# Declare Your Language

Eelco Visser

TU Delft

June 12, 2015 – June 14, 2015

This book is on github at https://github.com/MetaBorgCube/declare-your-language

## Contents

# Preface

This is a book about declarative language definition with the Spoofax Language Workbench. The aim of Spoofax is to separate the various concerns of language definition and implementation and provide high-level declarative meta-languages for each concern. These meta-languages are declarative in the sense that they abstract from the *how* of language implementation and focus on the *what* of language design. For example, "what is the syntax of my language?", instead of "how do I implement a parser for my language?". Thus, a language designer should not be distracted by language implementation details.

This is a book in progress, just as Spoofax itself is very much in progress. (Although Spoofax is much further along than this book :-) Spoofax is the product of an ongoing research project that investigates the nature of software language definition. We are continuously experimenting with better abstractions for various concerns. As a result, new features are being added to Spoofax and we are also always working on the guts of the workbench. Therefore, working with Spoofax can be a somewhat rocky experience.

Indeed, while I start writing this book in June 2015 using the nightly-build descendant of Spoofax 1.4, Spoofax is about to enter a new era. Within the coming months, we are planning to switch to a new version of Spoofax based on a complete overhaul of the internal architecture of the workbench. We are replacing IMP, the framework that provided the binding of Spoofax meta-languages to Eclipse, with Spoofax Core, a framework that defines IDE services independently from Eclipse. This will allow us to target other

IDE containers such as NetBeans and IntelliJ (modulo engineering work) as well as support robust command-line implementations of languages defined in Spoofax. So there is a risk that some of the text of this book and example project that comes with it, will have to be rewritten some.

This book and the accompanying Spoofax projects are on github. I will consider pull requests with minor or major contributions to the book and accompanying projects.

– Eelco Visser

June 14, 2015

Chapter 1

# Declarative Language Definition

This chapter examines the philosophical background of language workbenches in general, and Spoofax in particular. If you just want to get on with it and start building a language definition with Spoofax, just proceed to Chapter 2.

## 1.1 Software Languages

programming languages

expressing computation

development from directly programming machine code to high-level abstractions

not all languages used in software development can be characterized as programming languages

For example, HTML is clearly a language

data formats

domain-specific language

the term ‘software language’ captures this broader class of languages

Considering all these languages is useful since they share quite some

commonality

All software languages are characterized by a formal syntax. That is, the well-formed sentences of the languages can be formalized using a formal system. This can be a (context-free) grammar, but may also be a recognition algorithm.

Furthermore, the structure assigned to well-formed sentences is assigned a meaning. The semantics of the language

## 1.2 Understanding Software through Linguistic Abstraction

pragmatic goals

modeling

capturing domain understanding

## 1.3

internal, external, embedded languages

leaky abstractions

## 1.4 Language Workbenches

what is a language workbench

integrated tool for

- parsing
- abstract syntax tree representation
- static analysis
- code generation (compilation)
- interpretation

- editor services

while the functionality of a language workbench sounds pretty standard, there is a range of different approaches

different language implementation philosophies

long tradition of tools to assist language developers

**LEX/YACC** compiler-compiler

**Centaur**

**ASF+SDF MetaEnvironment**

**JastAdd**

**Xtext**

**MPS**

**Spoofax**

**Rascal**

**SugarJ**

### 1.5 Declarative Language Definition

declarative meta-languages

capture our understanding of the domain of language definition

separation of concerns

syntax, name binding, type constraints, dynamic semantics, source-to-source transformations

tension between declarative expression and programmability

Chapter 2

# Getting Spoofax

In this chapter we discuss how to install Spoofax and setting up language projects.

## 2.1 Eclipse + Spoofax

Spoofax is an Eclipse plugin. The regular way to install an Eclipse plugin is to use its update site to add the plugin to an existing Eclipse installation. However, Spoofax requires a few tweaks to be applied to the Eclipse configuration and it requires the separate installation of Java 7 or later. To avoid all this hassle, Spoofax is now also distributed as a complete Eclipse installation with Spoofax pre-installed and all configurations set corectly. The download page

`http://metaborg.org/download`

provides a link to the integrated distributions. Note that this distribution is currently only available for the bleeding edge continuous build version of Spoofax.

Download the `spoofax-<os>-<arch>-jre.zip` for your computer's operating system and architecture, unzip, and launch the Eclipse application inside.

The first thing that Eclipse will ask is which Workspace to use. The Workspace is the default directory where projects are created. Just create a new directory with an appropriate name (e.g. `Workspace-Spoofax`) in an appropriate location in your file system.

The first thing that I do when installing a new Eclipse is changing its appearance. In the Eclipse menu choose `Preferences`. In the dialog window go to `General > Appearance` and choose theme `Classic`. This is completely optional though.

Another setting that is useful to adjust is that for refresh. In the search box in the Preference dialog type 'refresh'. Under `Startup and Shutdown` select `Refresh workspace on startup`. Under `Workspace` unselect `Build automatically` and select `Refresh on access`.

The default font size is configured to be 11pt, which is too small for my eyes. Adjust the font size in the `Preferences > General > Appearance > Colors and Fonts`, and there select `Basic > Text Font` and choose something appropriate. I find 14pt Monaco to work out pretty well.

**JRE Error**:
At the first time that I try out the Eclipse with pre-packaged JRE, I get the following error:

> 'Update Installed JREs' has encountered a problem. Resource '/.org.eclipse.jdt.core.external.folders' already exists.

After ignoring the error, Spoofax appears to work fine. It is not clear at this point whether that is due to the fact that I had already installed JRE7.

### 2.2 Creating a Project

To start a new language with Spoofax you need to create an Eclipse project.

In the `File` menu select `New > Project ...`. In the dialog window select `Spoofax editor project` and hit the `Next >` button. This presents the `Spoofax Editor Project` wizard dialog in which you should indi-

cate the name and file extension of your project and language. This information is used to instantiate all the files that are needed in a Spoofax project.

The wizard will create a project directory in your Workspace with the following properties:

**Project name** This will be the name of your project and the directory that contains it.

**Language name** This is the name of your language, which means that it will be used as the basis for several file names.

**Plugin ID and package name** This is the name of the Java package and plugin that is generated from your project.

**File extensions** This is the file extension that the program files in your language will have

**Generate .gitignore file** Of course you will maintain your project's version history in git. Check the box so that generated code that needs not to be versioned is ignored by git.

**Generate minimal project only** Check this box to start with a fresh language. In this book I will walk you through building the various elements that you need for your language.

Choosing the name for your language is important. Unfortunately, the name that you choose will be hardwired at many places in your project. Therefore, **renaming** your project and language afterwards **is virtually impossible**. The usual way to achieve a renaming is to create a new project with the right name and manually copy over the files from your old project.

## 2.3 Importing an Existing Project

Another way to use Spoofax is to import an existing project. The github repository for this book provides a series of example projects which

are the basis for the text in the book. To use those projects check out the git repository and import the projects into Eclipse as follows. In the `File` menu select `Import ...`. In the import dialog select `General > Existing Project into Workspace`. Browse to the `declare-your-language/languages` directory and select a project to import (or select the entire directory, which will allow you to import all projects at once). In the `Projects:` are select all projects that you want to import. Hit the `Finish` button.

Importing a project will add it to your workspace without copying it to the Workspace directory.

You can remove a project from your workspace using `Edit > Delete`. This will not remove the files from the file system, unless you select that option using the check box.

### 2.4 Structure of a Spoofax Project

The Spoofax Editor Project wizard generates a complete Eclipse plugin project for your language. All that is left to do is fill in the language-specific bits. That is great, but the sheer number of directories and files in a project may seem rather overwhelming. However, it is not all that bad. You can ignore most of this stuff, certainly at first. Let's have a look what the wizard has generated.

- `.cache/`: A cache of intermediate results produced by processing the language definition. You should never have to look at this.
- `.externalToolBuilders/`: Automatically generated Ant files for building stuff.
- `.settings/`: Some Eclipse settings
- `editor/`: This directory contains `*.esv` files, which configure various aspects of the IDE for your language. Here you can change the color that syntax highlighting gives to certain tokens of your syntax, or define the outline view for programs in your language.

A basic definition of these configurations is generated automatically, so you can ignore this for now. But we will get back here.

- `editor/java`: This sub-directory of the `editor/` directory is unrelated to the configuration files. It contains some project-specific Java code inserted by the wizard that binds it to the language-independent Spoofax framework. The directory is also used as target for Java code generated from the DynSem meta-language for dynamic semantics. And later on in the book we will add some glue code for initialization of DynSem-based interpreters.

- `icons/`: This is where icons to be used in the outline view are stored. It is empty by default.

- `include/`: This directory contains files that are generated from the syntax definition for the language. (Eventually these files will end up in the `src-gen` directory.)

- `lib/`: This directory contains the common run-time library for Spoofax projects. (Eventually this should be a binary dependency.)

- `META-INF/`: This directory contains the manifest with configuration information for building the Eclipse plugin.

- `src-gen/`: This directory contains code generated from the language definition.

- `syntax/`: This is where one typically puts the modules making up the syntax definition. In today's Spoofax, syntax definitions are defined using the SDF3 syntax definition formalism.

- `target/`: This directory contains the class files resulting from the compilation of Java code.

- `trans/`: This is where one usually puts the transformations defining the non-syntactic aspects of a language definition, including source-to-source transformations, interpretation, and code generation. All these aspects used to be defined using the Stratego transformation

language. However, we will see that name binding and type checking are now done using the NaBL and TS meta-languages, and that operational semantics is defined using the DynSem meta-language.

- `utils/`: This directory contains Spoofax Java libraries.

Then there are some files at the top level of the project:

- `.classpath`: The Java class path for the project
- `.gitignore`: A specification of the (generated) files that can be ignored by git version management.
- `.project`: The file that makes the project directory into an Eclipse project.
- `build.properties`: Some parameter bindings for the Ant build that determines some of the directories above, and where they could be changed. But we will just stick to the standard layout.
- `build.generated.xml`: The generated Ant build file that defines the tasks for compiling language definitions.
- `build.main.xml`: The project-specific Ant build file that binds the generated build file to the project-specific properties. The file is instantiated by the wizard and is one of the places where your language name gets used.
- `plugin.xml`: Configuration of the Eclipse plugin for your language.
- `pom.xml`: A Maven file. In the next version of Spoofax (see Preface), building and dependency management will make heavy use of Maven.

In summary, only the `editor/`, `syntax/`, and `trans/` directories contain language-specific code. The other directories contain either standard Spoofax code that is copied into the project or code that is generated from language definitions.

## 2.5 Language Concerns

The structure of a Spoofax project does not reveal the conceptual structure of a Spoofax language definition. In the rest of this book we will be primarily be studying the definition of aspects of a language using declarative meta-languages. We distinguish the following concerns:

**Tests** As with any form of software development, developing a test suite is useful as a partial specification and for catching regressions. In Chapter 3 we study the SPT testing language.

**Syntax** A syntax definition describes the syntactically well-formed sentences (programs) of a language *and* the structure of these programs. Since all other operations on programs are driven by this structure, syntax definition are the corner stone of language definitions in Spoofax. We will study syntax definition in SDF3 in Chapter 4.

**Transformation** The parser derived from a syntax definition turns well-formed programs into abstract syntax trees. A wide range of semantic manipulations of programs can be expressed as transformations on abstract syntax trees. In Chapter 7 we will study the definition of basic transformations such as desugarings using the Stratego transformation language. In early versions of Spoofax, all semantic concerns were adressed using Stratego. In recent years we have been working to add higher-level languages that capture our understanding of particular aspects of semantics specification.

**Names** Abstract syntax trees do not take into account the graph structure induced by names in programs. Names are the key technique to facilitate abstraction in programming languages. Name resolution is concerned with resolving uses of names with declarations of names. In most tools, name resolution requires a programmatic encoding of the name binding rules of a language. In Chapter 5 we will study the definition of name binding rules using the NaBL name binding language, which abstracts from the implementation of name resolution algorithms.

**Type Constraints** Many languages apply restrictions to the set of programs that is considered valid beyond the syntactic well-formedness constraints imposed by a grammar. Such restrictions are typically formalized in terms of a type system. In Chapter 6 we study the formalization of such constraints using the TS type system specification language.

**Dynamic semantics** Language workbenches traditionally use code generation to define the semantics of a language. For many scenarios that is the appropriate thing to do. However, the definition of a code generator often obscures the intended dynamic semantics of a language. Thus, morally, it is a good idea to specify the dynamic semantics of a language directly. Such a specification can then be used to reason about the correctness of the code generator. Going further, the specification of the dynamic semantics may be interpreted directly to produce an *interpreter* for the language. In Chapter 8 we study the DynSem DSL for the specification of the dynamic (operational) semantics of programming languages.

Chapter 3

# Testing

**Chapter 4**

# Syntax

Spoofax takes a *syntax first* approach to language definition.

The syntax

We will illustrate the material in this book with a series of Spoofax projects names `LangA`, `LangB`, `LangC`, etc.

In this project

```
module
context-free syntax
  Exp.Var = ID
  Exp.Mul = [[Exp] * [Exp]] {left}
  Exp.Add = [[Exp] + [Exp]] {left}
context-free priorities
  Exp.Mul > Exp.Add
```

Chapter 5

# Names

Chapter 6

# Type Constraints

Chapter 7

# Transformation

Chapter 8

# Semantics

# Bibliography

**Appendix A**

# Cheat Sheets

cheat sheets for the meta-DSLs