On the Specification of Textual Syntaxes for Models

Frédéric Jouault Jean Bézivin
ATLAS team, INRIA and LINA
{frederic.jouault,jean.bezivin}@univ-nantes.fr

Abstract

Model-driven engineering is based on the use of models at all stages of the development. In many situations, these models have to be represented so as to be handled by a human being or a specific tool (e.g. a compiler). However, the default representation of models (XMI) is generally not adapted. There is consequently a need for tools enabling the specification of concrete syntaxes for models. The Eclipse Modeling project currently offers GMF (Graphical Modeling Framework) for the definition of visual syntaxes. This paper presents TCS (Textual Concrete Syntax) for the definition of textual syntaxes as a candidate for TMF (Textual Modeling Framework).

Categories and Subject Descriptors D.3.2 [*Language Classifications*]: Specialized application languages; D.3.4 [*Processors*]: Code Generation

General Terms Languages

Keywords Model Driven Engineering, Concrete Syntax

1. Introduction

Model-driven approaches are increasingly used for the implementation of Domain-Specific Languages [1] (DSLs) or "little languages". Several "model-based DSL frameworks" [2] provide tools to this end, such as: Microsoft Software Factories and DSL Tools [3], GME [4], and the Eclipse Modeling project [5]. With these three environments, the definition of a DSL typically consists of the specification of its metamodel and of a visual syntax. For instance, the Eclipse Modeling project uses the Ecore metametamodel, part of EMF [6], to define metamodels, and GMF [7] (Graphical Modeling Framework) for visual syntax specification.

Some languages may have textual rather than visual syntaxes, or they may even have both. However, support for the specification of textual syntaxes is usually not present in the model-based DSL frameworks. Thus, although the Eclipse Modeling project plans to define TMF (Textual Modeling Framework), the equivalent of GMF but for textual syntaxes, no tool is available yet.

The Eclipse Modeling project is actually a bit more complex than a combination of EMF and GMF. It is composed of several other sub-projects such as GMT [8], which is an incubator for research-originating technology. The AMMA (ATLAS Model Management Architecture) framework [2] is a model-based DSL framework available as part of GMT. AMMA does not provide support for visual syntaxes and relies for this on other solutions like GMF. For textual syntaxes the framework provides TCS [9], a language that enables the specification of textual syntaxes for metamodels.

We have shown in [9] and [10] that TCS can be used to specify the concrete syntax of a relatively complex DSL in the domain of internet telephony: SPL [11]. Moreover, many other DSLs already have a concrete syntax defined in TCS. This is the case of the model transformation language ATL, the metametamodel KM3 [12], TCS itself, and many more. In this paper, we present the main characteristics of TCS.

The paper is organized as follows. Section 2 gives some background considerations on modeling and concrete syntaxes. Section 3 defines a set of requirements for textual syntaxes. TCS is presented in section 4, and section 5 concludes.

2. Background

Model-Driven Engineering (MDE) is based on models considered as first-class entities. Each model conforms to its metamodel defining the concepts and relations that it can use. Let us, for instance, consider a metamodel that defines the concepts of *point*, *line*, *rectangle*, etc. and the relations between them (e.g. a line has two *point* ends). A model conforming to this metamodel could contain points, lines, and rectangles forming vectorial figures.

One central operation that needs to be performed on these models is transformation, which makes it possible to create high-level design models that are then transformed into lower-level implementation models. It may thus be necessary to convert points, lines, and rectangles into appropriate method calls to draw a figure on a screen. Several approaches have consequently been developed to perform model-to-model transformation: QVT [13], ATL [14, 15], etc. The representation format of the transformed models is typically some form of XMI [16], which is XML-based.

However, in most present concrete scenarios, source and target models are often either visual or textual. Metamodel elements need to be associated to syntactic elements (e.g. visual or textual). While the metamodel may be considered as an abstract syntax, the definition of this association may similarly be considered as a concrete syntax. For instance, a developer may visually design a model (e.g. a vectorial figure) from which actual code is generated (e.g. in Java or C#). On one hand, few developers would like to write or read XMI. On the other hand, neither the Java nor the C# compilers can process XMI. Since model-to-model transformation cannot deal with these concrete syntaxes, it is necessary to have tools that perform the extra work. Moreover, the combination of abstract and concrete syntaxes represented as models can be used as an implementation basis for DSLs, which was discussed in [2]. Such a definition of a DSL generally also includes some form of semantics mapping (e.g. as a model-to-model transformation), but this is out of the scope of this paper.

The Eclipse Modeling project currently offers some tools for the specification of concrete syntaxes. Thus, GMF [7] (Graphical Modeling Framework) is a tool that enables the specification of visual syntaxes for metamodels. GMF works by attaching visual elements such as lines and polygons to metamodel elements. The benefit is that a developer can create models without resorting to XMI but rather using a simpler syntax. For instance, a metamodel can be created using the well-known class diagram notation.

Several approaches like JET [6] (Java Emitter Templates) and Xpand [17] enable the generation of text (e.g. Java or C# code) from

models. The Eclipse Modeling project places them in the model-to-text sub-project, and the OMG as issued a RFP for a model-to-text transformation language [18]. These solutions work by specifying text pieces mixed with model traversal constructs. They are very effective for the generation of code or documentation from design models. However, we do not consider that these approaches provide support for textual syntaxes. The next section motivates this.

3. Requirements for Textual Syntaxes

The Eclipse Modeling project currently only contains a placeholder called TMF (Textual Modeling Framework) awaiting proposal. In this section, we start by defining some requirements for TMF. Then, we explain why model-to-text approaches are not possible candidates for TMF, and why textual syntaxes do not fit in the model-to-text sub-project.

By analogy to what GMF offers, we believe that TMF should have the three following properties:

1. **Customizability.** Unlike solutions like XMI or HUTN [19], TMF should enable the specification of arbitrary syntaxes. For instance, it must be possible handle user-specified keywords and symbols as well as to represent expressions using the infix notation.
2. **Bidirectionality.** A textual syntax can be used to create a new model but also to represent an existing model. Therefore, it is necessary that TMF provides a bidirectional mapping between the metamodel and the textual notation. Although bidirectionality can be achieved by a pair of mappings (one for each direction), we believe that using a single definition for both directions is simpler. Indeed, it reduces redundancy and inconsistencies.
3. **Model-based.** The specification of a textual syntax with TMF should be composed of a model or a set of models. These models could then be either interpreted or compiled in order to translate models to and from their textual representations.

With these requirements in mind, we can now again consider the model-to-text approaches:

1. **Customizability** of the syntax is at the basis of model-to-text. This requirement is therefore fulfilled.
2. **Bidirectionality** is not available. Although, some approaches offer text-to-model round tripping, this does not solve the problem of generating a model from code. Round tripping only takes into account limited modifications of the generated code on ad-hoc basis.
3. Some approaches are **model-based** and define a metamodel for their language, which is for instance required in [18]. But the concrete syntaxes of these languages are generally not specified by models. Moreover, it is not possible to specify these syntaxes using model-to-text approaches.

Usually, only the first requirement is fully satisfied while the third is taken into account by only some approaches. As for the second requirement, no model-to-text approach supports it. It is therefore necessary to have a specific and distinct solution for the specification of textual syntaxes within TMF. The next section presents one such solution: TCS.

4. Presentation of TCS

This section briefly presents how TCS can be used to attach a concrete syntax to the metamodel of a DSL. A more detailed presentation is given in [9].

We use the SPL [11] (Session Processing Language) internet telephony language as an example. Listing 1 shows a simple SPL program that forwards incoming calls to address `sip:phoenix@barbade.enseirb.fr`. The `SimpleForward` service (lines 1-11) declares the target address (line 3) and a registration session (lines 6-10). This session contains an `INVITE` method (lines 6-8) which forwards incoming calls to the declared address (line 7).

Listing 1. Simple SPL program

```
1  service SimpleForward {
2    processing {
3      uri us = 'sip:phoenix@barbade.enseirb.fr';
4
5      registration {
6        response incoming INVITE() {
7          return forward us;
8        }
9      }
10   }
11 }
```

Let us consider the excerpt of SPL metamodel given in Listing 2 using the KM3 [12] textual notation for metamodels. It starts with the declaration of the String data type. Then it specifies that an SPL *Program* (lines 3-5) contains (line 4) exactly one *Service* (lines 7-11). The latter has a name of type *String* (line 8), declarations of type *Declaration* (line 9), and sessions of type *Session* (line 10).

Listing 2. SPL metamodel excerpt in KM3: *Program* and *Service*

```
1  datatype String;
2
3  class Program extends LocatedElement {
4    reference service container : Service;
5  }
6
7  class Service extends LocatedElement {
8    attribute name : String;
9    reference declarations[*] ordered container :
       ↪Declaration;
10   reference sessions[*] ordered container : Session;
11 }
```

Listing 3 gives a TCS model excerpt specifying the concrete syntax of these elements according to Listing 1. Here is an informal description:

- **String.** Data type *String* is represented as an identifier corresponding to lexer non-terminal `NAME` (line 1).
- **Program.** Class *Program* is represented as its contained service (lines 3-5).
- **Service.** Class *Service* is represented as: keyword `service`, the `name` of the service, symbol `{`, keyword `processing`, symbol `{`, the `declarations` of the service, its `sessions`, and two symbols `}` (lines 7-14).

TCS elements are associated to their corresponding metamodel elements by their names. For instance, TCS template *Program* corresponds to KM3 class *Program* and TCS property *service* to KM3 feature *service*. This example shows that it is straightforward to encode such a simple syntax in TCS: syntactic elements are specified in syntax order.

Listing 3. SPL TCS model excerpt: *Program* and *Service*

```
1  primitiveTemplate identifier for String default using
     ↪NAME;
2
3  template Program main
4    : service
5    ;
6
7  template Service
8    : "service" name "{"
9        "processing" "{"
10         declarations
11         sessions
```

```
12          "}"
13        "}"
14    ;
```

From this TCS model, it is possible to transform a program such as given in listing 1 into a model conforming to the SPL metamodel (injection), and to transform a model conforming to the SPL metamodel into an SPL program (extraction). The current implementation works by interpreting the TCS model for extraction. Injection is handled by generating (via model transformation) a grammar such as the one given in listing 4. This excerpt is stripped from its annotations for readability reasons. However, the actual grammar contains annotations that generate the model while parsing.

Listing 4. Annotation-free SPL grammar excerpt: *Program* and *Service* (ANTLRv2 syntax with highlighted terminals)

```
1  identifier
2    : NAME
3    ;
4
5  program
6    : service
7    ;
8
9  service
10   : "service" identifier LCURLY
11       "processing" LCURLY
12         (declaration (declaration)*)?
13         (session (session)*)?
14       RCURLY
15     RCURLY
16   ;
```

Let us now consider TCS according to the requirements for TMF defined in section 3:

1. **Customizability** of the syntax is available: user-specified keywords and symbols can be inserted whereever. Moreover, TCS supports the infix notation of expressions, which mainly requires the specification of operators priorities.
2. TCS is **bidirectional** because from one specification (e.g. listing 3) both injection and extraction are possible. Although the implementation of each direction is separate and based on a different approache, this is totally transparent to the user.
3. TCS is **model-based** since its abstract syntax is defined by a metamodel and its concrete syntax by a TCS model. Moreover, the generation of a grammar from the TCS model is performed using model-to-model transformation.

In addition to its injection and extraction capabilities, TCS also offers a user-friendly editor: TGE (Textual Generic Editor). This editor is already available as part of the GMT/AM3 [20] subproject. Figure 1 shows how KM3 metamodels are edited using TGE. It can be seen on this figure that TGE supports syntax highlighting and an outline view as well as advanced features such as hyperlinks and text hovers.

A tool similar to TCS called XText is available as part of the GMT/oAW [17] sub-project. However, it seems that it does not provide as much customizability as TCS yet. For instance, it does not currently provide expression support.

5. Conclusion

We have presented some background information on MDE, and especially on the specification of concrete syntaxes. We have seen that the Eclipse Modeling project already offers GMF for visual syntaxes but lacks an actual tool for textual syntaxes in the TMF sub-project. By analogy to GMF, we defined some requirements for TMF.

Then, we presented TCS, which fulfills these requirements. TCS enables the specification of textual syntaxes for metamodels,

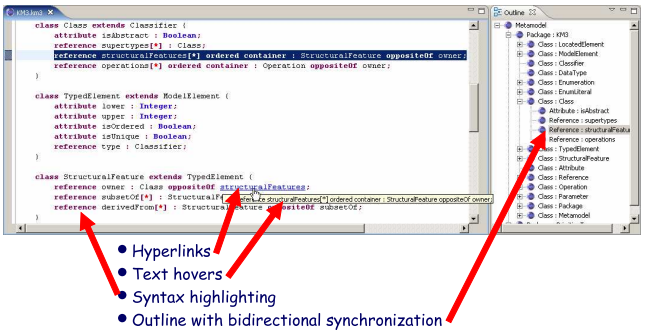

Figure 1. Screenshot of TGE

and thus the definition of textual DSLs with MDE. From a single definition it is possible to transform a program into a model and a model into a program. Additionally, textual editing is offered by TGE, which uses the TCS definition to automatically provide syntax highlighting, an outline view, hyperlinks and text hovers.

We believe that TCS provides a framework for textual syntaxes that is quite similar to what GMF offers for visual syntaxes. Therefore, we put TCS forward as a candidate for TMF implementation.

Acknowledgments

We would like to thank Charles Consel and his team who designed the SPL language, which we used to illustrate TCS. This work is being partially supported by ModelPlex, IST European project.

References

[1] van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: An annotated bibliography. SIGPLAN Notices **35**(6) (2000) 26–36

[2] Bézivin, J., Jouault, F., Kurtev, I., Valduriez, P.: Model-based DSL Frameworks. In: Companion to the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, OR, USA, ACM (2006) to appear.

[3] Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley (2004)

[4] GME: The Generic Modeling Environment, Reference site. (2006). Available at `http://www.isis.vanderbilt.edu/Projects/gme`

[5] Eclipse Foundation: Eclipse Modeling Project home page, `http://www.eclipse.org/modeling/`. (2006)

[6] Budinsky, F., Steinberg, D., Ellersick, R., Merks, E., Brodsky, S.A., Grose, T.J.: Eclipse Modeling Framework. Addison Wesley (2003)

[7] Eclipse Foundation: Graphical Modeling Framework (GMF) home page, `http://www.eclipse.org/gmf/`. (2006)

[8] Eclipse Foundation: Generative Model Transformer (GMT) Home page, `http://www.eclipse.org/gmt/`. (2006)

[9] Jouault, F., Bézivin, J., Kurtev, I.: TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering. In: GPCE'06: Proceedings of the fifth international conference on Generative programming and Component Engineering. (2006) to appear.

[10] Jouault, F., Bézivin, J., Consel, C., Kurtev, I., Latry, F.: Building DSLs with AMMA/ATL, a Case Study on SPL and CPL Telephony Languages. In: Proceedings of the 1st ECOOP Workshop on Domain-Specific Program Development (DSPD), July 3rd, Nantes, France. (2006)

[11] Burgy, L., Consel, C., Latry, F., Lawall, J., Palix, N., Réveillère, L.: Language Technology for Internet-Telephony Service Creation. In:

IEEE International Conference on Communications. (2006)

[12] Jouault, F., Bézivin, J.: KM3: a DSL for Metamodel Specification. In: Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, LNCS 4037, Bologna, Italy (2006) 171–185

[13] OMG: MOF QVT Final Adopted Specification, OMG Document ptc/2005-11-01. (2005). Available at `http://www.omg.org/cgi-bin/doc?ptc/2005-11-01`

[14] Jouault, F., Kurtev, I.: On the Architectural Alignment of ATL and QVT. In: Proceedings of ACM Symposium on Applied Computing (SAC 06), model transformation track, Dijon, Bourgogne, France (2006)

[15] Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Satellite Events at the MoDELS 2005 Conference: MoDELS 2005 International Workshops OCLWS, MoDeVA, MARTES, AOM, MTiP, WiSME, MODAUI, NfC, MDD, WUsCAM, Montego Bay, Jamaica, October 2-7, 2005, Revised Selected Papers, LNCS 3844, Springer Berlin / Heidelberg (2006) 128–138

[16] OMG: MOF 2.0 / XMI Mapping Specification, v2.1, OMG Document formal/2005-09-01. (2005). Available at `http://www.omg.org/cgi-bin/doc?formal/2005-09-01`

[17] Eclipse Foundation: openArchitectureWare project home page, `http://www.eclipse.org/gmt/oaw/`. (2006)

[18] OMG: MOF Model to Text Transformation Language, OMG Document ad/2004-04-07. (2004). Available at `http://www.omg.org/cgi-bin/doc?ad/2004-04-07`

[19] OMG: Human-Usable Textual Notation, v1.0, OMG Document formal/2004-08-01. (2004). Available at `http://www.omg.org/cgi-bin/doc?formal/2004-08-01`

[20] ATLAS team: ATLAS MegaModel Management (AM3) Home page, `http://www.eclipse.org/gmt/am3/`. (2006)