Teaching Model Driven Language Handling

Terje Gjøsæter, Andreas Prinz

5 pages
Teaching Model Driven Language Handling

Terje Gjøsæter¹, Andreas Prinz²

¹ terje.gjosater@ui.no
² andreas.prinz@ui.no
http://www.ui.no/
Faculty of Engineering and Science
University of Agder, Grimstad, Norway

Abstract: Many universities teach computer language handling by mainly focussing on compiler theory, although MDD (model-driven development) and meta-modelling are increasingly important in the software industry as well as in computer science. In this article, we share some experiences from teaching a course in computer language handling where the focus is on MDD principles.

Keywords: MDD, meta-modelling, language specification, teaching

1 Introduction

MDD (model-driven development) and meta-modelling are increasingly important in the software industry as well as in computer science. When language definition tools and technologies allow the language developer to work on a suitable abstraction level, they facilitate rapid development of powerful language tools. However, in the Norwegian universities, there is a strong emphasis on compiler theory and little or no focus on meta-modelling in most of the available computer language handling courses [GP10]. At the University of Agder, we have modified the course in computer language handling to focus not only on compiler development, but also on meta-model-based language design and definition, and particularly on finding the right level of abstraction for modelling as a basis for code generation. The main purpose of this article is to share experiences from teaching meta-model-based language description, and to discuss which tools and technologies are suitable for teaching computer language handling.

The rest of the article is organised as follows: Section 2 gives an overview of the different aspects of a meta-model-based language specification and outlines a course teaching these principles. Section 3 discusses issues related to choice of tools and technologies for use in teaching the different language aspects. Finally, we summarise our findings in Section 4.

2 Teaching Meta-model-based Language Handling

2.1 Overview

When a course in compiler theory was modified to also cover meta-modelling, it became clear that we needed to get a common understanding between the two paradigms; which parts of a compiler description correspond to which parts of a meta-model-based language description. In [NPT06], a language definition is said to consist of the following aspects: Structure, Constraints,
Presentation and Behaviour. Structure defines the constructs of a language and how they are related. This is also known as abstract syntax. Constraints bring additional constraints on the structure of the language, beyond what is feasible to express in the structure itself. Presentation defines how instances of the language are represented. This can be the definition of a graphical or textual concrete language syntax. Behaviour explains the semantics of the language. This can be a transformation into another language (denotational or translational semantics), or it defines the execution of language instances (operational semantics).

### 2.2 A Computer Language Handling Course

As described in more detail in [GP10], we have investigated how a course that primarily focused on compiler theory could be updated to include meta-model-based approaches to language definition, and a special focus on determining the optimal abstraction level for each language aspect. Based on this, we have defined the following course outline that covers both meta-model-based as well as compiler-based approach to language definition:

<table>
<thead>
<tr>
<th>Level:</th>
<th>MSc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prerequisites:</td>
<td>Object oriented programming, UML modelling.</td>
</tr>
<tr>
<td>Credits:</td>
<td>5 ECTS</td>
</tr>
<tr>
<td>Literature:</td>
<td>Aho, Lam, Sethi, Ullman: Compilers (2nd ed.)[ALSU07]; Clark, Sammut, Willans et. al.: Applied Metamodelling (2nd ed.) [CSW08]</td>
</tr>
<tr>
<td>Form:</td>
<td>8 parts; each part with lectures, practical and theoretical exercises, and an obligatory hand-in.</td>
</tr>
</tbody>
</table>

The course has the following parts:

- **Part 1 - Introduction**: Compilers, language aspects, grammars, NFA, DFA, T-diagrams.
- **Part 2 - Structure**: Models, meta-models, MDA, abstract syntax, attribute grammars.
- **Part 3 - Constraints**: Semantic analysis, type systems, static and dynamic checks, type safety.
- **Part 4 - Textual presentation**: Syntax analysis, top-down and bottom-up parsing, lexical analysis, mapping, symbol tables, error handling, textual presentation for meta-models.
- **Part 5 - Graphical presentation**: Graphical languages, graph grammars, graphical presentation.
- **Part 6 - Transformation behaviour**: Transformation, code generation, intermediate code, optimisation, handling of generated code, model-to-model and model-to-text transformations.
- **Part 7 - Execution behaviour**: Operational semantics, interpreters, runtime environments, storage allocation, activation records, parameter passing, dynamic binding.
- **Part 8 - Summary**: Repetition of the most important topics of the course.

The course has been implemented at the University of Agder in the spring term of 2010. In a related project course, the students have a choice of different projects building on this course.

After running the language handling course, the following experiences were gathered:

- It is good to use a running example where aspects are added to complete a simple example language. It is also beneficial to cover all language aspects within one platform. However, students can easily be demotivated by immature tools. We should not try to cover too many different tools in the practical exercises, but rather concentrate on the most important ones and give the students...
more time to try them out for themselves by modifying and extending provided examples. The understanding should be strengthened by giving different perspectives on the same issues in a lecture covering both compiler theory and meta-modelling. However, the connection between the two paradigms were sometimes difficult for the students to see.

2.3 Finding the Correct Abstraction Level

An important part of this course concerns finding a good abstraction level in order to facilitate code generation from models. In this respect, tools for language description are used as an example. However, it is a challenge to find tools and technologies that work on a high abstraction level for each language aspect. If the abstraction level is too low, there are too many seemingly irrelevant details, that create complications and complexities that will make it more difficult for the students to get started with the tools. On the other hand, if the abstraction level is too high, it may not be possible to generate working tools from the language specification. For Structure and Textual Presentation, there are tools that operate on a suitable level of abstraction, while it is more difficult to find good abstractions for the other language aspects. We will cover some of the available tools and technologies and their suitability for use in teaching in the following section.

3 Choice of Tools and Technologies for Teaching

3.1 Overview

Immature or overly complex tools and technologies can demotivate students and in some cases even make them avoid meta-model-based projects. A former Master student has described experiences from implementing a DSL in both Eclipse with suitable plugins, and in Visual Studio with DSL tools, and concluded that Visual Studio is good on integration, documentation and ease of use, while Eclipse allows the developer to operate on a more suitable high level of abstraction and has a good selection of plug-ins to extend its functionality. However, both platforms have weaknesses when it comes to stability and user-friendliness [IGP08].

We prefer free multi-platform tools and technologies to lower cost and to enable students to install the software on their home computers. We also wish to have a collection of tools that can co-exist in one platform, such as for example Eclipse. We have also seen that the stability and user-friendliness has increased for Eclipse over the last couple of years, so we have ended up using that as our preferred platform, and testing various plug-ins to cover the different aspects of a language specification. In the following, we will give a brief overview of our choices of tools and technologies for each of the language aspects listed in Section 2.

3.2 Tools and Technologies for Teaching Structure

There are different standards and recommendations for meta-modelling with different complexity and expressiveness. The most famous dialects are MOF 1.x, EMF/Ecore, and CMOF. It seems reasonable to start a course in meta-model-based language design with an introduction to structure definition, using for example Eclipse with EMF/Ecore (preferably with a graphical Ecore editor) for demonstrating relevant examples.
3.3 Tools and Technologies for Teaching Constraints

In meta-modelling, the most common way to express constraints is the Object Constraint Language, OCL, which has the expressiveness of predicate logic, in a programming-language-like syntax. A lecture on constraints can be illustrated by creating and adding OCL constraints to a sample meta-model, and using an OCL toolkit such as MDT OCL or the EMF Validation Framework.

3.4 Tools and Technologies for Teaching Presentation

One well-known framework for graphical notations is GMF. It features a language to define graphical notations, and generates Eclipse and GEF-based editors from these definitions.

Frameworks for textual notations can be divided into tools like XText, which provides editors based on language definitions consisting of grammars, and frameworks like TCS, TEF and EMFText, which combine meta-models and grammars. An advantage of EMFText, is that it can generate a HUTN-based (Human-Usable Textual Notation) parser and editor from an Ecore meta-model, that can be used as a starting point for developing a textual notation.

If a running meta-model-based example is used, it may be fruitful to show the students how an EMF-based example structure (with constraints) can be extended with both graphical and textual presentations, using editor generation frameworks like for example GMF for graphical editor generation and EMFText for textual editor generation.

3.5 Tools and Technologies for Teaching Behaviour

We have noted that it may be challenging to teach this language aspect since most of the tools available for supporting the theory of this aspect are relatively immature and/or hard to use, particularly for execution behaviour. Transformation languages like QVT or ATL can be used to create example transformations on the structure of the running EMF-based example, and for the latter, JET, Acceleo, or XPand can be used to generate textual code. The Eclipse plugin EProvide, provides support for developing visual debuggers and interpreters based on operational semantics defined in ASM, QVT/Relations, Java, Prolog or Scheme.

For illustrating the theory in this lecture, we may want to apply Model-to-Model transformations using QVT or ATL, and Model-to-Text with for example JET or XPand. We may also demonstrate operational semantics with ASM-based semantics in EProvide.

3.6 An Alternative Platform

Based on experiences from teaching, we have concluded that it may be useful to develop a very simple meta-model-based language definition platform, that attempts to remove some of the complexity of the more popular existing tools, in order to better allow the students to grasp the basic principles of meta-modelling. It should let the student operate on a suitable level of abstraction on each relevant language aspect, and facilitate making and modifying small example languages. In order to achieve this, we have started designing and prototyping a new platform named LanguageLab. It is planned to be a complete environment for experiments with modular language specification, particularly intended for use in teaching.
4 Conclusions

One of the main challenges of teaching meta-model-based language handling is finding tools that are simple, on a high abstraction level, and that work well together with other tools for other language aspects. It is our impression that some of the perceived complexity of meta-modelling comes from complex tools and technologies, rather than from the principles behind them.

It is possible to build a series of lectures in meta-model-based computer language handling supported by running examples based on Eclipse/EMF and other Eclipse-based plug-ins and frameworks, to cover all aspects of a language definition. However, we think that it may also be interesting and fruitful to develop and introduce a very simple meta-model-based language definition platform, that attempts to remove some of the complexity of the more popular existing tools, in order to better allow the student to grasp the basic principles of meta-modelling. It should let the student operate on a suitable level of abstraction on each relevant language aspect, and facilitate making and modifying small example languages.

Bibliography


