Creating Domain Specific Languages with Xtext

Andrés Carrasco

Andres.Carrasco@student.uantwerpen.be

University of Antwerp

Abstract

Even though visual languages have been often regarded as better suited for some Domain Specific Languages (DSL), we explore the possibilities textual languages provide to DSLs. Visual languages do excel in intuitiveness, however, they come in short in some of the mentioned advantages textual languages do have. Moreover, we also explore the Xtext framework for creating DSLs and how does it measure with the common requirements of DSLs.

Keywords: textual languages, Xtext, domain specific languages, DSL

1. Introduction

General Purpose Languages (GPL), e.g. Java, are designed for solving any kind of problem. However, due to their vast scope, in certain domains the solutions can become quite complex. For example, performing a query in a database in pure Java without using the domain specific language SQL, a big overhead would be added in parsing and interpreting the data coming out and to the database. Whereas simply using SQL reduces this overhead completely. SQL is just one example where a domain specific language (DSL) is more appropriate than a GPL. Nevertheless, a DSL does not always replace a GPL, as the DSL is often interpreted or compiled into a GPL. Therefore, the DSL is being utilized to simplify the complexity that a GPL might impose on the problem.

Both SQL and Java are textual languages, nevertheless, not all DSL are textual. Depending on your problem scope, a visual DSL might be better suited than a textual DSL, in some cases even a mixture of both. However, it is often argued that visual languages provide a better overview and are preferred above textual languages, as they are often referred to as more intuitive.

2. Visual vs Textual Languages

In their article Grönninger et al. (2014) [1] argue that textual languages have the following advantages over visual languages:

1. **Content compactness.** Grönninger et al. (2014) argue that graphical languages require more space to convey the same amount of content, thus when selecting an appropriate language type for a DSL, in which space is critical, textual languages have a clear advantage.
2. **Speed of creation.** the authors also argue that current text editor technologies are more efficient than visual editor technologies, due to their nature they often constrain the designer, leading to time loss.
3. **Integration of Languages.** in this point, Grönninger et al. (2014) argue that integrating different types of languages is easier if they are textual, as doing so in visual languages leads to a lack of developer efficiency.
4. **Speed and quality of formatting.** formatting text is a trivial task for standard formatting algorithms. In contrast to this, formatting a visual model is not, as depending on the semantics there can be a special layout thought of, probably requiring a custom layout algorithm.
5. **Platform and tool independency.** textual languages have the advantage that they can be manipulated by any text editor available, this gives them a high degree of flexibility, whereas visual languages usually need their own environment.
6. **Version control.** the last advantage Grönninger et al. (2014) point out, is the possibility of using standard version control systems. Version control systems are widely used for text, whereas with visual languages they have not, as they have not proven to be very reliable.

Although some of the previous points seem to be very convincing, in some cases visual languages might still be a better choice. Whenever deciding between one or the another, all advantages and disadvantages should be considered, as to decide on the best type for the proposed DSL.

3. Creating a DSL with Xtext

A Domain Specific Language requires the ability to read the input text, parse it, process it and possibly interpret or generate code in a GPL. However, as most programming languages a DSL also needs to have a good IDE support, syntax highlighting, continuous background parsing, error markings in the input text, auto-complete features, hyperlinking between references, and even possibly suggesting quickfixes, among other functionalities [2].

As it seems so far, developing a DSL is also quite a task, here is where Xtext [1] comes in handy. Xtext can generate all of the requirements of DSLs automatically, after specifying grammar rules utilizing their grammar language. Moreover, Xtext is able to create a compiler from your DSL to any other language. Xtext also provides special support for targeting the Java Virtual Machine through Xbase. In other words, utilizing the Xtext framework, reduces the overhead of creating a DSL and potentially enabling your DSL to unlock its full potential. Xtext makes it specially easy, because of their well chosen default configuration, which usually covers all of the common needs, however it is completely configurable.

Xtend itself uses a grammar language based on the Eclipse Modeling Framework (EMF). The input grammar is parsed using Another Tool For Language Recognition (ANTLR). The latter is widely used by programming language like Java and Python. From the grammar parsed by ANTLR a new linker and parser are created for the language specified by itself, alongside other popular IDE features. Creating thus a full blown DSL with IDE support.

References

[1] H. Grönninger, H. Krahn, B. Rumpe, M. Schindler, S. Völkel, Textbased Modeling, ArXiv e-prints`arXiv:1409.6623`.

[2] L. Bettini, Implementing Domain-Specific Languages with Xtext and Xtend, Packt Publishing Ltd, 2013.

[1] http://www.eclipse.org/Xtext/