High Performance XML/XSLT Transformation Server
Fall 2016 Progress Report

Zixun Lu (luzi), Shuai Peng (pengs), Elijah Voigt (voigte)

CS 462 | CS Senior Capstone | Winter 2017

February 13, 2017

Abstract

An update on the development of the High Performance XML/XSLT Transformation Server named *XZES40 Transformer*.

Figure 1: Source: Wikimedia Commons [2]

Figure 2: Source: Apache Software Foundation [1]

1 Project Purpose

XZES40 Transformer is an implementation of a standard XML document transformer. This type of application takes two documents as input: an XML document and an XSLT document, and performs and XML document transformation. This is similar to how in an Excel document one might take a series of columns, perform some operation on that data, and set the output to be a new column. Just like Excel, XML document transformation is used to perform operations on data, except that it can be highly automated done in much larger quantities with programs like ours.

XZES40 Transformer is unique in that it increases performance by both caching already-processed XSLT documents and it performs transformations in parallel. The use of an in-memory cache will cut down considerably on the time required to perform a document transformation. This is because much of the time transforming documents is spent compiling the (commonly re-used) style-sheet. The application will also accept transformation jobs over the internet via a web API; this allows users to perform document transformations without installing the application locally.

2 Project Status

2.1 Zixun Lu

2.2 Shuai Peng

Because all of our teammate is busy in this term, we are slow than our schedule that we planed in the design document. So far, we just complete the basic function of the XZES40-Transformer. The XZES40-Transformer can get one XML file and XSLT file and generate a new XML file. The cache and the parallel computation will be completed during the beta version.

2.3 Elijah C. Voigt

As mentioned above, our group has made unfortunately slow progress on development. I made considerable progress over winter break before the term started. This work included outlining the structure of the code, the code repository, setting up automated testing, and documenting what had been and had yet to be done.

Unfortunately we quickly reached a show-stopping bug which, with two out of three developers playing catch-up, halted development for a week. I made the decision after that week to gut the incorrect code and fix the bug by re-designing the problematic portion from the ground up. This fixed the bug and got us to completing the core function of our project: XML document transformation.

3 Remaining Tasks

3.1 Zixun Lu

3.2 Shuai Peng

The remaining tasks that I am responsible for is complete the cache function, web interface, and Windows package.

3.3 Elijah C. Voigt

Having completed basic document transformation, my remaining tasks include turning the application into a daemon (long-running process), exposing it over the web with a CGI script, creating a Command-line interface, and creating a FreeBSD package for the application. The latter two tasks being stretch goals, the former two being hard requirements necessary for the beta release of our application.

4 Problems Encountered

4.1 Zixun Lu

4.2 Shuai Peng

When we are working on the basic document transformation, we face a strange problem. The structure and the implementation of our code should be no any problem. However the compile also say that we have expected unqualified-id bugs for the XalanC++ transformer function. Elijah and me search on the Google, and most of answers said that you should check the syntax of code and some semicolon error. We double check the code, but there is no error such that syntax and semicolon problem. So, we come together and write a new prototype for test XalanC++ transformer, and the prototype is work correctly. Thus, we decided to re-write the transformer.cpp, and make sure this code is only doing one things – transformer document. We change the structure of transformer.cpp from OOP to normal function code, and also make sure the document.cpp pair this file. Finally, we get over this bugs, and our program can transformer document successful.

4.3 Elijah C. Voigt

Although we purposely scoped our project to be relatively simple and straight forward, we have encountered a lot of stumbling blocks. Technically we encountered the aforementioned unqualified-id compilation error, this halted development for far too long. Non-technically Shuai Peng and Zixun Lu struggled with getting caught up after break, Shuai fairing much better than Zixun. All together the biggest thing that has bitten us so far is prioritizing work on this project on-par or above that of other class-work.

5 Interesting Code

5.1 Zixun Lu

5.2 Shuai Peng

This is the structure of the XZES40-Transformer code repository.

```
shell> ls
bulid doc examples include lib Makefile README.md src test
```

This is the usage of the XZES40-Transformer.

```
shell> ./main
Usage:
  a.out --xml=input.xml --xsl=style.xslt [--out=output file]
```

This is the result of the XZES40-Transformer. We print the new xml file on screen for easy debug, but it will generate a xml file in the future.

```
shell> ./main --xml=simple.xml --xsl=simple.xsl --out=result.xml
<?xml version="1.0" encode="UTF-8"?><out>Hello</out>
```

5.3 Elijah C. Voigt

While it might not seem like much, this is the core of the XZES40-Transformer application. This function is passed a struct of three arguments, the XML file path, the XSLT file path, and the (optional) output XML file destination. For technical reasons it currently prints the output document to standard output.

```
int xzes::transform_documents( xzes::cli_arguments_t *args )
{
    // Initialize Xalan
    XMLPlatformUtils::Initialize();
    XalanTransformer::initialize();

    // create a xalantransformer
    XalanTransformer theXalanTransformer;

    // Allocate objects on the heap so they can be cached in the non-prototype version.
    // XSLTInputSource  *xml = cache.get(args.xml);
    Document xml(args->xml);
    Document xsl(args->xsl);

    // Perform transformation and capture results
    int theResult = theXalanTransformer.transform( *xml.get_content().obj ,
                                                   *xsl.get_content().obj ,
                                                   std::cout );

    // Terminate xalan
    XMLPlatformUtils::Terminate();
    XalanTransformer::ICUCleanUp();

    return theResult;
}
```

Eventually this will perform document transformation and setup in a separate thread for each requested transformation. That functionality will be the core of the 'Parallel transformation', an anticipated speed-boost of our application.

The Document class used to instantiate the xml and xsl variables are custom written classes. They encapsulates the Xalan-and-Xerces-specific code as well as handling the parsed-document caching. This creates a clean separation of concerns, which (in theory) makes our code modular for current development and more maintainable for future development and maintenance.

6 Relevant Media

We have not completed development enough to have screen-shots and demos to show. We hope to have a functioning website by the end of the term.

References

[1] *ASF Press Kit: Apache Software Foundation Logo*. URL: https://www.apache.org/foundation/press/kit/.

[2] *Wikimedia Commons: Oregon State University Logo*. URL: https://commons.wikimedia.org/wiki/File:Oregon_State_University_log