TOWARDS THE AUTOMATED CONVERSION OF NATURAL-LANGUAGE USE CASES TO GRAPHICAL USE CASE MAPS

Jason Kealey
SITE, University of Ottawa
800 King Edward Avenue
Ottawa, ON, Canada, K1N 6N5
email: jkealey@shade.ca

Daniel Amyot
SITE, University of Ottawa
800 King Edward Avenue
Ottawa, ON, Canada, K1N 6N5
email: damyot@site.uottawa.ca

Abstract

Use cases are popular software engineering artifacts because their simplicity facilitates the comprehension of a system by all stakeholders. However, use cases written in natural language are inherently ambiguous and do not support automated reasoning.

This article presents a new tool that implements the transformation from the textual use cases represented in UCEd, a Use Case Editor, to a graphical notation that is designed to be as simple to learn and understand as use cases, while offering enough formality to enable automated reasoning. The target language we selected is the Use Case Map (UCM) notation, part of the User Requirements Notation. We define the mapping between use case constructs and their UCM counterparts. The described translation, together with a suitable auto-layout mechanism, is implemented as a plug-in to a recent Eclipse-based UCM editor: jUCMNav.

Keywords*: Use Case, Natural Language, UCM Model, Eclipse*

1. Introduction

Software engineers often utilize use cases during requirements elicitation even though they are inherently ambiguous and do not support automated reasoning, which would allow for completeness and consistency verification of a set of use cases. However, the main problem with textual use cases is that they are frequently left untouched once software development moves on to design and implementation. Use cases integrated with software engineering artifacts later in the development process leverage the effort invested in writing them; this paper is a small step in this direction. Although use cases can be written directly in other notations, such as activity diagrams or finite state machines, many are reluctant to change their process for various reasons.

This paper builds upon previous work on semi-formal analysis of use cases [11] to visualize textual use cases once their semantics have been analyzed. The visual notation presented here, along with its accompanying tool, allows software engineers to quickly evaluate design alternatives via a high level, gray-box view of the system. This transformation does not aim to replace UML Use Case Diagrams, as their purpose is to express the relationships between the use cases, but not the use cases themselves.

The main goals of this research are to define the mappings between the various use case and Use Case Map (UCM) constructs, to identify the UCM meta-model's limitations in this respect and to prototype a conversion utility from textual use cases written in UCEd [15] to jUCMNav [9].

2. Tools and Techniques

2.1. Use Case Analysis

In order to extract semantics from use cases, one first needs to have a standard use case format. A few years ago, the use case syntax was very liberal, too liberal in fact. However, the use case writing style presented by Alistair Cockburn [3] is now one of the most commonly used formats (see example in Figure 1).

Title: Cash Withdrawal
Primary Actor: User; Participants: Bank
Precondition: ATM is ON AND ATM Display is operation menu
Postcondition: ATM is ON AND ATM Display is operation menu
1. USER selects cash withdrawal
2. ATM asks withdrawal amount
3. BEFORE 60 sec, User enters amount
4. ATM asks customer account update
5. IF USER withdrawal is ok THEN
ATM dispenses cash in the ATM cash dispenser slot
6. USER takes cash
7. ATM ejects card
8. ATM displays operation menu
2. a. AFTER 60 sec
2. a. 1. ATM displays operation menu
4. a. USER withdrawal is not ok
4. a. 1. ATM displays error message
4. a. 2. Goto Step 2.

Figure 1. Example Use Case.

Furthermore, the actual use case steps must be written in a particular format. The key concept here is to write them in very simple English for them to be easily understood by readers [14] while enabling automated reasoning [16]. The SVDPI pattern (Subject, Verb, Direct Object, Preposition, Indirect Object) allows for enough flexibility while avoiding ambiguity. The Use Case Editor tool (UCEd) [15] takes advantage of this pattern and is able to extract a domain model from a set of use cases and convert use cases into finite state machines on which simulations can be performed.

2.2. Use Case Visualization / Evaluation

The Use Case Map (UCM) notation [18] is a part of the User Requirements Notation (URN) [7] currently undergoing standardization by the ITU-T. UCMs visually illustrate scenarios cutting through a system's component structure, a high-level gray-box view of a system (see Figure 2). Because of their visual cleanliness and apparent simplicity, UCMs are quickly picked up by most stakeholders. Furthermore, being a structured notation with a specified syntax, one makes use of UCMs in various contexts. For example, UCMs have been used for performance modeling [19], test generation, scenario generation [2], and requirements engineering [1] in general.

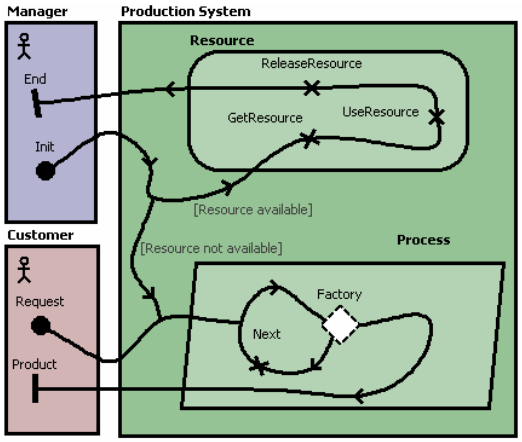

Figure 2. Example Use Case Map.

jUCMNav [9][10] is a new visual editor for the UCM notation, destined to replace UCMNav [12], the original UCM editor. This tool makes use of the Eclipse platform [6] and its plug-ins, such as the Graphical Editing Framework (GEF) [5] and the Eclipse Modeling Framework (EMF) [4]. It also supports GRL [17], the Goal-Oriented Requirements Language found in URN .

3. Transformations by Example

The following sections demonstrate the mappings by example. The UCMs shown are automatically generated by the prototype conversion utility, which uses jUCMNav's auto-layout mechanism. We have manually tweaked the final UCM layout for increased readability. UCEd's domain extractor was previously run on these use cases to extract the actors, their associated actions, and any relevant conditions.

3.1. Components and Simple Paths

Title: PaperSubmission
1. Author writes a paper
2. Conference receives submission
3. Conference Reviewer reviews the paper
4. Conference ProgramCommittee informs author of outcome
5. Author forwards response to supervisor

Figure 3. Simple use case.

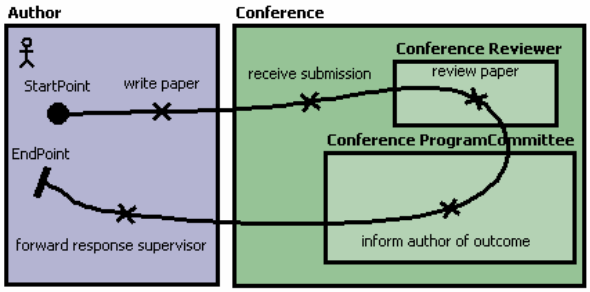

Figure 4. UCM equivalent to Figure 3.

UCEd's domain model defines the different entities that can be used in use cases. UCEd calls these *Concepts*, divided into *System Concepts* and regular concepts. The former represents the system and the latter, actors. UCEd refines these into sub-concepts, sub-components and instances. In the Use Case Map notation, these are represented as *Components* of different types. Actor components are visually distinguishable by the actor icon in their top left corner of the component reference (Figure 4). Furthermore, the component hierarchy is represented visually using containment.

A use case's main successful scenario is represented by a linear path in UCMs. Note that this path cuts through the various components to illustrate where the actions are to be performed (responsibilities in UCM language).

3.2. Alternatives

Title: PaperSubmission
1. Author writes a paper
2. Conference receives submission
3. Conference Reviewer reviews the paper
4. Conference ProgramCommittee informs author of outcome
5. Author forwards response to supervisor
2. a. Conference Reviewer is too busy
2. a. 1. Conference Reviewer delegates work
2. a. 2. Conference Reviewer confirms review
2. a. 3. GOTO 4

Figure 5. Use case with alternatives.

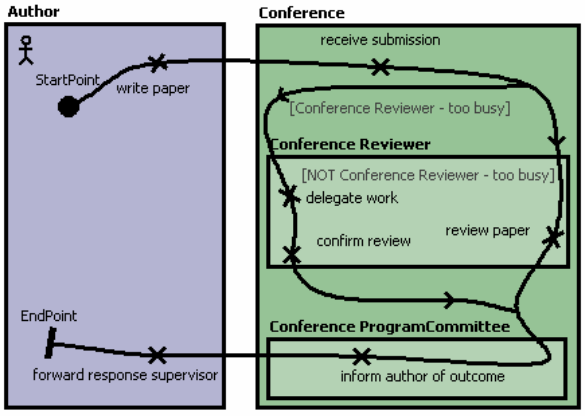

Figure 6. UCM equivalent to Figure 5.

Pre and post-conditions can be defined on use cases; their UCM counterparts are conditions that accompany the start and end point of the main path (not shown here). UCMs use *Or-*

Forks to represent alternatives and *Or-Joins* to merge paths together again (Figure 6). Alternatives to the same step are represented as branches of an Or-Fork, each with appropriate conditions.

3.3. Inclusions and Extensions

Most notations allow some sort of modularity and encapsulation. In use cases, one uses the inclusion relationship to refer to details presented in another use case (Figures 7 and 8). This allows for better readability and use case re-use. The same concept is available in UCMs via *stubs* and *plug-in maps*. Here, the diamond represents a *Static Stub*, linked to one and only one plug-in map. Via *plug-in bindings*, scenarios follow the UCM path into the plug-in and back (Figures 10 and 11).

Title: PaperSubmission
1. Author writes a paper
2. Conference receives submission
3. INCLUDE PaperReview
4. Conference ProgramCommittee informs author of outcome
5. Author forwards response to supervisor
ExtensionPoint==> response reception

Figure 7. A use case with inclusion and an extension point.

Title: PaperReview
1. Conference Reviewer receives paper
2. Conference Reviewer reviews the paper
3. Conference Reviewer sends in evaluation
1. a. Conference Reviewer is too busy
1. a. 1. Conference Reviewer delegates work
1. a. 2. Conference Reviewer confirms review
1. a. 3. GOTO 3

Figure 8. The included use case.

Title: PaperReception
PART 1. At Extension Point response reception
1. Author updates publication list

Figure 9. An extension use case.

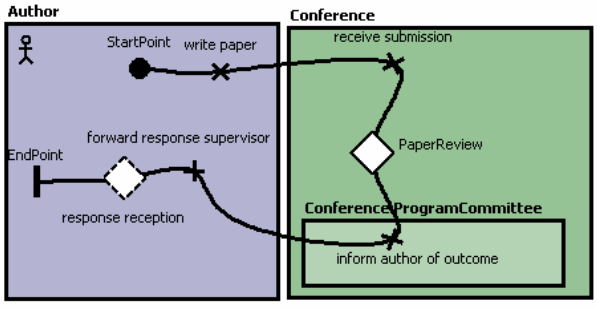

Figure 10. UCM equivalent to Figure 7.

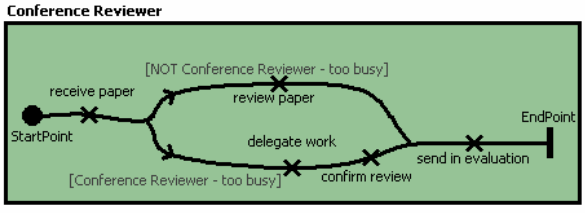

Figure 11. UCM equivalent to Figure 8.

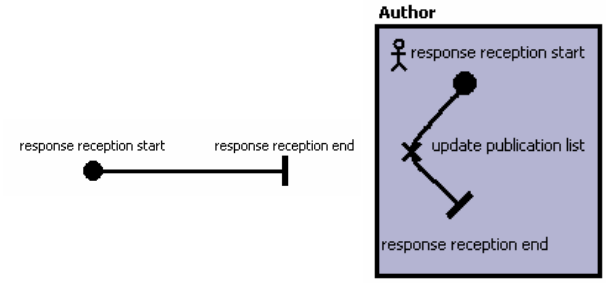

Figure 11. Default UCM for extension point in Figure 7 (left) and UCM equivalent to Figure 9 (right).

Use cases also feature extensions that refine other use cases at specific locations called extension points (Figure 9). UCMs, for their part, offer *Dynamic Stubs* that contain references to one or more plug-ins (Figure 11). A selection policy is defined to determine, at runtime, in which plug-in the path should continue. The current UCM notation and the dynamic stub selection policy revolve around the concept that one and only one plug-in map should be chosen at runtime. However, the use case model could offer multiple extensions to the same extension point. Nothing determines if these extensions should be run concurrently, sequentially, or a mix of both.

UCMs can address this by adding an intermediate plug-in map that combines in a particular way the sub-maps that correspond to the extensions. However, we chose not to implement this solution in our prototype because we wish to address common patterns such as parallel or sequential execution directly in the dynamic stub in our future work. The next section presents other limitations.

4. Current Limitations

The UCM notation is not fully equivalent to the use case notation. Both present features that are not available in the other. Researching the various mappings makes this impedance mismatch more apparent.

As mentioned previously, UCM dynamic stubs should be enhanced to allow more than a simple mutual exclusion selection policy. Passing control to multiple plug-ins either concurrently or sequentially should be made easier. For more complex solutions, expansion capabilities should be added to jUCMNav to give the designer full control over the model. These capacities are important in the context of performance analysis and feature interaction review. Also concerning concurrency, UCMs allow for explicit concurrent execution, via *And-Forks* and *And-Joins*, whereas use cases only offer implicit concurrency via use case preconditions. Although use case enhancements have been proposed, they are not widely used.

The UCMNav editor supports scenarios which operate on Boolean variables. The definition and execution of scenarios will soon be ported to jUCMNav, completely deprecating its ten-year-old ancestor. Once this is done, jUCMNav's scenarios will complement UCEd's FSM-based simulations. It will also allow for completeness and consistency verification thanks to the integration of first order logic manipulations.

UCEd supports time-based use cases and integrates this information in its FSMs. Although not presented here, some work has been performed by mapping these timing constraints to UCM *Timers*. However, the lack of semantics on UCM conditions in jUCMNav leaves the conversion in the embryonic stage, until scenarios are migrated into jUCMNav.

One use case feature that is required in UCMs is the addition of an equivalent to "any alternatives", alternatives that can be executed at any time during the use case, given the defined precondition is met. In UCMs, this is equivalent to the concept of exceptions, which is not currently well supported.

What does jUCMNav offer that is not present in textual use cases other than visualization? We mentioned concurrency issues previously, but it is also important to mention that mappings have been defined from UCMs to, among others, the Core Scenario Model (CSM) for performance analysis [19] and Telelogic DOORS for requirements management and traceability analysis [13]. More recently, jUCMNav's support of GRL opens the door for comparative evaluation of architectures and functional scenarios via non-functional goals and strategies.

5. Future Work

As our interest lies in round-trip requirements engineering, we are interested in researching the inverse transformation. The generation of textual use cases from UCMs or, ideally, the synchronization between the two views, would free the software engineer from having to use a particular process and allow him to use the model best suited for the task. Transforming complex UCMs into use cases would require a decomposition mechanism which takes into consideration the defined scenarios. Research has already been done in this area with UCM Requirement Slicing [7]. Furthermore, in order to obtain better synchronization, UCEd's needs to limit the user input required during the domain extraction process.

In addition, an automated UCM synthesis mechanism could help uncover feature interactions earlier in the development process. Moreover, this synthesis mechanism would be a useful asset for the current research being done on aspect-oriented UCM patterns.

6. Conclusions

We have defined and prototyped in jUCMNAv the mapping from textual natural language use cases in UCEd to the Use Case Map notation. Our research discovered some limitations in the UCM notation and gave us insight on the issues to be resolved while progressing towards round-trip requirements engineering with regards to textual use cases and UCMs.

Acknowledgements

This research was supported by NSERC, through its Canada Graduate Scholarship program. We are grateful to G. Mussbacher, J.-F. Roy, and S. Somé for their various contributions throughout the project.

References

[1] D. Amyot and G. Mussbacher, "URN: Towards a New Standard for the Visual Description of Requirements". *Telecommunications and beyond: The Broader Applicability of SDL and MSC (SAM 2002)*. LNCS 2599, Springer 2003, 21-37.

[2] 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.

[3] A. Cockburn, *Writing Effective Use Cases*, Addison-Wesley Professional, USA, January 2000, 270p.

[4] Eclipse, *Eclipse Modeling Framework (EMF)*, http://www.eclipse.org/emf/

[5] Eclipse, *Graphical Editing Framework (GEF)*, http://www.eclipse.org/gmf/

[6] Eclipse, http://www.eclipse.org/

[7] J. Hassine, R. Dssouli, and J. Rilling, "Applying Reduction Techniques to Software Functional Requirement Specifications", *System Analysis and Modeling (SAM 2004)*. LNCS 3319, Springer 2005, p. 138.

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

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

[10] 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.

[11] V. Mencl, "Deriving Behavior Specifications from Textual Use Cases", *Proceedings of Workshop on Intelligent Technologies for Software Engineering (WITSE04)*, Linz, Austria, September 2005, 331-341.

[12] A. 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] G. Mussbacher, B. Jiang, D. Amyot, and M. Woodside, "Importing and Updating of Scenario Models in DOORS", *Telelogic User Group Conference*, Hollywood, USA, October 2005.

[14] D. Richards, K. Boettger, and O. Aguilera, "A Controlled Language to Assist Conversion of Use Case Descriptions into Concept Lattices", *Proceedings of AI 2002*, Canberra, Australia, Dec. 2-6, 2002. LNCS 2557, Springer, 2002.

[15] S. Somé, *Use Cases based requirements engineering with UCEd, UCEd User Guide*. http://sourceforge.net/projects/uced/

[16] S. Somé, "Beyond Scenarios: Generating State Models from Use Cases", *ICSE 2002 Workshop Scenarios and state machines: models, algorithms, and tools*, May 2002.

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

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

[19] Y. X. Zeng, *Transforming Use Case Maps to the Core Scenario Model Representation*, M.Sc. thesis, SITE, University of Ottawa, June 2005.