Textual Modeling Tools:

Overview and Comparison of Language Workbenches

Bernhard Merkle

SICK AG
Research & Development
Software Engineering
Erwin-Sick-Str.1
79183 Waldkirch, Germany
bernhard.merkle@gmail.com

Abstract

Domain Specific Languages (DSL) attract more and more users as they are specialized and optimized for a certain problem area. Currently the number of new emerging Programming Languages is significant [1] but GPL (General Purpose Languages) do often not fit the specific need of the end-user. DSL are one way to solve this problem. DSLs can be divided into different independent dimensions: e.g. internal vs. external or textual vs. graphical or tabular. In this paper we focus on textual syntaxes as they have several advantages like easy information exchange via e.g. mail, integration into existing tools like diff, merge and version control and most important the fast editing style supported by the "usual" IDE support like code completion, error markers, intentions and quick fixes. While Fowler described the initial vision of Language Workbenches [2], several mature Textual Language Workbenches have emerged in recent years. In this paper we will compare them with a consistent example and look at pros and cons.

Categories and Subject Descriptors D.3.3 [**Programming Languages**]: extensible languages, domain-specific languages

General Terms Design, Languages, Textual modeling, Eclipse.

Keywords *language workbenches, domain-specific languages, textual modeling, , Xtext, TEF, TCS, EMFText, MPS.*

1. Introduction

Model driven development enables programming/modeling on a higher level, and generate low level stuff via a code generator. Essentially the idea is not really new and similar to the former transition from assembler to high-level programming languages (where code generators were compilers). Meanwhile developers want to express problems for a certain domain more appropriate, hence general programming languages (GPL) are not enough which led to the adoption of domain specific languages (DSL). Textual Languages are beneficial for many reasons. They enable productivity because of their easy and fast editing style, usage of code completion, error markers and other facilities people are used to meanwhile. Textual editors for GPL like eclipse, netbeans, IntelliJ are powerful and set new standards for textual editing. However for a wide adoption of DSL in day to day developments, IDEs for DSL should be easy to create (for language designers) and easy to use (for end-users). Martin Fowler described the idea of Language Workbenches [2], however that term focused on Projectional editing [3]. In this paper we compare the current state of Textual Language Workbenches, mainly based on the eclipse platform.

The subsequent chapters are structured as follows: Chapter 2 talks about various forms and representations of Domain Specific Languages and presents a specific DSL example. Chapter 3 describes a DSL classification model which is applied later and Chapter 4 shortly discusses textual and Projectional editing approaches. Several Textual Language Workbenches are presented and discussed in Chapter 5, following the example and criteria outlined before (in 2+3) and finally Chapter 6 summarizes and concludes.

2. DSL Overview and DSL Example

DSLs can be divided into different independent dimensions e.g. internal vs. external or textual vs. graphical or tabular.

2.1 DSL Overview

Internal DSL are language extensions built with the language itself and directly embedded into the host language. During recent years, several languages like Ruby, Clojure or the

SPLASH'10 October 17–21, 2010, Reno/Tahoe, Nevada, USA.

classical Lisp got much attention for supporting internal DSLs. On the other hand it requires a good amount of discipline and unfortunately often convenient features like IDE support (code completion, syntax highlighting, cross-referencing, etc) are typically not supported by internal DSLs. Also fluent interfaces [4] support some kind of internal DSL/API, they even can be generated from a higher level abstraction, like an Ecore model as [5] shows.
Another possibility is embedding a DSL into a general purpose programming language (GPL). A classical example is embedded SQL (or LINQ from C# language) for database access from a host language. A major drawback of this approach is the proprietary GPL extension and usually a vendor lock-in, also editing and debugging support is usually not very good (e.g. often some kind of preprocessor or pre-compiler is required).

External DSLs on the other hand are another approach to support the user with a powerful language, adopted specifically for a certain domain (e.g. state machines). Language designers and end users get support for external DSL with textual [6] and graphical [7] concrete syntax. Most of the tools mentioned in this paper, create or deliver the "usual" infrastructure like parser, model-creation and code-emitters (generator) as well as IDE convenience features for free. The main drawback with external DSL is that they start again from scratch and initially lack constructs like flow-control, type systems and other language features usually expected by end-users or users with a GPL background. Figure [1] shows some textual DSL example.

```
//SQL
SELECT firstname, lastName from
  employee where age = 42;

// Regular Expressions
([+-]?[0-9]*) | ([A-Z][a-z]+)

// fluent interface in java

public Collection<Student> findByNameAge (String name, int age, {
  return em.createNamedQuery("Student.findByNameAge")
           .setParameter("name", name) .setParameter("age", age)
           .setFirstResult(1) .setMaxResults(30)
           .setHint("hintName", "hintValue") .getResultList();

}
```

Figure 1. examples of textual DSL

While we focus on textual DSL here, often graphical DSL are also used (interestingly most modeling/DSL initially started graphical). As an example for a non-technical or non-IT DSL see Figure [2] where a graphical DSL for music (basso continuo) is used. It saves writing 80% of the notes and several music notes get derived from the sequence of events or numbers below the bass note. Actually this shows that this particular DSL is a combination of graphical (notes) and textual (numbers, symbols) domain syntax.

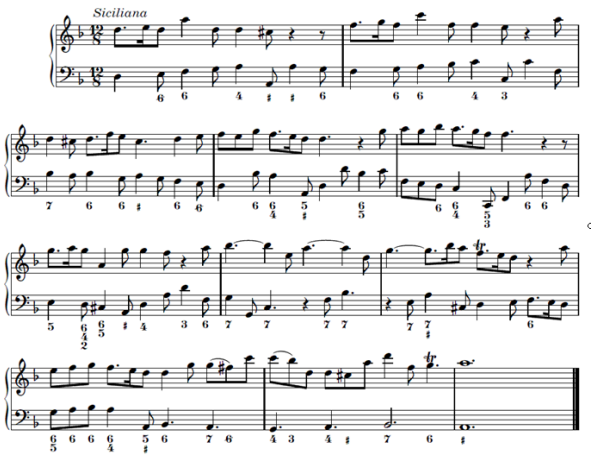
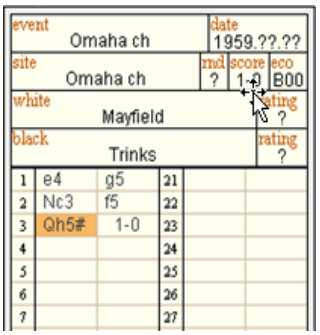


Figure 2. graphical DSL

2.2 DSL Example

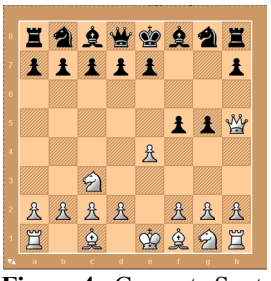
We will present several Textual Language Workbenches [6] in Chapter 5 with a uniform example. It should be a textual language with rules and additional constraints. Chess games often use a textual DSL to exchange and document their moves (e.g. wikis, e-mail, irc etc.). To keep the sample short, we use the shortest tournament game, at the Open Championship of Omaha: Mayfield vs. Trinks. Figure [3] shows the five moves in tabular notation, and subsequently as plain text in two forms (algebraic and spoken move). This is the concrete syntax we want to use for our DSL.




```
// algebraic move
Pe2-e4
// spoken move
pawn at e2 moves to e4
```

Figure 3. DSL notation of chess game

Analyzing the complete game, written in algebraic and spoken moves statements we can derive the underlying meta-model.



```
P e2 - e4
p g7 - g5
Knight at b2 moves to c3
pawn at f7 moves to f5
Q d1 - h5
# 1-0
```

Figure 4. Concrete Syntax (CS) of chess game

The meta model is represented in an abstract syntax (AS) in Figure[5] and consists of a Game, Moves and Pieces.

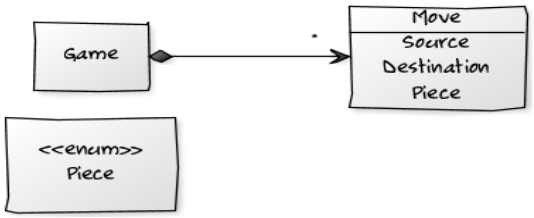


Figure 5. Abstract Syntax (AS) of chess game

3. DSL Language classification

To better compare the textual language workbenches we will first introduce a DSL language classification schema. This schema follows a DSL feature model defined by Langlois [8] and serves for a "as a neutral as possible" comparison. Figure [6] shows that the feature model covers various aspects from a DSL e.g. language, transformation, tooling and process. In this sense we are following the approach of [9] but we compare a different set of textual language workbenches and use a more extensive example.

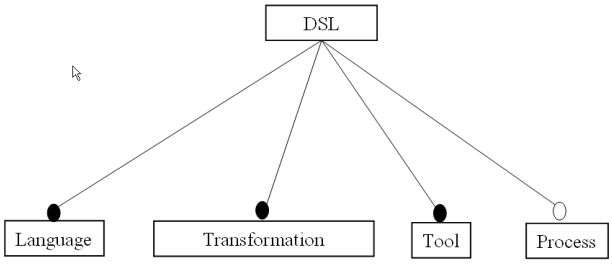


Figure 6. DSL classification model

We focus on the first three criteria as process is optional. An alternative DSL classification model is described in [10]

3.1 Language

The language considers criteria about the abstract syntax (AS) and concrete syntax (CS). Figure [7] show the main parts.

We evaluate which representation is used for the AS (graph or tree), which syntax is used the for definition of the AS (grammar or meta-model), issues about composability and how the AS to CS mapping happens. As we are looking at textual language workbenches only, the representation style of the CS will be text.

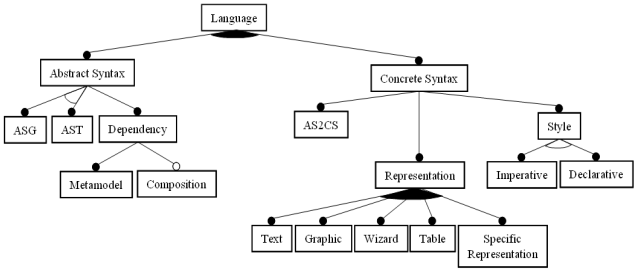


Figure 7. DSL classification model: Language

3.2 Transformation

This section is further divided into aspects about:

- Specification of the Transformation
- Target Asset and
- Operation Transformation

Generally the transformation realizes the correspondence from problem to the solution domain. Because of size restriction we only show Target Asset in Figure [8] as an example.

In Target Asset questions like: which representations of the target asset are possible (text, graphic, binary), and what support of asset update is available (regenerate or incremental) are answered.

In Operation Transformation we look at different transformation techniques, mode for trafo execution (compile vs. interpreted) and also the environment for the trafo (internal or external). Also scheduling, location and automation level are interesting points.

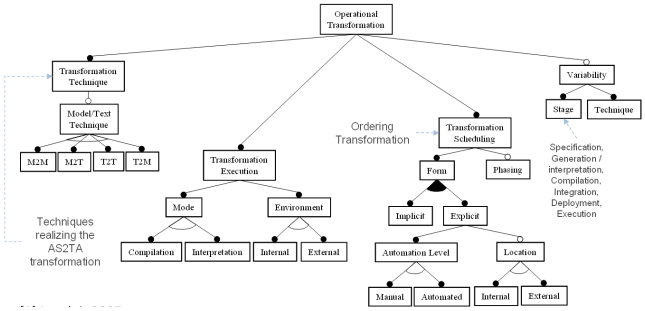


Figure 8. DSL classification model: Transformation

3.3 Tool

This category covers the overall tool support, e.g. what assistance is supported, (static or adaptive), is there process guidance (step or workflow) and what kind of checking is supported (consistency or completeness).

While the mentioned criteria serve as a good comparison catalog other aspects should not be forgotten like documentation, updates, activity of development, support via newsgroups/mail etc. We omitted the Process category as this is often project specific.

4. Textual Language Workbenches

In this chapter we discuss two kinds of Language Workbenches: pure text based (based on the usual scanner/parser approach) and Projectional based with a textual projection.

4.1 Textual Language Workbenches (TLWB)

A number of different TLWB are available, especially on the Eclipse [11] modeling project. Xtext currently is most well-known. It generates an EMF model, a full featured editor and parser from an enhanced EBNF notation. Other examples are TEF (Textual Editing Framework), EMFText and TCS (Textual Concrete Syntax). Most of them use EMF as underlying abstract syntax technology and some kind of scanner/parser technology like ANTLR[12] or RunCC[13]. Especially TCS is interesting as it has a generic editor and interprets the model at runtime, hence avoids as much code generation as possible and enables short turn around cycles during the language development.

4.2 Projectional Language Workbenches (PLWB)

Jetbrains offers an open source solution with MPS (Meta Programming Systems) [14]. Three steps typically define a new language, 1. the "concept" defines the Abstract Syntax (AS), 2. an editor supports the Concrete Syntax (CS) via a cell based editing style and 3. the generator emits new artifacts (e.g. a GPL code like java). The Intentional Language Workbench [15] is a similar solution which is also used for real world projects, especially in the insurance and banking domain, however it is not really widespread and has a higher learning curve for language creators.
Spoofax/IMP [16] is not really a Projectional editor but uses scanner less parsing and also enables arbitrary language composition. In summary the number of PLWB is still relative small but we expect them to increase soon.

5. Workbench Comparison

For creating languages (by a designer) and applying them (by an end user) we need some ingredients. Martin Folwer described this [2] and created the term Language Workbench (LWB).

Essential requirements for LWB are e.g.:
- ability to freely define languages
- which are fully integrated with each other,
- primary source of information should be the abstract syntax,
- DSL is defined by schema, editor and generator
- it can also persist incomplete or contradictory information

Fowler also talks about
- manipulation of the DSL via a Projectional editor

This is a kind of prerequisite for some of the other requirements. While LWB were still in a "visionary stage" in 2005, meanwhile there are plenty of solutions around. Admittedly the variation amongst them is still large but one basic criterion is if it is a text/parser based or a Projectional language workbench. We will discuss pros and cons in the following subchapters.

5.1 Xtext

Xtext [17] is developed actively by a group of developers in itemis' Kiel office and offered as open source. Itemis also offers professional training, consulting and support.
The first version of Xtext was initially developed as part of the openArchitectureWare [18] framework, however after a rewrite of Xtext, it is now a major part of the Eclipse TMF (Textual Modeling Framework) project. Since version 0.7 Xtext is self hosting. The current version 1.0 delivered with Helios is used in lots of projects, also in Eclipse projects itself in B3 [19].
The Xtext supported workflow in shown in Figure[9]. The user starts to describe the AS and CS in the .xtext file. The CS is specified as context-free grammar including the terminal symbols and production rules while the AS is mixed into the same .xtext file and later generated into a corresponding Ecore model. Xtext does not support left-recursive meta-models. Also a corresponding .mwe (Modeling Workflow Engine) file describes necessary build steps like loading the model, run checkers or code generator, transform/layout of generated code. Figure [10] shows the Xtext grammar with our Chess DSL.

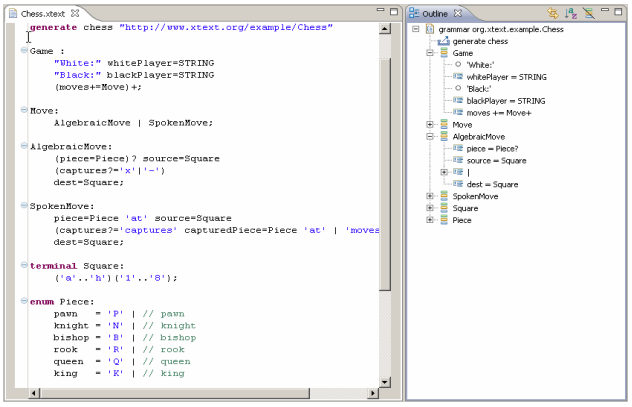


Figure 9. Xtext workflow

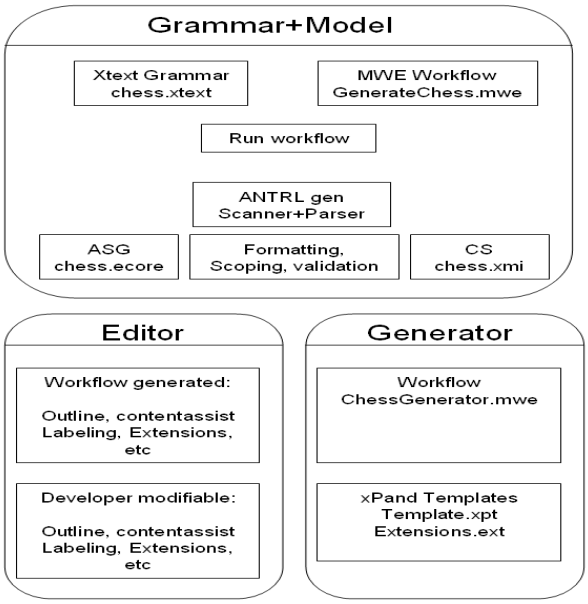


Figure 10. Xtext: Concrete Syntax and Abstract Syntax

From the .xtext file, Xtext generates a ASG (.ecore file), an ANTLR based scanner and parser for DSL-to-model transformation, model-to-text generator with Xpand (.xpt) support and a full fledged editor the DSL based on eclipse. The generated editor supports nearly all features you are used from a textual editor and could compete with the eclipse JDT java editor (without the refactoring support). The editor supports syntax highlighting, code completion, navigation and reference, folding,

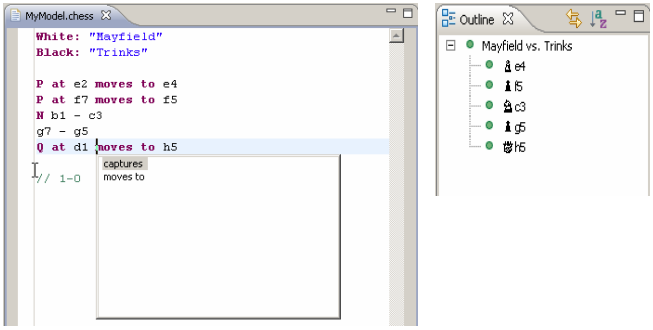


Figure 11. Xtext: Chess DSL Editor

bracket matching, styled label providers, incremental codegen, and much more. Xtext also supports qualified name support and referencing existing java elements from your DSL. Figure [11] shows the resulting Xtext Chess DSL editor.

5.2 TEF (Textual Editing Framework)

TEF (Textual Editing Framework) [20] was initially developed by Markus Scheidgen during is PhD at the University of Berlin. For the concrete syntax (CS), TEF provides a syntax definition language called TSL (textual syntax language). TSL describes the textual notation for an existing Ecore meta-model (AS) is in the .etslt file. Via the usual Eclipse EMF facilities (the gen-model) the necessary EMF support is generated. For DSL-to-model transformation, TEF creates a RunCC parser which is interpreted at runtime (RunCC avoids code generation).
Figure[12] show the TEF workflow and relevant artifacts.

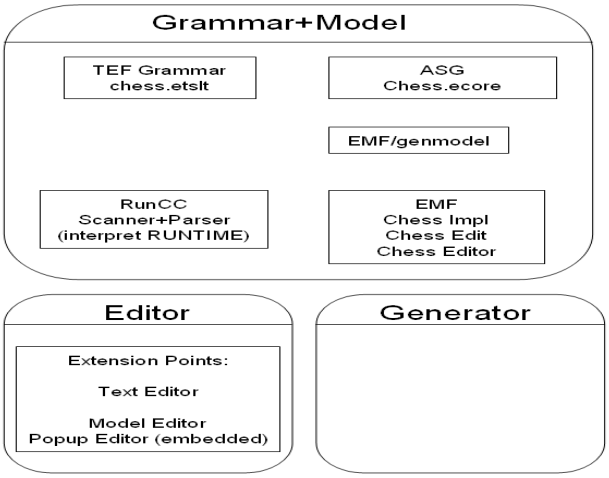


Figure 12. TEF workflow
Figure[13] show the etslt description (CS) for the Chess example. Note that we reference the AS in a Ecore model.

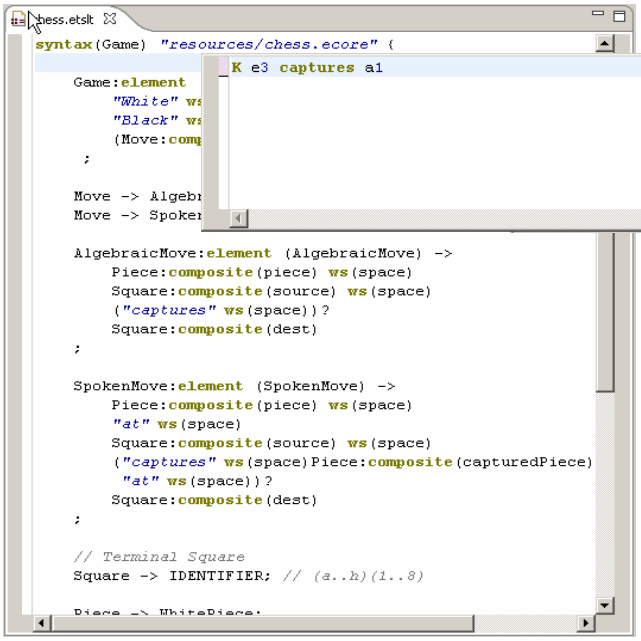


Figure 13. TEF Concrete Syntax
TEF generates three different editors via eclipse extension points:

- a textual editor

This editor parses textual models and allows editing them in a comfortable way. Features of the generated editor are outlined below. Figure[14]

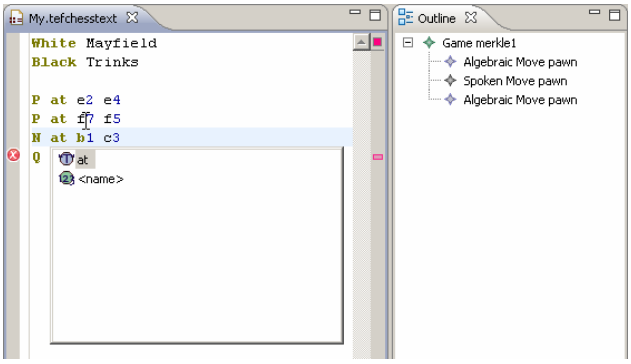


Figure 14. TEF: textual editor with Chess DSL

- a model based editor

The model based editor acts like an enhanced generic Ecore editor. Initially it is a tree based editor but other representations are also possible. Figure[15]. There is no text parsing involved here as the editing "style" does not allow it.

- a embedded editor

This is a textual editor embedded into the model based tree editor. On each model element the user can open a textual editor with a hotkey (Alt-T). Figure[15]. TEF hence combines different editing styles (treebased/textbased). Depending on the situation, the best editor is offered for the user (we call this "convergent" editor).

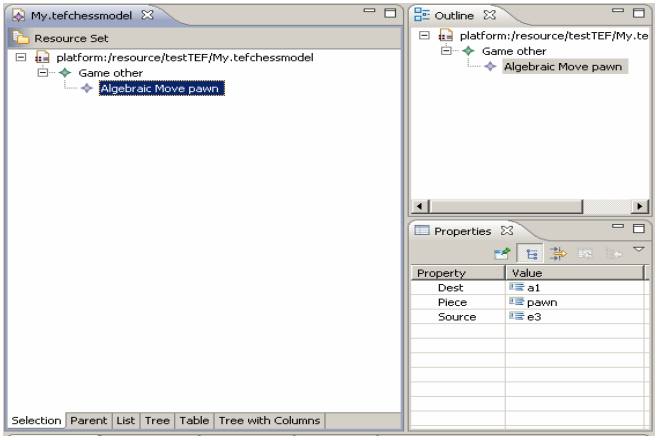


Figure 15. TEF: model editor and embedded text editor

The generated editor supports syntax highlighting, code completion, navigation and reference, folding, error annotation and several other features. Model validation is possible via the Eclipse Modeling projects.

5.3 TCS (Textual Concrete Syntax)

TCS (Textual Concrete Syntax) [21] was developed by Frederic Jouault at EMN (Ecole des Mintes in Nantes) and the ATLAN-Mod team. It is also part of the eclipse TMF project and used in other eclipse projects, e.g. ATL2 [22]. TCS is also self hosting.

Figure[16] show the TEF workflow and relevant artifacts.

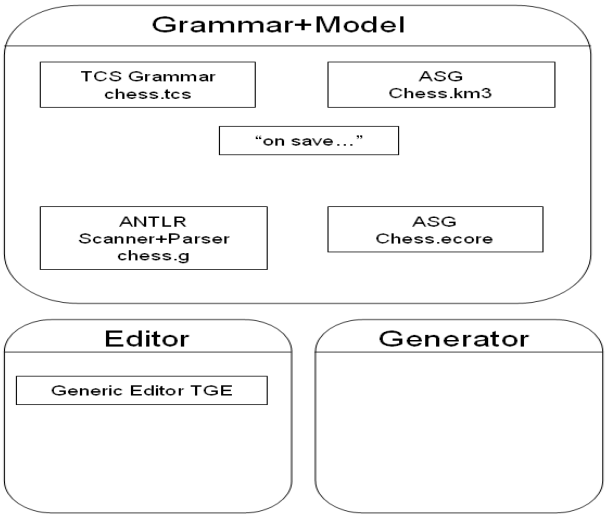


Figure 16. TCS workflow

The abstract syntax is (also) specified in a textual language for meta-modeling, called KM3 [23]. When saved, TCS then generates a corresponding Ecore file on the fly.

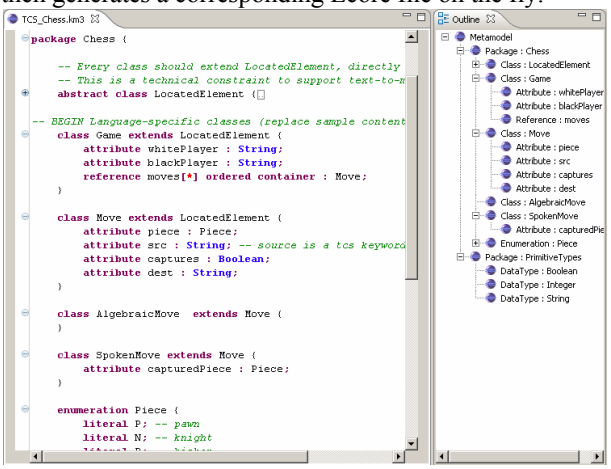


Figure 17. TCS: Abstract syntax

The concrete syntax is specified in a .tcs file. Again, when saved a corresponding ANTLR parser is generated on the fly. TCS hence avoids lots of (unnecessary) code generation often found in other tools.

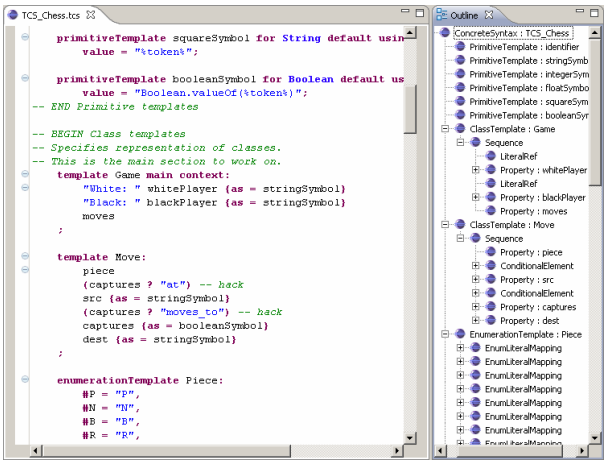


Figure 18. TCS: Concrete syntax

TCS avoids as much code generation as possible and hence allows very fast and short turn around cycles. There is no need to start an additional embedded eclipse instance with the plugin, instead everything is updated "on save" and then reinterpreted.

To edit and create DSL conforming to the one the user specified, TCS offers the Textual Generic Editor (TGE). TGE supports syntax-highlighting, text hovers, hyperlinks, and an outline view for every language that has its textual syntax specified in TCS. If necessary, TGE can be further customized to specific needs/layout. Model validation is possible via the Eclipse Modeling projects or preferably via the ALT language directly.

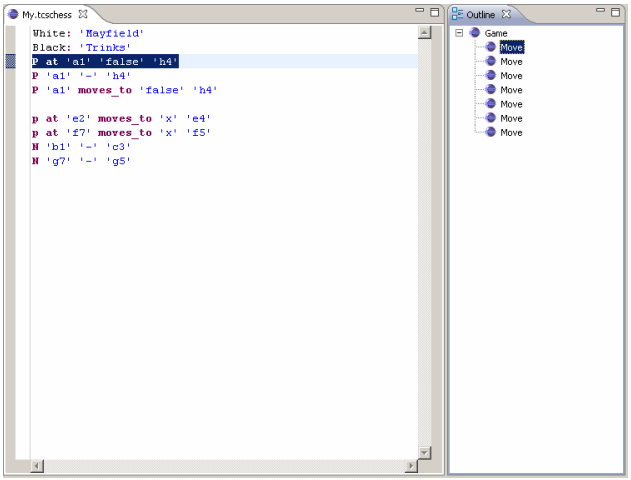


Figure 19. TCS: Generic Editor (TGE) with Chess DSL

TCS supports a language Zoo with over 50 languages on their website. TCS is also reused by Furcas, another TMF tool.

5.4 EMFText

EMFText [24] was initially developed as part of the resuse-ware composition framework [25] at University Dresden. It was later extracted into an own, independent tool. Similar to TCS also EMFText allows specifying a concrete syntax for an existing EMF model (abstract syntax). Figure[20] shows the main workflow. Note that some of the workflow/ANT properties can be specified in the .cs (concrete syntax) file already (e.g. reload properties).

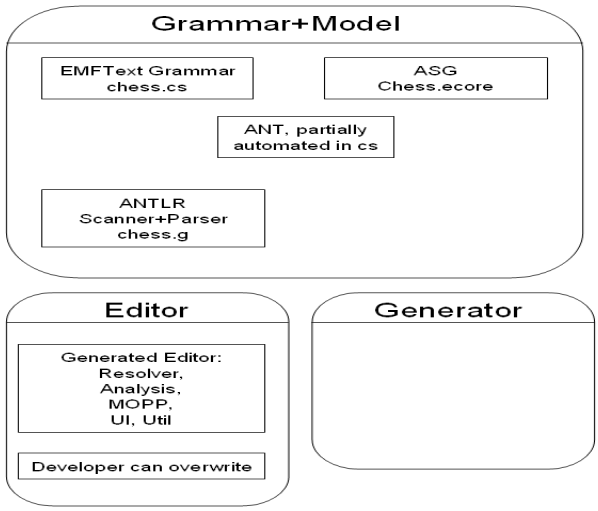


Figure 20. EMFText: workflow

To enable EMFText to use models at runtime, a EMF model plug-in must be generated (following the well-known gen-model).

The concrete syntax specification (.cs file) consists of 3 blocks:

- A configuration block, which contains the name, the base model and the root Meta class (start symbol).
  Optionally other syntaxes and metamodels can be imported and generation options can be specified.
- A (optional) TOKEN section.
  tokens for the lexical analyser can be specified.
- A RULES section, which defines the syntax for each concrete Meta class.

EMFText has some special support for the syntax definition:

- Automatic generation of default syntaxes
- Modular specification
  (Support for abstract syntaxes and syntax imports)
- Default reference resolving mechanisms
- and comprehensive syntax analysis to warn about potential syntax problems

The concrete syntax for the Chess Example is shown in Figure [21].

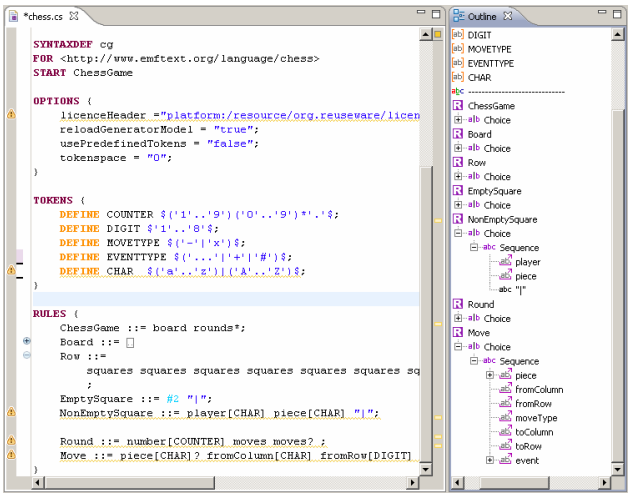


Figure 21. EMFText: Concrete Syntax

Via the build process a default editor is generated by EMFText. Developers can overwrite or customize special behavior. Out of the box the editor supports several IDE features like outline view, customizable syntax highlighting (also via the .cs file), code completion, bracket handling, text hovers and the usual hyperlink and reference support. Figure [22] shows the EMFText Chess.

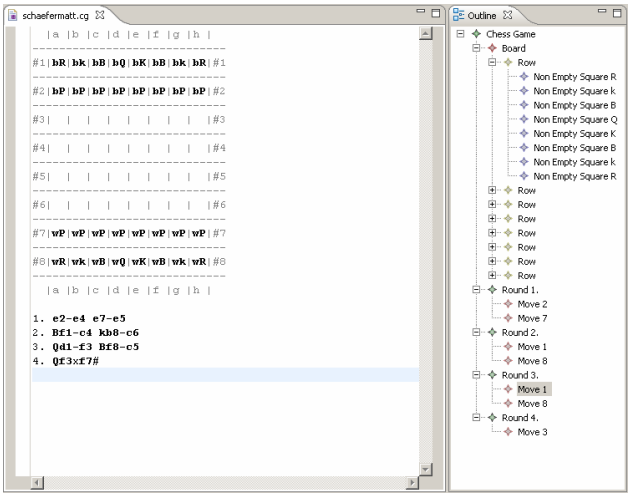


Figure 22. EMFText: Editor with Chess DSL

EMFText supports also a language Zoo with about 50 languages, with real world languages like Java5 or .e.g. a textual Ecore syntax.

5.5 Other text based approaches

On the eclipse platform there are several other interesting text based approaches which should be mentioned like IMP[26], Spoofax/IMP [27] (a project at the university Delft) or ETMOP and CAL [28] (Andrew Eisenbergs PhD.)

5.6 MPS (Meta Programming Systems)

Jetbrains offers an open source solution with MPS (Meta Programming Systems) [14]. Unlike the previous introduced parser based approaches of Xtext, TEF, TCS and EMFText, MPS is a Projectional editor. This approach also follows the "language oriented programming" idiom described in [29].
MPS offers a projection from the Abstract Syntax Tree (AST) to Text, however under the hood the user edits (indirectly the AST). Editing the tree as opposed to "real text" needs some accustomization. Without specific adaptations, every program element has to be selected from a drop-down list and "instantiated". However, MPS provides editor customizations to enable editing that resembles modern IDEs that use automatically expanding code templates. So the user does not really feel that he is editing an AST. Using the Projectional approach avoids a lot of problems like scanning/parsing, refactoring support etc.

Three steps typically define a new language, 1. The "concept" or "structure" defines the Abstract Syntax (AS), 2. An editor supports the Concrete Syntax (CS) via a cell based editing style and 3. The generator emits new artifacts (e.g. a GPL code like java).
Within MPS there is direct support to use the generated Editor or Generator. CTRL-F9 generated/compiles e.g. the DSL Editor and reloads in on the fly. There is no need to start an additional instance (like the eclipse plugins). MPS also supports important features like Constraint checking, support for real type system, etc. The Workflow is depicted in Figure [23]. MPS is self hosting and used for several real world products (e.g. the youtrack MPS bug tracker[30]) and was developed and used for about 7 years internally within Jetbrains. The MPS product is now open sourced.

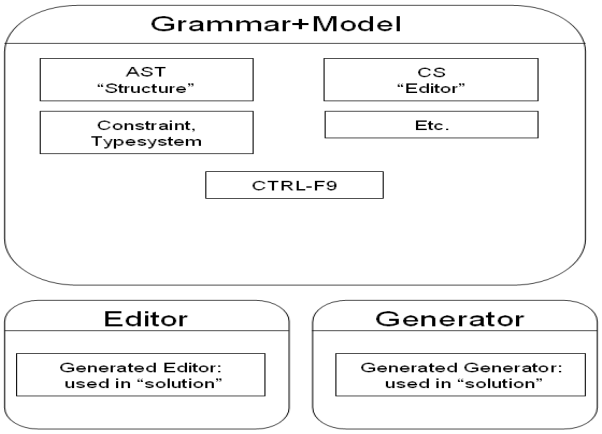


Figure 23. MPS: Workflow

Defining a new language start with the abstract syntax which is called a "concept", located under the "structure" node. As shown in Figure [24], MPS uses also a textual syntax to describe the AS. (n.b. This editor is also described with MPS (self hosting)).

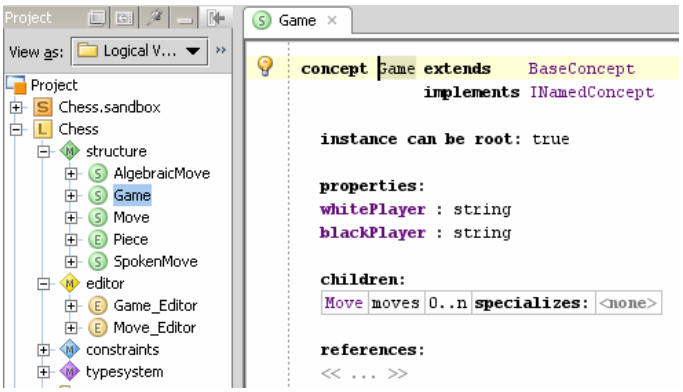


Figure 24. MPS: Abstract Syntax

Next, the concrete syntax has to be defined in an "editor". As MPS uses the Projectional approach there is no parser/scanner. Editing is only based on "cell-editing", hence the programmer describes the cell layout (horizontal/vertical list/collection etc) and the mapping to the abstract syntax. The concrete syntax is shown in Figure [25]. Note that other, alternative projections (e.g. to table, graphs, spreadsheets etc) would be possible.

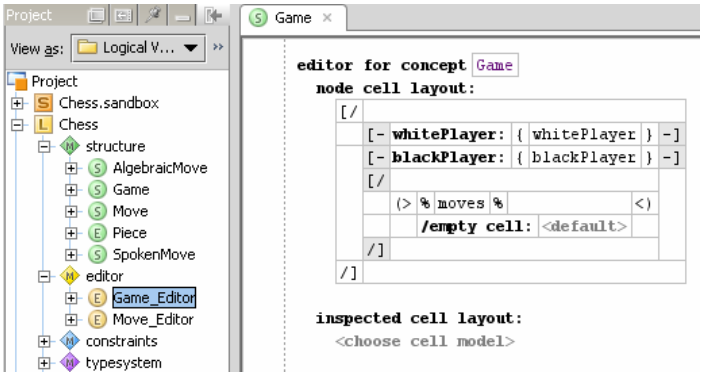


Figure 25. MPS: Concrete Syntax

Finally we can generate a DSL Editor with CTRL-F9 for our Chess language in MPS (Figure [26]) and create the hess game.

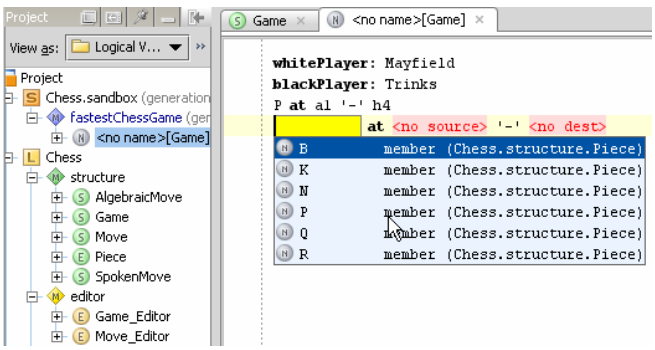


Figure 26. MPS: DSL Chess editor

6. Conslusion / Summary

In this paper we presented several different language workbenches for textual DSL. Using a feature model for DSL one can compare them on a "neutral" and unbiased platform. However this is not to declare a winner or the "best" textual language workbench.

The number of parser based textual language workbenches is significant in the meantime and eclipse seem to be the common host. However in terms features and approaches we were able to identify several differences (e.g. pure generation based approach of Xtext vs. a generic, interpreted approach of TCS).

While Projectional editors are currently still the exception (MPS and Intentional), we assume that they gain a huge increase of use. The combination of modular languages, different DSLs combined is much easier with Projectional editor and classical parser based approaches will reach their limits. Currently there is not yet a real projectional editor for eclipse but we are sure that this is already on the roadmap.

Acknowledgments

Special thanks go to the developers from Xtext, TEF, TCS, EMFText and MPS. They were very helpful with my questions and bug reports for their tools. Thank you again!

References

[1] Announcing The Emerging Languages Camp at OSCON http://radar.oreilly.com/2010/05/announcing-the-emerging-langua.html

[2] Fowler, M.: Language Workbenches - The Killer-App for Domain Specific Languages?

http://martinfowler.com/articles/languageWorkbench.html

[3] Fowler, M.: Projectional Editing

http://martinfowler.com/bliki/ProjectionalEditing.html

[4] Fowler, M.: Fluent Interfaces

http://www.martinfowler.com/bliki/FluentInterface.html

[5] Garcia, Automating the embedding of Domain Specific Languages in Eclipse JDT

http://www.eclipse.org/articles/Article-AutomatingDSLEmbeddings/index.html

[6] Textual Modeling Tools for ecliplse

Xtext http://ww.eclipse.org/Xtext/

TCS: www.eclipse.org/gmt/tcs/

TEF:http://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tool.html

EMFText: http://emftext.org

[7] Meta Edit http://www.metacase.com

[8] B. Langlois, C.E. Jitia, E Jouenne: DSL Classification. In 7th OOPLA Workshop on Domain-Specific Modeling, 2007

[9] M. Pfeiffer, J. Pichler A Comparison of Tool Support for Textual Domain-Specific Languages, In 8th OOPSLA Workshop on Domain Specific Modeling, 2008

[10] T. Goldschmidt, S.Becker, A. Uhl: Classification of Concrete Textual Syntax Mapping Approaches, In ECMDA-FA 2008

[11] www.eclipse.org/modeling

[12] www.antlr.org/papers

[13] runcc.sourceforge.net

[14] www.jetbrains.com/mps

[15] Intentional Software, Intentional Domain Workbench, http://intentsoft.com/technology/IS_OOPSLA_2006_paper.pdf

[16] L.. Kats, E. Visser. The Spoofax Language Workbench. Rules for Declarative Specification of Languages and IDEs. In OOPSLA 2010

[17] Xtext http://ww.eclipse.org/Xtext/

[18] openArchitectureWare http://www.openarchitectureware.org/

[19] eclipse B3: http://www.eclipse.org/modeling/emft/b3/

[20] TEF: http://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tool.html

[21] TCS: www.eclipse.org/gmt/tcs/

[22] ATL: www.eclipse.org/m2m/atl

[23] KM3: F. Jouault, J. Bezivin: KM3: a DSL for Metamodel Specification, Formal Methods for Open Object-Based distr. Systems 2006

[24] EMFText: http://www.emftext.org

[25] Reuseware: http://www.reuseware.org/

[26] Eclipse/IMP (Safari project) http://eclipse-imp.sourceforge.net

[27] Spoofax/IMP http://strategoxt.org/Spoofax

[28] CAL, http://www.cs.ubc.ca/~ade/research.html

[29] S. Dmitriev: Language Oriented Programming: The Next Programming Paradigm

[30] Jetbrains youtrack bugtracker

http://youtrack.jetbrains.net/dashboard