srcML Tool Implementation on a C# Environment

Jesus Eduardo Jaime Gandarilla
Department of Computer Science
The University of Texas at Dallas
jxj132730@utdallas.edu

1. Introduction

This project's main goal is to detect Common Coupling in a C# language code base using the set of tools provided by the srcML Framework.

Coupling is defined as a measure of interdependence among modules in a computer program [1]. In our case we chose, Common Coupling, also know as Global Coupling which occurs when two modules share the same global data or variable.

On the other hand, srcML is an infrastructure for the exploration, analysis, and manipulation of source code [2] which translates code from a project in Java, C++, C or C# into a single XML format document. All original text is preserved so that the original source code documents can be recreated from the XML file [3] and even keeping the directory structure of the project. The latest version of this toolkit was posted in May, 2015.

2. Implementation

As a first step, we searched for the different options available to carry out code analysis for a project in C# and more precisely, with the intention to use these tools inside a Visual Studio Project to create an automated tool for code analysis.

At the end of the day, we found that there are three viable options to implement srcML in a C# code base.

2.1 Option 1 - Command Line Tool

This is the tool available to download in the project's website: srcml.org. The installation and use is very simple:

1. Go to www.srcml.org
2. Click on Downloads
3. Select the version for your OS
4. Install the executable.
5. Open a command line.
6. Execute srcML command as shown in figure 1.

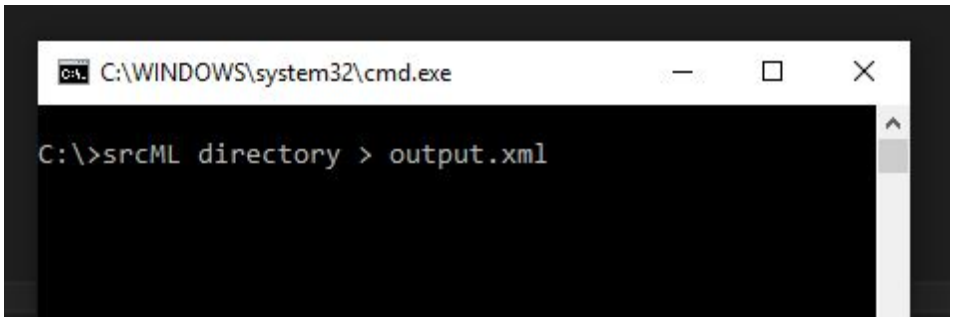

Figure 1. Converting a project into a single xml file in srcML Format.

This toolkit also offers the following commands:

1. src2srcml: Converts a directory or a file in any of the supported programming languages into a XML File in srcML Format.
2. srcml2src: Converts a XML File in srcML Format into a file or project files in the original programming language and preserving directory structure.
3. xpath: By passing a XPATH query we can get nodes in the XML File.

In our case, we will not implement this option since we are interested in creating a tool in Visual Studio to automate this process. We are specially interested in offering XPath query capabilities in our tool, but we may use this option to create the XML File only.

2.2 Option 2 - As a Visual Studio Service

srcML Service can be installed in Visual Studio by click on tools, selecting extensions and updates and then looking for it in the online tab.

This option provides analysis services for other visual studio extensions to consume. It is useful if you want to develop Visual Studio plugins, such as Sando Search tool which is an improved text search tool, also sponsored by ABB Research Center and uses srcML Service to get notified of file changes. [4]

This option is not useful for our project's needs. However, it is worth mentioning so that the audience knows that there are some standalone tools available that implement srcML.

2.3 Option 3 - As a package in a Visual Studio Project.

ABB Research Center provides a framework that they have been using within themselves to do both program transformation and code analysis.

This framework is named srcML.NET, and can be downloaded using GitHub or directly 'installed' into a Visual Studio Project using the Package Management Console.

To do so, one must follow the following steps:

1. Open visual studio and create a new project.

2. Click on View > Other windows > Package Management Console
3. Enter the following command:

```
PM> Install-Package ABB.SrcML
```

4. Now you can import ABB.SrcML and code a small tool. Figure 2 shows an example of how to create an instance of the ABB.SrcML class that converts a project into a srcML XML File.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace srcML_Test
{
    class srcML_TestApp
    {
        public void main() {
            ABB.SrcML.SrcML srcml = new ABB.SrcML.SrcML();

            srcml.GenerateSrcMLFromDirectory("c:\\path\\codefolder", "results.xml");

        }
    }
}
```

Figure 2. Creating a class that converts a project into an XML file.

In this project, we will choose this options as it is the one that fits best our goals:

1. Create a small tool that converts a project in C# into a XML file in srcML Format.
2. Provide code analysis capabilities with the same tool in order to detect Common Coupling in the project.

2.4 More on ABB srcML.NET

srcML.Net is a framework used within ABB Corporate Research and it's based on the srcML project from Kent State University Software Development Laboratory.

It is used to do both program transformation and code analysis, and includes the same tools and functionality offered by srcML but allows integration to Visual Studio Projects to develop code analysis tools. [5]

3. Projects to be analysed in this project

At this point, we used option 1, the command line tool, to complete the first step in this project which is converting a code base in C# into a single XML file in srcML Format.

As you can see in table 1, the conversion from C# to srcML inputs a XML file that is between 2.5 and 2.8 times larger than the code base. In our biggest size project, Mono, the output XML file is 455 MB in size which is still considered, in our opinion, a processable size and does not require splitting the file into more files.

The Mono project took around a minute to be converted into XML and all the others were converted almost instantly by the tool.

Project	Number of .cs files	Total Size (MB)	Total Size of XML
CodeHub	344	1.64 MB	4.55 MB
MonoGame	1,229	9.24 MB	24 MB
Newtonsoft.Json	2,014	2.46 MB	6.57 MB
Mono (<1min)	24,819	161 MB	455 MB

Table 1: Code base conversion results

If enough time is available, we are planning to include one of the following Java Projects: Freemind or OpenCMS.

It is also worth mentioning that, for our analysis and tool creation, we will be carried out using a computer with the following characteristics:

- 6th Generation Intel Quad Core i7 Processor
- 16 GB DDR4 of RAM memory.

4. Structure of the selected approach

In this section we will define the structure of the selected approach for the implementation of srcML in a C# environment.

After some tests we found out that the functions available through srcML.NET help to convert the code base easily from source code to srcML format. However, the functionality available to query the file is more complex than the standard XPath implementation of the .NET Framework.

srcML.Net has a class named srcMLFile wich can execute a query using LinQ. LinQ stands for Language-Integrated Query and is a set of features introduced in Visual Studio 2008 that extends the capabilities to the language syntax of C#. Visual Studio includes LinQ provider assemblies that enable the use of LinQ with SQL Server Databases, ADO.NET Datasets, and XML Documents. [6]

Since LinQ is we want to keep our srcML implementation as universal and easy to understand for everybody as possible, we opted to use the simple XPathDocument class that receives an XPath Expression as a parameter to execute a Query. We will then go on and create a tool in C# and Visual Studio which loaded the converted code base in srcML XML Format and offered querying capabilities.

The tool shall offer the following functions:

1. Load a XML file in srcML Format.
2. Execute custom queries using the XPathDocument class available in the .NET Framework.
3. Execute a specific scan type:
    a. Detect Common Coupling with Method granularity.
4. Convert a folder to srcML Format.

5. Analysis of the Code Bases

The next step was to analyse a code base sample in order to detect the attributes and tags that we want to identify and then based on this information, generate the XPATH queries necessary for our tool:

By taking a look at the srcML File structure, we identified the following components:

Units (Files)

Units represent the individual source code files by using the tag <unit>. Some attributes like language and filename add useful information to identify the file.

Each unit includes, among other components, comments, blocks of code, declaration statements, and functions. The following is an example of the <unit> tag:

```
<unit xmlns:cpp="http://www.srcML.org/srcML/cpp" revision="0.9.5"
                language="C#"
                filename="ConstructorHandling.cs"
                hash="336e2f0959f373b22eab5cdef109f7a2d72c196a">
        <decl_stmt>(Declaration statement)</decl_stmt>
        <function>(Method definition)</function>
</unit>
```

Variable Declarations

The variable declaration statement is simply represented by the tag <decl_stmt> and includes components like type and name. The following example represents the declaration **"int myInt;"** :

```
<decl_stmt>
        <decl>
                <type> <name>int</name> </type>
                <name>myInt</name>
        </decl>;
</decl_stmt>
```

Function Structure

A function is represented by the tag <function> and includes the specifiers such as static or private, type, name and a code block that contains methods calls, variable declarations, etc. The following example represents the function **"static void doWork() { CODE BLOCK... }":**

```
<function>
        <specifier>static</specifier>
        <type> <name>void</name> </type>
        <name>doWork</name>
        <parameter_list>()</parameter_list>
        <block> {  (Comments, variables declarations and method calls)  }  </block>
</function>
```

XPath Querying

The first functionality available in our tool was the Custom Query. We decided to code this requirement first in order to figure out the queries needed for the coupling detection.

After getting familiar with the XML Structure used by srcML, we proceed to code the first path of the tool in order to offer simple custom query capabilities. The following table shows the results of this process, when we were finally able to retrieve a list of filenames, declared fields, and declared methods in the code base:

Objective	**XPath Expression**
Get all the files in the code base.	//src:unit/src:unit
Get filename for each unit.	//src:unit/@filename

Get all the fields.	//src:class/src:block/src:decl_stmt/src:decl/src:name
Get all the methods that use those fields.	//src:function[descendant::src:name='FIELDNAME']/src:name

6. Code Analysis

At this point and after a few modifications, our analysis algorithm resulted as follows:

1. Get all the units (source files) contained in our code base.
2. For each unit get all the declared Fields.
3. For each Field, detect if it's being used at least once in a function.
4. Display the results if it is the case, ignore otherwise.

For the first part of the analysis, we created a test XML file using selected source files that we would use to measure the correctness of our analysis output.

For this test code base, we included 10 source files, ran the analysis using our tool and then manually checked the fields used. We then verified that our analysis tool skipped get and set methods, and fields used only in one method, since these don't represent coupling. The tool also skipped the interfaces and classes that do not contain coupled methods.

After this step, we can say that the tool successfully detects coupling in the methods inside a class. No further adjustment was necessary at this point.

Next, a full analysis was carried out using the converted code base of the following projects:

1. CodeHub
2. Mono
3. MonoGame
4. Newtonsoft.Json

We then proceeded to manually check a random sample of the output and we did not detect any false positives although we can not say that our tool is 100% accurate due to the random nature of our test sample. However, we can say that we are confident that we achieved the goal of this project in a great extend. Maintenance is part of the software development life cycle and improvements we expect to make some improvements to our tool in a later time.

7. Results

At the end of this project, we were able to create a tool using C# and Visual Studio that offers querying capabilities. In summary:

- The tool allows the user to enter a custom XPath Expression and execute it. The tool will display a list of results and this functionality can be used to better understand the structure of the code base or simply to obtain information about the composition of the classes in the project.
- The tool allows the user to load the code base in XML format and then carry out a “Coupling Analysis with Method Granularity” simply by clicking the button “Execute Scan”.
  The results of this scan are presented in the following format, and show the filename followed by the field and the methods in which it is used.

```
Filename : json\Src\Newtonsoft.Json\Utilities\CollectionWrapper.cs
The field _list is being used in the methods: Add, Clear, Contains, CopyTo, Remove, GetEnumerator,
IEnumerable.GetEnumerator, IList.IndexOf, IList.RemoveAt, IList.Insert, .
The field _genericCollection is being used in the methods: Add, Clear, Contains, CopyTo, Remove, GetEnumerator,
IEnumerable.GetEnumerator, IList.IndexOf, IList.RemoveAt, IList.Insert, .
```

Figure 3. Coupling analysis output format.

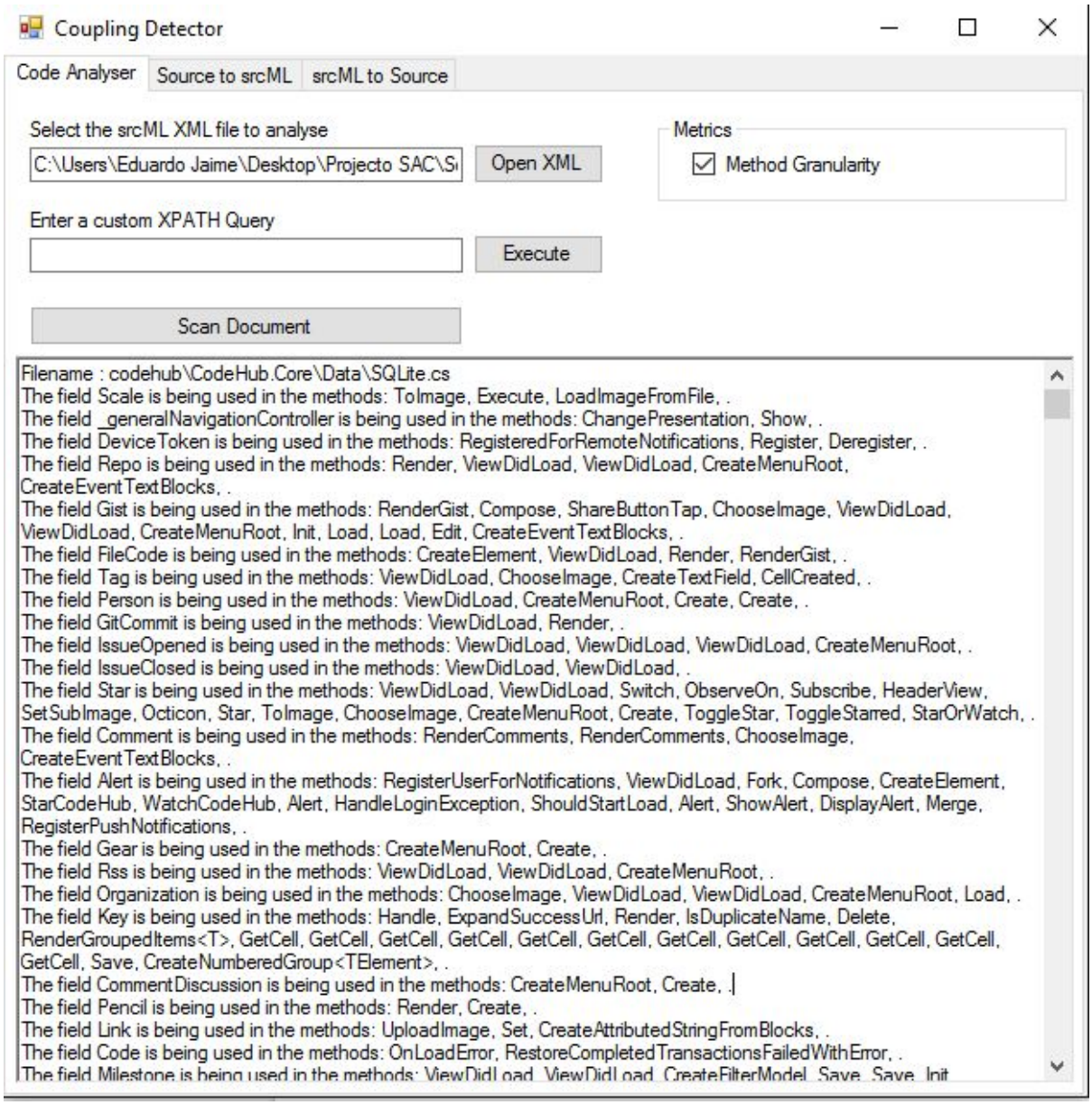

Figure 4. Tool after displaying results.

8. Conclusion and Future Work

In this paper, we presented our approach and tool implementing srcML in a Visual Studio and C# environment. During this project we learned that srcML can easily be implemented in any programming language that you like. Java, C#, VB.NET, or whatever language with XPath support to design your code analysis tools.

srcML makes it really simple to analyse the code and it was a huge difference from the Java JDT approach. The main difference is that when using srcML you are dealing with an XML document and the implementation of XPath expression makes it really easy to query the document and get the desired information.

In our future work, we would like to expand our tool's functionality in order to offer different kinds of analysis and even include analysis of JAVA code.

References

[1] IEEE, Guide to the Software Engineering Body of Knowledge (SWEBOK v3), 2014.
[2] srcML.org Home Page. Retrieved from: http://www.srcml.org/
[3] Collard, M. L. and Maletic, J. I., "Lightweight Transformation and Fact Extraction with the srcML Toolkit.", 2011.
[4] Visual Studio Gallery. Retrieved on April 14th, 2016 from: https://visualstudiogallery.msdn.microsoft.com/ca1c4937-170f-472b-b251-11a9f4d2a161
[5] srcML GitHub repository. Retrieved from: https://github.com/abb-iss/SrcML.NET
[6] Microsoft MSDN on LinQ. Retrieved on April 30th, 2016 from: https://msdn.microsoft.com/en-us/library/bb397926.aspx