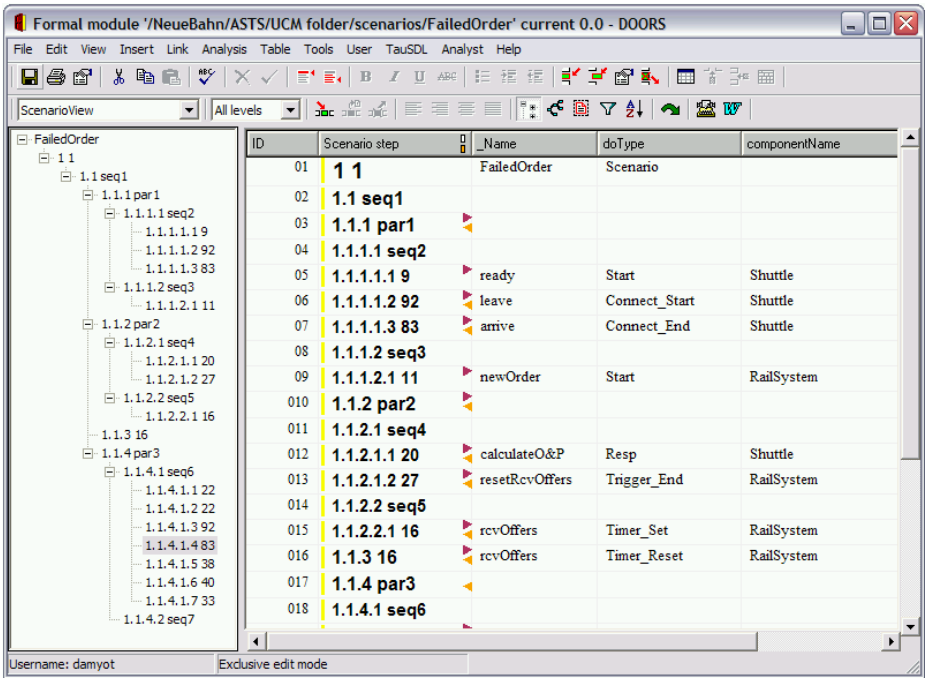

Fig. 9. *FailedOrder* scenario imported into DOORS & linked to other requirements

The same *FailedOrder* scenario was imported into DOORS, as shown in Fig. 9. This scenario view provides the means to connect UCM elements and external requirements in a way that would be difficult otherwise. Instead of manually linking each pair of relevant external requirements directly (there would be too many pairs, and many might be missed by requirements engineers), the traceability can be done more efficiently via UCM scenarios. For instance, the in-

formal descriptions of shuttle and agents (respectively section 2 paragraph 1 and section 2.1.0 paragraph 4 of the informal document), discussed in the previous examples, can be linked in the following way:

- UCM element Shuttle to section 2 paragraph 1 (manual, but obvious)
- UCM element BankAgent to section 2.1.0 paragraph 4 (manual, but obvious)
- UCM scenario SuccessfulOrder (not shown here) to UCM element Shuttle and to UCM element BankAgent (not obvious, but automatic with scenario import)

Such links created automatically provide very helpful support when performing traceability and impact analysis on requirements. A RMS tool could hence answer questions such as "What is connected to this requirement, directly or indirectly?" or "What scenarios and external requirements would be directly or indirectly affected if we removed this responsibility or this component?". Additionally, this automated process would prevent missing non-obvious links, would be easier to use in a scenario/requirement evolution context, and would lead to clearer explanations to questions such as the ones above because of the availability of link types (providing rationales).

5 Performance Evaluation of UCM Scenario Models

Performance requirements represent an interesting application area for the types of links discussed in this paper. UCM scenarios can easily capture functional and operational requirements, but they can also be supplemented with annotations to describe various aspects of performance requirements. This combined view is sufficient to enable the generation of performance models [18]. Analysis of such models can be used to detect hot spots and trace them back to the scenarios and, indirectly, to the components requirements and environment requirements to which these scenarios are linked. This can help prioritize important issues which may lead to the evaluation of alternative requirements for (COTS) components, execution environments, and performance requirements altogether. A strategy where requirements are linked to scenarios analysed outside the RMS is likely to be more profitable and agile than a total integration strategy (scenario tool within the RMS) because the analysis complexity remains outside of the RMS environment. We are currently exploring this strategy.

UCMNav incorporates a built-in export filter that generates Layered Queueing Network (LQN) performance models [15]. The path traversal and transformation algorithm for the generation of LQNs is explained in detail in [19]. Several path detail changes were made to version 7 of the ASTS UCM (Fig. 4) in order to comply with the usage rules for creating UCMs that are well-formed for the purpose of performance model generation, as described in [18].

Fig. 10 shows the LQN model generated from the ASTS UCM. The trapezoids in the diagram represent *tasks* and the arrows represent calling relationships between them – full arrow heads denote *synchronous calls* while half arrow heads

denote *asynchronous calls*. LQN tasks are subdivided into *entries* which represent services that the task provides, as well as optional *activities* that represent the detailed breakdown of the workload for a given entry. For visual clarity, entry and activity details for the ASTS are left out of the LQN figures presented here. Instead, dashed lines are used to provide a graphical shorthand for the entry and activity sequencing inside tasks.

The documents provided at the requirements oracle session did not provide the workload parameters required to do a complete performance analysis of the ASTS. The LQN model was therefore generated with default parameter values as explained in [19]. Even with the use of these default parameters, running the ASTS LQN model through the LQNS analytical solver does provide some interesting non-quantitative insights into the system architecture.

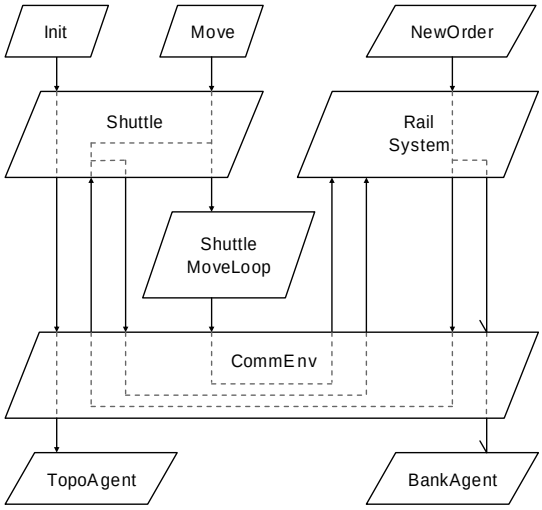

Fig. 10. ASTS LQN showing the calling relationships between tasks

The LQNS solver tool can be configured to automatically detect call cycles in a model [15]. In the case of the ASTS LQN, it detected a cyclical calling pattern between the Shuttle and RailSystem tasks. These cycles can be seen in Fig. 10 and are representative of a breakdown in the layering of a system. Further inspection of the LQN reveals that these cycles are due to the bundling of the track segment management and the order management functions in the RailSystem task. This bundling is due to a lack of detail in the ASTS requirements. Since the documents were focused on explaining the shuttle behaviour requirements, there was no detailed description of the RailSystem itself. Thus the two functions are not actually required to be bundled together and can be separated.

Fig. 11 shows a repartitioned LQN for the ASTS. The RailSystem has been divided into an OrderMgr task to handle new orders and assign them to shut-

tles, and a TrackMgr task to deal with permissions for shuttles to use individual track segments. This repartitioning gives the system a well-layered architecture. In addition it also separates two functions that may have different performance requirements. The track permission functionality is safety-critical and should definitely have hard real-time constraints in term of response times and deadlines. The order management functionality is related to the overall usability of the system and only needs to perform within soft real-time constraints.

This evaluation could hence lead to modifications to the ASTS UCM (not shown here), such as the definition of two sub-components for RailSystem, with partitioning of the paths and responsibilities. This new version of the UCM, together with new versions of the resulting scenario files, could then be imported again into DOORS, where the requirements objects and links would be updated. Specific and appropriate performance requirements could then be created for the new sub-components.

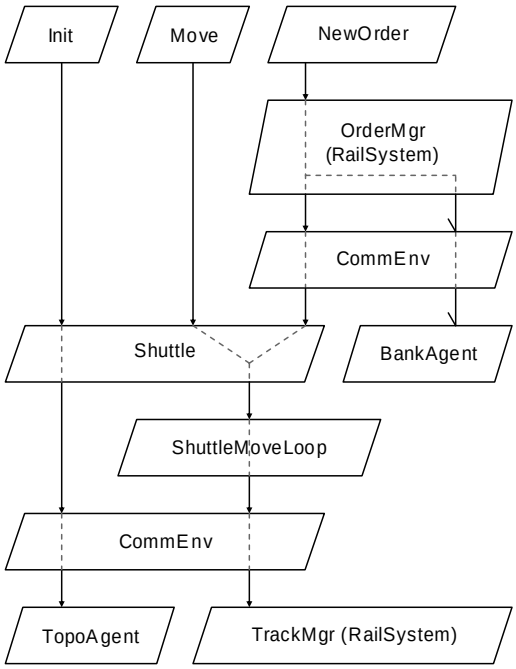

Fig. 11. Repartitioned ASTS LQN without cyclical calls

6 Conclusions

This paper has presented a framework for rapidly creating UCM scenario models from requirements documents, for rapidly refining those UCMs using the

UCMNav editor while maintaining traceability links between versions and to the original requirements through the use of a requirements management system, and for analysing the software architecture of the system based on an evaluation of the LQN performance model generated from the UCMs. The framework was illustrated using the ASTS as an example. Section 2 explained how the requirements documents were interpreted in order to create the initial ASTS UCM. Section 3 described the steps used in rapidly prototyping the resulting UCM so as to capture as much of the system behaviour as possible (at a high level of abstraction) and to incorporate additional details and thinking about the system resulting from discussions among requirements oracle participants.

Section 4 introduced a new, tool-supported iterative process for combining UCM scenarios with other types of requirements in the DOORS RMS. The UCM notation provides an appropriate means of capturing the important scenarios for a given system, of integrating them in a single model, and of linking them to external requirements and documentation. Such traceability to a scenario view can help assess the validity and the completeness of requirements. Since both scenarios and external requirements evolve over time, our tool also maintains the existing links whenever this is possible.

The novelty of the approach is also partly due to the open and flexible import interface with the RMS. Others have shown similar interests in combining graphical scenario models with an RMS. With *ScenarioPlus*, Alexander has extended DOORS to support various notations including UML 1.x Use Case diagrams and class diagrams [1]. However, the diagrams must be drawn directly within the RMS, causing substantial usability and performance problems. Earlier this year, a DOORS plug-in called *Analyst* became available [22], which supports most UML 2.0 diagrams. Analyst also uses a separate model editor and then synchronizes the updated models with the DOORS database, where links to other requirements objects are created. The number of supported modelling languages and the integration with the RMS are impressive, but this tool uses a rigid synchronization model and proprietary interfaces. The approach presented here is more open in the sense that one can freely adapt the RMS library or the UCMNav export mechanism to import exactly the information that is needed. However, we see a lot of potential in combining our tools with the Analyst as this would provide a way to connect requirements and UCM scenarios with more detailed design aspects, in UML 2.0.

Finally, Section 5 builds on previous work to show that simple analysis and evaluation of performance models generated from UCMs can influence several requirements and architectural decisions early in the development process. The detection of cyclical calling dependencies between ASTS tasks and the resulting repartitioning of the system in order to remove those cycles illustrates the value of early performance analysis even on incomplete models, as well as the value of being able to automatically generate the performance models from tools such as UCMNav.

This work has demonstrated the feasibility of the approach and has led to several additions to existing tools, especially to handle interoperability. Future

work will involve the strengthening of the current prototypes in terms of coverage of UCMs, robustness, usability, and interoperability with performance tools and with UML 2.0 tools. We also plan further validation of the approach through industrial case studies.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada, through its programs of Strategic Grants and Collaborative Research and Development Grants. We are grateful to Telelogic for making their tools available via the ASERT lab.

References

1. Alexander, I.: ScenarioPlus - Tools for Requirements Engineering. http://www.scenarioplus.org.uk
2. Amyot, D.: Introduction to the User Requirements Notation: Learning by Example. Computer Networks, 42(3), 285–301, 21 June 2003.
3. Amyot, D. and Eberlein, A.: An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development. Telecommunications Systems Journal, 24(1), 61–94, September 2003.
4. Amyot, D., Echihabi, A., He, Y.: UCMExporter: Supporting Scenario Transformations from Use Case Maps. NOuvelles TEchnologies de la RÉpartition (NOTERE'04), Saïdia, Morocco, June 2004. http://ucmexporter.sourceforge.net
5. Amyot, D. and Logrippo, L.: Use Case Maps and LOTOS for the Prototyping and Validation of a Mobile Group Call System. Computer Communication, 23(12), 1135–1157, 2001.
6. Bordeleau, F. and Buhr, R.J.A.: UCM-ROOM Modeling: From Use Case Maps to Communicating State Machines. Proc. of IEEE Engineering of Computer-Based Systems (ECBS'97), 169–179, Monterey, California, March 1997.
7. Bordeleau, F., Leue, S., and Systä, T.: Dagstuhl Seminar 03371 – Scenarios: Models, Transformations and Tools. Wadern, Germany, September 2003. http://www.dagstuhl.de/03371/
8. Breitman, K. and Leite, J.C.S.P.: Scenario Evolution: A Closer View on Relationships. Proc. of the Fourth Intl Conf. on Requirements Engineering (ICRE 2000), 95–105, Schaumburg, USA, 2000.
9. Buhr, R.J.A. and Casselman, R.S.: Use Case Maps for Object-Oriented Systems, Prentice Hall, 1996.
10. He, Y., Amyot, D., and Williams, A.W.: Synthesizing SDL from Use Case Maps: An Experiment. Reed, R., Reed, J. (Eds) 11th SDL Forum (SDL'01), Stuttgart, Germany, July 2003. Volume 2708 of Lecture Notes in Computer Science, 117–136.
11. ITU-T: Recommendation Z.150 (02/03), User Requirements Notation (URN) – Language Requirements and Framework. International Telecommunication Union, Geneva.
12. ITU-T, URN Focus Group: Draft Rec. Z.152 – UCM: Use Case Map Notation (UCM). Geneva, Switzerland, Sept. 2003. http://www.UseCaseMaps.org/urn/

13. Jarke M., Bui X.T., and Carroll J.M.: Scenario Management: An Interdisciplinary Approach. Requirements Engineering, 3(3/4), 155–173, 1998.
14. Lamsweerde A.v.: Requirements Engineering in the Year 00: A Research Perspective. Proc. of 22nd Intl Conf. on Software Engineering (ICSE), Limerick, Ireland, ACM Press, 5–19, 2000.
15. Layered Queueing Resource Page. http://www.layeredqueues.org/
16. Leite, J.C.S.P., Rossi, G., Maiorana V., Balaguer, F., Kaplan, G., Hadad, G., and Oliveros, A.: Enhancing a Requirements Baseline with Scenarios. Requirements Engineering, 2(4), 184–198, 1997.
17. Nuseibeh B. and Easterbrook S.: Requirements Engineering: A Roadmap. A. Finkelstein (Ed) The Future of Software Engineering, ICSE 2000, ACM Press, 35–46, 2000.
18. Petriu, D.B., Amyot, D., and Woodside, M.: Scenario-Based Performance Engineering with UCMNav. Reed, R., Reed, J. (Eds) 11th SDL Forum (SDL'01), Stuttgart, Germany, July 2003. Volume 2708 of Lecture Notes in Computer Science, 18–35.
19. Petriu, D.B. and Woodside, M.: Software Performance Models from System Scenarios in Use Case Maps. Proc. 12 Intl Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation (Performance TOOLS 2002), 141–158, London, April 2002.
20. Software Engineering Group: Autonomous Shuttle Transport System Case Study. University of Paderborn, Germany, January 2003. http://tele.informatik.uni-freiburg.de/dagstuhl03371/CaseStudy.html, http://www.cs.tut.fi/t̃systa/Dagstuhl03371/SWTPRA-case-study-v04b.pdf
21. Telelogic AB: DOORS/ERS. http://www.telelogic.com/products/doorsers/
22. Telelogic AB: DOORS/Analyst. http://www.telelogic.com/products/doorsers/analyst/index.cfm
23. Telelogic AB: DXL Reference Manual, 2001.
24. UCM User Group: Use Case Maps Navigator 2 (UCMNav). http://www.usecasemaps.org/tools/ucmnav/index.shtml