Zen-RUCM: A Tool for Supporting a Comprehensive and Extensible Use Case Modeling Framework

Gong Zhang[1], Tao Yue[2], Ji Wu[1] and Shaukat Ali[2]

[1] School of Computer Science and Engineering, Beihang University, Beijing, China
zhanggong@sei.buaa.edu.cn, wuji@buaa.edu.cn
[2] Simula Research Laboratory & University of Oslo, P.O. Box 134, Lysaker, Norway
{tao, shaukat}@simula.no

Abstract. The Restricted Use Case Modeling (RUCM) approach is composed of a set of well-defined restriction rules and a new template, aiming to reduce ambiguity and facilitate automated analysis. Zen-RUCM is an RUCM-based framework to tackle the challenges of requirement specification in different application domains (e.g., real-time systems) and from various requirement specification concerns (e.g., variability). In this demonstration, we discuss an implementation of the Zen-RUCM framework with the focus on its lightweight design architecture and extension mechanism. (Demonstration video link: http://youtu.be/a8YZ_wuVxQg)

1 Motivation and Overview

Requirements state the necessary attributes, capabilities, characteristics, or qualities of a system in order for it to have value and utility to a user, and thus play a central role in the communications between different stakeholders in the system engineering value chain. More precise, consistent, and complete requirements can significantly improve the quality of the system being developed. Vice versa, vague, inconsistent, and inadequate requirements can result in significant consequences including system failures, excessive maintenance costs, and future loss of credibility and business.

There is a wide range of techniques that one may use for requirements specification from informal to formal [1]. Informal specifications in unstructured and unrestricted Natural Language (NL) tend to be easier to understand by many stakeholders and no special training is required. However, requirements in unstructured and unrestricted NL are often ambiguous and therefore different stakeholders may interpret the same requirement in different ways, in turn affecting the quality of subsequent system development activities such as design and testing. In addition, the absence of formalization precludes any form of automated analysis. In contrast, formal specification languages (which are grounded on formal logic) have higher precision than informal specifications. Sophisticated forms of validation and verification can then be built on top of such specifications and automated by tools. However, formal specification languages have limited expressiveness, are hard to write and read without extensive training, and thus may be very difficult, if not impossible, to communicate to end users and domain experts. Disciplined specifications in structured and restricted NL strike a fine balance between informal and formal specifications. They capture requirements in a more precise way than informal textual specifications and thus enable automated processing by tools.

Use case modeling is one of the most promising and widely-used, disciplined specification techniques in structured NL. By combining diagrammatic and textual descriptions, use case models offer a very intuitive and yet precise foundation for requirements specification. Tao et al. [4, 5] have devised a Restricted Use Case Modelling approach, RUCM, which is composed of UML use case diagrams, a set of well-defined restriction rules and a use case template. The goal is to reduce ambiguity and facilitate the automated analysis of use case models.

As shown in Fig. 1, built on the top of RUCM, Zen-RUCM aims to tackle the challenges of requirement specification in different application domains (e.g., real-time systems, distributed systems, communication systems) and from various requirement specification concerns (e.g., variability, Non-Functional Requirements (NFR), crosscutting concerns). RUCM is a generic framework and has not been tailored for use in any particular domain. Specifically, the use case template in RUCM captures only the generic aspects of use cases. They are not specific to a particular problem or application domain. We believe, and as strongly suggested by our previous studies [3, 5-7], that RUCM has substantial room for improvement by making use of domain-specific abstractions. Introducing these abstractions is expected to bring several major benefits, including more succinct use case descriptions, less ambiguity, and more precision in automated analysis. In addition, RUCM does not provide solutions for commonly arising requirement specification concerns like NFR.

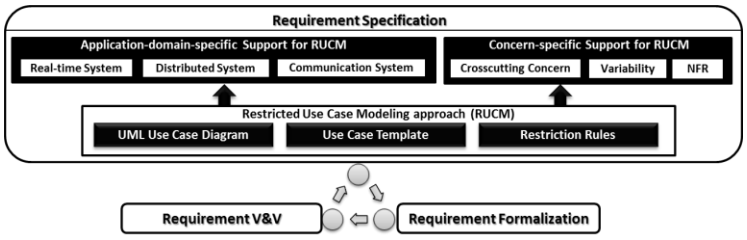

Fig. 1. Overview of the Zen-RUCM Framework

As the first step towards the full realization of the Zen-RUCM framework, in this demonstration, we presents the Zen-RUCM tool, which includes an RUCM modeling editor and mostly important has an extensible, lightweight architecture design to facilitate future extensions of RUCM for e.g., having crosscutting concerns and variability modeling capabilities as shown in Fig. 1.

2 Background: RUCM and UCMeta

RUCM encompasses a use case template and 26 well-defined restriction rules [5]. Rules are classified into two groups: restrictions on the use of NL, and rules enforcing the use of specific keywords for specifying control structures. The goal of RUCM is to reduce ambiguity and facilitate automated analysis, which have been empirically evaluated [4, 5] and positive results were obtained.

A RUCM use case specification has one basic flow and can have one or more alternative flows. An alternative flow always depends on a condition occurring in a specific step in a flow of reference, referred to as reference flow, which is either the

basic flow or an alternative flow itself. We classify alternative flows into three types: A specific alternative flow refers to a specific step in the reference flow; A bounded alternative flow refers to more than one step (consecutive or not) in the reference flow; A global alternative flow refers to any step in the reference flow. The 26 restriction rules of RUCM are classified into two categories: restrictions on the use of natural language and rules enforcing the use of keywords for specifying control structures. Eleven restriction rules are to reduce ambiguity in use case specifications and 15 rules defines a set of keywords to specify concurrency sentences (MEANWHILE), condition checking sentences (VALIDATES THAT), etc.

UCMeta is the intermediate model in aToucan [7], used to bridge the gap between a textual use case model and a UML model including class, sequence, activity, and state machine diagrams. It can be also used as a formal representation of textual RUCM models. UCMeta is hierarchical and contains five packages: UML::UseCases, UCSTemplate, SentencePatterns, SentenceSemantics, and SentenceStructure. UML::UseCases is a package of UML 2 superstructure [2], which defines the key concepts used for modeling use cases such as actors and use cases. Package UCSTemplate models the concepts of the use case template of RUCM. SentencePatterns describes different types of sentence patterns. SentenceSemantics is a package modeling the classification of sentences from the aspect of their semantic functions. Package SentenceStructure takes care of NL concepts in sentences such as subject or noun phrase. The detailed description of UCMeta is given in [7].

3 The Zen-RUCM Tool

3.1 Architecture

The architecture of the Zen-RUCM tool is provided in **Fig. 2**. Its components and their relationships are illustrated in the left side of the figure. As part of the solution, we proposed our own modeling framework (i.e., LMF), which implements similar kinds of functionalities that Eclipse EMF has, but with a lightweight design. The main objective of designing such a framework is that it is easier for a small development team (with few developers like the current setting we have) to transplant the tool to different platforms. Though EMF can also be used, it is impractical for a small team, as EMF is huge to compare with LMF. Note that my implementation of LMF only has around 5000 lines of code, which is significantly smaller than EMF and therefore easy to maintain and extend. With such a design, Zen-RUCM can then be easily deployed to different platforms such as Java Web Applications and C++. As shown in Fig. 2, the LMF architecture is similar to EMF in the sense that they both have components such as Generators and Editors to facilitate the development of metamodel-based modeling environment; however the main difference is in terms of the degree of coupling with the EMF architecture and Eclipse platform.

Both LMF and EMF have two editors: reflective model editor and metamodel editor. LMF Reflective Editor is a simple editor that can automatically adapt metamodel changes. It is based on the LMF metamodel reflection mechanism. When a user registers a domain-specific metamodel extension (e.g., Real-time) to the framework, the reflective editor is instantly ready for editing model instances that conform to the newly registered metamodel. The editor presents the model instances in a tree structure. A user can add or delete elements on that tree with pop-up context

menu. With a property view, a user can edit the attribute values for a selected node. New user interface items for the metamodel extension will automatically appear in context menu and property view without introducing new implementation. This is useful for domain experts who don't want to customize the implemented RUCM editor via directly coding and it can be used to check if a metamodel extension is correctly defined.

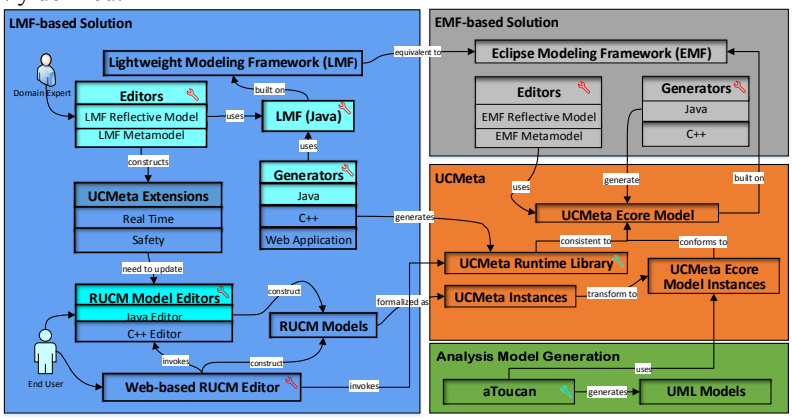

Fig. 2. Architecture of the Zen-RUCM tool

One of the key features of Zen-RUCM is to support extensions to the generic RUCM methodology. As we discussed in Section 1, there is a need to extend the RUCM template and restrictions for various purposes, such as capturing domain-specific information and supporting specific concerns (e.g., variability modeling). Zen-RUCM is developed to account for this need by carefully designing the LMF metamodel editor, which allows users to implement UCMeta extensions easily. With this editor, users can create new packages, new meta classes and enumerations and it is also possible to append new attributes to existing meta classes. Notice that these elements are reflected as keywords, restrictions and fields of the extended RUCM specification methodology. The editor can also automatically generate Java code for the newly introduced metamodel or extension. See Section 3.2 for details.

RUCM models are formalized as UCMeta Instances as shown in **Fig. 2**. These UCMeta instances can be automatically transformed into UCMeta Ecore Model Instances, which are consumed by aToucan as input to automatically generate UML analysis models such as UML class, sequence, activity, and state machine diagrams. It is important to bridge the gap between LMF and EMF to integrate the Zen-RUCM tool with EMF-based applications such as aToucan [7].

End users can rely on the provided RUCM Model Editors to construct requirements as RUCM models. Equivalently a user can also construct RUCM models using the web-based RUCM editor. All the RUCM editors have two parts: a use case diagram editor and a use case specification editor. It is worth mentioning that to ease the adoption of Zen-RUCM in practice, the use case specification editor is designed to look like a table in Word, with some features such as automatically highlighting RUCM keywords and indenting for nested sentence structures when using keywords such as IF-THEN-ELSE-ENDIF and syntax checking of keywords.

The current implementation of the Zen-RUCM tool is highlighted using the light

green color in the LMF-based solution. UCMeta extensions are ongoing projects and Web-based RUCM Editor is currently under development and will be made available online soon. All the UCMeta artifacts are implemented and the analysis model generation part is fully implemented as well.

3.2 LMF Extension Mechanism

The LMF extension mechanism is the most important feature of the Zen-RUCM tool. As shown in **Fig. 3**, the LMF metamodel editor is used by domain experts to construct new metamodels. Extension points are implemented in the metamodel editor to facilitate the addition of new classes and properties, deletion and modification of existing classes and properties. Extension points are also implemented in the RUCM model editor to facilitate the introduction of graphical notations in use case diagrams and new fields to the RUCM use case template. Based on the extension mechanism implemented both in the metamodel and model levels, the generic RUCM model editor and the corresponding UCMeta can be easily extended, without requiring much modifications and coding. This feature nicely enables the lightweight implementation of the Zen-RUCM framework described in **Fig. 1**.

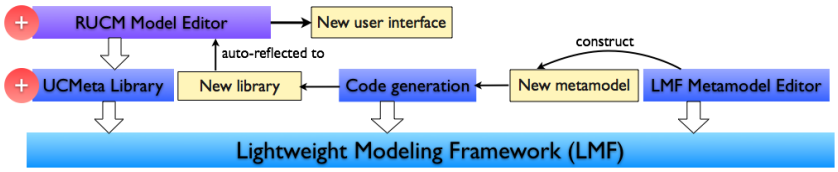

Fig. 3. LMF Extension Mechanism

4 Conclusion

We presented a tool called Zen-RUCM, which provides extensible mechanisms for specifying use case models for various domains (e.g., Real Time) and facilitates easier generation of editors in different platforms (e.g., Java Web Applications).

5 References

[1] Lamsweerde A. V., Requirements Engineering From System Goals to UML Models to Software specifications, Wiley, John&Sons, Incorporated, 2009.
[2] OMG, "UML 2.2 Superstructure Specification (formal/2009-02-04)."
[3] Yue T., Ali S. and Briand L., "Automated Transition from Use Cases to UML State Machines to Support State-based Testing," Proc. 7th ECMFA, 2011.
[4] Yue T., Briand L. and Labiche Y., "Facilitating the Transition from Use Case Models to Analysis Models: Approach and Experiments, TOSEM 22(1), 2013.
[5] Yue T., Briand L. C. and Labiche Y., "A Use Case Modeling Approach to Facilitate the Transition Towards Analysis Models: Concepts and Empirical Evaluation," Proc. MODELS2009, 5795/2009, pp. 484-498, 2009.
[6] Yue T., Briand L. C. and Labiche Y., "An Automated Approach to Transform Use Cases into Activity Diagrams," Proc. 6th ECMFA, LNCS 6138, pp. 337-353, 2010.
[7] Yue T., Briand L. C. and Labiche Y., "Automatically Deriving a UML Analysis Model from a Use Case Model," Simula Research Laboratory, Technical Report 2010-15, 2010.