Generic Framework for Integration of Programming Languages into NetBeans IDE

Jan Jančura
Sun Microsystems
The Park, Building 3, V Parku 2308/8
148 00 Prague 4, Czech Republic
jan.jancura@sun.com

Daniel Průša
Sun Microsystems
The Park, Building 3, V Parku 2308/8
148 00 Prague 4, Czech Republic
daniel.prusa@sun.com

Abstract

We present a generic framework that can be used for an easy integration of editing and visualization support for a programming language or files with a structure into NetBeans IDE. Needed features are defined using a simple declarative language. It is also possible to provide custom Java methods to enhance the definition's capabilities. The concept aims at good maintenance and performance of implemented languages. We proved it to work well by integrating over twenty languages.

Categories and Subject Descriptors D.2.11 [*Languages*]

General Terms algorithms, design

Keywords framework for visualization of programming languages, parser

1. Motivation

Nowadays, it is a common trend to have a support for many file types within one IDE. IDE's are able to recognize and handle files with structure like HTML, XML, Shell scripts, BAT files, etc. They also quite often contain support for not one, but several programming languages. A substantial contribution is surely given by the growing number of scripting languages connected to web technologies.

In the following text, let the language denote a set of structured files of one type. To provide a hand-coded support for several languages into an IDE can be a problematic task from several points of view – the important factors are time of development, maintenance, performance (scalability), memory requirements. If we would like to support about 100 languages, an easy, generic framework of implementing a language, covering the editing features like syntax coloring or code folding (and many more) would be a wise solution. This is exactly what project Schliemann offers.

2. Project Schliemann

Project Schliemann comprises of an engine that provides a generic framework for a language definition and its integration into NetBeans IDE. The support concerns mostly editing and visualization features. Except for the generic features, custom features can be implemented on the top of the engine output for a particular language.

The engine has been inspired by the support which is present in many programmer's editors like Emacs, vi or JEdit. However, these editors typically implement basic features only (syntax coloring, indentation, code folding). There is usually a proprietary way how to define lexical analysis, but syntax analysis is missing. The goal of project Schliemann is to go far beyond this approach and to offer many more possibilities. On the other hand, the ambition of the project is not to provide a framework for a complete programming language support, including compile/debug/run ability.

3. NBS Language

To integrate a particular language using Schliemann engine requires to describe the language by so called NBS file (standing for NetBeans Schliemann) which is a text file consisting of sections. The structure of a language is defined by the lexical section comprising of regular expressions that determine tokens of the language, and the syntactic section, which contains grammar productions. The other sections define the language visualization.

Common syntax is used for regular expressions and grammar productions in NBS files. The form of grammar productions follows extended Backus-Naur form. $LL(k)$ grammars are allowed. Regular expressions are enriched by states definitions that simplify tokens description. In addition, both analysis have a possibility to call Java code that handles a portion of the analysis, returns detected tokens, resp. derivation subtree and passes control back to the engine. This mechanism allows, e.g., to handle languages which of tokens cannot be described by regular expressions (like Ruby or JavaScript).

Features are defined based on recognized tokens and also grammar's non-terminals. For example, in a programming language, syntax coloring can be defined for a keyword (which is a token detected by the lexical analysis) as well as for a method name (a non-terminal detected by the syntactic analysis).

Format of a feature definition is intuitive and easily readable. To demonstrate this, let us show fragments of a NBS file:

```
TOKEN:number:(["0"] | ["1"-"9"] ["0"-"9"]*)

COLOR:number {
    foreground-color:"orange";
    font-type:"bold";
}
```


PEPM'08, January 7–8, 2008, San Francisco, California, USA.
ACM 978-1-59593-977-7/08/0001.

The first line defines token "number" by the given regular expression. The second part defines coloring for this token, foreground color and font type are specified within this definition.

The rich nature of features is boosted by providing Java code as a part of their definition. The code is required each time, when a language specific behavior that cannot be described by a common pattern has to be implemented. A good example is feature Hyperlink. A hyperlink is defined by a token or non-terminal on which it can be enabled. The action to be performed on clicking the hyperlink is specified by a static method, which is the action performer. It is referenced in the hyperlink definition.

4. Supported Features

We give a list of the most important generic features, together with their brief descriptions.

- Syntax coloring – to distinguish tokens and possibly non-terminals of the language by a color in editor.
- Code folding – to wrap and unwrap pieces of code (e.g. methods) in editor.
- Navigation – to browse logical elements of the language in Navigator window.
- Code completion – to offer how to complete a piece of code based on the typed prefix.
- Brace matching – for a bracket located under the cursor, to highlight the pairwise bracket.
- Actions – to define language specific actions over documents.
- Tooltips – to display tooltips on elements of the language.
- Hyperlinks – to implement language element driven jumps into a logically bound part of a document (the same or a different one).
- Indentation – to properly indent documents based on the language structure.
- Annotations – to annotate specific lines of documents (e.g. error lines).

Except the generic features, it is possible to implement custom features based on the output of the lexical and syntactic analysis.

5. Languages Embedding

A very important feature provided by the Schliemann engine is support for languages embedding. There are several languages, especially scripting languages, that allow to embed another languages. To give an example, we can consider HTML containing JavaScript. When editing such sources, we expect all the features listed above to work for both languages (HTML and JavaScript) in their own way. It is also convenient to have pieces of different languages visualized to easily distinguish them. This is exactly what can be implemented using our framework.

6. Notes on Engine Implementation

The engine contains a general parser for $LL(k)$ grammars. We have decided to have our own implementation to meet our requirements on the parser input, output, error recovery, internal architecture, grammar correctness checking, performance tuning, etc. Produced abstract syntax trees include information on comments, whitespaces and positions. One of the important features the engine's internal architecture supports is the languages embedding mentioned above.

The lexical analysis can be connected to the incremental analyzer provided by NetBeans editor module. The engine also has its own analyzer, but it is not incremental.

Languages are defined in separated modules. They are detected by the engine using a NetBeans specific way.

7. Project Status

Currently, the Schliemann engine is implemented in the development builds of upcoming NetBeans 6.0. We have proven the concept works well. Over 20 languages have been already integrated into NetBeans using the engine. Full, grammar based support has been done for JavaScript. This can be considered as the main achievement. As for the other languages, it is worth to mention HTML, YAML, CSS, PHP, Groovy, BAT files, Shell scripts and several NetBeans specific file types. The integration of some of these languages is based on the lexical analysis only.

Provided that a grammar is available for a language, we have proved that to implement all the supported features is really an easy task which can be completed by one person during a week. Of course, adequate knowledge of the theory of formal languages is required to do this. As for the syntactic part of a language definition, an existing grammar that fulfills $LL(k)$ criterion can be adopted. If the criterion is not fulfilled, the grammar can be still adopted after some modifications.

Performance of the engine is quite good – it scales over the number of integrated languages as well as over the length of a document. We can conclude that the achieved results are promising. We plan to continue on integrating more languages into NetBeans, and also to extend the engine's capabilities (e.g. to support LR grammars, etc.).

A. Tool Demonstration Overview

The tool demonstration consists of the following parts:

- Short introduction into the problematic, motivation. Possibilities offered by the Schliemann engine.
- The support for JavaScript in NetBeans implemented using the engine. A demonstration of the features. Enhanced features like semantic coloring (local variables, parameters, fields, unused variables and methods) or refactoring (find usages, rename).
- Languages embedding support examples.
- A demonstration of how to integrate a new language into NetBeans.
  - Wizard helping to generate a new language support module.
  - NBS file generated from the template.
  - An example of the lexical and syntactic section.
  - How to define editing and visualization features.
  - Running the new language support module in NetBeans.

References

[1] http://www.netbeans.org

[2] http://wiki.netbeans.org/wiki/view/Schliemann