Colorer: Syntax Analysis Framework for Source Code Editors and Integrated Development Environments

Igor Russkih
irusskih at gmail.com

Yuli Ketkov
ket at unn.ru

Nizhny Novgorod State University
23 Gagarin ave.
603950 Nizhny Novgorod, Russia

ABSTRACT

The paper reviews a number of existing solutions for source code incremental syntax analysis in the Integrated Software Development Environments (IDE) targeting software engineer's efficiency improvements. Colorer library is presented as an solution in this area for many of the interaction problems between the developer and development environment.

Categories and Subject Descriptors

D.2.3 [**Software Engineering**]: Coding Tools and Techniques—*Program editors*; D.2.6 [**Software Engineering**]: Programming Environments—*Integrated environments*; D.3.2 [**Programming Languages**]: Language Classifications—*Specialized application languages*; D.3.4 [**Programming Languages**]: Processors—*Parsing*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

General Terms

Languages, Algorithms

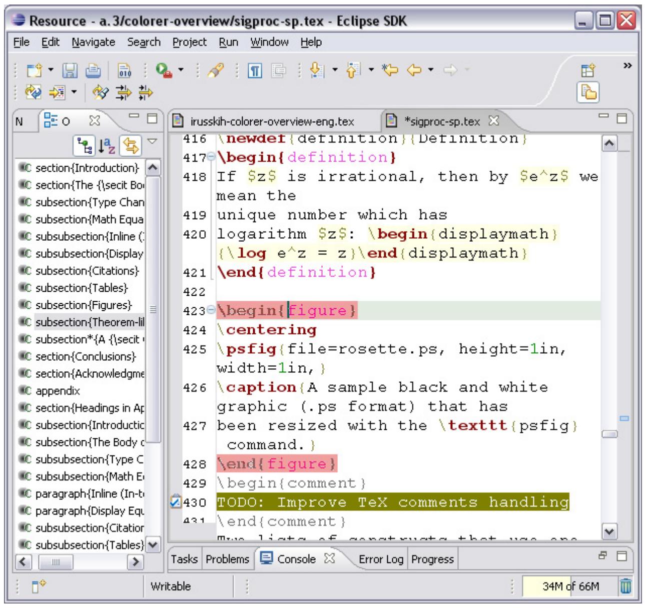
Keywords

syntax analysis, grammar description, programming environments

1. INTRODUCTION

Software development today is an active and a valuable industry. Environments which help and simplify a process of creation and debugging application's source code is one of the most actively developed type of software. All the popular programming languages are bundled and shipped with their own multi-functional IDEs, allowing a developer to easily understand and adapt the target language.

Increased complexity of modern languages, compilers and architectures requires much more help and automation from



```
\newdef{definition}{Definition}
\begin{definition}
If $z$ is irrational, then by $e^z$ we
mean the
unique number which has
logarithm $z$: \begin{displaymath}
{\log e^z = z}\end{displaymath}
\end{definition}

\begin{figure}
\centering
\psfig{file=rosette.ps, height=1in,
width=1in,}
\caption{A sample black and white
graphic (.ps format) that has
been resized with the \texttt{psfig}
 command.}
\end{figure}
\begin{comment}
TODO: Improve TeX comments handling
\end{comment}
```

Figure 1: EclipseColorer within the Eclipse IDE

Integrated Development Environments today. IDEs improve efficiency and speedup a process of programming language usage and study [7, 11]. Additional program tools like realtime source code syntax analysis, code construction highlighting, context dependent assistance and help are the common and well known parts of any mainstream IDE.

Unfortunately for many task-specific and brand new programming languages IDE development or even integration within an existing IDE is a complex and time-consuming task. This paper presents the *Colorer library* — a tool for description and implementation of any programming language source code editor. Colorer is a framework for source code fast syntax analysis, visual highlighting, annotation and structural navigation. It provides "out of box" support for variety of existing programming, markup and scripting languages via special declarative HRC language.

In contrast with regular "multilingual" program text editors syntax highlighting is only a trivial part of all the functionality Colorer library provides. Syntax analysis in

Colorer allows to build abstract token trees from source text and map these tokens with visual attributes (color, font size, style). Beside this library provides extended structural text analysis: outline of the program structure, recursive and paired constructions markup, code folding. This extra information allows target editing system or IDE to implement enhanced source code navigation and editing experience (Fig. 1).

Colorer library is not a complete IDE but a set of services, framework. It can be easily integrated into the target text editing environments and provides a set of services based on incremental syntax analysis of the edited text.

2. RELATED WORKS

Probably, Cornell Program Synthesizer [10] is the one of the first programming oriented environments which could be treated as an "IDE". This system is basically a structure-oriented editor which allows to construct the desired program structurally. It uses target language's grammar description to generate structure-directed program editor. Ultimately this allows to exclude syntax errors and lets an user to concentrate on the program's logic.

Today pure structure-oriented editors is rare, the idea basically migrated into the area of visual programming (UI-constructors, model designers) where target code is generated automatically.

In general the evolution of editing systems has continued with pure plain text presentation [9]. The most important areas of development here now are visual assistance, background model extraction, realtime incremental code validation. More and more attention is gained to the framework environments, libraries, which allows to build custom source code editors and IDEs, based on common services and features.

Popular Eclipse platform [4], for instance, provides a common extensible framework of text editing core components including text coloring, context assistance, advanced text decoration and presentation.

This core is extended by a set of language specific environments. The most advanced and powerful is Eclipse JDT — a development platform for java language. The core of this system is a full-featured incremental java compiler. Complete syntax, semantic analysis and compilation of full project under development allows to implement extended assistance features like realtime syntax errors indication, context dependent text input assistance and code refactoring. A few other Eclipse-based environments offer a similar set of functionality. These are Eclipse CDT, Web Tools, PDP and others.

Although the above system is at most an industrial-level work, there exist a few academic and research projects, featuring new ideas in the area of source code editing and presentation. Barista [5] project makes an accent on the advanced visualization of the edited text. It presents and stores text internally within structure-oriented format, however it doesn't restrict user from breaking language grammar rules and allows inconsistent or invalid documents to be saved.

Internal structured presentation allows system to provide user with a number of multimedia and visual interactive features. This is achieved by precise target grammar parser usage and by grammar rules attribution with particular UI-related actions. Rather complex target language's grammar description and customization limits this system with the only Java language support.

Another active research project in the area of programming environments is a Harmonia [1] system. Harmonia chooses more general approach. It is based on a CF description of a target language, which is used to generate a common editing functionality. The resulting partially generated and partially hand-written target language's model includes lexical, syntax and semantic analyzers. Incremental syntax analysis algorithm supports generated AST in the consistency with edited text. This allows to implement rich text editing framework with strong "near-compiler" validation. Harmonia is one of the most complex systems in sight of syntax description and implementation. Today it supports only a few common programming languages.

These and many other source code editing IDEs are at most support only mainstream languages. Implementation of full-featured complete support for some new, relatively rare or specific language is very complex and time consuming task. This is true even within the usage of the frameworks, described above.

Modern development environments often do not support wide set of languages, scripts and markups, falling down only into particular technology area specifics. But the real projects under development often include a wider variety of technologies. Beside the main project language, mixed language projects are not rare, the mixup happens even in a single file, like in the template systems — JSP, ASP, PHP — these include both language and markup with HTML, CSS, JavaScript and other syntaxes. Project's building environment often uses build and other scripts. All this forces developer to search for a set of editing tools, suitable for his/her particular needs.

To fulfill this there exist a number of general text code editors with a ranging set of functionality [3, 13]. The most common feature here is a syntax highlighting. Some of the text editors provides a scripting support for compilation/validation, however it is often limited and can't be compared with the features IDEs can provide. EMacs [9, 6], VIM, jEdit systems could be noted as the most powerful in this area. Language syntax and structure analysis are the important features, implemented in these systems.

3. COLORER LIBRARY OVERVIEW

Colorer library is a set of components, most important of which are the syntax description language (HRC) and an incremental syntax analyzer. Description language is a XML-based declarative syntax, it is used to declare lightweight regular and context free constructions. Unlike traditional forms of grammar description (BNF, EBNF), HRC allows to easily declare important aspects of the target language without going into the deeps of the formal grammars.

HRC provides a set of trivial constructions (plain keyword lists, regular expressions) to define token extraction rules. With use of a "scheme" concept (also referenced as a context or non-terminal) it allows easily to describe most of today language syntaxes.

In addition to these common (and widely used) constructions, HRC introduces a new concept of scheme reuse and dynamic redefinition (virtualization). These concepts basically allows to interpret different language's syntax declarations as a linked modules and to reuse syntax elements in multiple places. Ultimately this leads to a very compact definition of "mixed" languages, which consume more and more space in today's software industry.

The example here is Web-area with the well-known languages of ASP, PHP, JSP. Mostly all the popular scripting languages implement today their own templating systems (including Perl, Ruby etc.) Another area of combined and mixed syntax is XML group of languages. XML specification internally allows to mix different syntaxes within the single document (using a namespace specification). Moreover, these syntaxes are essentially the same at the grammar level (they are all XML) — the difference is on the semantic level. HRC and Colorer easily handle this by generating grammar declaration directly from XML Schema definitions.

Colorer's syntax analyzer allows "on the fly" support of any mixed syntax, described above. Together with the support implemented for a great variety of traditional languages, this makes Colorer library a good choice for the programming environments extention and improvement.

Today's software systems may use a variety of programming languages, configuration scripting, project documentation in different formats. Colorer allows to implement a support for all of this inside of a single IDE. This greatly improves programmer's experience and productivity since there is no more need for user to learn a different environment for each of the used technology or language.

Colorer library has a flexible internal architecture, allowing to quickly and easily integrate it into the target editor or IDE. Syntax analyzer output tree is separated from text storage and is updated on each text modification. Library requires only a simple interface from the target system to provide text storage incremental access and a set of notifications of user actions. This differs Colorer from other systems, where syntax analyzer is often tightly linked with editor's internals.

Syntax analysis output is a tree-like presentation of the edited text partitioning. It contains a hierarchy of tokens and contexts which then could be mapped with visual and structural attributes: color, font, style or program element. Separation between analysis information and different types of presentation allows to use colorer in a wide set of applications. Library's implementation core is C++, and it's available for all hardware platforms. Java and Perl mappings are also available.

3.1 HRC Language

The basis of syntax analyzer functionality is matching a sentence in the input language with a set of syntax rules. And these are essentially describe the target language. Colorer library uses a special declarative syntax "HRC" [8] to describe these rules.

The noticeable characteristic of HRC model and Colorer's analyzer is "positive" syntax description and parsing. In comparison with the traditional grammar descriptions HRC declares the "desired" structure to be found in the target text. Syntax analyzer then extracts this structure in terms of regular expressions and schemes and ignores all the unknown input. It doesn't fail if an input sentence contains extra or unknown tokens. Below is a simple HRC grammar for Ruby language:

```
<scheme name="ruby">

 <block start='/\#/' end='/$/'
        scheme='def:Comment'
        region='Comment'/>
 <block start='/^=begin/' end='/^=end/'
        scheme='def:Comment'
        region='CommentDoc'/>

 <inherit scheme="def:Number"/>

 <block start='/([\x22\x27\`])/' end='/\y1/'
        scheme="String" region='String'/>

 <regexp match='/ (?{Keyword}alias) \s+
               (?{AliasOutline}
               \S+ \s+ \S+)/x'/>
 <regexp match='/\M class \s+
                (?{ClassOutline}\S+)/x'/>
 <keywords region="Keyword">
   <word name="BEGIN"/>
   <word name="END"/>
   <word name="class"/>
   <word name="ensure"/>
   <word name="nil"/>
   <word name="self"/>
 ...
```

This syntax declaration can be easily understood by any person with common knowledge of regular expressions and XML markup principles.

HRC declarations extend traditional concepts of syntax analysis with a new model of inheritance, virtualization and parameterization. Within these new concepts any HRC scheme (grammar context) can inherit properties and behavior of any other scheme, even related to the syntax of another language. Inheritance in HRC is not a blind copy — thats a dynamic linkage between schemes of different syntaxes. Inheritance operation can even redefine and partially modify the behavior of the modified scheme.

All this allows to extremely easy describe a syntax of mixed and complex languages. HRC separates a description of each language and at the same time allows to reuse constructions

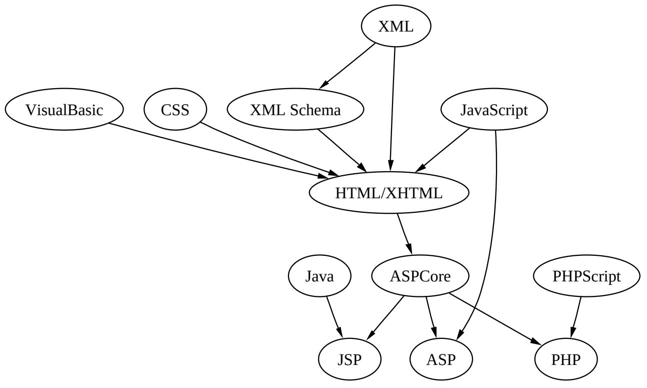


Figure 2: Linked syntaxes within templating systems

of the already defined language in a new one. In such a way, for instance, separate definitions of the "Web" syntaxes are constructed: HTML, CSS, JavaScript, VBasicScript. These are then combined with traditional languages definitions (Java, PHP, Perl, etc.), leading to the JSP, PHP, ASP and others mixed syntax definition (Figure 2).

`ASPCore` here is a common HRC template, it is essentially a

```
<scheme name="Insertion">
  <block start="/(&lt;\%)/" end="/(\%&gt;)/"
         scheme="targetLanguage"
         ... />
</scheme>
<scheme name="InverseInsertion">
  <block start="/((\%&gt;))/" end="/((&lt;\%))/"
         scheme="asp"
         ... />
</scheme>
<scheme name="asp">
 <include scheme="Insertion"/>
 <include scheme="html:html">
  <virtual scheme="html:htmlCore"   prolog="Insertion"/>
  <virtual scheme="html:htmlString" prolog="Insertion"/>
  <virtual scheme="html:html"       prolog="Insertion"/>
  <virtual scheme="css:Property"    prolog="Insertion"/>
  <virtual scheme="css:RuleContent" prolog="Insertion"/>
  <virtual scheme="css:RulesList"   prolog="Insertion"/>
  <virtual scheme="def:Comment"     prolog="Insertion"/>

  <virtual scheme="vbScript:vbScript"
           prolog="Insertion"/>
  <virtual scheme="vbScript:vbMETA"
           subst-scheme="Insertion"/>
  <virtual scheme="jScript:jScript"
           prolog="Insertion"/>
  <virtual scheme="jScript:jsMETA"
           subst-scheme="Insertion"/>
  <virtual scheme="perl:perl"
           prolog="Insertion"/>
  <virtual scheme="perl:META"
           subst-scheme="Insertion"/>
 </include>
</scheme>
```

This abstract syntax uses a `prolog` directive to advise derived scheme execution with ASP specific processing. The syntax is then extended with the concrete implementation of a "targetLanguage" scheme stub:

```
<import type="asp"/>

<scheme name="jScript">
   <regexp match="/\/\/.*? \M (\%&gt;|$)/ix"
           region="def:Comment"/>
   <inherit scheme="InverseInsertion"/>
   <include scheme="jScript:jScript"/>
      <virtual scheme="jScript:jScript"
               subst-scheme="jScript"/>
   </include>
</scheme>

<scheme name="asp.js">
   <inherit scheme="asp:asp">
      <virtual scheme="targetLanguage"
               subst-scheme="jScript"/>
   </inherit>
</scheme>
```

HRC inheritance and virtualization are used here to advise a lower syntax with peculiarities of a new behavior. Many of other technology areas are described in HRC in a similar fashion. "C++" is declared as an extended HRC over the plain "C", a number of a different Assembler language syntaxes are derived from the single core.

Another example is a strong HRC support of XML [12] specification and its derivates. XSLT, XML Schema, XPath, XQuery — all these are just the most actively involving applications of XML. Using traditional grammar descriptions is pointless here — target IDE should support semantic validation and analysis.

HRC introduces editing support for arbitrary XML with DTD or XML Schema definition available. This is based on an automatic XML Schema translation into the HRC grammar description. This transformation is based on a Colorer's XSLT [2] module "xsd2hrc" which allows also a transformation partial customization. Based on this module Colorer and HRC mostly automatically provide full-featured editing support for XHTML, SVG, XSLT 1/2, RelaxNG, ANT, DocBook, MathML and mostly any XML syntax with XSD/DTD available.

All these generated HRC declarations support syntax and semantic validation of XML document's structure, attribute types and values validation, document outline and folding information extraction and other features (see Fig. 3).

3.2 Parser algorithm

Colorer's syntax analyzer works incrementally based on source text modification events. It doesn't build a parse tree in it's traditional form. Internal parser's logic only stores and updates a partial cache, which is reduced according to per-line layout of the source. This parse cache keeps only "between-line" context changes. This allows to reduce memory usage when running the incremental parser algorithm.

Distinct parsed tokens are also never saved internally. Parser uses a "push" model to inform external modules about token flow and context changes. This model has a number of practical advantages against a traditional parsing approach, where parser itself controls the creation and modification of the parse tree.

Because of the specifics of the final results of parsing in Colorer, it is much more efficient to allow external librarie's client to construct the efficient presentation of the parse tree. For instance, at most all the editing systems use their own visual presentation storage implementation. Using Colorer's "push" method it is possible to store the parse results directly in the target system's presentation model. This gives good performance and memory usage improvements.

Additional parse tree listeners could be used for any required information extraction and aggregation. Colorer framework includes a number of predefined parse tree listeners for source code structural analysis, code outline creation, filtered token search, text folding model extraction.

To handle HRC declarations, they are initially transformed into the internal presentation, suitable for parser's usage. The transformation happens during library loading, it layouts HRC syntax elements in a simplified form to allow parser to run over them. Some optimizations are also being held, targeting on syntax elements prediction rules. The resulting static HRC model tree is used by a parser to build a parse tree. These two trees are linked and the linkage information is used to allow incremental text processing. Incremental analysis uses the information about the links between parse tree and a model (grammar) tree to recover from modification events and to provide parse listeners with the updated tree.

3.3 End-user functionality overview

Colorer framework implements a wide set of end-user functionality based on the syntax analyzer and HRC language expressive power. This functionality is delivered to the end user via the Integrated Development Environment. Basically IDE's functionality and flexibility limit, how deeply the Colorer's features could be integrated.

One of the actively evolving IDEs now is Eclipse platform, which is built on concepts of dynamic extensions and API flexibility. Eclipse provides a powerful commons to build extended programming editors. These include different document presentations, structural and outlining views, powerful annotation framework, folding and of course a flexible visual presentation management of the edited document.

Colorer framework links all these features with the source code analysis information and provides an Eclipse plugin. This effectively adds a support of more than 150 different languages and syntaxes, presently defined in HRC.

The basic functionality — syntax highlighting — comes with a lot of user configurable visual styles. Colorer also provides an extended document navigation mechanism. Because of the deep analysis, running via HRC, it detects an arbitrary paired constructions, starting from trivial bracket and ending with XML tags matching.

Structure outlining and navigation is also one of the actively used features. Colorer derives and builds a structure outline for great number of languages automatically, based on HRC output analysis.

Many of the languages in HRC include a syntax and semantic validation. of the parsed document. All the possible errors are automatically shown in Eclipse editor with the visual annotation rulers.

Although the Eclipse is one of the most powerful platforms for IDE construction, Colorer framework usage is not limited only with this solution. Mostly the same level of functionality is provided by library's integration into the well known shells — FAR Manager (Windows platform) and Midnight Commander (unix flavors). Although the code editors in these systems are not as powerful, as in Eclipse, they still benefit from Colorer library's wide range of features.

Another actively evolving area of Colorer usage is a source code publishing. Colorer is used in a wide set of blogging and source code browsing Web-engines due to it's ability to easily generate HTML-based presentation of parsed code.

4. CONCLUSION & FUTURE WORK

Although Colorer library could be considered as a stable work, there still exist a wide area of research. The most important areas are HRC language expressive power, optimizations targeting HRC pre-compilation and parser's code generation.

The area of the end-user experience is also a wide field of research. This includes researches in compact and efficient code presentation in a very big and a complex systems, alternative code presentation approaches.

5. REFERENCES

[1] M. Boshernitsan. Harmonia: A flexible framework for constructing interactive language-based programming tools, technical report csd-01-1149, 2001.

[2] J. Clark. *XSL Transformations (XSLT)*. W3C Recommendation http://www.w3.org/TR/xslt, 1999.

[3] N. Hodgson. Scintilla. http://www.scintilla.org.

[4] IBM and contributors. Eclipse integrated development environment, http://www.eclipse.org.

[5] A. J. Ko and B. A. Myers. Barista: An implementation framework for enabling new tools, interaction techniques and views in code editors. In *CHI 2006 Proceedings*, pages 387–396.

[6] C. Rhodes, R. Strandh, and B. Mastenbrook. Syntax analysis in the climacs text editor, 2005.

[7] T. L. Roberts and T. P. Moran. The evaluation of text editors: Methodology and empirical results. *Communications of the ACM*, April 1983.

[8] I. Russkih. Hrc language reference, http://colorer.sf.net/hrc-ref/, 2004.

[9] R. M. Stallman. *EMACS, The Extensible, Customizable, Self-Documenting Display Editor*. Massachusetts Institute of Technology, June 1979.

[10] T. Teitelbaum. The cornell program synthesizer: a syntax-directed programming environment. *SIGPLAN Not.*, 14(10):75–75, 1979.

[11] R. C. Thomas. *Long Term Human-Computer Interaction*. Springer-Verlag, 1998.

[12] J. P. Tim Bray and C. M. Sperberg-McQueen. *Extensible Markup Language (XML) 1.0 Second Edition*. W3C Recommendation http://www.w3.org/TR/2000/REC-xml-20001006, 2000.

[13] S. R. Wood. Z - the 95% program editor. In *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, pages 1–7, New York, NY, USA, 1981. ACM Press.