String-Embedded Language Support in Integrated Development Environment[*]

Semen Grigorev
St. Petersburg State University
198504, Universitetsky prospekt 28
Peterhof, St. Petersburg, Russia.
rsdpisuy@gmail.com

Ekaterina Verbitskaia
St. Petersburg State University
198504, Universitetsky prospekt 28
Peterhof, St. Petersburg, Russia.
kajigor@gmail.com

Andrei Ivanov
St. Petersburg State University
198504, Universitetsky prospekt 28
Peterhof, St. Petersburg, Russia.
ivanovandrew2004@gmail.com

Marina Polubelova
St. Petersburg State University
198504, Universitetsky prospekt 28
Peterhof, St. Petersburg, Russia.
polubelovam@gmail.com

Ekaterina Mavchun
St. Petersburg State University
198504, Universitetsky prospekt 28
Peterhof, St. Petersburg, Russia.
emavchun@gmail.com

ABSTRACT

Most general-purpose programming languages allow to use string literals as source code in other languages (they are named string-embedded languages). Such strings can be executed or interpreted by dedicated runtime component. This way host program can communicate with DBMS or web browser. The most common example of string-embedded language is Dynamic SQL or SQL embedded into C#, C++, Java or other general-purpose programming languages. Standard Integrated Development Environment functionality such as syntax highlighting or static error checking in embedded languages can help developers who use such technique, but it is necessary to process string literals as a code to provide these features. We present a platform allowing to create tools for string-embedded languages processing easily, and compare it with other similar tools like IntelliLang. We also demonstrate a plug-in for ReSharper created by using the platform. The plug-in provides code highlighting and static error checking for string-embedded T-SQL in C#.

[*]Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

CEE-SECR '14, October 23 - 25 2014, Moscow, Russian Federation
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2889-0/14/10$15.00
http://dx.doi.org/10.1145/2687233.2687247

Categories and Subject Descriptors

D.3.4 [**Programming Languages**]: Processors—*parsing*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*

General Terms

Algorithms, Parsing

Keywords

String-embedded language, abstract parsing, parser generator, lexer generator, integrated development environment, IDE, Dynamic SQL.

1. INTRODUCTION

Multiple languages are often used in large software system development. In this case there are one *host* language and one or more *string-embedded* or just *embedded* languages. String expressions of host language form programs in another language and then they are interpreted in runtime by special component such as database management system or web-browser. The majority of general-purpose programming languages can play role of both the host and the embedded programming language. Examples of embedded languages are presented below.

- Dynamic SQL:

```
CREATE PROCEDURE [dbo].[MyProc]
  @TBLRes   VarChar(30)
AS
```

```
  EXECUTE ('INSERT INTO ' + @TBLRes + ' (f1)'
  + ' SELECT ''Additional condition: '' + f2'
  + ' from #tt where sAction = ''100''')
GO
```

- Multiple embedded into PHP languages (MySQL, HTML):

```
<?php
  $query = 'SELECT * FROM '. $my_table;
  $result = mysql_query($query);
  echo "<table>\n";
  while ($line =
    mysql_fetch_array($result, MYSQL_ASSOC)){
    echo "\t<tr>\n";
    foreach ($line as $col_value) {
      echo "\t\t<td>$col_value</td>\n";
    }
    echo "\t</tr>\n";
  }
  echo "</table>\n";?>
```

String-embedded languages may help to compensate the lack of expressivity of general-purpose programming language in domain-specific settings. However, it is rather hard to develop, support or reengineer string-embedded code using this technique. Dynamically generated expression are often constructed of string primitives of the host language by concatenations in loops, conditional expressions or recursive procedures. Dynamically generated expressions are simple strings in the point of view of the host language analyser and even syntactic analysis is undecidable in general case. Impossibility of static check for dynamic expression results in high possibility of getting errors in runtime.

Common practice in software system development is Integrated Development Environments using that provides such features as code highlighting, autocompletion, error handling, different kinds of refactoring. All these significantly simplifies development and debugging process. Thus the tools being able to perform *abstract analysis* — static analysis of dynamically-generated expression value set — may be really helpful.

Grammarware research and development project YaccConstructor [?] is now aimed to create an infrastructure for development of string-embedded language processing tools. In this paper we provide overview of the tools for embedded languages processing, describe the infrastructure under development and its components, and pay attention to multi-language support problems. We also describe error reporting and semantic calculation for embedded languages. Finally, we illustrate infrastructure features with developed plug-in to ReSharper[1].

2. RELATED WORK

There are several approaches for string-embedded languages processing. The first is based on comparison of the specification of the language obtained by regular or context-free approximation of dynamically generated expression with some language reference grammar [?, ?]. This approach answers the question of the dynamic expression correctness, but cannot provide meaningful error report for an incorrect expression. The most languages being used as embedded are at least context free and it is rather hard to find context free grammar for them. This circumstance leads to the second drawback: low analysis precision due to the type of approximation being used.

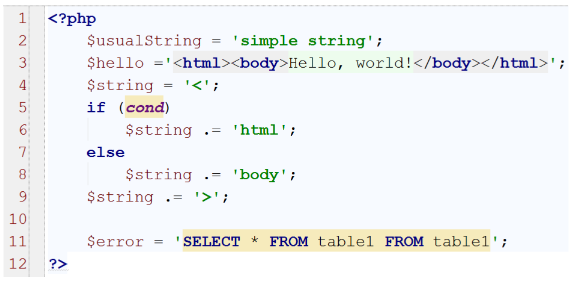
Second approach — *abstract analysis* [?] — is a static analysis of some representation of dynamic expression value set. This representation can be data-flow equation, regular expression, or finite automaton — as it is in our platform. The tools implementing this approach can be separated into two categories: *analysis-and-parsing* and *analysis-then-parsing*. The first one stands for performing of analysis and parsing at the same time on the fly. And the second one, which is also named *step-by-step analysis*, means that first the analysis of source code — constant propagation, approximation construction — is performed and after that the parsing itself is executed. We use step-by-step analysis in our framework.

2.1 Existing Tools

There are several tools for processing of concrete string-embedded languages. They differ in the approaches used and the ease of new language extension support. We provide a brief overview of these tools below.

2.1.1 PhpStorm

PhpStorm[2] — integrated development environment for PHP which implements code highlighting and autocompletion of string-embedded HTML, CSS, JavaScript or SQL code. However, only if the tool deals with string primitive (no string operation is used for expression construction), it is able to provide this support. You can see PhpStorm screenshot that illustrates this feature in figure 1. "." is concatenation operator and ".=" is concatenating assignment operator. As you can see, PhpStorm recognises `$hello` value as HTML expression and provides code highlighting, but no highlighting is performed for `$string` values. PhpStorm provides a separate code editor for every string-embedded language. The drawback is that the tool does not provide error reporting for incorrect expressions (see `$error` in figure 1).



```
<?php
    $usualString = 'simple string';
    $hello ='<html><body>Hello, world!</body></html>';
    $string = '<';
    if (cond)
        $string .= 'html';
    else
        $string .= 'body';
    $string .= '>';

    $error = 'SELECT * FROM table1 FROM table1';
?>
```

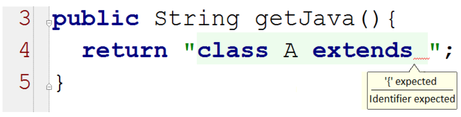
Figure 1: HTML code embedded in PHP code in PHPStorm

2.1.2 IntelliLang

[1]ReSharper is plug-in to Microsoft Visual Studio, extending standard IDE functionality. Site (accessed: 07.07.2014):`http://www.jetbrains.com/resharper/`

[2]IDE for PHP programming language. Site (accessed: 07.07.2014): `http://www.jetbrains.com/phpstorm/`

IntelliLang[3] — plug-in to PhpStorm IDE and IntelliJ IDEA[4] performing code highlighting and error reporting for string-embedded languages. It provides a separate code editor for embedded language processing by analogy with PhpStorm. The screenshots of IntelliLang are presented in figures 2 and 3.

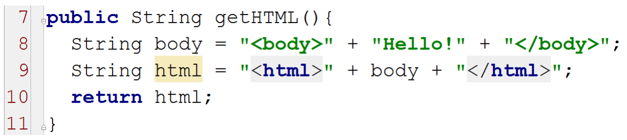


```
public String getJava(){
  return "class A extends ";
}
```

'{' expected
Identifier expected

Figure 2: Embedded Java code fragment in IntelliJ IDEA

The drawback of the plug-in is that it is necessary to specify manually the language of every string expression to be analysed. Figure 3 illustrates the drawback: if substring `"<html>"` is marked as HTML expression, IntelliLang highlights this tag and the one matching it, but it does not highlight variable s value despite the fact it is used in HTML code construction.



```
public String getHTML(){
  String body = "<body>" + "Hello!" + "</body>";
  String html = "<html>" + body + "</html>";
  return html;
}
```

Figure 3: `"<html>"` is marked as HTML language and `"<body>"` is not marked

2.1.3 Alvor

Alvor[5] [?] — plug-in to Eclipse IDE — intended to statically validate SQL expressions embedded into Java code. It does not require specification of dynamic expression language. The tool structure is presented in figure 4.

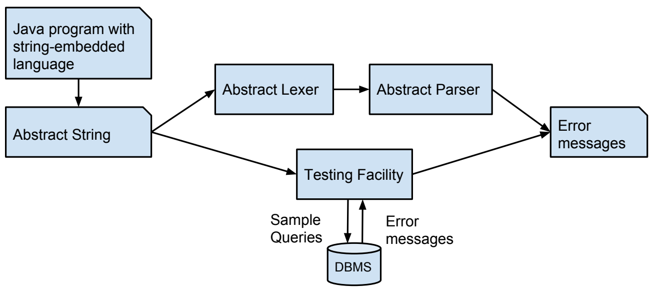
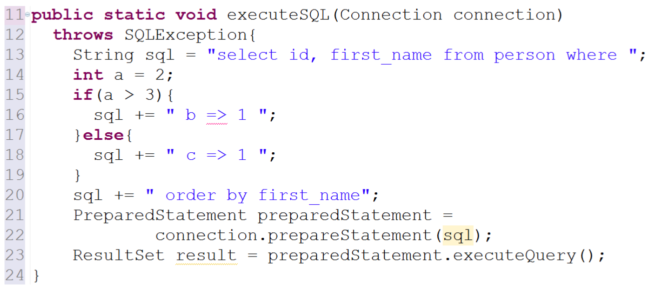


Figure 4: Alvor structure

The value set of dynamic expression is represented with abstract string — in the context of Alvor it is regular expression. After the representation is created, abstract analysis based on GLR-analysis is performed. The analysis reports lexical and syntactic errors. If there are more than one error in the expression, only the first of them is reported (see figure 5).



```
public static void executeSQL(Connection connection)
  throws SQLException{
    String sql = "select id, first_name from person where ";
    int a = 2;
    if(a > 3){
      sql += " b => 1 ";
    }else{
      sql += " c => 1 ";
    }
    sql += " order by first_name";
    PreparedStatement preparedStatement =
            connection.prepareStatement(sql);
    ResultSet result = preparedStatement.executeQuery();
}
```

Figure 5: Error reporting in Eclipse IDE using Alvor

Alvor statically validates SQL-queries constructed of string primitives using concatenations and conditional statements in interactive mode, and performs interprocedural analysis. Alvor supports several SQL dialects (Oracle PL/SQL, MySQL, PostgreSQL), but adding of the new languages support requires source code modification.

2.1.4 Java String Analyzer

Java String Analyzer[6] [?] — tool answering the question of syntactic correctness of dynamically generated expressions embedded in Java. The structure of JSA is presented in the figure 6.

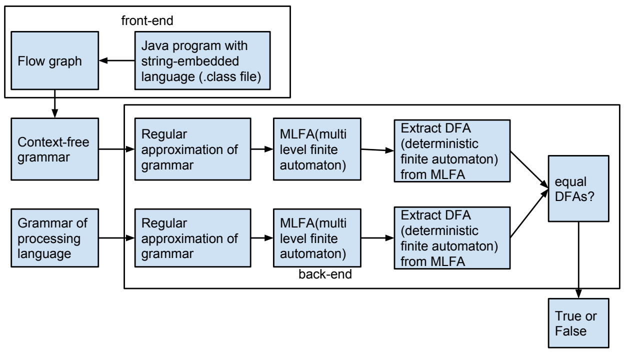

front-end
Flow graph
Java program with string-embedded language (.class file)
Context-free grammar
Grammar of processing language
Regular approximation of grammar
MLFA(multi level finite automaton)
Extract DFA (deterministic finite automaton) from MLFA
Regular approximation of grammar
MLFA(multi level finite automaton)
Extract DFA (deterministic finite automaton) from MLFA
back-end
equal DFAs?
True or False

Figure 6: JSA structure

JSA allows to process several embedded languages by modifying only front-end of the tool which creates flow-graph representation of the input data. Back-end of the tool represents a flow-graph as a context-free grammar which is approximated with regular grammar, and then a finite automaton is constructed by regular grammar. This automaton approximates value set of dynamic expression. The regular approximation and finite automaton are constructed similarly for the reference grammar of the language being analysed. After that, two automata are compared to each other, and

[3]IntelliLang is a plug-in offering a number of features related to the processing of embedded languages. Site (accessed: 07.07.2014): `http://www.jetbrains.com/idea/webhelp/intellilang.html`

[4]IntelliJ IDEA is an IDE for JVM-based development. Site (accessed: 07.07.2014): `http://www.jetbrains.com/idea/`

[5]Alvor project site (accessed: 07.07.2014):`https://code.google.com/p/alvor/`

[6]Java String Analyzer project site (accessed: 07.07.2014):`http://www.brics.dk/JSA/`

if they are equal, then the analysed expression is assumed to be syntactically correct.

2.1.5 *PHP String Analyzer*

PHP String Analyzer[7] [?] — tool for static validation of dynamic expressions, constructed by PHP programs. HTML and XML expressions can be analysed as embedded. The tool is based on the ideas of JSA algorithm, but the authors proposed to use context-free approximation instead of regular.

2.2 String-Embedded Languages Processing

We perform step-by-step static abstract analysis of value set of dynamically generated expression approximated with finite automaton in our framework. Finite automaton can be represented as a graph and this representation is very intuitive, so we say hereinafter that it is a graph which approximates input. The description of analysis steps is provided below.

The first step is *approximation* — creation of compact representation of dynamic expression possible values. We use graph representation of finite automaton: edges labeled by strings and nodes corresponded with concatenations used to build dynamic expression. Note that this step is frontend-specific, so approximation function should be implemented for each tool created with our platform. Another important restriction: the result of approximation must be a **direct acyclic graph** (DAG) because core functions such as abstract parsing can process DAGs only in current implementation [?].

The next step is *abstract lexing* or *tokenization* of approximation result. This step transforms an input graph with string edge labels to a graph with token labels. Token part positions in source code data — references to string literals which contain parts of token and position in this literals — are preserved during tokenization. Abstract lexer can be generated from lexical specification of the language to be processed. Generator is based on FsLex [8] — lexer generator for F# [?].

Further, *abstract parsing* is applied to tokenized graph. Abstract parsing algorithm is based on GLR parsing algorithm which can process ambiguous context-free grammars [?]. The result of abstract parsing step consists of a parse forest for all correct values and a set of errors occurred in incorrect values. We use classical GLR **graph structured stack** (GSS) which allows to fork and merge stack according to forks and merges in input graph. We use standard generator for parsing tables building from grammar specified in Yard language [?] which allows to use attributed grammars making semantic specification possible.

As a result of abstract parsing, one gets a forest containing trees for each correct value produced by finite automaton approximating input. In the context of embedded language analysis the size of such forest is usually huge. A big number of nodes are common for different dynamic expression values and may be reused so we use **shared packed parse forest** (SPPF) [?] introduced by Rekers to compress parse forest. One often needs to calculate semantic for parsing results and it meens manipulation with separate trees, not SPPF. It is enough to enumerate not all the trees from SPPF but some subset for some tasks. But sometimes enumeration of all the trees from parse forest is required, and it can lead to memory and performance issues. We propose lazy enumeration of trees to solve them. One of possible function of forest subset extraction and lazy enumeration was implemented and will be described below.

3. THE PLATFORM

The majority of IDEs can support more than one programming language and provide the possibility of extension to support new languages. The similar functionality is necessary to extend existing IDEs and code editors with string-embedded languages support. We propose to use classical modular mechanism implying that new language support should be provided as single package named *language extension package* or *language package*. This way is suitable for end users and for language package developers because it allows to develop and use different language extensions independently.

Existent tools for string-embedded language processing are targeted to one or several concrete languages or require modification of existing code to add new language extension package. This approach limits the following development of the instrument, so it is necessary to provide developer toolkit (SDK) for language package developers and a simple package installation mechanism for users. Moreover, we should provide an ability to specify language for string variables in user code. It is necessary to provide project-depended data on mapping between strings and embedded languages used in it.

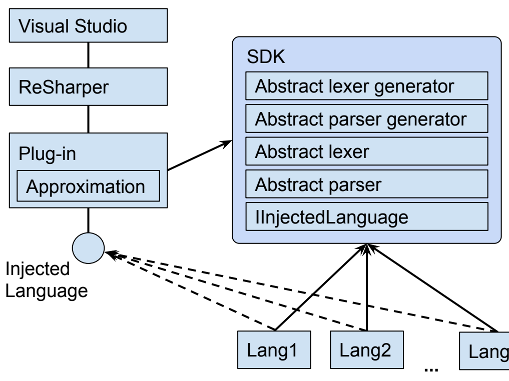


Figure 7: High-level platform structure

In figure 7 you can see a general structure of our platform. IDE integration part is based on ReSharper — Microsoft Visual Studio plug-in which provides additional functionality for code refactoring, code navigation, etc. We integrate string-embedded language support as plug-in for ReSharper.

[7]PHP String Analyzer project site (accessed: 07.07.2014):`http://www.score.cs.tsukuba.ac.jp/~minamide/phpsa/`

[8]FsLex is lexer generator for F#. Documentation (accessed: 07.07.2014): `https://fsharppowerpack.codeplex.com/wikipage?title=FsLex%20Documentation`