Thesis Title

Cheuk Chuen Siow
School of Computer Science
McGill University, Montréal

November 2017

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Abstract

Abrégé

Acknowledgements

Preface

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since 1960s, software development has been evolving rapidly to address the increasing demands of complex software. The complexity of modern software brings about difficulties in developing and maintaining quality software. Software engineering as a discipline ensures that developers follow a systematic production of software, by applying best practices to maximize quality of deliverables and minimize time-to-market. Various methodologies exist through the efforts of active research by theorists and practitioners, but the core of software development process typically consists of the following six phases—requirements gathering, design, implementation, testing, deployment, and maintenance.

Conceptual models help illustrates complex systems with a simple framework by creating abstractions to alleviate the amount of complexity. Hence, the use of models is progressively recommended in representing a software system. This simplifies the process of design, maximizes compatibility between different platforms, and promotes communication among stakeholders. Model-Driven Engineering (MDE) technologies offer the means to represent domain-specific knowledge within models, allowing modelers to express domain concepts effectively [1]. MDE advocates using the best modeling formalism that expresses relevant design intent declaratively at each level of abstraction. During development, we can use models to describe different aspects of the system vertically, in which the models are refined from higher to lower levels of abstraction through model transformation. At the lowest level, models use implementation technology concepts, and appropriate tools can be used to generate code from these platform-specific models [2].

Modularity is key in designing computer programs that are extensible and easily maintainable, but concerns that are crosscutting and more scattered in the implementation are more likely to cause defects [3]. This poses obstacles for MDE because modeling such crosscutting concerns in a modular way is difficult from an object-oriented standpoint. Furthermore, reusability is also a main factor in allowing developers to leverage reusable solutions such as libraries and frameworks provided for a given programming language, thereby improving the development speed without having to implement existing software components from first principles. Model reuse is still in its early stage, but modeling libraries are emerging as well [4].

Concern-Oriented Reuse (CORE) introduces a modeling technique that focuses on concerns as units of reuse [5]. CORE enables large-scale model reuse by utilizing the ideas of MDE, Separation of Concerns (SoC), and Software Product Lines (SPL). This is possible because CORE uses aspect-oriented modeling techniques to allow complex models to be built by incrementally composing smaller, simpler models. Current state of CORE supports models at the design phase [6], but models of other development phases can also be supported by integrating with the CORE metamodel to benefit from advanced modularization and reuse support.

This thesis focuses on models at the requirements phase, typically built earlier than design models, and the User Requirements Notation (URN) sets the standard as a visual notation for modeling and analyzing requirements [7]. URN formalizes and integrates two complementary languages: (i) Goal-oriented Requirements Language (GRL) to describe non-functional requirements as intentional elements, and (ii) Use Case Map (UCM) to describe functional requirements as causal scenarios. GRL and UCM are used to capture goal and scenario models, respectively. Since CORE already supports the use of goal models to analyze the impact of choosing features [5], we are interested in examining the possibility of having scenario models as part of the CORE toolkit. The goal of this thesis is to determine how UCMs can be integrated with the concepts of CORE. This leads to the question that begs to be investigated—whether actual MDE, i.e., software development with models at multiple levels of abstraction and model transformations that connect them, is compatible with CORE.

The remainder of this thesis is structured as follows. Chapter 2 offers background information on CORE and UCM. Chapter 3 presents the integration of CORE with UCM. Chapter 4 validates the resulting integration process. Finally, Chapter 5 concludes the thesis and discusses future work.

Chapter 2

Background

URN consists of two notations: GRL for modeling non-functional requirements, and UCM for modeling functional requirements. In this chapter, we describe UCM as part of the requirements engineering tool and its use in specifying scenarios in section 2.1. Then we detail about CORE as a reuse technique with the current state of development in section 2.2. Finally, we provide an overview of related work to modeling tools, applicable to the use of UCM and concern-orientation, in section 2.3.

2.1 Use Case Map (UCM)

This section describes UCM in detail.

Brief overview and what UCMs are used for. Maybe one example UCM (that is used later on)

.

UCM metamodel (or a simplified version of it)

.

UCM Weaving? Could also be explained later before you talk about your new weaving algorithm

.

2.2 Concern-oriented Reuse (CORE)

This section describes CORE in detail.

You can reuse some of our text from previous papers here. Focus on what is really interesting for us in this thesis, i.e., interfaces, features + realization models, customization mappings, weaving algorithm. You should also show the CORE metamodel here (or at least a simplified version of it that is used as a basis for the integration in chapter 3. You can use some of the text from our SoSyM paper.

.

2.2.1 Reusable Aspect Models (RAM)

No need to explain too much here, since we don't use RAM really in the rest of the thesis, no? Explain mostly that RAM is the only language currently integrated with CORE, and that the models are about design. You could use class diagrams to give an example of weaving, if you think it is necessary

.

2.3 Review of Related Modeling Tools

Literature review.

Most of your related work will be Gunter's AoUCM stuff, which you probably mention above already. jUCMNav as well, but this you could also mention above when you talk about UCMs.

.

Last paragraph leads to integration chapter.

Chapter 3

Adding Support for UCM to CORE

In this chapter, we describe the integration of UCM as a new modeling language into existing modeling languages of TouchCORE. We describe the definition of UCM metamodel in section 3.1. We also present the weaving algorithm for UCM in the context of CORE in section 3.2.

Yes that's it. Don't talk about implementation. The integration you are describing here, i.e., metamodel and weaver, constitute the conceptual integration. Here you explain why you are doing things this way, why we are mapping connection points, for example, why we need slightly different weavings for extend / reuse. Talk about TouchCORE / EMF later on in the following chapter.

3.1 UCM Metamodel

Here you can describe your UCM metamodel, i.e., how and why it differs from the original shown in chapter 2, and how you subclassed the important CORE metamodel classes to integrate your UCMs with CORE.

We follow the URN specification [8] closely in designing the UCM metamodel.

Figure **??**

The basic structure of a UCM

3.2 Weaver

Again, maybe we need a short intro here to explain why weaving is needed. I assume there is also a common algorithm to extensions and reuses, or are they completely separate? Then describe the algorithm

Explain that we need to specify composition specifications, i.e., mappings.

3.2.1 Responsibility Mapping

3.2.2 Connecting Point Mapping

Chapter 4

Validation

The definition of UCM metamodel and the specification of weaving algorithm described in previous chapter provide the foundation for the implementation of UCM in TouchCORE. UCM allows us to describe precise sequences graphically display a workflow model. In this chapter, we illustrate the realization of scenario models in TouchCORE through the use of UCM notation in section 4.1. Then we attempt to validate our proposed approach of concern-oriented UCMs by means of case studies in section 4.2. Finally, we demonstrate that concern-oriented UCMs are able to cover the workflow patterns in section 4.3.

4.1 UCM Implementation in TouchCORE

Is there a lot of interesting details to say about your implementation in TouchCORE / EMF / Java? If yes, then keep this chapter. If not, you could add a section about the implementation to the validation chapter, as an implementation is a form of validation (in the sense that if you implement your weving algorithm, and can run it on models to compose them, and the result is "correct", i.e., the algorithm terminates and the resulting model is as expected, then your algorithm must at least not be completely flawed :)

The project uses Java SE Development Kit 8 as the implementation language and Eclipse Modeling Framework (EMF) [9] as the modeling facility for building TouchCORE. To support a new language, we need to specify its metamodel through EMF.

4.1.1 Model Weaving with UCMs

Model Extension

Model Reuse

4.2 Case Studies

4.2.1 Authentication

4.2.2 Online Payment

4.3 Workflow Patterns

Chapter 5

Conclusion

5.1 Summary

Recap of thesis: what I did so far and what can the tool achieve.

5.2 Future Work

Remaining tasks such as components, path drawing, validation, semantics, etc.

Appendix A

Complete Metamodels

A.1 CORE Metamodel

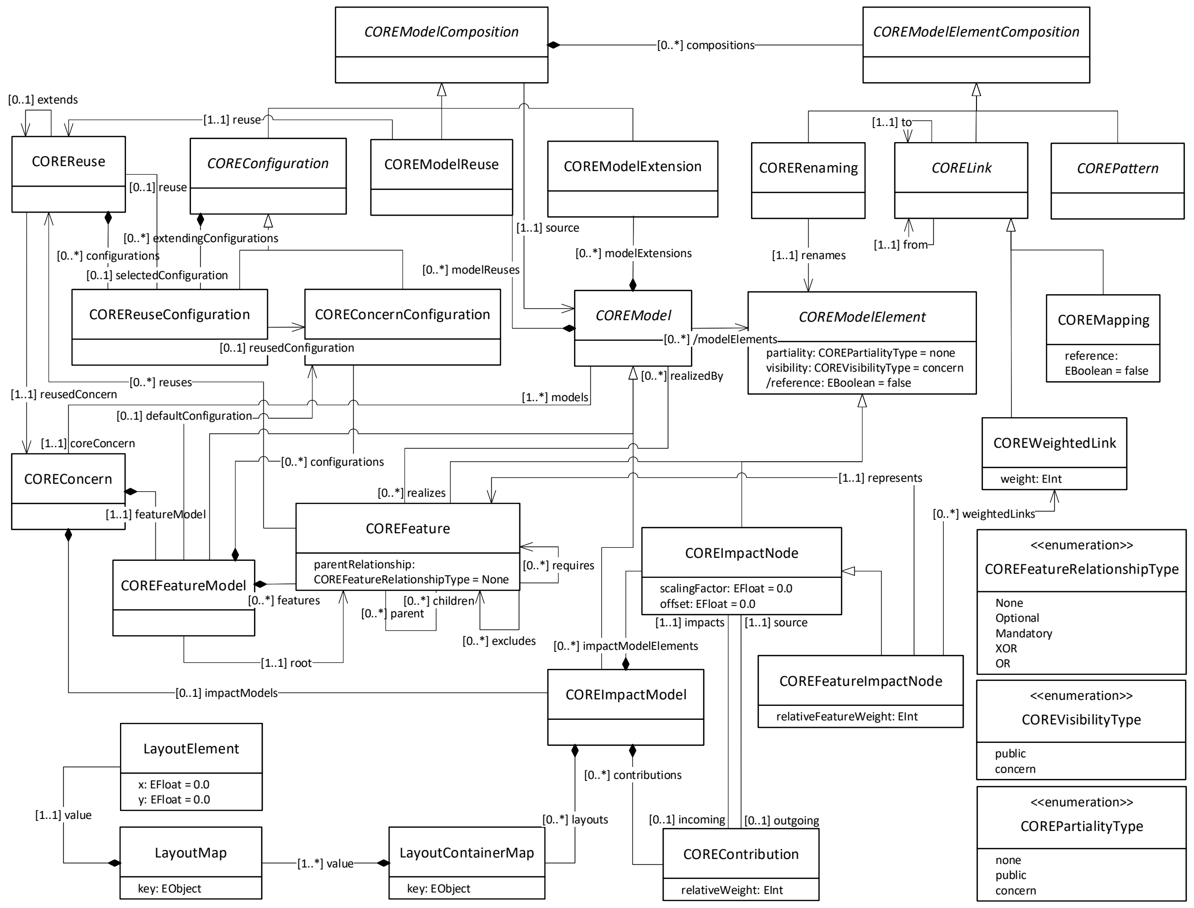

Figure A.1: Abstract grammar: CORE metamodel overview

A.2 UCM Metamodel

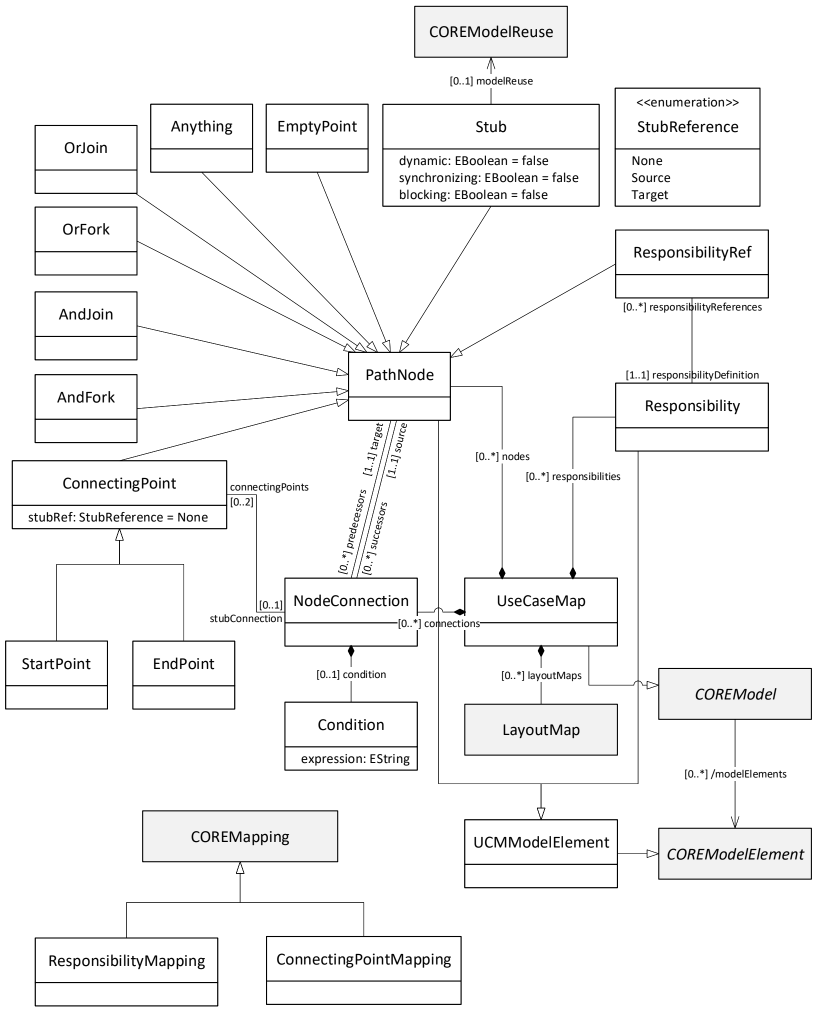

Figure A.2: Abstract grammar: UCM metamodel overview

Appendix B

Complete Case Study Models

B.1 Authentication Model

B.2 Online Payment Model

References

[1] Douglas C Schmidt. “Model-driven engineering”. In: *COMPUTER-IEEE COMPUTER SOCIETY-* 39.2 (2006), p. 25.

[2] Shane Sendall and Wojtek Kozaczynski. “Model transformation: The heart and soul of model-driven software development”. In: *IEEE software* 20.5 (2003), pp. 42–45.

[3] Marc Eaddy et al. “Do crosscutting concerns cause defects?” In: *IEEE transactions on Software Engineering* 34.4 (2008), pp. 497–515.

[4] Robert B France et al. “Repository for model driven development (ReMoDD)”. In: *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press. 2012, pp. 1471–1472.

[5] Omar Alam, Jörg Kienzle, and Gunter Mussbacher. “Concern-oriented software design”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2013, pp. 604–621.

[6] Jörg Kienzle et al. “Aspect-oriented design with reusable aspect models”. In: *Transactions on aspect-oriented software development VII* (2010), pp. 272–320.

[7] Daniel Amyot and Gunter Mussbacher. “URN: Towards a new standard for the visual description of requirements”. In: *International Workshop on System Analysis and Modeling*. Springer. 2002, pp. 21–37.

[8] Z ITU-T. “151 User requirements notation (URN)–Language definition”. In: *ITU-T, Oct* (2012).

[9] Dave Steinberg et al. *EMF: eclipse modeling framework*. Pearson Education, 2008.