

Background

It has been established in the software engineering industry that Code-Generation is a viable approach to Agile Software Development.

Several products currently available in the market leverage concepts of code-generation primarily to reduce development time, improve performance and stability, and generally to disseminate best-practice-approaches in an easy-to-use form. Products available are diverse in origin and philosophy – ranging from Open Source Frameworks such as Django, to Industry-leading products like Microsoft Visual Studio.

All these tools provide or incorporate some level of “canned” functionality which expresses itself as code which specifically addresses a particular application. BrightSword Designer was a tool built along these lines to heavily assist the development of database-driven web applications.

However, there is no generally available tool to generically build such code-generation frameworks. Significant effort in the development of BrightSword Designer was applied to building the code-generation engine, which is heavily linked to the problem domain itself.

Therefore, we took it upon ourselves to build a general-purpose code-generator-generator, which would be easy to use by moderately-equipped developers to assist them in building their own custom code generators.

Synopsis

BrightSword LongReach™ is a suite of development tools which can be used to build language- and platform-agnostic code-generators for metadata-parametrizable patterns.

It includes cstone.exe, a brief discussion of which follows. Discussion of the other tools included in the suite is outside the scope of this document.

Technical Discussion

Metadata-driven pattern-based code-generators are roughly analogous to “mail-merge” letters.

An instance of data conforming to some metadata is applied to a pattern (or template) parameterized by that same metadata to obtain a “merged” document containing the constant portion of the template and the variable portions filled in from the data.

There may be many approaches to doing the “merge”. One of the ways of performing the merge is to somehow convert the template into a program that takes the data instance as input and generate the merged document as output. In effect, there would have to be a code-generator-generator to transform the given template into a code-generator that can be executed.

Our method of performing the merge operation can be deconstructed into the following actions:

1. At Code-Generator Creation Time
 - a. Define the metadata that parameterizes the template.
 - b. Develop the template, fixing the boilerplate portion and marking out the variable portions with reference to the metadata
 - c. Apply some tool that converts the template into a program that takes an instance of the metadata as input and generates the merged document as output. The output is the code generator we're seeking. The tool is `cstone.exe` – a part of the BrightSword LongReach™ Suite.
2. At Code-Generator Usage Time
 - a. For each time we want to exercise the code-generator, develop a specific instance of the metadata.
 - b. Exercise the code-generator on this metadata to generate the specific merged documents. This is the target application.

It's easy to see that when Step 1 is applied to commonly used pattern, the investment performed is leveraged effectively by multiple invocations of Step 2.

The key take-away is that we have a tool to build such pattern-based generators with minimal effort and basic programming knowledge.

We use XML to define the metadata in our approach, and the CST “markup” language to define templates.

Description of Suite Components

CST “markup language”

We have developed an easy way to effectively create templates from existing code-snippets or patterns. The CST “markup” language allows a template developer to quickly annotate code and specify placeholders for parameter values. Since the metadata is encoded in XML, we can specify parameter values as XPath expressions in the code. The syntax of the CST markup language minimizes the traditional suffering associated with hand-writing XML or XSLT to the extent possible.

A sample CST script may look like this:

```

emacs@GROMIT
<TEMPLATE>
MATCH : class
MODE : Handler
</TEMPLATE>
<DECLARE>
<xt:comment/>
</DECLARE>
<BODY>
namespace [%../@name%]
{
    using System;
    using BrightSword.olive.Core;
    using BrightSword.Reminisce.Core;
    using BrightSword.SoftwareEngineering.Core;

    <INLINE>
    <xt:variable name="class-t">
    <xt:apply-templates select="." mode="normalize"/>
    </xt:variable>
    <xt:variable name="class" select="msxsl:node-set($class-t)/class"/>
    </INLINE>
    public class [%$class/@name%]Handler : BrightSword.Reminisce.Core.ReminisceHandler
    {
        public [%$class/@name%]Handler(ReminisceRequest Request)
        {
            base.Request = Request;
        }
    }
}

```

Figure 1: A simple CST fragment

The .cst file above generates C# code (in fact, it is an actual template used to generate IHttpHandlers in the BrightSword Reminisce™ Web Server). Notice that the C# code is largely “native”, in that the formatting, indentation and such are exactly what a developer might write and expect to read.

The following succinct description of all the “tags” that comprise the CST “markup” language underscores its simplicity. There aren’t too many exceptions, and a cursory examination of a few sample CST files is generally sufficient to become conversant with CST:

- The <TEMPLATE> tag defines the CST template. The name of the CST template is traditionally the same as the value of the “MODE” attribute in the tag.
- The <DECLARE> tag allows for declaration of global-scoped templates, variables and parameters. If none is to be used, an empty <xt:comment/> suffices.

- The <BODY> tag delimits the actual boilerplate of the template. It can contain formatted text in any language (programming or otherwise), which is retained through the CST transformation process.
- Placeholders in the boilerplate are marked with the [%%] delimiters, which contain an XPath expression relative to the node matched by the MATCH attribute in the TEMPLATE tag.
- The <INLINE> tag is used to delimit sections of native XSLT code, which operate in the context of the data document, and get executed inline.

When we consider the following CST fragment, which generates a T-SQL stored procedure to delete a record in a database, we make the following observations:

1. The entire XSLT language set is available within CST. The *xt* namespace refers to <http://www.w3.org/1999/XSL/Transform>.
2. The <xt:for-each> seen executes exactly as expected, ensuring that the column name and type are appropriately computed for each property satisfying the condition.
3. A whole library of XSLT routines (<xt:template> and <xt:transform>) are available to the template developer, and more can be easily added by the developer to the CST-processing environment. For example, the <xt:call-template> tag seen binds at CST-processing-time to the appropriate template in the environment.
4. The color-coding seen in the editor comes from the *cst-mode* major mode for Emacs which switches dynamically between *xslt-mode* and *c#* mode depending on the cursor location. XSLT code is presented in orange, and SQL code in yellow.
5. In summary, the *only* programming skills required to use the CST “markup” language are a working knowledge of XSLT and sufficient skill in the target language in which the template code is to be generated.

As an example, using the CST processing environment, we at BrightSword were able to build the template which generates the exact code for Visual Studio 2005's TypedDataset class *in 6 hours*.

```

emacs@GROMIT
CREATE PROCEDURE [[%$class/@deleteproc-name%]]
<INLINE>
  <xt:for-each select="$class/deep-properties/property[@is_parent_id='true' or not(@is_reference='true')][@column-name='ID']">
    <xt:variable name="sql-type-v">
      <xt:call-template name="get-sql-type"/>
    </xt:variable>
    <xt:variable name="length">
      <xt:apply-templates select="." mode="get-length"/>
    </xt:variable>
    <xt:variable name="sql-type">
      <xt:choose>
        <xt:when test="$length=''">
          <xt:value-of select="$sql-type-v"/>
        </xt:when>
        <xt:otherwise>
          <xt:value-of select="concat($sql-type-v, '(', $length, ')')"/>
        </xt:otherwise>
      </xt:choose>
    </xt:variable>
  </xt:for-each>
  @p_[@column-name%] [%$sql-type%],
<INLINE>
</xt:for-each>
</INLINE>
  @p_currentTime datetime
AS
BEGIN
  BEGIN TRANSACTION TX_DELETE
  <xt:for-each select="$class/contained-tables/table">
    <xt:for-each select="IF EXISTS (SELECT # FROM [[%@name%]] WHERE ID = @p_ID)">
      BEGIN
        UPDATE [[%@name%]] SET
          [Internal_IsDeleted] = 1,
          [Internal_LastModifiedTime] = @p_currentTime
        WHERE ([ID] = @p_ID)
      END
    </xt:for-each>
  </xt:for-each>
  COMMIT TRANSACTION TX_DELETE
END

```

Figure 2: A more complex CST fragment

cstone.exe

This is the tool that converts CST files and specified data XML into the generated code.

```

c:\Work\svn\projects>cstone --help
Setting environment for using Microsoft Visual Studio .NET 2003 tools.
(If you have another version of Visual Studio or Visual C++ installed and wish
to use its tools from the command line, run vcvars32.bat for that version.)

c:\Work\svn\projects>cstone --help
*****
*** cstone - Applies the Cornerstone Code Generation Engine according to the given parameters
***
*** Usage: cstone --help --verbose --xml=<value> --cst=<value> --utils=<value> --xmlfile=<value> --match=<value> --mode=<value>
***
*** --help          : [Optional] Prints this message and quits
***
*** --verbose       : [Optional] Emits more information
***
*** --xml=<value>    : [Optional] Specifies the directory in which the xml files are stored. If not specified, uses the 'xml' directory.
***                   --xml has a default value of [xml]
***                   The effective value of --xml is [xml]
***
*** --cst=<value>    : [Optional] Specifies the directory in which the cst files are stored. If not specified, uses the 'cst' directory.
***                   --cst has a default value of [cst]
***                   The effective value of --cst is [cst]
***
*** --utils=<value>  : [Optional] Specifies the directory in which the utility and include files are stored. If not specified, uses the 'utils' directory.
***                   --utils has a default value of [utils]
***                   The effective value of --utils is [utils]
***
*** --xmlfile=<value> : [Optional] Specifies a single xml file to generate code for.
***                   --xmlfile has a default value of []
***                   The effective value of --xmlfile is []
***
*** --match=<value>  : [Optional] If --xmlfile is specified, specifies a single xml node in the xml file to generate code for.
***                   --match has a default value of []
***                   The effective value of --match is []
***
*** --mode=<value>   : [Optional] Specifies a single cst pattern to generate code for.
***                   --mode has a default value of []
***                   The effective value of --mode is []
***
*** Please ensure that there is no space except between arguments. Quote values that embed a space in them. Use '--arg=value' and not '--arg = value'
***
*** Exiting...
***
*****

```

Figure 3: cstone --help -The command line options for the tool.

cst-mode for Emacs

As mentioned above, there is an Emacs Major Mode based on multi-mode.el, which switches between XSLT and C# based on the position of the cursor. This is useful because the sub-modes provide syntax coloring and indentation appropriate to their native languages. It is possible to extend this to other language syntaxes as well.

Summary

The CST “markup” language can be used to develop templates, which, when coupled with XML metadata, and processed through cstone.exe, will result in generated code.

cstone.exe is platform- and language- agnostic in that the code that it generates exactly follows the pattern specified. cstone.exe adds no further information and imposes no further constraints than those imposed by XSLT and the native language in question.