COMP-533
Model-Driven Software Development
Course Outline

Instructor

Jörg Kienzle, McConnell 327
Phone: (514) 398-2049
Email: Joerg.Kienzle@mcgill.ca
Office hours: Monday: 9:30 - 11:00 or by appointment (send email)

Teaching Assistants

Omar Alam, McConnell 322
Phone: (514) 398-7071 ext. 00116
Email: Omar.Alam@mail.mcgill.ca
Office hours: Tuesday: 2:30 - 3:30 or by appointment (send email)

Course Homepage

http://www.cs.mcgill.ca/~joerg/SEL/COMP-533_Home.html

Course Contents

Model-Driven Engineering (MDE) is a unified conceptual framework in which the whole software life cycle is seen as a process of *model production*, *refinement* and *integration*. Models are built representing different views of a software system using different *formalisms*, i.e. modeling languages. The formalism is chosen in such a way that the model concisely expresses the properties of the system that are important at the current level of abstaction. During development, high-level specification models are refined or combined with other models to include more solution details, such as the chosen architecture, data structures, algorithms, and finally even platform and execution environment-specific properties. The manipulation of models is achieved by means of *model transformations*. Model refinement and integration continues until a model is produced that can be executed. This model can take the form of source code, but does not need to.

The object-oriented software development industry has gone through the process of standardizing visual modeling notations. The Unified Modeling Language (UML), a modeling language for specifying, visualizing, constructing, and documenting, is the product of this effort; it unifies the notations that currently exist in the industry. Therefore, every computer scientist, and especially every software engineer should have had contact with UML during his or her studies. Using UML notations has many benefits, including:

- It offers a common language uniting different object-oriented software development methodologies in terms of notation and vocabulary.
- It provides a rich set of notations that can be used to describe many different aspects of the software under development, including complex concurrency and distribution.

- It contains extension mechanisms and a constraint language called the Object Constraint Language (OCL). OCL is based on standard set-theory and is free of side-effects. It can be used to place additional constraints on models and describe pre- and postconditions of operations.
- UML provides a semantic base in the form of a metamodel that defines well formed models, and possible relationships between models and model elements.
- UML is a OMG standard.
- It allows tool interoperability between different vendors.

However, UML is only a language: it is process independent and therefore does not prescribe how and when during the software development cycle its notations should be used. Using UML to model software systems still requires a method - a choice of models and a process of their elaboration.

The goal of this software engineering course is expose the students to model-driven engineering by teaching them a rigorous UML-based software development method. The emphasis of the course is on how to put the different pieces of the UML puzzle together, i.e. which UML notations are the most appropriate to model the system under development within each software development phase. It teaches the students how to choose a coherent subset of UML to produce complete and consistent analysis and design models, and in which order the different models should be produced, and how models from one development phase are used as input / transformed into models at the next development phase.

The course teaches Fondue, a UML extension of the second-generation object-oriented development method Fusion (Hewlett Packhard). Fondue uses a coherent subset of UML to establish complete and precise analysis and design documents for a software system. Fondue's requirements engineering is based on use cases. The analysis phase establishes a Domain, a Concept, an Environment, a Protocol, and an Operation Model. During design, a Design Class Model, an Object Interaction Model, a Dependency Model and an Inheritance Model are constructed. Finally, Java-specific mapping strategies lead to the system implementation.

Most models use graphical UML notations (e.g. class diagrams, object-interaction diagrams, state diagrams). The course covers these UML notations in detail. In addition, a major part of the lectures present how to elaborate one model based on the others, and ensure completeness and consistency among the different views. When the graphical notations are not strong (or "formal") enough to produce a precise model, the models are augmented using the Object Constraint Language (OCL). OCL is a formal, set-theory based language that allows software engineers to augment UML models with additional constraints, or state preconditions, postconditions and invariants of operations in a precise way.

Finally, since model-driven engineering advocates to use the most appropriate modelling formalism(s) at each development phase to express the concerns at hand, this course also exposes the students to more advanced modelling notations. In particular, I am planning to introduce this year alternative modelling notations for requirements elicitation and analysis, in particular URN and AoURN (the User Requirements Notation and its aspect-oriented extension), and an aspect-oriented software design notation called Reusable Aspect Models.

Tentative Course Schedule

Course Overview

- Introduction
- UML and Method Overview

Requirements Elicitation

- Object-Orientation and Aspect-Orientation
- Use Cases
- URN and AoURN
- Domain Model

Analysis

- Concept Model
- Environment Model
- Protocol Model
- OCL
- Operation Model
- Check of Analysis Models

Design

- Interaction Model
- Dependency Model
- Design Class Model
- Inheritance Model
- Principles of Good Design / Design Patterns

Aspect-Oriented Software Development

- Aspect-Oriented Programming
- Aspect-Oriented Modelling

Implementation

- Implementation Class Model
- Mapping to Java

Reading List and Handouts

All course slides and exercises will be made available for download on the course webpage.

Most Useful Textbook

- Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. First Edition, Prentice Hall 1998. ISBN: 0137488807

Note: The new "second and third editions" of the book are based on the Rational Unified Process (RUP), rather than the Fusion process.

Other Textbooks

Fusion

The following book describes the Fusion object-oriented development method. (Fondue is based on Fusion, but uses the UML notation. Therefore, diagrams presented in these books will seem unfamiliar.) The book has also been translated into French.

- English Version: D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes and P. Jeremaes. Object-Oriented Development - The Fusion Method. Prentice Hall, 1994.
- French Version: D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes et P. Jeremaes. Fusion: la méthode orientée objet de 2ème génération. Masson, 1996.

UML Reference Manual

- James Rumbaugh, Ivar Jacobson and Grady Booch. The Unified Modeling Language Reference Manual. 2nd edition. Object Technology Series, Addison Wesley, 2004. ISBN: 0-321-24562-8

Download the UML specification from the web!

OCL

- Jos Warmer and Anneke Kleppe. The Object Constraint Language - Getting your models ready for MDA, 2nd edition. Object Technology Series, Addison Wesley, 2003. ISBN 0-201-37940-6

Prerequisites

- COMP-335 Software Engineering Methods, or
- ECSE-321 Introduction to Software Engineering, or
- Consent from the Instructor

Course Format

The course will be offered in the traditional lecture format, i.e. 3 hours of lectures per week.

Grading

There will be 3 graded homework assignments (3 * 10%), a mid-term exam (30%), and a take-home final in November (40%).

Academic Integrity:

McGill University values academic integrity. Therefore all students must understand the meaning and consequences of cheating, plagiarism and other academic offences under the Code of Student Conduct and Disciplinary Procedures (see www.mcgill.ca/integrity for more information).

L'université McGill attache une haute importance à l'honnêteté académique. Il incombe par conséquent à tous les étudiants de comprendre ce que l'on entend par tricherie, plagiat et autres infractions académiques, ainsi que les conséquences que peuvent avoir de telles actions, selon le Code de conduite de l'étudiant et des procédures disciplinaires (pour de plus amples reseignements, veuillez consulter le site www.mcgill.ca/integrity).

Students Rights and Responsibilities

Regulations and policies governing students at McGill University can be downloaded from the website:
http://www.mcgill.ca/deanofstudents/rights/

Email Policy

E-mail is one of the official means of communication between McGill University and its students. As with all official University communications, it is the student's responsibility to ensure that time-critical e-mail is accessed, read, and acted upon in a timely fashion. If a student chooses to forward University e-mail to another e-mail mailbox, it is that student's responsibility to ensure that the alternate account is viable. Please note that to protect the privacy of the students, the University will only reply to the students on their McGill e-mail account.