An eXecutable Metamodelling Facility for Domain Specific Language Design

Tony Clark, Andy Evans, Paul Sammut, James Willans
Xactium Limited
andy.evans@xactium.com

Introduction

Domain specific language definition involves the application of metamodelling technologies to rapidly generate and integrate semantically rich languages and tools that target domain specific modelling requirements. The aim is to provide developers with modelling abstractions appropriate to their development needs, thus enabling them to clearly focus on the problem domain in isolation from implementation details.

This article discusses limitations with the current standards that support language definition (MOF, QVT, UML) and looks at a particular approach to extending the standards to support it. This approach is based on the construction of a layered, executable metamodelling framework called XMF that provides semantically rich metamodelling facilities for designing languages. This architecture has been implemented in a commercial tool, full details of which can be found at: www.xactium.com. A book on the subject of language-driven development and metamodelling can be also be found at this web-site [4].

Finally, we argue that any approach to designing DSL's must be all encompassing in its ability to design languages in general.

The Role of Languages in Systems Development

Developers use a surprisingly varied collection of languages during the systems development process. This includes high-level modelling languages that abstract away from implementation specific details, to languages that are based on specific implementation technologies. Many of these are general-purpose languages, which provide abstractions that are applicable across a wide variety of domains. Of particular relevance here, are domain specific language (DSL's). DSL's are languages that provide a highly specialised set of domain concepts.

Languages typically support many different capabilities that are an essential part of the development process. These include:

- *Execution*: allows the model or program to be tested, run and deployed.
- *Analysis*: provides information of the properties of models and programs.
- *Testing*: support for both generating test cases and validating them must be provided.
- *Visualisation*: many languages have a graphical syntax, and support must be provided for this via the user interface to the language.
- *Parsing*: if a language has a textual syntax, a means must be provided for reading in expressions written in the language.

- *Translation*: languages don't exist in isolation. They are typically connected together whether it is done informally or automatically through code generation or compilation.
- *Integration*: it is often useful to be able to integrate features from one model or program into another, e.g. through the use of configuration management.

The important point to make is the diversity of ways in which languages are used in practice. Any language definition approach must be expressive enough to capture all these aspects.

Features of Languages

Although there are many different types of languages, there are some common features that they all share. These must be understood if we are to develop a generic approach to language definition. The primary features are:

Concrete Syntax

All languages provide a notation that facilitates the presentation and construction of models and programs in the language. This notation is known as its concrete syntax. There are two main types of concrete syntax: textual and visual. A textual syntax enables models and programs to be described in a structured textual form. A visual syntax presents a model or program in a diagrammatical form. The advantage of a textual syntax is that it is good at representing detail, while a visual syntax is good at communicating structure.

Abstract Syntax

The abstract syntax of a language describes the vocabulary of concepts provided by the language and how they may be combined to create models or programs. It consists of a definition of the concepts, the relationships that exist between concepts and may also include rules that say how the concepts may be legally combined. It is important to emphasize that a language's abstract syntax is independent of its concrete syntax and semantics. Abstract syntax deals solely with the form and structure of concepts in a language without any consideration given to their presentation or meaning.

Semantics

The semantics of a language describes what models or programs in the language actually mean and do. In the context of programming languages, an execution semantics is essential in order to run programs written in the language. Semantics are also important in the context of modeling languages. Without semantics, modeling languages like UML offer little more than a collection of notations and their usefulness is reduced.

Language Definition and Metamodels

The need to capture languages independently in a platform independent format is not new. One of the most widely known standards for this purpose is the MOF (the Meta Object Facility) [3]. The purpose of the MOF is to define a common way of capturing meta-data. Languages expressed in terms of MOF can be related to each other simply because they are defined in the same way. For example, if one wants to move from a model written in UML to a model that describes a Java program, the process is greatly eased because they are represented in the same way.

The way that MOF describes languages is through *metamodels*. Typically, a metamodel is a model of the concepts that are provided by the standard. In this case of UML, this might include such concepts as Class, Attribute, Operation, and so on.

Although the MOF is a good starting point for defining languages, it has a number of limitations:

- It is not rich enough to capture semantic concepts in a platform independent way. For instance, the execution of a state machine or a business process cannot be expressed in MOF. The tool designer must resort to implement these aspects in an external implementation technology such as Java.
- MOF does not provide a means of expressing the concrete syntax of a language, whether it is a textual or diagrammatical syntax. Whilst MOF models can be exported in terms of XML, this is of limited use to modellers, who require a more human readable form.
- MOF does not provide abstractions for capturing user interfaces and tools in a generic fashion. This means that either the language designer has little control over the user interface of a tool that supports the language, or these aspects must be encoded in a platform specific way.
- There is currently no way of defining both uni-directional and synchronised mappings between MOF models. These are essential to describe language transformations and the synchronisation of language elements. Whilst the QVT standard is aiming to address the issue of a uni-directional language, the need for a synchronised mapping language is paramount in order to describe synchronisation between language elements and diagram editors, user interfaces, and so on.

An eXecutable Metamodelling Facility

While a simple meta-data approach to defining language is insufficient, what can be done to address the issue? How can we rapidly create the rich modelling languages that are required for language driven development?

What is required is a metamodelling environment that is sufficiently rich to capture all aspects of a language in a platform independent way. Moreover, this language should be self describing and self-supporting, thus making it possible for any language to be defined completely independently of external technology.

A central feature of such an environment is executability – this must be built in from the start in order to be able to capture the operational semantics of languages, and to be able to define the semantics of the metamodelling language itself. Without it, language definitions become reliant on external implementation technologies in order to define language semantics.

To support domain specific language definition, we propose the following components of an executable metamodelling facility (an overview is shown in figure 1):

- A virtual machine for executing metamodels. Its purpose is to act as a platform independent execution environment for language definitions.

- A small, precise, executable metamodelling language that is bootstrapped independently of any implementation technology. It should support the minimum language definition capabilities necessary to be self-supporting. Thus includes a generic parsing language and diagramming language, a compiler and interpreter, and a collection of core executable MOF modelling action primitives. Such a language can be built on top of MOF with the addition of appropriate executable primitives.
- A layered language definition architecture, in which increasingly richer languages and development technologies are defined in terms of more primitive languages via operational definitions of their semantics or via compilation to more primitive concepts or extension of existing concepts: These will include:
    - Definitions of richer metamodelling languages, such as mapping languages (QVT), UI languages, constraint languages, testing languages and pattern languages.
    - Definitions of general-purpose language primitives such as components, aspects, patterns, etc, that can be combined into general purpose modelling languages such as UML.
    - Definitions of bespoke, domain specific languages that are built from the above definitions.
- Support for the rapid deployment of metamodels into working tools. This involves linking the UI metamodels with appropriate user-interface technology. An open source toolset like Eclipse and GEF can be used for this purpose.

The aim is to build support from the ground up for language definition in a layered fashion. Using the facility it is possible to rapidly implement many different modelling languages and associated tools in a platform independent way. In other words, the metamodel becomes a complete definition of the language and tool that supports it.

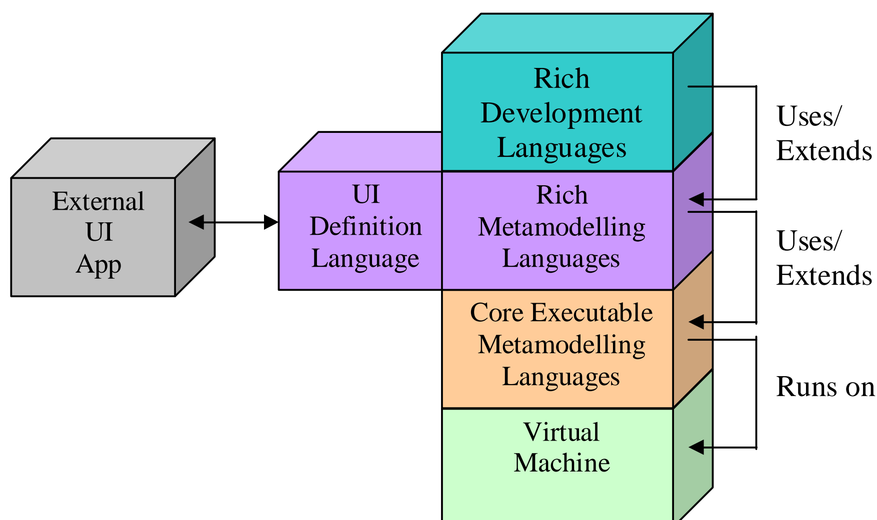

Figure 1: A Metamodel Facility that supports Language Driven Development

Example

Imagine a domain specific inventory modelling language that supports the specification of inventory entities, including products, services and resources within a teleco environment. Specifications (expressed as constraints) can be associated with entities. Inventory entities can also have operations and attributes, can be associated with other inventory entities, and can specialise inventory entities of the same type. The language is to be integrated into a component modelling language because of the distributed nature of the telco environment. Furthermore, inventory models should be deployable into Java and an associated user interface that permits creation/deletion of instances of the models and the checking of specifications.

In order to implement a development environment for this scenario, a number of different languages will be required, each with their own metamodel definitions:

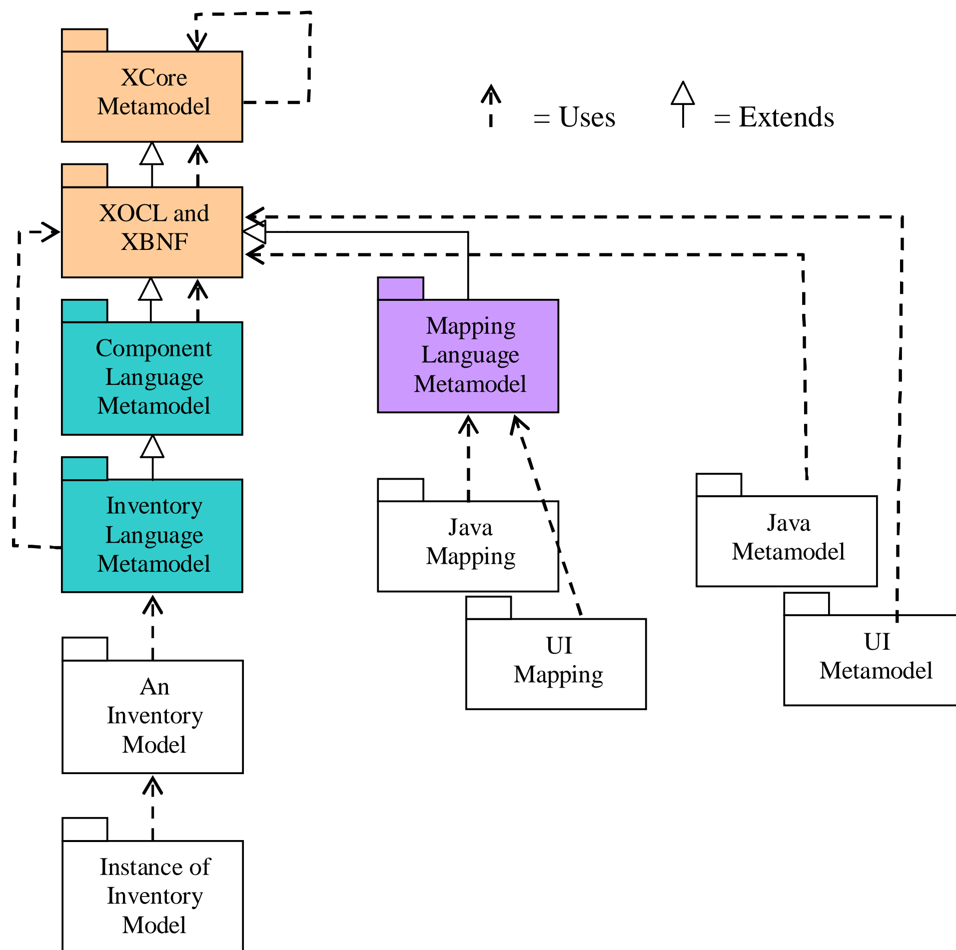

Figure 2: Inventory Example

At the heart of the language definition is the XCore metamodel. This provides a minimal collection of executable OO metamodelling concepts that can be run on the virtual machine. This is extended by XOCL, an executable version of OCL for expressing constraints and actions, and XBNF, a generic parser language for describing the concrete syntax of a language. All other metamodels will extend and

instantiate (use) these metamodels (including the XCore metamodel, which instantiates itself).

The mapping language metamodel is an example of a rich metamodelling language definition as it adds the functionality to express mappings between metamodels more concisely than using the core metamodelling facilities alone.

The component language metamodel is an example of a rich (general purpose) development language definition, while the inventory language is an example of rich (domain specific) development language definition. Note that the domain specific language is also dependent on a number of general-purpose languages, such as OCL for capturing specifications.

Finally, mappings are instances of the mapping language metamodel that relate inventory models to Java and UI metamodels.

The key point about the languages used in this example is that they are all defined in terms of a common executable metamodelling language. Thus, models written in these languages can leverage the executable semantics provided by the metamodelling language. For instance, the inventory language is a fully executable language: models can be created, and instances can be created from those models. Thus, specifications and standards can be verified for correctness very early in the lifecycle. Moreover, the completeness of the inventory metamodel means that a complete implementation can be generated in Java.

Development Experiences

We have been successful in implementing a tool that fully supports the architecture described above. This tool is beginning to realise the many benefits of an executable metamodelling architecture. These include being able to flexibly and quickly design new modelling environments, which provide semantically rich modelling capabilities. A good example of this is a language for designing user-interfaces, which is being used to design new tools. This language has been modelled such that it can be used to generate new tools and interfaces to existing tool components. However, the language is fully bootstrapped, thus enabling its diagrammatical syntax to be described in its own representation.

Language Driven Development and DSL's

Many papers, for example [2] have proposed that domain specific languages can provide significant advantages to the systems development process. DSLs aim to provide targeted domain specific modelling concepts, which can be used to accelerate development. However, our experience tells us that developers need to access a wide variety of languages and to be able to use those languages in flexible, integrated ways. As shown in the inventory modelling language example, there will often be the need to utilise general purpose languages in DSL's and vice versa.

Instead we must seek ways in which general purpose and domain specific languages can be defined in a common metamodelling facility that is rich enough to express all aspects of the languages, including their semantics. In [4] we call this approach *language driven development*, emphasising the fact that effective systems

development involves the use of multiple languages and multiple language types to be most effective.

Conclusion

Standards like MOF and research on DSL's are driving us forward to a world in which languages are managed in a unified and semantically rich ways. However, to achieve this, we must understand better how generic language driven development techniques can be layered on top of existing metamodelling approaches. As outlined above, this can only occur if we raise the bar with respect to what metamodels can represent, and build a layered metamodelling architecture that can support semantically rich language definition capabilities.

In this paper, we have given an overview of a generic executable metamodelling facility called XMF that aims to realise this approach. The XMF toolkit is an implementation of XMF that is currently being used in a variety of application domains. Further details of XMF can be found in [4] and at www.xactium.com.

References

[1] www.omg.org/mda
[2] Cook, S. Domain-Specific Modeling and Model Driven Architecture, MDA Journal, January 2004 (http://www.bptrends.com/publicationfiles/01-04%20COL%20Dom%20Spec%20Modeling%20Frankel-Cook.pdf).
[3] www.omg.org/mof
[4] Clark et al. Applied Metamodelling: a Foundation for Language Driven Development. Available from www.xactium.com.