There's kind of a drop&drag interface.

– Eelco Visser

9 Interactive Transformation and Editing Environments

Many programmable software transformation systems are based around novel domain-specific languages (DSLs) with a long and successful history of development and deployment. Despite their reasonable maturity and applicability, these systems are often discarded as esoteric research prototypes partly because their languages are frequently based on less familiar programming paradigms such as term and graph rewriting or logic programming, and partly because modern development environments are rarely found for these systems. The basic and expected interactive development aids, such as source code navigation, searching, content completion, real-time syntax highlighting and error checking, are rarely available to developers of transformation code.

This chapter describes Spoofax, an interactive development environment based on Eclipse for developing program analyses and transformations with Stratego/XT. The chapter illustrates how the new language and system abstractions introduced in Part III and Part IV of this dissertation are useful when constructing interactive editing and transformation environments. Spoofax provides, in addition to the aids mentioned above, a code outliner and incremental building of projects. This significantly eases the development of language processing tools using Stratego/XT. Moreover, Spoofax is extensible with scripts written in Stratego that can be executed within Eclipse and allow live analyses and transformations of the code under development.

This chapter is based on the the paper *"Spoofax: An Interactive Development Environment for Program Transformation with Stratego/XT"*, written together with Eelco Visser [KV07b].

9.1 Introduction

Developing and maintaining frameworks and libraries is at the core of any modern software development project and the development of transformation programs is no different. When the code size creeps over a certain limit, it becomes difficult to keep track of, and navigate the source, without reasonable editor support. Unfortunately,

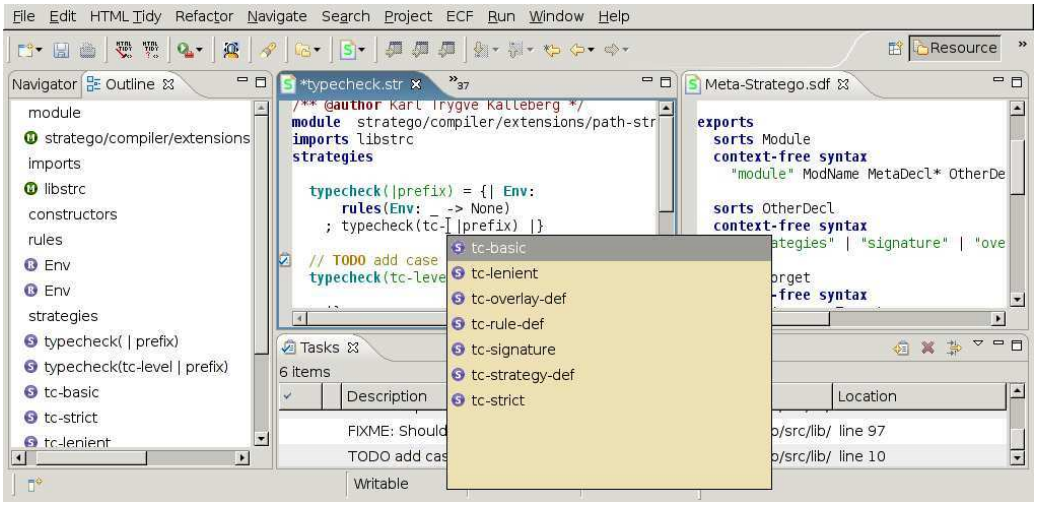


Figure 9.1: Screenshot showing Syntax Definition Formalism (SDF) and Stratego editors with outline view.

most editing tools for domain-specific software transformation languages offer little assistance for editing larger programs. The basic and expected interactive development aids, such as source code navigation, content completion, syntax highlighting and continuous error checking, are rarely available to developers of transformation code.

This lack of development aids keeps the entry barrier for new developers high; DSLs for program transformation use their own syntax and language constructs which may be unfamiliar to many. In addition, most editing environments support these languages rather poorly, providing only limited syntax highlighting. Even skilled developers may be less effective because errors are reported late in the edit-compile-run cycle; that is, only after compiling. It is generally held that errors should be reported immediately after a change has been made while the human programmer is still in a relevant frame of mind. Also, error reporting should ideally be customisable and check project-specific design rules, where possible. This problem also exists with Stratego/XT. Until recently, a good editing environment did not exist for Stratego. This made development with Stratego/XT harder than necessary.

This chapter describes Spoofax, an extensible, interactive environment based on Eclipse for developing program transformation systems with Stratego/XT. Spoofax supports Stratego/XT by providing modern development aids such as customisable syntax highlighting, code outlining, content completion, source code outlining and navigation, automatic and incremental project rebuilders.

Spoofax is a set of Eclipse plugins – a Stratego and an Syntax Definition Formalism (SDF) editor, a help system and the Stratego/J interpreter from Chapter 6.

It supplements Stratego/XT, which must be installed separately, by providing an extensible, interactive development environment. Figure 9.1 shows an example session with an SDF editor (top right), a Stratego editor (top middle), a list of pending tasks extracted from all project files (bottom), and a code outline view (left) displaying all imports, rules and strategies defined in the edited file. The popup is a content completer showing alternatives for the `tc-` prefix.

In this dissertation, the contributions of the Spoofax environment include user extensibility with scripts written in Stratego that allow live analyses and transformations of the code under development; syntax highlighting, navigation and content completion that eases the learning curve for new users of Stratego; and, integration into a mainstream tools platform that is familiar to developers and that runs on most desktop platforms.

9.2 Core Functionality

The Spoofax environment is built around a program model of a Stratego project. This model is called a *build weave* and is discussed below. An important task of the core functionality of Spoofax is to maintain this build weave as users and tools modify the various artifacts that make up the Stratego project including: Stratego files, syntax definitions and build files. Another task of the core functionality is to provide support for user preferences and project-specific settings, and make certain that these are saved across editing sessions in the Eclipse preference store.

9.2.1 Architecture

As depicted in Figure 9.2, Spoofax is divided into a handful of separate components called plugins. Each plugin provides a specific piece of functionality. Some plugins depend on others as indicated in the figure by directed arrows. The full composition of all plugins provides the Spoofax "feature"; that is, an Eclipse-specific term for a set of plugins that together provide a well-defined tool or application.

The editor plugin (`org.spoofax.editor`) provides the interactive editors for Stratego and SDF as well as other interactive capabilities such as configuration menus and various views (shown later). The Stratego/J plugin (`org.spoofax.interpreter`) provides the execution engine for running all the user-provided Stratego scripts. The jsglr parser plugin (`org.spoofax.jsglr`) is used to produce ASTs from the editor buffers. Both the compiled scripts and the ASTs are represented as terms using a slightly modified version of the ATerm [vdBdJKO00] library (`org.spoofax.aterm`). The plugin `org.spoofax.help` provides a manual for Stratego/XT. It may be accessed and read through the Eclipse help system.

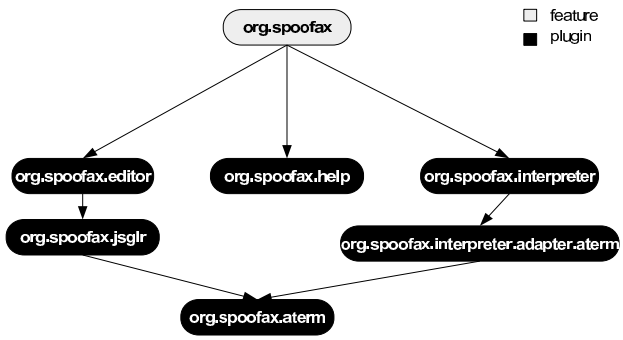


Figure 9.2: Relationship between the plugins that make up Spoofax.

9.2.2 Build Weave

The build weave is essentially a module dependency graph. Each graph node corresponds to a Stratego module. A module, in this case, is either a Stratego source file with the `.str` suffix or a compiled module with the `.rtree` suffix. Each directed edge corresponds to an import declaration in a module. The build weave also contains information about how to resolve import names to actual files on disk using the include paths defined in the project build system. The main purpose of the build weave is to provide searching capabilities to the editor so that the user can search a project for definitions, e.g. find definition locations for a given identifier. It is also used for incremental project building, as discussed below.

The weave is constructed by parsing all `.str` files of a Stratego project and the build system (in the form of `Makefiles`). The build system declares all include paths. There are for resolving module import names to actual files from the file system. Once constructed, the build weave activates logic which listens for events pertaining to the modules or build system files. When a relevant change event is seen, the affected parts of the build weave are marked dirty. These parts will be lazily updated when subsequent requests require updated information. For example, the source code navigation feature of the editor may ask for the list of all visible definitions from a given module. If this module is marked dirty (or depends on nodes marked dirty), then the dirty nodes will be reparsed (possibly adding new nodes and edges in the graph). The weave will be marked as up to date. Only after this update process is finished will the list of visible definitions be computed.

9.2.3 Project Rebuilding

Whenever a module is changed that is referenced by the build system, i.e., it contributes to the final deployable program, the build weave will signal the project builder to commence a rebuild of the project. The current Stratego compiler is a

whole-program compiler. This means that a project rebuild may take several minutes. For convenience, it is possible to turn off automatic project rebuilding.

9.3 Editor

The principal user-interface component provided by Spoofax is the Stratego editor. It is built on top of the Eclipse editor framework and provides a range of editor features from syntax highlighting to source code navigation.

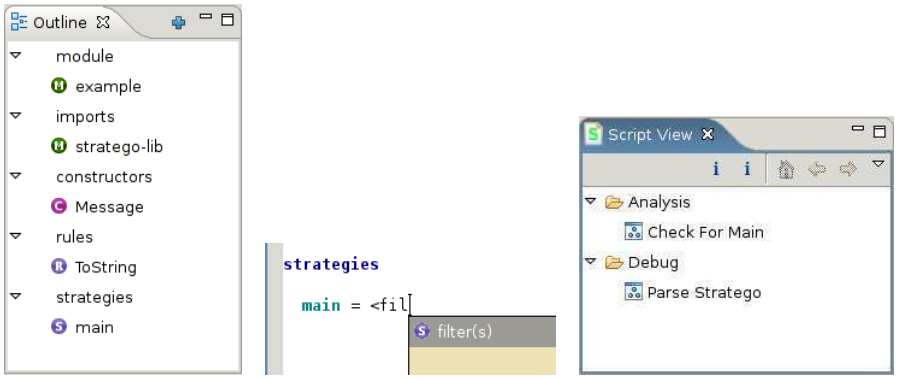
9.3.1 Content Completion

The purpose of content completion is to help the developer writing source code by suggesting possible textual completions based on the surrounding source code context. For example, if a developer asks for a completion for the prefix "`fil`" in a location where a strategy or rule is applicable, then the strategy `filter(s)` may be suggested, provided that the current module imports the standard library where the `filter(s)` strategy was defined. See Figure 9.3(b).

The Spoofax content completer is not perfectly context-aware. For example, it cannot always guess whether a strategy, variable or constructor name is expected. In these cases, it will suggest all possible choices. However, the completer is aware of sections in a Stratego module. If completion is requested in the `imports`-section, only valid module names are suggested. Further, in the `rules` and `strategies` sections, constructor, strategy, rule or overlay names are in general possible. Only closer inspection of the context can determine which is applicable. The Spoofax content completer will automatically suggest overlays and constructors if the prefix before the cursor is a `!` (a build operator), since that uniquely identifies the expression which follows as a term. Similarly, if the immediate prefix before the cursor is a <, the following expression must be a strategy expression, so strategies and rules are the only possible choices.

9.3.2 Syntax Highlighting

Perhaps the most basic development aid expected by an editor is the proper syntax highlighting of source code. Spoofax provides syntax highlighting for both SDF and Stratego. The user can configure the visual attributes, such as slant, boldness and colour, of the different syntactical categories, which includes keywords, rule or strategy declarations, comments, documentation and built-in primitives. Due to the flexible syntax definition formalism used for Stratego, some uncommon corner cases must be dealt with. One of these is the multiple meanings of the character `'`. It may be the start(and end) of a character literal, e.g. `'a'`. It is also a valid suffix of an identifier, e.g. `decl'`. Fortunately, it is possible, though a bit complicated, to



(a) Outline View (b) Content Completion (c) Script View

Figure 9.3: Outline and script views, and a content completion popup.

distinguish between these cases purely at the lexical level. Figure 9.1 shows syntax highlighting for both Stratego and SDF.

9.3.3 Parenthesis Highlighter

The job of the parenthesis highlighter is to find a matching parenthesis for any parenthesis next to the cursor. By convention, most editors (including Spoofax) will first check to see if there is a closing parenthesis before the cursor and, if so, find the first opening parenthesis of the same type at the same nesting level. If there is no closing parenthesis in front of the cursor, an opening parenthesis is looked for immediately after the cursor. This logic accounts for the different allowed parenthesis types in Stratego (`{`, `[`, `(` and `<`), and will indicate a mismatch by colouring the offending match red. In the case of a good match, a pink outline is used to highlight the matching parenthesis.

```
nasty = id < id + id  ; <id> < <id> + <id ; <id> +> id >(); !("(",")")
```

A noteworthy complication is the meaning of the character `<`, which is allowed as both a (nested) application operator, e.g., `<s1 ; <s0> >` and in the choice, e.g. `s0 < s1 + s2` as well as part of the left (`<+`) and right (`+>`) choice.

9.3.4 Outline

The Outline View, Figure 9.3(a), is a helper view which, when open, always applies to the currently active editor. It shows all definitions found in the module being

edited. The definitions are sorted under their respective types including rules, strategies, module, imports and constructors. The purpose of the Outline View is to allow quick navigation inside larger modules, and to show the structure of a module at a glance. By clicking on any of the defined names, the cursor is moved to the corresponding definition inside the module.

9.3.5 Source Code Navigation

In any non-trivially-sized project, developers spend a lot of their time navigating the source code to find, understand and fix existing definitions or to add new definitions. Cross-module source code navigation is supported by Spoofax due to the build weave discussed in Section 9.2.2. By placing the cursor under an identifier, the "go to definition" action can be invoked. This action will compute the possible definition sites for the identifier requested and, if multiple applicable locations are found, produce a popup window to ask the user to select which definition location to go to. This mechanism allows the user to easily navigate to rule, strategy, overlay and constructor definitions.

A complementary action, "open definition", allows the user to open a popup listing all definitions visible from within a module. This list may be searched by prefix and wildcard strings, e.g. `fi*ter`. It is only populated with definitions which would be visible through the import graph of the current module. A final action, "open module", allows the user to open any module visible from the import graph of the current module.

9.3.6 Build Console

Whenever a project (re)build is started, all output from the build process is redirected to a special Build Console as shown in Figure 9.4. This console keeps the history of all output messages from all previous builds until the user explicitly resets the log.

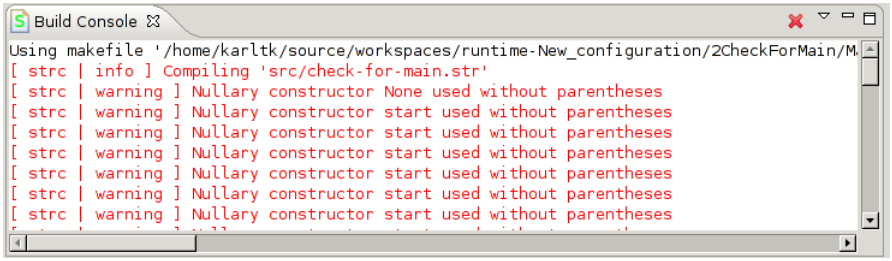


Figure 9.4: The build console shows an example output from the Stratego/XT build system.

9.4 Scripting

Software transformation systems such as Stratego/XT are powerful vehicles for implementing program analyses and transformations. Spoofax offers this power to Stratego programmers in the form of user-supplied editor scripts. By writing a script in Stratego using a few hooks provided by Spoofax, users can extend Spoofax with custom functionality. These scripts run on the Stratego/J execution engine and are used to rewrite ASTs obtained from Stratego source code using the jsglr parser. Scripts may be loaded into Spoofax and executed later by invoking them from a catalogue of scripts found in the scripts view, as discussed below.

Consider the example script in Figure 9.5. Line 4 imports the Stratego syntax and lines 5 and 6 are required for hooking into Spoofax. The strategy `main` on line 15 defines a trivial Stratego program which first obtains the AST for the current editor buffer (line 17), then traverses this AST and collects all definitions of a `main` strategy using the rule from line 22 (here, concrete syntax is used). The resulting list is printed to the script console on line 19. On line 20, a popup is displayed if the list is empty.

9.4.1 Script View

All scripts loaded by the user are visible in the Script View, as shown in Figure 9.3(c), and may be invoked from this view by double-clicking on the script's name. Each script is classified under a category. This aids in organising the script collection. Referring back to Figure 9.5, line 12 declares a user-visible name, which will be shown in the Script View. On line 13, the category is declared. Line 16 declares to the transformlet infrastructure that the strategies `xlet-script-name` and `xlet-script-category` are meta information, not part of the script logic. Additional meta-information definitions, such as author and license, are also possible and will be extracted and placed in a cache. This means that all script definitions need not be reloaded upon every restart of Eclipse for the Script View to be populated.

9.4.2 Script Console

All scripts executing inside Spoofax will have their output redirected to the Script Console. For editor scripts, this console is mainly useful for debugging and simple logging of non-essential information. On line 18 in Figure 9.5, the current term (a [possibly empty] list of strategy definitions) will be printed to the Script Console. This console is visually very similar to the Build Console that was shown in Figure 9.4.

```
1  module check-for-main
2  imports
3    stratego-lib
4    Stratego-Sugar
5    org/spoofax/editor/editor-common
6    org/spoofax/editor/transform
7    org/spoofax/xlet/core
8    org/spoofax/bindings/eclipse/eclipse-ui
9
10 rules
11
12   xlet-script-name = !"Check For Main"
13   xlet-script-category = !"Analysis"
14
15   main =
16      xlet-meta(xlet-script-name ; xlet-script-category)
17    ; spoofax-current-ast
18    ; collect(FindMain)
19    ; where(spoofax-debug)
20    ; try(?[] ; <eclipse-ui-show-popup> ("Malformed", "Missing main"))
21
22   FindMain = ?|[ main = s ]|
```

Figure 9.5: Simple analysis script that checks for the presence of `main` in a module

9.4.3 Analysing and Transforming Source

Analysis and transformation of Stratego code takes place on the AST associated with the current program being edited. In other editors, such as Emacs, code transformation is purely textual, which severely limits the possible transformations and analyses. An AST is obtained from the current editor buffer using the strategy `spoofax-current-ast`. Internally, the source code is parsed using the jsglr parser and, if successful, the AST is returned. A limitation of the current implementation is that the source code must be syntactically correct, since the standard grammar for Stratego does not have any error-correcting productions. The resulting AST can be analysed and modified by the user scripts. Once modification is done, the result can be pretty printed back to the buffer. Layout is not currently preserved well enough for everyday use, but known techniques, such as those explained in [BV00] could be applicable.

9.4.4 Transformation Hooks

In addition to user-initiated execution, scripts can be triggered to automatically execute when certain events in Eclipse are seen. Currently, three events are supported.

OnLoad – Whenever a Stratego file is loaded, a script can execute before the editor displays the file. It is generally recommended, though not required, that the script only performs analysis and the results of this analysis be made visible in the default Eclipse Problems View with other compiler errors and warnings.

OnSave – Scripts can also execute when files are saved. This is frequently useful for (re)generating dependent – or derived – files.

OnTimer – At intervals determined by the script meta information, a script may be executed. These scripts cannot expect any editors to be open, but may query for their existence and should gracefully fall asleep if none are found.

9.5 Implementation

Spoofax is implemented as an Eclipse plugin, written in Java and Stratego. The interfacing between Java and Stratego is handled using the Stratego/J interpreter introduced in Chapter 6. The user-provided extensions are written as transformlets, using the transformlet infrastructure also introduced in Chapter 6.

In the current release, most of the core functionality from Section 9.2, and the editors in Section 9.3, are written in pure Java. The scripting support is mostly written in Stratego (except for the Stratego/J interpreter). Since the editor code is heavily dependent on the libraries and abstractions provided by Eclipse (especially for actions and callbacks using inner classes) it is unlikely that Stratego can provide a worthwhile alternative here. The build weave implementation, however, would clearly benefit from being reimplemented using GraphStratego presented in Chapter 7. The current

implementation predates Stratego/J, and was therefore written in Java. The propagation rules for change events and the graph navigation code are prime candidates for a rewrite.

Parsing – The editor is built on top of three different parsers of Stratego. The ones used for syntax highlighting and code outlining are hand-written in Java because they must work work well for syntactically incorrect programs. A scannerless GLR parser (jsglr) is used to extract the abstract syntax tree from source files and are available for user scripts to inspect. Modification is also possible, but layout is not (yet) always properly preserved. Most contemporary syntax highlighters are written using a mishmash of regular expressions and state-keeping helper code. The Stratego syntax highlighter is, sadly, no exception. It is especially important that the highlighting works well when the program is syntactically incorrect, thereby aiding the programmer when needed the most. The Stratego syntax highlighting is context-dependent, so using a purely declarative tokeniser is not feasible.

Help – The help system is a packaging of the official Stratego/XT reference manual, along with the official Stratego tutorial, and a collection of detailed examples [BKVV05].

9.6 Related Work

Many program transformation systems provide some form of interactive environments. The paragraphs which follow briefly discuss some program transformation systems that are advanced and actively developed.

The *Meta-Environment* is an open and extensible framework for language development, source code analysis and source code transformation based on the ASF+SDF transformation system [vdBvDH+01]. The environment provides interactive visualisations, editors with error checking and syntax highlighting. *Tom* is a software environment for defining transformations in Java [MRV03] and comes with a basic Eclipse editor plugin that provides syntax highlighting, context-specific help, error checking and automatic compilation, but no source navigation. *JTransformer* is a Prolog-based query and transformation engine for Java source code, based on Eclipse. It provides a Prolog editor with syntax highlighting, auto-completion, code outlining, error checking and context-specific help. *HATS* is an integrated development environment for higher-order strategic programming [Win99]. *HOPS* is a graphically interactive program development and program transformation system based on term graphs [Kah99]. The environment is a mix between literal and visual programming. *ANTLRWorks* [BP] is a graphical development environment for developing and debugging ANTLR grammars, with an impressive feature list that includes code navigation, visualisations, error checking and refactoring.

All these systems have feature sets overlapping with Spoofax, but to my knowl-

edge, only the Meta-Environment was also designed to be extensible using a transformation language.

9.7 Summary

This chapter introduced and described an extensible, interactive development environment for Stratego/XT that provides modern development aids like content completion, source code navigation, customisable syntax highlighting, automatic and incremental project building. Users can extend the environment with scripts written in Stratego, and these can perform analysis and transformation on the code under development.

The transformlet techniques introduced in Chapter 6 enabled the extensible scripting features.

Spoofax is still evolving and maturing, but already over a dozen of active Stratego programmers are using it. The feedback so far suggests that the environment lowers the entry level for new users and makes existing developers more productive. Increased productivity comes both from offering source code navigation for Stratego and from the close integration with editors for other (subject) languages that are already available for Eclipse.