

# A Novel Formalization Process for Use Case Maps

Yahia Menassel<sup>1</sup> and Farid Mokhati<sup>2</sup>

<sup>1</sup>*Department of Mathematics and Computer Science, University of Tébessa, Algiers, Algeria*

<sup>2</sup>*Department of Mathematics and Computer Science, University of Oum El Bouaghi, Oum El Bouaghi, Algeria*

**Keywords:** UCM, Maude Strategy Language, Formal Specification, Translation.

**Abstract:** This paper presents a novel process for formalizing UCM notation as an executable formal specification described in the Maude language Strategy, a recent extension of Maude. The main motivation of our work is essentially to provide a sound and rigorous description of complex systems described by UCM, which can help analysts, designers and developers, to automating their verification and validation processes and to assuring their quality.

## 1 INTRODUCTION

Use Case Maps (UCM) (Buhr and Casselman, 1995) is an integral part of User Requirements Notation (URN) (ITU-T, 2003); (Amyot, 2003) standards proposed to the International Telecommunication Union-Telecommunication (ITU-T) for describing functional requirements as causal scenarios. UCM allows developers to model dynamic behavior of systems where scenarios and structures may change at run-time. For these reasons, UCM have successfully been used for service-oriented, concurrent, distributed, and reactive systems (Mussbacher, 2007).

In recent years, some works are proposed in order to deal with UCM formalization, validation and testing, we cite among others (He et al., 2003); (Hassine et al., 2005). Those works have considerably forwarded the domain by proposing novel strategies for improving complex systems development process based on UCM notation. Although the formalization of UCM notation is not new problem, we present in this paper, a novel process for formalizing UCM notation as an executable formal specification described in Maude Strategy language (Marti-Oliet et al., 2009), a recent extension of Maude. The main motivation of our work is essentially to provide a sound and rigorous description of complex systems described by UCM, which can help analysts, designers and developers, to automating their validation and verification processes and to assuring their quality.

The proposed formalization process consists in a

preliminary specification phase of transformation of UCM modeling elements into Maude-Strategy.

The remainder of this paper is organized as follows. In Section 2 we give a summary of UCM notation. Section 3 presents a brief introduction to Maude Strategy language. In section 4 we present the translation process we propose. We discuss in section 5, our actual research, draw some conclusions and give some future work directions.

## 2 OVERVIEW OF USE CASE MAPS

Use Case Maps (UCM) is a semi-formal notation for describing scenarios of a system. This notation is a high level scenario based modeling technique that can be used to specify functional requirements and high-level designs for wide range of systems (Liu and Yu, 2001).

UCM expressed by a simple visual notation allow for an abstract description of scenarios in terms of causal relationships between responsibilities along paths allocated to a set of structural elements (components) (Mussbacher, 2007).

UCM consists of one or more paths describing the causal flow of behavior of a system. Furthermore, behavioral aspects (scenario paths) are superimposed over components which represent the architectural structure of a system. UCM abstract from the details of message exchange and communication infrastructures while still showing

the interaction between architectural entities (Mussbacher, 2007). UCM are integrated with goal models described with the Goal-oriented Requirement Language (GRL), allowing for the seamless capture of stakeholders' goals, rationale, and alternative solutions to achieve such goals. The solutions are reasoned about with GRL and their behavior and structure described in more detail with UCM (Mussbacher and Amyot, 2008).

### 3 MAUDE STRATEGY LANGUAGE

Maude strategy language (Marti-Oliet et al., 2009), a recent extension of Maude, introduces some regular expression combinators: concatenation (;), union (|), and iteration (E\* for zero or more iterations and E+ for one or more iterations). Additionally, there is the combinator *orelse* is a typical if-then-else. These combinators can be used to control how rules are applied to rewrite a term in an attempt to control the non-determinism in the execution process. For more detail see (Marti-Oliet et al., 2009).

### 4 TRANSLATING UCM TO MAUDE-STRATEGY

In this section, we present our translation process in order to give a formal semantics of UCM notation using Maude's strategy language.

Table 1 presents the description of each element of the UCM notation and the corresponding formal semantics.

### 5 CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, a novel formalization process is proposed for formalizing UCM element modeling in the Maude strategy language. The proposed process consists in a preliminary specification phase of transformation of UCM modeling elements into Maude-Strategy.

Currently we are developing a formal framework which is based on transformation rules presented in Table 1 for translating complex systems functional requirements described by UCM into a formal specification written in the Maude strategy language. The Maude language is supported by a tool, which

will allow us to validate the generated code by simulation.

## REFERENCES

- Amyot, D., (2003) 'Introduction to the User Requirements Notation: Learning by Example', *Computer Networks*, Vol. 42(3), pp. 285-301.
- Buhr, R. J. A. and Casselman, R. S. (1995) '*Use Case Maps for Object-Oriented Systems*', Prentice-Hall, USA. [http://www.UseCaseMaps.org/UseCaseMaps/pub/UCM\\_book95.pdf](http://www.UseCaseMaps.org/UseCaseMaps/pub/UCM_book95.pdf).
- Hassine, J., Rilling, J. and Dssouli, R., (2005) 'An ASM Operational Semantics for Use Case Maps', In *RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05)*, IEEE Computer Society, Paris, p. 467-468.
- He, Y., Amyot, D., and Williams, A. W. (2003) 'Synthesizing SDL from Use Case Maps: An experiment', In *Proc SDL 2003: System Design, 11th International SDL Forum*, Stuttgart, Germany, p. 117-136.
- ITU-T, (2003) 'User Requirements Notation (URN) – Language Requirements and Framework', *ITU-T Recommendation Z.150*. Geneva, Switzerland.
- Liu L., and Yu E., (2001) 'From Requirements to Architectural Design - Using Goals and Scenarios', From *Software Requirements to Architectures Workshop (STRAW 2001)*.
- Mussbacher G., (2007) 'Evolving Use Case Maps as a Scenario and Workflow Description Language', *10th Workshop on Requirements Engineering (WER'07)*, Toronto, Canada, p. 56-67.
- Mussbacher G., and Amyot D., (2008) 'Assessing the Applicability of Use Case Maps for Business Process and Workflow Description', *3rd Int. MCETECH Conference on e-Technologies*, p. 219-222.
- Marti-Oliet N., Meseguer J., Verdejo A., (2009) 'A Rewriting Semantics for Maude Strategies', *Electronic Notes in Theoretical Computer Science*, vol. 238(3), pp. 227-247.

Table 1: Mapping from UCM specifications to Maude strategy.



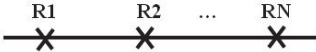

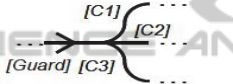




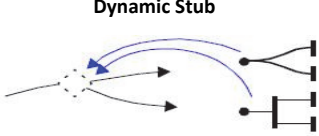

Elements of the UCM notation	Elements of the UCM description	Maude Strategy Language
<p><b>Start Point</b></p> 	<p>Begin of the path. Formally, a start point is specified in terms of:</p> <ul style="list-style-type: none"> <li>• <i>Preconditions</i> that need to be satisfied to enable the triggering of the path, and/or</li> <li>• A set of possible <i>triggering events</i>.</li> </ul>	<p>CRL [Start] :</p> $Ed_1 \dots Ed_n \text{ Conf}_1 \Rightarrow \text{Conf}_2 \text{ if } \text{Prec}_1 \dots \text{Prec}_n .$
<p><b>Responsibility</b></p>  <p><b>Sequence of responsibilities</b></p> 	<p>A path describes a sequence of responsibilities that need to be executed by system components in response to a given triggering event. Formally, a responsibility is defined in terms of:</p> <ul style="list-style-type: none"> <li>• A <i>responsibility label</i>, which gives a textual description of the responsibility, and</li> <li>• A <i>responsibility identifier</i>, which allows uniquely identifying a responsibility in a UCM model.</li> </ul>	<p>RL [RespID] : Conf<sub>1</sub> =&gt; Conf<sub>2</sub> .</p> <p>strat sequence : @ Configuration .</p> <p>sd sequence := RL<sub>1</sub> ; RL<sub>2</sub> ; ... ; RL<sub>N</sub> .</p>
<p><b>End Point</b></p> 	<p>End of the path. Formally, an end bar is specified in terms of:</p> <ul style="list-style-type: none"> <li>• <i>Postconditions</i> that must hold after the execution of the path, and</li> <li>• A set of possible <i>resulting events</i>.</li> </ul>	<p>CRL [End] :</p> $\text{Conf}_1 \Rightarrow \text{Conf}_2 \text{ Er}_1 \dots \text{Er}_n \text{ if } \text{Pstc}_1 \dots \text{Pstc}_n .$
<p><b>OR-fork</b></p> 	<p>Is used to show a point along a path where alternative branches may be followed. Each branch is associated with a distinct path segment.</p>	<p>strat OR<sub>Fork</sub> : @ Configuration .</p> <p>sd OR<sub>Fork</sub> :=</p> $RL_1 ; ( CRL_2 \text{ orelse } CRL_3 \dots \text{ orelse } RL_N ) .$
<p><b>OR-join</b></p> 	<p>Captures the merge of two or more incoming path segments into a single one without requiring any synchronization or interaction between the incoming path segments.</p>	<p>strat OR<sub>Join</sub> : @ Configuration .</p> <p>sd OR<sub>Join</sub> :=</p> $( CRL_1 \text{ orelse } CRL_2 \dots \text{ orelse } RL_N ) ; RL_{N+1} .$
<p><b>AND-fork</b></p> 	<p>Is used to illustrate a point along a path where a single path segment forks into two or more concurrent path segments.</p>	<p>strat AND<sub>Fork</sub> : @ Configuration .</p> <p>sd AND<sub>Fork</sub> := RL<sub>1</sub> ; ( RL<sub>2</sub>   ...   RL<sub>N</sub> ) .</p>
<p><b>AND-join</b></p> 	<p>Is used to illustrate a point along a path where several concurrent path segments synchronize together and result a single path segment.</p>	<p>strat AND<sub>Join</sub> : @ Configuration .</p> <p>sd AND<sub>Join</sub> := ( RL<sub>1</sub>   ...   RL<sub>N</sub> ) ; RL<sub>N+1</sub> .</p>
<p><b>Static Stub</b></p> 	<p><i>Stub</i> is a mechanism for path abstraction, hence enabling hierarchical decomposition of complex maps.</p> <p><i>Static stub</i> can contain only one plug-in.</p>	<p>(smod Static-Stub-NAME is</p> <p>strats OR<sub>Fork</sub> Plug-in Static-Stub : @ Configuration .</p> <p>Plug-in := OR<sub>Fork</sub> .</p> <p>Static-Stub := Plug-in .</p> <p>endsm)</p>
<p><b>Dynamic Stub</b></p> 	<p>May contain several plug-ins, whose selection can be determined at run-time according to a <i>selection policy</i> (often described with pre-conditions).</p>	<p>(smod Dynamic-Stub-NAME is</p> <p>strats OR<sub>Fork</sub> AND<sub>Fork</sub> : @ Configuration .</p> <p>strats Plug-in1 Plug-in2 Static-Stub : @ Configuration .</p> <p>Plug-in1 := OR<sub>Fork</sub> .</p> <p>Plug-in2 := AND<sub>Fork</sub> .</p> <p>Dynamic-Stub := Plug-in1 orelse Plug-in2 .</p> <p>endsm)</p>
<p><b>Component</b></p> 	<p>A component is an object associated to a part of system. The different tasks located inside a component are performed only by this component.</p>	<p>(omod Composant-NAME is</p> <p>RL<sub>1</sub> .</p> <p>...</p> <p>RL<sub>N</sub> .</p> <p>endom)</p>

Table 1: Mapping from UCM specifications to Maude strategy (cont.).

Elements of the UCM notation	Elements of the UCM description	Maude Strategy Language
<p><b>Waiting Place</b></p>	<p>A <i>Regular waiting place</i> identifies a point along the path at which the progression of a path is blocked until a predefined unblocking (or triggering) event occurs.</p>	<pre>start Wait : @ Configuration . sd Wait := ( WP   TP ) ; WaitPlace : CP .</pre>
<p><b>Special cases</b></p>	<p><b>Special cases description</b></p>	<p><b>Maude Strategy Language</b></p>
<p><b>Generic Version of AND-Join/Fork</b></p>	<p>Can be used to illustrate cases where an arbitrary number, say N, of incoming path segments need to synchronize together to trigger the execution of an arbitrary number, say M, of outgoing path segments.</p>	<pre>start VG : @ Configuration . sd VG := (RL1   RL2   ...   RLN) ; (RL'1   RL'2   ...   RL'M) .</pre>
<p><b>AND-Fork/Join</b></p>	<p>AND-Fork with AND-Join.</p>	<pre>start ANDForkJoin : @ Configuration . sd ANDForkJoin := ANDFork ; RL_{N+1} .</pre>
<p><b>OR-Join/fork</b></p>	<p>OR-Join with OR-Fork.</p>	<pre>start ORJoinFork : @ Configuration . sd ORJoinFork := ORJoin : ( CRL1 orelse CRL2 ... orelse RL_M ) .</pre>
<p><b>OR-Fork/join</b></p>	<p>OR-Fork with OR-Join.</p>	<pre>start ORForkJoin : @ Configuration . sd ORForkJoin := ORFork ; RL_{N+1} .</pre>
<p><b>Loop</b></p>	<p>Loop can be modeled implicitly with OR-fork and OR-join.</p>	<pre>start Loop : @ Configuration . sd Loop := (CRL1)+ orelse CRL2 ... orelse RL_N .</pre>
<p><b>UCM interactions</b></p>	<p><b>Description</b></p>	<p><b>Maude Strategy Language</b></p>
<p><b>Trigger after completing path execution</b></p>	<p>is used to illustrate cases where the completion of the execution of a path triggers another path that is waiting on a waiting place. The waiting can either be a start point or a waiting place along a path.</p>	<pre>start TAE : @ Configuration . sd TAE := RL1 ; RL2 .</pre> <p>Such as:  <math>RL_1 : Conf_1 \Rightarrow Conf_2 . RL_2 : Conf_3 \Rightarrow Conf_4 .</math>  and <math>Conf_2 \cap Conf_3 \neq \emptyset</math></p>
<p><b>Trigger In Passing</b></p>	<p>is used to illustrate cases where a waiting place, positioned either at the beginning (start point) or along a path, is triggered by another path in an asynchronous manner.</p>	<pre>start TIP : @ Configuration . sd TIP := ANDFork .</pre>
<p><b>Rendezvous</b></p>	<p>This type of interactions is used to illustrate cases where two, or more, paths synchronize together to execute a certain path segment (sequence of responsibilities) before returning to the execution of their own respective path.</p>	<pre>start RendezVous : @ Configuration . sd RendezVous := ANDJoin ; (RL1   ...   RL_N) .</pre>
<p><b>Synchronize</b></p>	<p>This type of interactions is used to illustrate cases where two, or more, paths synchronize together and then return to the execution of their own respective path.</p>	<pre>start Synchronize : @ Configuration . sd Synchronize := (RL1   ...   RL_N) ; (RL1   ...   RL_N) .</pre>
<p><b>Abort</b></p>	<p>Is used to illustrate cases where the execution of a path interrupts the execution of another. Top path aborts bottom path after R1.</p>	<pre>start Abort : @ Configuration . sd Abort := (RL1   RL2) .</pre> <p>Such as:  <math>RL_1 : Conf_1 \Rightarrow Conf_2 . RL_2 : Conf_3 \Rightarrow Conf_4 .</math>  and <math>Conf_1 \cap Conf_3 \neq \emptyset</math></p>