

Usage of XML as AST templates

Anakreon Mentis

September 21, 2004

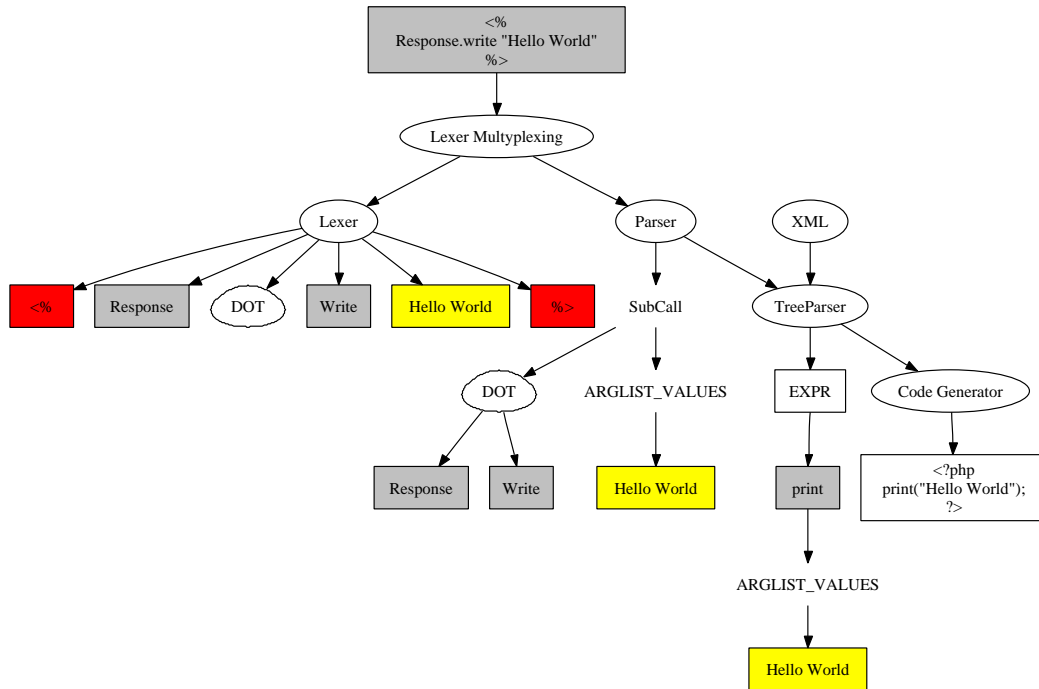
Abstract

ASPAs is a source to source compiler which translates ASP into PHP code. ASPAs aims to produce PHP code of equal functionality with the initial ASP source. In order to achieve this, beside the proper translation of JScript and VBScript language structures into the proper language structures of PHP, functionality provided by built-in methods and Objects should be provided also in the PHP version of the code. An other important request which the translator fulfills is the translation of ActiveX components available to the ASP programmer. Because of the wide adoption of various ActiveX components from ASP programmers, ASPAs had to be capable of translating the ActiveX method calls and properties in order to be useful.

1 Overview of ASP

An ASP page contains directives to the ASP engine (like the language definition and page includes), plain text and source code in a programming language. The programmer can virtually use any programming language as long as an implementation for that language exists. ASPAs recognizes only JScript and VBScript which are the most commonly used languages in ASP pages. The code is enclosed inside the `<%` and `%>` tokens or the `<language=lang runat=Server>` directive.

2 Overview of the translation process



The fact that ASP contains code for different languages implies the usage of different Lexers for every language and the combination of the token streams produced by the Lexers for syntax analysis. Each section of the ASP page is handled by the appropriate Lexer. Antlr provides such a mechanism, via the `TokenStreamSelector`, which enables the appropriate Lexer on every occasion and then transfers the control to the next Lexer when the task of the first one is over. The token stream produced by the Lexers of JScript and VBScript are used as input for the Parsers of the above languages respectively. The product of the Parsers will be a forest of AST and each of the trees of the forest is used as input for a `TreeParser`. The `TreeParser` transforms the input tree into another AST based on the rules of the grammar and information provided by the XML files.

3 Description of method calls and ActiveX components

An ActiveX is a collection of properties and methods. Each property can be read-only, write-only or support both read and write operations. The methods may accept parameters and also may alter the internal state of the ActiveX instance.

We made the assumption that every method or property can be emulated by a call to a PHP function or a code block, which should provide the same functionality. If a PHP function is used, the order of the parameters provided may be different or some parameters may be discarded or others may be added as constants in case the PHP function requires more parameters than that of the ActiveX.

Since the desired output of the TreeParser is an AST which encodes PHP code, the ActiveX function definition along with its translation could be stored in a treelike form. A well accepted standard which describes trees is XML, and for this reason was chosen.

4 Structure of XML documents

```
<method name="Write" type="VOID">
<arg type="DSTRING" />
<ast name="METHOD_CALL" text="print">
  <ast name="ARGLIST_VALUES">
    <arg index="1" />
  </ast>
</ast>
</method>
```

The first line defines a method called Write which belong to the Response built-in ActiveX component and returns nothing (the return type of the method is VOID). In the second line the argument of the method is defined but currently the argument's type is not used. The next five lines define the structure of the produced AST.

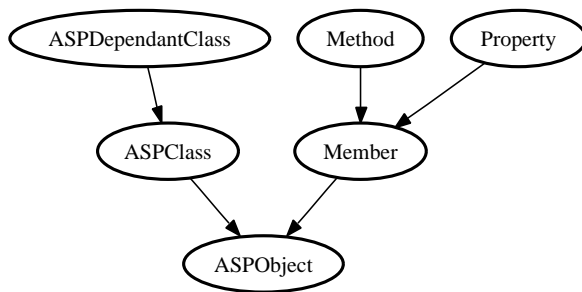
Each "ast" element represents an AST node who's type will be equal to the "name" attribute and the "text" attribute defines the text which the AST node holds. The children of the "ast" elements will be children nodes of the produced AST.

The element "arg" is used as a place holder and is substituted by the argument the method actually receives, which is also an AST. The "index" property of the element defines the index of the parameter which is used.

5 Translation of method calls

The XML files are loaded in runtime during the application initialization, and the information contained in those files are stored internally as Java Objects. When the TreeParser visits the AST nodes the Parser produced, locates nodes which refer to instances of built-in objects, method calls or ActiveX components. Based on the name of the method or object, the appropriate Java Object is found. If the object represents a method, the parameters which are provided to the method are passed to the Java Object as a List. The Java Object now has all the information it needs to produce an AST which contains the translation of the method or property.

The image below displays the classes which store the information provided by the XML files.



We will utilize a few examples in order to demonstrate the translation process.

1. Response.write "Hello World".

The AST produced is:

```
(SUB_CALL (DOT Response write) (ARGLIST_VALUES "Hello World")).
```

The Response node should refer to an instance of a class or an ActiveX component since a member is requested from it. The TreeParser attempts to find an instance called Response or a class with this name. The result of the search will be an ASPClass instance, which stores the properties of the Response ActiveX component. The Response object is queried for the member named "write" and a Method instance is returned. From the (ARGLIST_VALUES "Hello World") subtree, a List is created with only one element, the "Hello World" AST node, which is passed to the Method instance. The Method now has all the information it needs to produce the translated AST, which is (print (ARGLIST_VALUES "Hello World")) and substitutes the original tree.

2. The second example demonstrates the translation of built-in VBScript functions. Consider the expression len(someString). The TreeParser

attempts to locate a Method instance which describes the method called "len". Since the method is described in XML, the search will be successful. The parameter provided to method "len" is passed to the Method object and the translation is the AST (`strlen (ARGLIST_VALUES someString)`).

3. Beside the methods and ActiveX, we should consider the built-in objects of JScript like Array, String and others. Consider the expressions:

```
a = "Hello";  
b = a.length;
```

When a value is assigned to a variable, the TreeParser checks if the value is an object or an object reference. In this case, the variable "a" keeps a String object. In the second expression, the value of the property length of the String object is stored in the variable b. The TreeParser already knows that the identifier "a" references a String class, and queries the class for the member "length". The result will be an instance of the Property class. Obviously, the second expression is reading the value of the property "read", so the TreeParser requests the AST for the read process of the property which is: (`strlen (ARGLIST_VALUES a)`).

6 Conclusions

The usage of XML in the translation process of ASPA, provides greater flexibility, because the support of an other ActiveX component is only a matter of adding an other XML AST template for the component. The user does not need to learn Java or Antlr in order to add functionality to the project.

An other benefit is the ability which the user has, to alter the translation ASPA performs on an ActiveX or on a specific method or property of the ActiveX. For example, if the user prefers the method "echo" instead of "print", he can easily impose this preference by simply altering the template of the method.