High Performance XML/XSLT Transformation Server

Spring 2017 Final Project Report

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

OSU CS Senior Capstone 2016-2017

June 1, 2017

Abstract

Abstract text.

CONTENTS

This document is intended to present a detailed description of the high performance XML/XSLT transformation server being developed by the Oregon State University CS Capstone Team "XZES40". The intended audience for this document are the developers and sponsors of the project.

2.1.1.2 Scope

The name of this software, for lack of a better one, will be XZES40-Transformer.

The core product being delivered is a high performance XML/XSLT transformation server. This server will be able to perform repetitive document transformations quickly and efficiently by caching previously processed and compiled documents. Time is saved by pulling from an in-memory cache of documents and their compiled state rather than downloading and compiling documents which have already been processed, as current systems tend to do. XZES40-Transformer will also carry out transformations in parallel. It will transfer documents to and from clients via the HTTP or HTTPS protocol.

The target platform for XZES40-Transformer will be Debian Linux 8 ("Jessie"). The core product will be designed to allow the program to be ported easily from Linux to other operating systems like Windows and BSD.

In addition to the core server there will also be a command-line interface and web-interface developed to interact with the application, these will be called XZES-CLI and XZES-Web respectively.

2.1.1.3 Definitions, acronyms, and abbreviations

Below is a list of acronyms and abbreviations used throughout the document:

- Extensible Markup Language XML: The human-readable data format used and processed by our application.
- Extensible Stylesheet Markup Language (XSLT): The human-readable format used to transform documents in our application.
- Xerces-C [**xerces**]: One library used to perform XML transformation in C.
- Xalan-C [**xalan**]: One library used to XML transformation in C.
- ICU [**icu**]: One library used to process UTF-8 character formated documents.
- Hypertext Transfer Procol (HTTP/HTTPS): The protocol over which XZES40-Transformer will interact with remote clients.

- HTTP Application Programming Interface (HTTP API): A standard way of communicating with a web application.
- Unified Resource Locator / Identifier (URL/URI): Addressable location of a resource over the internet (e.g., a website address).
- Debian 8 (“Jessie”): The target Linux-based operating system XZES40-Transformer will run on.
- Unicode Transmission Format 8 (UTF-8): The international standard for encoding text-based data.
- Apache Web-server: A Free and Open Source web-server.
- Common Gateway Interface Script (CGI Script): Server-side scripts that can run applications on client’s behalf.

2.1.2 Overall description

The following sections of this document outline the factors that affect the creation of XZES40-Transformer at a high-level.

2.1.2.1 Product perspective

System interfaces

The XZES40-Transformer application will interface with the outside world over the internet via the HTTP networking protocol. The application will recieve HTTP POST requests to the application URI endpoint containing the documents to be transformed. Once the transformation is completed the transformed document will be sent to the user for download.

In addition to the transformed file the application will respond with an HTTP **OK** status. If an error occurs it will respond with a **SERVER ERROR** status and no file.

User interfaces

XZES40 will have two user interfaces which will access it’s functionality over the internet.

- A Website to access XZES40-Transformer via a web-browser. This interface will be called “XZES40-Web”.
- A CLI to access XZES40-Transformer via a terminal interface. This interface will be called “XZES40-CLI”.

Both interfaces will not perform local document transformation. They will instead access the transformation service over the internet, making the HTTP API convenient to use.

Hardware interfaces

XZES40-Transformer will not have any direct physical interfaces as it is meant to be interacted with over HTTP or HTTPS. Any computer with an internet connection, monitor, input methods, and web-browser will be able to access XZES40-Transformer via the web interface. Any computer with an internet connection, monitor, input methods, and which has the CLI installed will be able to access XZES40-Transformer via the CLI interface.

The application will be targeted to run on a Debian Linux 8 (“Jessie”) X86_64 CPU architecture server. This machine should have one port open for communicating over HTTP (port 80) and one for HTTPS (port 443) if that is configured.

Software interfaces

Below is a list of software required for the XZES-40 (on the host and on remote systems).

Name: Xerces-C++ XML Parser

Mnemonic: Xerces-C

Specification Number: XML 1.0 specification is implemented.

Version Number: 3.1.4 (Recent)

Source: http://xerces.apache.org/xerces-c/

Name: Xalan-C++ XSLT Processor

Mnemonic: Xalan-C

Specification Number: XML 1.0 specification is implemented.

Version Number: 1.10 (Recent)

Source: http://xalan.apache.org/xalan-c/

Name: International Components for Unicode

Mnemonic: ICU

Specification Number: Unicode 9.0

Version Number: ICU 58

Source: http://site.icu-project.org/download/58

Name: Apache CGI Processing

Mnemonic: Apache CGI

Version Number: Apache 2

Source: http://apache.org

Communications interfaces

An internet connection between the host and client is required for use of XZES40-Transformer. The host and client will be communicating over HTTP or HTTPS through the web interface or CLI interface. The application may be deployed behind a firewall.

Administrators may deploy multiple instance of the application, however additional instances will not be designed to communicate together, so they will each act autonomously.

Memory constraints

XZES40-Transformer will depend heavily on an internal caching system, so an appropriate amount of memory should be dedicated to the application. The specific amount of memory will depend on how much an instance of the application is expected to be used, however a minimum of 4GiB should be dedicated to the machine it is running on. That said, the more the merrier.

Operations

XZES40-Transformer will initially target the Debian Jessie operating system. To install the application a system administrator will acquire a Debian installation package (`xzes40-transformer.deb`), run the installation file, and begin the newly installed service.

The steps will roughly be as follows:

```
Download xzes40-transformer.deb
$ wget http://example.com/xzes40-transformer.deb

Install the package
$ dpkg -i xzes40-transformer.deb

Enable the xzes40-transformer Systemd service
$ systemctl enable xzes40-transformer

Start the Systemd service
$ systemctl start xzes40-transformer
```

Installation on an additional Debian system would require the same procedure as listed above.

Installation on a non-Debian operating system will require a system-specific installation file, which we will create. The non-Debian systems we will include:

- Windows 7+
- MacOS
- FreeBSD
- RedHat Enterprise Linux

Once installed and setup, XZES40-Transformer will require minimal user-interaction by the system administrator. The application will run as a daemon on the host system. If a fatal error occurs the daemon will restart.

While users are not interacting with the application it will idle in the background. During periods of intense use the application will manage it's own resources to avoid breaking.

As the application does not store ephemeral data, there will not be a need for data backup nor data restoration.

Site adaptation requirements

XZES40-Transformer will use Apache to manage web-requests. If users want to setup secure communication over HTTPS they will need to do this manually using Apache.

The application will include a configuration file to specify resource limits, and other relevant information. This file should be tailored to a given user's installation and needs.

In addition to the configuration file, the user may configure the daemon through the daemon manager (`systemd` for instance) to set hard-limits on the application's resource usage.

2.1.2.2 Product functions

XZES40-Transformer will perform one function: XML/XSLT document transformation. Given one XML and one XSLT documents it will return a transformed XML document.

This functionality will be remotely accessible via an HTTP web-page and CLI interface.

2.1.2.3 User characteristics

The **user** of our application is expected to have common web-interface knowledge (e.g., they should know how to navigate a website, upload a file, and download a file).

2.1.2.4 Constraints

XZES40-Transformer will be subject to the following limitations:

- The code must be licensed under Apache 2.0.
- It must handle memory limitations gracefully.
- It must restart if a fatal error occurs.
- It must run on Debian 8.
- It must be accessible over a network.
- It must have an accessible interface.

2.1.2.5 Assumptions and dependencies

XZES40-Transformer will be written to interface with an OS agnostic API for any operating-system level operations (e.g., reading and writing from the cache). The application will be portable to new operating systems by writing an OS-specific interface layer and compiling the binary for the given target platform (e.g., Windows or FreeBSD).

XZES40-Transformer will assume that the relevant libraries and languages listed under Software Interfaces are already installed. The installation package we create will resolve these dependencies if they are not already installed with the correct version.

2.1.2.6 Apportioning of requirements

Development of the application, user interfaces, and installation packages will be carried out over a 19 weeks, split into three development cycles: Alpha, Beta, and Release.

The Gantt chart can be seen in the Appendix, Figure 4.

Alpha

During the Alpha phase of development we will collect benchmark data, create our basic transformation functionality, and begin work on Cache and Parallel computation optimizations. By the end of week three the application will be able to accept two input documents and output a transformed document. After the initial transformation functionality is complete work on optimizing this process will take place by adding caching and parallel processing to the transformation cycle.

Beta

During the beta phrase of development the XZES40 team will begin work on the HTTP API, the web interface, the Debian package, and further optimizations on the transformation process. Work on the web interface is not possible without the CGI interface first being added, however most other development can take place in parallel.

Release

During the Release phrase we will work exclusively on stretch goals including the CLI interface as well as the RedHat, BSD, and Windows packages. While these goals would be nice to achieve, we understand that there will probably be overflow from the Alpha and Beta phases of development, so we hope to complete all required deliverables well before the release deadline.

2.1.3 Specific requirements

2.1.3.1 External interfaces

XSLT40-Transformer will have two user interfaces: a **Web Interface** and a **Commandline Interface (CLI)**.

Web Interface

The website interface for XZES40-Transformer will include of a form with the following fields:

XML File A file upload field for the XML document.

XSLT File A file upload field for the XSLT document.

Output Filename (optional) The filename of the output document. If one is not specified a name will be generated of the following format `document-transform-<date>.xml>`.

The website will make an HTTP `POST` request to the server. This `POST` request will include an XML and XSLT document in it's payload.

The page will redirect the user to a new page where they can download the transformed file.

The Web Interface will require a web-browser (supporting HTTP4.0+).

A prototype of the web interface can be seen in the Appendix, Figure 2.

Commandline Interface (CLI)

The CLI will give users a text-based interface with XZES40-Transformer. The with the following flags:

`-xml-file=` Specifies the input XML document.

`-xslt-file=` Specifies the input XSLT document.

`-server=` Specifies which server to connect to (`e.g., http://servername.ext`)

`-output-file=` (optional) Specifies a file to write out to. Otherwise writes to a file of the following format `document-transform-`

-port= (optional) Specifies which port to connect through if non-standard (`e.g., 8001`)

-help= (optional) Prints out a help menu (describing these flags)

The CLI will take the following arguments and make a `POST` request to the server. The transformed file will be automatically downloaded to the user's desired location, or to the current working directory with the automatic file name.

The CLI requires a UNIX terminal and UNIX shell.

An example of some CLI interactions can be seen in the Appendix, Figure 3.

Both interfaces XZES40-Transformer will require a method of input (keyboard and mouse or touchscreen), an internet connection, and monitor.

As input both interfaces expect one XML 1.0 formatted document and one XSLT 1.0 formatted document. These files can be UTF-8 or ASCII character encoding.

As output both interfaces will send the user an XML 1.0 formatted document of UTF-8 character encoding.

2.1.3.2 Functions

The following functions will be the core functionality of XZES40-Transformer.

- `int transform(string XML_filename, string XSLT_filename, string output_filename)`: This function is called to transform the given XML_FILE with the XSLT_FILE. If output_file is defined the file will be written to that location. If output_file is not defined the new file will be written to STDOUT.
  Returns a status macro (SUCCESS or FAILURE).
- `type_cache* get_cache(input_filename)`: Returns a pointer to the cached file.
- `type_cache check_cache(string input_filename)`: The system will check if the given file is in the cache. Returns TRUE if the file is in the cache and FALSE if the file is not in the cache.
- `type_cache set_cache(string input_filename)`: The new XML file will be saved in the cache. Returns SUCCESS or FAILURe macro if the document was or was not successfully cached.
- `int compile(string input_filename)`:
  The system will compile the given XML/XSLT file into machine code to later be transformed.
- `int delete_cache()`: Removes old documents from the cache which are not being used. Triggered when the cache is filled to a certain capacity.

2.1.3.3 Performance requirements

XZES40-Transformer will perform better than existing Open Source XML transformation software. The application will have a higher rate transformations per minute, normalizing for input file-size. Given a standard set XML + XSLT document pairs, the application will complete the transformations on average faster than it's leading competitors.

The transformations will also be verified for correctness. It is expected that the application will perform document transformation with high correctness.

2.1.3.4 Logical database requirements

XZES40-Transformer application does not require database.

2.1.3.5 Design constraints

For the XML compilation and transformation process we are restricted to using the Xerces-C and Xalan-C libraries. For document encoding and decoding we will use the ICU UTF-8 libary.

Standards Compliance

Input documents must be correctly formatted XML and XSLT documents. Malformed documents will be rejected by the application.

Correctly formatted XML documents follow the W3C outlined XML 1.0 and XSLT 1.0 formats. [1] [2]

Our application will also communicate with users over HTTP/HTTPS, however we are not implementing these standards, just using them to communicate over the internet.

2.1.3.6 Software System Attributes

Reliability

XZES40-Transformer will be reliable if it's cache is up to date, to avoid transforming documents incorrectly.

Availability

Since XZES40-Transformer will be run as a web-service it should be highly-available during business hours. It will be configured to handle heavy workloads and restart if it crashes. Upon restart it's cache will be cleared.

Security

XZES40-Transformer may be used to handle sensitive data, however it is not this team's job to account for that.

If administrators want the application to be secure they may choose to deploy it behind an organization firewall or configure the web-service to use HTTPS instead of HTTP. The application's HTTP traffic will be handled by Apache via a CGI script, this means that any Apache webserver configuration can be used with XZES40-Transformer's web API.

Maintainability

Our application will be deployed as a daemon which will be configured upon installation. This configuration may be modified after installation, but should work "out of the box".

Portability

As an Open Source project XZES40-Transformer will be designed for portability. It will perform all operating-system specific operations via an OS agnostic API. When the application is compiled on a new platform it will compile against the given OS API (e.g., Windows, Linux, MacOS, etc).

As for installation packages, one must be manually created for each platform, however this is not an urgent requirement and can be carried out platform-by-platform after the initial development is completed.

2.1.3.7 Organizing the specific requirements

System Features

Transformation

The core feature of XZES40-Transformer will be the XML + XSLT document transformation.

Stimulus:

As input the application will accept two files, one XML 1.0 formatted document and one XSLT 1.0 formatted document.

Response

As output the application will return one XML 1.0 formatted document. This will be the output of the program.

If a mal-formatted document is given as input the output of the program will be an appropriate error.

Caching

To increase performance the application will cache certain compiled documents.

Stimulus:

As input the in-memory cache will accept either a compiled file for caching or a hash of a compiled file for retrieval.

Component	Owner
Document Transformer	Elijah C. Voigt
Website Interface	Shuai Peng
Web API	Elijah C. Voigt
Document Cache	Shuai Peng
Daemon Process	Elijah C. Voigt
Parallel Document Transformation	Zixun Lu
Benchmarking	Zixun Lu
Debian Software Package	Zixun Lu

Figure 1: Required project components ordered by importance

Response

If the cache is given a hash and "asked" for a cached document it will either respond with the contents of the compiled file or a FALSE response, signifying the document is not in the cache.

If the cache is given a file for storage it will respond with either a SUCCESS of FAILURE response if the document was successfully cached or not appropriately.

Parallel Computation

To further increase performance the application will carry out independent calculations in parallel.

This feature does not have easily defined "stimulus + response" pairs.

The application will need to be designed to avoid race-conditions during parallel computations.

Web API

The application will have a web-accessible API.

Stimulus

As input the API will accept either GET or POST requests. POST requests will contain a payload with the XML + XSLT document pair as listed above for the transformation feature.

Response

As response to a GET request the API will respond with a SUCCESS, saying that the application could be reached.

As response to a POST request the application will respond with either a transformed XML document or an appropriate error.

User Interface

The application will have two user interfaces. These will allow the user to provide the above stimuli to the web API, download the response file, or display the error message from the API.

2.1.3.8 Additional comments

There are no additional comments to be made at this point.

2.1.4 Supporting Information

The following table outlines **required** project components ordered by importance:

Component	Owner
CLI Interface	Elijah C. Voigt
CentOS Linux software package	Elijah C. Voigt
Windows software package	Shuai Peng
FreeBSD software package	Elijah C. Voigt

Figure 2: Stretch goal project components ordered by importance

2.1.4.1 Appendixes

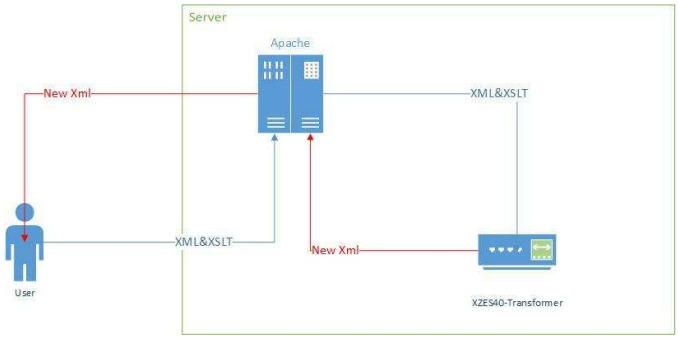


Figure 3: Diagram of dataflow. Documents are sent to the application through the Apache gateway interface. Once processed documents are returned to the user through the gateway interface again.

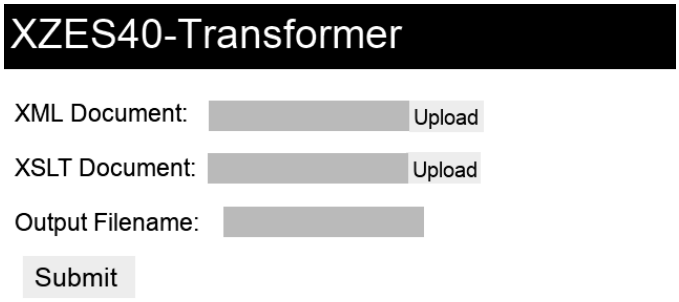


Figure 4: Prototype of the Web Interface. Demonstrates the simplicity of the interface and required form fields.

```
Normal usage:
$ xzes40cli --xml-file='./my-file.xml' --xslt-file='./my-other-file.xslt' --server='http://example.com/
    xzes40-transformer' --output-file='./newfile.xml' --port='8001'
Sending XML and XSLT files to http:/example.com:8001/xzes40-transformer
Transformation complete. Downloading response file to newfile.xml

Sending a bad file:
$ xzes40cli --xml-file='./badfile.jpg' --xslt-file='./badfile.txt' --server='http://example.com/xzes40-
    transformer'
Sending XML and XSLT files to http:/example.com:80/xzes40-transformer
ERROR: Server was unable to transform the requested files.

Using inadequate parameters
$ xzes40cli
Please provide an xml file (--xml-file), xslt file (--xslt-file) and a host (--server).
```

Figure 5: Example use-cases of CLI interface.

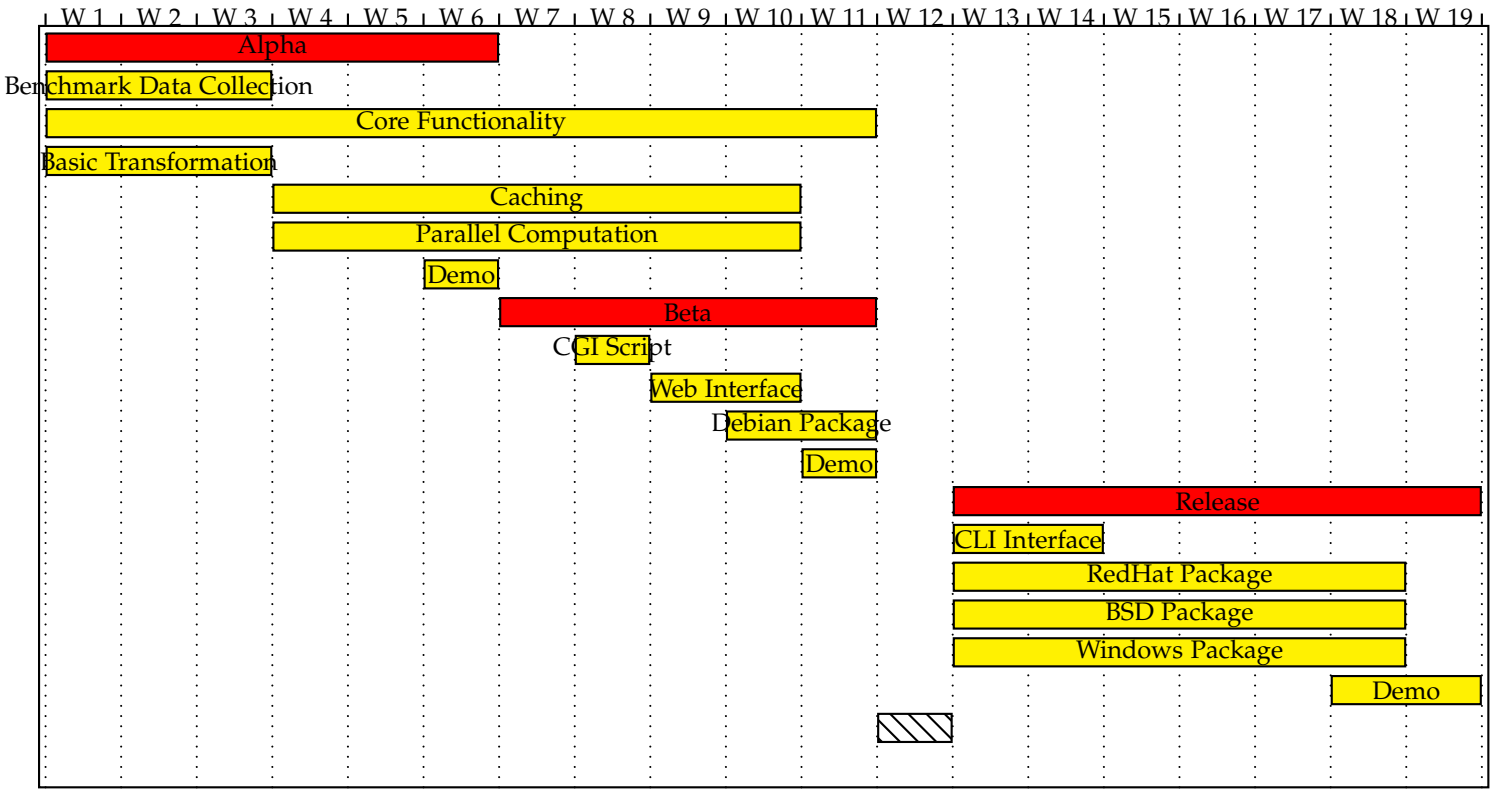


Figure 6: Development Gantt Chart Timeline.

would function, and what it would do differently than existing systems. The just of it being that a few layer of caching would be added to the transformation sever to speed up re-compilation of old documents (a common procedure).

Since last week we have finalized our Problem Statement document, confirmed it with our sponsor, and signed.

We have encountered no problems so far.

This coming week we will meet with our sponsor again to get setup with development environments. We decided on initially targeting Debian Linux as our platform, so we will be given VMs that are more or less the environment our application will be running on in production.

5.3.1.2 2016-10-21

This last week we continued editing our problem statement and met with our client to obtain a virtual machine for future development. The virtual machine was created by the client and included a slew of development tools for the package we will be developing and LaTeX development if we need.

There we no show-stopping problems encountered this week. One difficulty may be with the size of the virtual machine, which is almost 25GB, but once we have a stable development environment configured it shouldn't be an issue.

This coming week we will complete the Client Requirements document. We will also get a revised copy of our Client Requirements doc and Problem Statement doc signed and turned in on time.

5.3.1.3 2016-10-28

Since last week we have begun work on our Client Requirements documentWe turned in a rough draft, but will have to make a lot of edits before turning in a final document next week.

There was a lot of confusion about our what our application is, how it will be implemented, what it will do, and its core purposeTo fix this our group had a meeting. We decided to have daily meetings to improve our document writing workflow. Future documents will be written with all three of us so we're on the same page. We will have daily hour+ long writing meetings.

We will spend next week refactoring the Client Requirements documentOnce we complete a draft we like we will get it signed and submit this document.

5.3.1.4 2016-11-04

This week we completed the client requirements document and had it approved by our clientHe seemed impressed with the scope of the project.

Our client was too busy to have a meeting with us, but not too busy to sign the document, so it worked out.

This coming week we will begin work on the technology reviewNow that we have a solid workflow for writing documents as a team I think this will be straight-forward.

5.3.1.5 2016-11-11

Since last week we have started working on our Technology Assessment document, and to an extent (at least internally) our design documentThis document is not yet complete, but good work has definitely been put into it. Since bridging the knowledge gap I think that we've made good progress on unifying the design, and technologies being using in the project.

Our client has also supplied us with ample development scripts for our projectSince we have not started development these are not yet useful, but will be come winter term.

I don't recall any problems yetI was unfortunately unable to participate in daily meetings because of travel on the holiday weekend, but I don't think this will cause tremendous problems.

This next week (starting Sunday) we will complete the technology assessment document and begin work on the design documentI hope to have a first draft of the design document by this coming Friday, but that may be wishful thinking.

5.3.1.6 2016-11-18

Since last week we finished our tech review and started the design document.

We were sort of down to the wire when we submitted the tech review, but got it in before the midnight on Monday deadline.

We started working on the design document but are a little confused about the exact format of the document.

This coming week, before thanksgiving, we will try to get a rough draft of the design document done.

5.3.1.7 2016-11-25

Since last week we have begun working on our Design document and we created a repository to store code by our Clients request.

We are confused about the exact format of the design document, we will ask the TA about this to get it cleared up.

This coming week we will finish the design document and make an initial draft of the term summary due during finals week.

5.3.1.8 2016-12-02

Since last week we completely re-wrote our Design Document and turned it in, unfortunately unsigned due to time constraints with out clientA large amount of time was spent restructuring the document to more closely fit the IEEE standard we were adhering to.

We also began work on the Progress report document in hopes of getting it done before the start of Finals Week.

The largest problem we encountered this week was having to re-structure our documentWe misunderstood the IEEE structure for this assignment so we spent a lot of time re-writing the document after talking with the TA.

This coming week we will finish the Progress Report and Progress Report Presentation.

Over break we will also make headway on the actual assignment portion of the project, hopefully setting us ahead of schedule if all goes well.

5.3.2 Winter Term

5.3.2.1 2016-01-13

Over break I completed a large chunk of the project's structureThis included:

- Outlining the source code file-structure
- Added skeleton code for key classes, functions, and headers in the code.
- Added initial project documentation as well as code-documentation.
- Added a Makefile which currently compiles the project.
- Added a Vagrant virtual machine for lighter-weight developmentThis includes a setup script which can be used to provision a Continuous Integration system when we start using one.

Unfortunately I was the only member who was able to (or chose) to work on the project so there are a lot of decisions left to be made (changes to the design of the project including whether to use Redis or a library provided by Steven Hathaway for caching, weather to use Boost for cross-os compilation)

This week I will complete the structure of the project so work can begin on transformation and caching.

The team-members who did not engage with the project over the beak will catch up by reading what has been written and documented, getting familiar with important libraries, and setting up their development environments.

5.3.2.2 2016-01-20

Since last week we went started going over what we need to do for the project in finer detail, learning our parts of the project, and learning the required tools we need to complete the projectWe also started to plan in greater detail what and when we need to get tasks done? Tracking our progress better.

Shuai and Lu did not work on the project over the break so they are still catching upHopefully we won't be eventually behind and will eventually get back on schedule.

This coming week we will hopefully complete the basic transformation capabilities of our project, start bench marking competing products, and add continuous integration to pull requests.

5.3.2.3 2016-01-27

Since last week we have added Travis CI support and made progress in completing our basic functionality.

Shuai was unable to complete the basic transformation functionalityI spent the majority of Sunday January 28 (I know this is due Friday but I was late and might as well include this info) making major refactors to Shuai's contributions.

Lu has still not gotten any work done on this project.

This coming week I hope to get the basic transformation complete so that we can complete our alpha release on time.

Once we complete the basic transformation then there is a very short list of features to implement for the beta and final releaseThese are namely:

- Caching parsed documents.
- Daemonizing the application.
- Exposing the application to the web over an HTTP with a CGI script.
- Creating a Website for users to use the application.

Once those features are complete we will have finished the most important parts of our applicationTo finish any of those though we need to complete the core feature of document transformation.

5.3.2.4 2016-02-03

We have made very little progress since last weekAs mentioned previously, I worked for most of a sunday debugging the code that shuai worked on. Since then we have not figured out a solution to the bug.

Lu has not been communicating nearly enough with the group, and we have not solved this bugOnce we fix a show-stopping bug we should be able to start making progress individually on the application.

I have asked Lu to work on the bugIf he is not able to fix it I will do my best with Shuai to fix the bug before the end of next week so we have a working alpha release.

5.3.2.5 2016-02-10

This week our group (Shuai and myself) finally tackled and completed the core of our project.

The biggest problem was completed, but we still have a long way to goOur biggest problem now is finishing all of the work we have ahead in the time we have available.

Strictly speaking though we didn't encounter any new problems, just fixed existing problems.

That said Zixun Lu has still not completed the benchmarking for the projectThis is a problem as we will not be able to assess the success of our project without that information.

This coming week I will try to start and finish the daemon-ization of the program, but with the progress report assignment most actual development is practically suspended.

5.3.2.6 2016-02-17

Since last week we have made minimal progress on the applicationI have begun work on demonizing our application.

Process daemonizaion was not a problem we thought out well enough so a lot of planning has to happen pretty late in the game for that part of our application to work correctlyI think I know a way to execute the idea using best practices, but so far it's not been easy.

Tomorrow (Sunday) we will meet and develop for a full day to hopefully power through some problems we've been havingI am not optimistic that we will complete anything meaningful. I hope to be proven wrong.

5.3.2.7 2016-02-24

Since last week I made progress on and eventually merged the daemon codeI also helped Shuai Peng with the Caching code.

The daemon works fairly wellIt has not been battle tested, nor has it been designed to handle errors or problems well. These have not cause any headaches yet.

This sunday Shuai and I will complete and merge the Caching codeI will begin work on the Web API. We will also demo what we have this Thursday.

5.3.2.8 2016-03-03

Since last week we have not made substantial technical progressAfter merging the Caching code Shuai and I have focused on other coursework. I personally have been researching how to write the CGI script for our application, but not as much time has been spent as I would like.

We also met with our ClientThis was productive and expectations were tempered as we have been missing deadlines. I still believe we will be able to get a working prototype completed by our deadline, however it will not be nearly as polished as I hoped and expected given the complexity of the project.

Zixun Lu has not been able to complete his portion of the project and we are as of yet far past our deadlines for the Benchmarking sprintHe has been reassigned to other parts of the project which do not require C++ coding (benchmarking, website frontend, and the system package) as this seems to be intimidating for him.

This coming week I will finish a prototype, and hopefully merge, the CGI script portion of our project as well as any setup scripts to start an apache server.

This will include:

- The actual Python CGI script(s).
- Setup scripts for installing and enabling an Apache server CGI service.
- Possibly some systemd services.

Actually, may I go on a quick tangent about how useful a systemd service can be for this project? Using systemd we can limit the CPU and RAM usage of an application, thus giving us a 'first line of defense' against runaway cacheNot that I think that is a likelyhood, however it is a nice 'stopgap' just in case. We can set the application, in the init system layer, to restart and wipe the cache whenever it reaches some threshold. A more ideal solution would be to garbage collect the cache every so often so we don't need emergency safeguards, but if it works it works!

5.3.2.9 2016-03-10

Is it already week 9? That term flew by...

Since last week we haven't made a terribly large amount of progressI've gotten the CGI script working so we can (hopefully soon) start communicating with the outside world and running jobs over the 'net.

Originally I tried implementing the CGI script using 'fastcgi' which was touted as being the way to do this, but it didn't workOur client helped out and told us to just use the regular old boring 'cgi'.

I am very close to getting this working, so I'll get that done before we need to demo.

5.3.2.10 2016-03-17

Good news! We finished the demo!

As the clock struck 10:30 I finally got the demo workingYou could upload two files and get the transformed output in your browser. It took a long time, but it was totally worth it.

5.3.3 Spring Term

5.3.3.1 2016-04-07

We encountered many problems on the way, and have many more problems to come, but the big ones related to this were:

1) CGI scripts are non-trivial to setup.
2) Error handling and debugging cgi script was non-trivial.
3) Slightly tweaking (or in some cases entirely re-writing) parts of the code was time consuming.

This coming week I am going to finish finals and take a brief vacation, but starting next term I will spend a substantial amount of time fixing up our pull codebase for Expo and making sure it is ready for the client to take overThis includes documenting the codebase, 'sanding the edges', and any tools Steven needs to own he project when we leave the project.

5.3.3.2 2016-04-14

Since last week we have completed benchmarking of our application (or at least a first draft) and implemented a prototype of parameter passing (https://github.com/XZES40/XZES40-Transformer/pull/38).

The biggest problem we encountered was that we had to hack together an environment to actually do the benchmarkingA complete application would allow us to upload header files for our benchmarked XML documents and pass parameters, but we have yet to complete these features.

This coming week I would like to complete the parameter passing and integrate it into our website interfaceI will add the web interface and form processing / sending this week.

Next week we will try to add dependency file processingThis will be similar to the parameter passing feature, but adding files to the build job instead of key=value parameters.

5.3.3.3 2016-04-21

Since last week I have begun working on passing parameters to the application through the web UI.

The hardest part of this is that I have to format the data in such a way that the input form data is sent as JSON to the end CGI scriptThis can be done with JQuery, but I'm not a frontend person so it's taking longer than the rest of my components did.

This coming week I want to merge parameters and a big docs pushWe should be code-complete by friday the 28th (my birthday!) so I'll make sure we have something presentable by then.

5.3.3.4 2016-04-28

Since last week we have made leaps and bounds.

- We (Shuai and I) fixed a lot of last minute bugs and added some much needed revised documentation.
- I made our website dynamically submit transformation jobs and load the response content.
- I learned JQuery (for the website).

We ran into a few bugs, especially around form processing and displaying results to the user on the frontendThankfully we fixed most of those bugs and by Sunday we should have all of that ironed out for Code Completion.

This coming week we are going to merge the last pull requests we have for Code Completion and begin working on our demo and written documents for the course.

As I mentioned last week, today is my birthday, so I'm going to take a day offWe got a lot done and really brought it all together at the last minute. Now we just need to put a bow on it.

5.3.3.5 2016-05-05

Since last week we completed our code and poster and submitted both of thoseWe did not work on the code for our project, favoring instead to work on other homework to get ahead for the end of the term.

We had a few hiccups with our code, trying to merge a lot of changes together at the last minuteThankfully it passed the smoke tests so we were feature complete.

This coming week I would like to add more tests to the application, however as a team we will probably not do this.

5.3.3.6 2016-05-12

Since last week we finished a progress report (ahead of schedule!) and started prepping for Expo.

No problems have been encountered this week.

Up next: Expo!

5.3.3.7 2016-05-19

Since last week we just made sure everything was ready for expo.

We didn't encounter any problems at expoThank god.

This coming week I will get the three short writings out of the way if possible.

5.3.3.8 2016-05-26

My biggest regret in the course of this project was over-engineering solutions which did not need to be madeThe biggest example of this was when I spent all of winter break outlining the code we needed for our project, creating skeleton code for the majority of our C++ work, and in the end a large swath of that was removed to get the project done.

This was a blessing and a curseI of course learned a good lesson, and had a good understanding of our project going into winter term, but it was a blow to the ego when a bunch of my freetime was wasted.

The biggest skill I've learned over the course of this project was time management, and allocating work fairlyI tend to feel an urge to over-work myself when others are underperforming. This is unfair to myself and the people under-performing. So my new skill is probably something like "tend to your own garden".

The biggest skills I can use in the future on the technical side are some nifty javascript skills and some nice apache system admin skills.

On the non-technical side I've gotten very good at managing a small team and I've learned a lot of lessons the hard wayAs mentioned above, I have a new understanding of how to allocate jobs and stick to that allocation. Don't over work yourself just to get the job done, hold those who agreed to a job accountable. Otherwise you'll pull your hair out trying to get somebody to do a job you're already doing for them.

I am glad to have contributed to the Apache Software FoundationI am also glad that this project is over.

I learned from my teammates many lessons about management, and how as a manager you should meter your expectations a lotDon't expect anything from your teammates, make everything explicit, and make sure you stick to your job.

I believe the client is satisfiedAlthough we did not know this going in, this was more or less a prototype project and to that end we definitely finished the prototype.

I will volunteer myself as a contributor for this project going forwardI want to support the Apache Software Foundation and this is a pretty simple way to do that.

At Expo we were surrounded by really exciting project so we didn't get much attention at ExpoThat said I did meet someone from Nvidia who was very excited about our project and told me to send her an email when I was looking for a job, so that was a success!

6 PROJECT DOCUMENTATION

The following section includes the project's documentation.

The document has not been reformatted to best preserve the original content.

If you would like to view the documentation in it's 'pretty print' formatting, visit https://github.com/XZES40/XZES40-Transform

The headings of each section here reflect the original file's location in the directory structure of the source code. The documentation co-mingles with the source code.

Each soruce file also includes documentation blocks, which are not included here.

6.1 README.md

```
# XZES40-Transformer

*This repository contains the Code, Code Documentation, Tests, Examples, and Build/Dev infrastructure
    for the Oregon State University + Apache Software Foundation Computer Science 2017 Capstone project
    .*
For more information about this project, see the project's [LaTeX document repository][capstone-repo]

## Build status

- Linux: [![Build Status](https://travis-ci.org/XZES40/XZES40-Transformer.svg?branch=master)](https://
    travis-ci.org/XZES40/XZES40-Transformer)

## About the project

This project's purpose is to create an XML/XSLT transformation server which is

- *Fast*
- *Cross-platform*
- *Packaged for your OS*
- *Open Source*

It achieves these by:

| Feature                                                  | Status              |
| -------------------------------------------------------- | ------------------- |
| Processing documents in parallel.                        | **IMPLEMENTED**     |
| Using an in-memory cache to store processed documents.   | **IMPLEMENTED**     |
| Using [FPM][fpm] to package the application.             | **NOT IMPLEMENTED** |
```

| Using the [Apache Webserver][apache] to handle requests. | **IMPLEMENTED** |
| And of course developing in the Open! | **IMPLEMENTED** |

For a complete guide on the *design* and *purpose* of this project refer to our [LaTeX Documents Repository][capstone-repo].

A what transformation server?

XML is a standard format for storing data, like JSON or YAML.
XSLT, or XML Style Sheets, are used for performing transformations on data in XML sheets.
For example:

Problem

- You have an XML spreadsheet with information about all cars sold at a used car lot.
- You want to know how many *red* cars were sold between 2000 and 2015.

Solution

Create an XML stylesheet (XSLT document) which specifies "if the <color> is red, and the <date sold> is between 2000 and 2015, include it in the output" (in psuedo XSLT).

The XZES40-Transformer application is used to perform these transformations over the internet so one can process data without installing heavy XML transformation applications locally.

How do I use this?

The easiest way to use the application is to run a small Virtual Machine (VM).
The application is designed to be run on Apache, but it is not encouraged that you install, run, or develop the application on your personal machine.

The tool the development team uses to run the application in a VM is Vagrant.
Vagrant is a Command Line Interafce (CLI) interface for pulling, provisioning, and running VM with arbitrary back-ends.
This means you can use the same commands to run a VirtualBox, VMWare, or Docker machine.
Vagrant runs on Linux, BSD, Windows, and MacOS, so it is highly accessible for development.

Once you have Vagrant installed and setup on your system you should be able to run ‘vagrant up‘.
Once the machine is setup it will give you a URL to visit in your local web browser.
Visit that page and you’ll be using the application!

For information on running the application with Vagrant, see the [‘vagrant‘ directory’s README file][vagrant-code].

Where is everything?

There are three components to the XZES40-Transformer project.

Component	Description

`transformer`	Core of the project. Actually performs XML transformations.
`cgi-glue`	Bridge between the transformer and web UI.
`frontend`	Web interface for the application. Accepts XML transformation requests, displays results to user.

Each of these components can be found under the `xzes` directory.
Each component has an explanatory README which explains what it in greater detail is and how it works.

How can I test?

In the directory `xzes/transformer/test/simple_test_file` we have provided a simple test script that you can run on your local system which connects to the server running via Vagrant..
Just run `./simple_test.sh` and then you should see HTTP 200 OK responses and transformed XML documents ..

Contributing

If you have interest in contributing to this project, **great**!

Please create an [issue][issues-url] and see if the developers respond.
If you have a fix for your issue, make a pull request by forking the project, doing your work on your fork, and clicking the `Create a Pull Request` button once you push your changes to that fork.

We have a [Vagrant Box][vagrant-code] which is probably your best way to test code-changes.
See the `vagrant` directory for details about that work-flow.

A note about names

There might be a bit of confusion about what is named what and why:

- The project is called XZES40-Transformer because the Capstone group name is XZES40 (a combination of our names, our group number, and the letter X).
- The application is just called `xzes` because programs shouldn't have numbers in their name.
- The transformer daemon is called `xzesd`, and it's systemd service is called `xzesd.service.`
- The CGI script used by apache is called `xzes.py`.
- The component that runs between Apaache and the daemon is `cgi-glue` because it's just enough code to get `xzesd` exposed to the internet.
- The frontend is called `frontend` because that is an intuitive name.

Most things, unless they are the group or project, are just `xzes`.
The project as a whole is `XZES40-transformer`.
There are no components called simply `XZES40`, although during development some components *were*.

License

This software is licensed under `Apache 2.0`. Read the `LICENSE` file for more information.

[capstone-repo]: /XZES40/cs-capstone-project
[issues-url]: /XZES40/XZES40-Transformer/issues

```
[fpm]: /jordansissel/fpm
[apache]: https://httpd.apache.org/
[vagrant-code]: vagrant/
```

6.2 scripts/README.md

```
# Scripts

This directory includes scripts used for development that didn’t really fit with a specific part of a
    project **or** had multiple purposes.

The scripts are exclusively written in Bash and focus on operatics.

The ‘ci‘ directory includes scripts intended for Continuous Integration.
If you are interested in how a component is used, see if it has a script associated with that.

The CI system we intended to use heavily was TravisCI.
Most of the ‘ci‘ scripts do not get used except for the one that builds the software (‘transformer.sh‘).
That said most of the scripts are [mostly] accurate, just not robust enough for an automation build
    system.
```

6.3 xzes/README.md

```
# XZES Projects

These components include:

- ‘transformer‘
- ‘cgi-glue‘
- ‘frontend‘

Each project includes it’s own README which self-describes the component, what it does, and how it works
    .

## xzes.conf

This file is the Apache config for the XZES40-Transformer application.
Feel free to edit this to change the port the application runs on, add security certificates, pretty
    much anything you would do in an Apache HTTPd config.

## xzesd.service

This is the systemd service file for xzesd.
It automatically restarts the application, manages forking the process into the background, and
    aggregates xzesd stdout to the Linux logging service.
```

6.4 xzes/cgi-glue/README.md

```
# cgi-glue

This component of the project is a python CGI script which accepts HTTP requests via Apache, processes
    them into a format that the xzesd daemon can read, saves uploaded files, waits for the
    transformation to complete, and responds with the output.mdt from the daemon.
```

‘xzes.py‘ is the meat of it. Read the comments or more information.

‘simple_test.sh‘ is used to test to see if the application is running correctly. Pass it an xml and xsl file (in that order) and see if it responds correctly.

‘benchmarks.sh‘ was used to benchmark the speed of the application for grading purposes.

This is essentially treated by the frontend as a *very* simple.mdt-based web API.
A form with two files (‘xml‘, ‘xsl‘) and parameters (‘{key: val, key2,val2}‘) are accepted, processed, serialized, and sent to the daemon over a local network connection (‘localhost:40404‘).

6.5 xzes/transformer/README.md

```
# Transformer

This is the C++ application which transforms and XML and XML style-sheet documents into an output XML document.
This transformation is requested over a local network connection on ‘localhost:40404‘ via an Apache CGI script found in ‘cgi-script‘ (located in the parent directory).

## Usage

### Building

Once you’re in a VM with the correct packages installed (check the ‘vagrant‘ directory for a reference as to what those packages are), you can build the project.

‘‘‘
(transformer/)$ make
[... lots of verbose output ...]
‘‘‘

### CLI Usage

Now that you’ve built the project you can *use* it.
How fancy.

‘‘‘
user@host:transformer/$ ./build/xzesd &    # This runs the daemon in a background process
[2] 14876
user@host:transformer/$ ./build/main.py ‘pwd‘/examples/simple.xml ‘pwd‘/examples/simple.xsl /tmp/example.xml
user@host:transformer/$ cat /tmp/example.xml
<?xml version="1.0" encoding="UTF-8"?><out>Hello</out>
‘‘‘

## Development

To develop the application you will need
```

- A Debian Linux host (probably a virtual machine).
- The C++ dependencies listed in the ‘vagrant‘ directory in the ‘setup.sh‘ script.
- The ability to host a website locally.

Using Vagrant

Vagrant us a convenient tool for using a portable Virtual Machine.
Our development environment depends heavily on Vagrant.
It automatically runs a setup script which installs the dependencies, builds the binary, installs apache , sets up the daemon service, and starts the webserver to be accessed on your local virtual machine.

```
(XZES40-Transformer/)$ cd vagrant
(XZES40-Transformer/vagrant)$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'debian/jessie64' is up to date..
[...]
==> default: Vanilla Debian box. See https://atlas.hashicorp.com/debian/ for help and bug reports
(transformer/vagrant)$ vagrant ssh
[...[
vagrant@jessie:~$ cd xzes/xzes/transformer
```

See the file ‘vagant/README.md‘ for more information about the Vagrant development environment.

Even if you are not using Vagrant, the README and setup script should help you start contributing if you want to do that.

Structure

This project is organized according to a [’better than nothing’ standardized structure][cpp-project].
This looks as follows:

```
project/
|-- build
|   ‘-- generated binaries (*.o, main)
|-- doc
|   ‘-- documentation (*.txt, *.md, *.rst)
|-- examples
|   ‘-- code examples not necessary for building but helpful
|-- include
|   ‘-- headers (*.h, *.hpp)
|-- lib
|   ‘-- external dependencies (*.h, *.hpp, *.c, *.cpp)
|-- Makefile
|-- src
|   ‘-- code (*.c, *.cpp)
‘-- test
    ‘-- unit tests (*.c, *.cpp)
```

```
```

In addition to this structure, each source code file has

- A `.hpp` file in the `include` directory which outlines the classes, methods, and functions for that component.
- A `.cpp` file in the `src` directory which implements the classes, methods, and functions in that component.
- A `.md` file in the `doc` directory whichcovers the design, co.mdt, and oddities of that component.

All components in the project are compiled into `.o` files in the `build/` directory.
The entrypoint binary is compiled to `build/main`.

Documentation

Documentation about the code is written in the `doc` directory but the source files are also self-documented.
For reference the `doc` directory tends to talk about the high-level concepts (what a compent is supposed to achieve and how it fits in) while the code documentation is targeted at helping developers find bugs and implement features.

[cpp-project]: http://stackoverflow.com/questions/10782554/how-to-organize-a-c-project#10782577

6.6 xzes/transformer/test/simple_test_file/README.md.md

```
# How to use this test?

## In the command-line

To auto-run the tests run the following commandin this directory:

```
./simple_test.sh
```

This can be run in the virtual machine, or on your local system.
It depends on the application being avaliable on `192.168.33.22:8000`.

In the browser

Once the application is up and running using the Vagrant setup script, go to the address `192.168.33.22:8000`.
For each pair of files in this directory (`sample1.xml`+`sample1.xsl`, etc), upload the pair, click 'Transform Files' and when the button becomes available click 'View Raw XML'.
Fore each sample you can also add parameters.
The accepted parameter keys are `param1`, `param2`, etc. up to `param5`.
The key must be one of those but the value may be anything you want.

`sample1` does not use parameters, but all of them can accept parameters.

Click the 'View Raw XML' button to refresh the document preview.
```

Click 'Download Output' to download a copy of the transformed file.

How do I know it works?

In the CLI you should see 200 OK response output form `curl`, and transformed XML documents.

For the web-browser testing you should see the output of the transformation in the preview.

If you upload a *non* XML/Style Sheet you should see an error message.

6.7 xzes/transformer/doc/lib.md

Lib

The library (`lib`) is used for miscelaneous functions used across the project.
This is kept as small as possible since any large objects and groups of similar methods will be broken out into their own components (like the Document class).

6.8 xzes/transformer/doc/README.md

Documentation

These documents are designed for developers to get an understanding of what the application is, what each file/component is doing, and in some cases why it was designed that way.

Read at your own peril.

6.9 xzes/transformer/doc/daemon.md

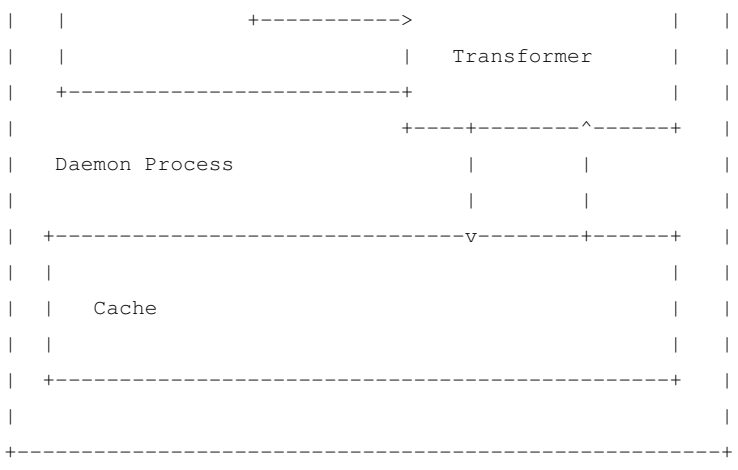
Application Daemon

NOTE In production the daemon is called `xzesd` and is stored on disk at `/usr/local/bin/xzesd`.

In order to preserve state in the in-memory cache, XZES has a long-running daemon.
This could have been implemented a number of ways: Redis, a database, carrier pigeons.
We chose an in memory cache as the volatility is not hyper detrimental to the goals of the application (convenience, then speed).

Here is roughly how that daemon works and how it recieves jobs.

```
 +----------------------+
 |                      |
 |   Main Executable    |
 |                      |
 +--^-----------------+--+
    | network socket |
+---------------------------------------------------------+
|   |                |                                    |
|   |                |                                    |
|   |                |         +--------------------+     |
```



Usage

The daemon is included as a systemd service in the application's codebase (`xzesd.service`).
This service is setup and enabled in the `vagrant` setup script.

The daemon can be started manually with the following commands:

```
$ make              # Build the daemon
[... successful build ...]
$ ./build/xzesd &  # spawns a daemon process in the background
[1] 14877
```

... and tested like so:

```
$ ./build/main.py `pwd`/examples/simple.xml `pwd`/examples/simple.xsl # check that the dameon is running
<?xml version="1.0" encoding="UTF-8"?><out>Hello</out>
```

This assumes that the application is not running with the web-app component. To test that see the `cgi-bin` directory's README.

The Daemon process startup

1. When the `daemon` binary is executed it sets up a long running process and goes into a state of cybernation.
2. This process starts by:
  - Opening and listening on a `localhost:8000`
  - Allocating a block of memory for the cache.
  - Spawning a set of threads on which transformations may occur.

Spawning a Transformation

1. When the `main.py` script recieves a request it is passed two files, input xml and xslt file-paths, as well as some parameters.
   This is either done via the web API (tested with `simple-test.sh` or a CLI directly talking to the daemon... probably the first of those two.
2. The `main.py` script opens a local socket connection with the daemon process.
3. The request is passed in a tuple with the following format: `'"request-id#,input-filename.xml,input-filename.xsl,output-filename.xml,{param1:value2},{param2:value2}..."'`.
4. The daemon process the arguments checking for errors.
5. The daemon process spawns a thread and calls the `transform_documents` method in that thread.
6. Once the thread completes the transofrmation it returns a status code which is propagated up to the `main.py` script and returns to the calling process.

6.10 xzes/transformer/doc/main.md

```
# Main

**NOTE** This file has been deprecated in favor of `daemon.cpp`.
Please read the daemon document instead of this one.

Main is the entrypoint for the entire application.
It essentially just parses command-line arguments and either passes those to the Transformer or yells at the user for using the application wrong.
```

6.11 xzes/transformer/doc/cache.md

```
# Cache

The caching component of XZES40-Transformer is designed to create an easily usable Caching API for the Transformer and Document components.

Cache uses the XercessC keyList to manage the stored state.
The data structure of the keyList is link list.
Although link list is slower than hash map, it was used becuas it was provided by the XercessC Library.

## API

The current interface looks like this:

```cpp
#include <lib.hpp>
#include <keylist.hpp>
#include <cstdlib>

        class Cache {
        private:
                KeyListEntry *theList;

        public:
                Cache();
            bool search(id_t);
            doc_t* get( id_t );
            int set( id_t,doc_t*,uri_t );
```
```

```
            int print_name();
            int print_id();

    };
```

To clarify:

- `Cache()` which is the default constructor.
- `bool search(id_t)` will search the content inside of cache, return true if there is file exist.
- `doc_t* (id_t)` return the content from the cache.
- `int set(id_t,doc_t*,uri_t)` Set the content to cache. id_t is id, uri_t is name, doc_t is content.
- `int print_name()` is a debug tools for printing all content from the cache.
- `int print_id()` is a debug tools for printing all id from the cache.

Usage

The only component of the Cache which is used is for storing Documents.
When a document is created it is parsed and cached automatically, given an input file.

The data flow is as follows:

1. The `Document` constructor sets the URI and hash for the given document.
2. The `Document` constructor checks if the document is already in the cache for the hash key.
    - if true, run cache.get and document.set_content for `Document` class.
    - if false, run document.complie(), and cache_set() for `Cache` storing.
5. Done

6.12 xzes/transformer/doc/main.py.md

main.py

Main.py is a test python script for ensuring the application daemon communicates correctly over the
    local netowork.

This assumes that the daemon us running on `localhost:40404`.

6.13 xzes/transformer/doc/keylist.md

KeyList

The keylist file contains functionality used by the cache to store state, specifically parsed documents,
     in-memory.

6.14 xzes/transformer/doc/transform.md

Transform

The `transform_documents` method wraps the use of the Transformer class.

Transformer class

```
The Transformer's job is to perform the actual application's purpose: transforming an XML document with
    an XSLT document and writing that to disk or printing it to the screen.

### Members

The three main members of the Transformer are:

- `Document xml`
- `Document xsl`
- `Document out`

These are created with the Document class initializer outlined in the Document ... err... doc.

### Methods

When the Documents are created they contain three components themselves `id`, `uri`, and `contents`.
The `contents` is the part we care about.

Using the Xerces and Xalan libraries we perform a DOM transformation with `xml.contents` as the XML
    component of the transformation, `xsl.contents` as the XSLT component and `out.contents` as the
    output.

### Caching output documents

Once the `xml` and `xsl` documents are initialized the cache gets requested for a document with the id `
    xml.contents + xsl.contents`.
This is a unique identifier at which all output documents are stored.
If the output document already exists then we bypass the entire transformation process and return this
    cached output document.

Once saved the output document is saved to the cache by it's own `id` **and** by the `id` `xml.contents
    + xsl.contents`.

Only output documents have a double-length id so there should be no collisions.
```

6.15 xzes/transformer/doc/parse.md

```
# Parse

The file `parse.cpp` contains functionality for parsing input recieved from the user via the python CGI
    script into a struct which is easy to operate on.

That input is in the following format:

`"job_id,xml_file_path,xsl_file_path,output_file_path,key1,val1,key2,val2,key3,..."`

This means that the daemon recieves what essentially amounts to a stream which is then parsed.

The `key1,val1,key2,val2` are tuples of key=val pairs passed to the transformer for custom
    tranfsormations.
```

6.16 xzes/transformer/doc/parallel.md

```
# Parallel Computation

Parallel computation is how XZES40-Transformer hold the faster transformer performance.

Parallel computation will hold multiply threats for XZES40-Transformer application.
We will using Unix-socket lock function to handle the time conflict when we have mutiply file at the
    same time.

Here is how Parallel computation run.

1. First we have CGI script receive request from the front side.
2. CGI script may receive mutiply request at the same time, but they only can write at one time.
3. Daemon receive request and have mutiply thread to transfer the document, but only can return
   a transfered file at one time for one request.

---


## Here is a map how parallel looks like:


                        +-------------------+
                        |  daemon running   |
    +--------+          |    multiply       |
    |        |  request1|     thread        |
    |  CGI   |  request2|                   |
    | script +---------->                   |
    |        |  request3|                   |
    |        |          |                   |
    |        |          |                   |
    +---^----+          |                   |
        |               +---------+---------+
        |                         |
        |                         |
        |                         |
        |                         |
        |                         |
        +-------------------------+
                  return


---


## Here is the time slot represent how parallel looks like:

### Daemon:

Request 1 received ->> create thread ->> do transformer ->> lock socket ->> return file ->> unlock
```

```
    socket

Request 2 received ->> create thread ->> do transformer ->>   __wait,locked socket__    ->> lock socket
    ->> return file ->> unlock socket

### CGI script:

Request 1 send ->> lock socket ->> write to the daemon ->> unlock socket ->>receive return file

Request 2 send ->> __wait,locked socket__ ->> lock socket after detect unlock ->>  write to daemon ->>
    unlock socket ->> receive return file
```

6.17 xzes/transformer/doc/document.md

```
# Document

Document is a C++ class with three main components, as well as methods which operate on those components
    .

Documents are stored and retrieved from the Cache and are operated on by the Transformer.
While we could use the Xerces and Xalan object natively, this wrapper allows for a seperation of
    concerns.
The Cache can worry about storing an arbitrary `class`, the Transformer worries about transforming the `
    Document.contents`, and the each component can be developed (mostly) seperate.

## Members

Here is the rough outline of the variables in a Document:

- `uri`: The file path or remote address that the contents of a Document are stored.
- `id`: Is a **unique** ID, probably created by hashing some safe metadata about the source file.
- `contents`: Are the **parsed** XML or XSLT Document.

The raw contents of a file are not stored, only the parsed version.
This may change going forward.

## Methods

The Document class has a few methods:

- `Document()` is the initializer.
- `set/get_uri()` set and get the URI respectively.
- `set/get_id()` set and get the ID respectively.
- `set/get_contents()` set and get the contents respectively.
- `to_str()` processes the `contents` member and outputs and XML/XSLT document (useful for `output`
    documents).
- `to_file()` processes the `contents` member and outputs the XML/XSLT document to a file (useful for `
    output` documents).

## Dataflow
```

```
The Document class is the component of the project which interfaces with the Cache the most.
Here is why:

1. For a Document to be created, it must be passed a URI (file path or HTTP address).
2. The URI is parsed and basd on that an ID is created by either hashing the file contents or hashing
    metadata about the remote URI.
3. The construcor then checks to see if the Document ID is a key in the Cache.
    - If the Document is *not* in the Cache it is parsed and stored in the Cache.
    - If the Document *is* in the Cache the value of `contents` is filled with the Cached value.
4. Once the Document is initialized it is returned (probably to the Transformer).

The Document component was broken out into it's own file as it pertained to both Caching and
    Transformation and seemed complicated enough not to need it's own source file.
```

6.18 xzes/frontend/README.md

```
# xzes frontend

This is the frontend web interface for xzes40.
It is served via an Apache httpd server.
The Apache conf can be found in this component's parent directory.

## Dependencies

The frontend depends on [FileSaver.js][filesaver.js](git commit 4db4a78) and [Blob.js][blob.js](git
    commit 079824b) , both licensed under the MIT license, for downloading transformed files.

To upgrade these dependencies:

1. Visit their GitHub pages
2. download `FileSaver.js` and `Blob.js` respectively.
3. replace the current copies of those files in this directory with the updated versions.

The frontend also depends on JQuery, which is resovled by a Content Delivery Network (CDN).
If you need to use a local copy of JQuery, please consult the JQuery documentation for this task.
It roughly looks like:

1. Downlaod JQuery 3 (latest, full, minified).
2. Place it in this directory.
3. Replace the JQuery `script` tag with a refernce to the downloaded copy.

[blob.js]: https://github.com/eligrey/Blob.js
[filesaver.js]: https://github.com/eligrey/FileSaver.js/
```

6.19 vagrant/README.md

```
# Vagrant development environment

This development environment requires the installation of [Vagrant][vagrant]
(and probably [VirtualBox][vbox]).

## About the dev environment
```

The Vagrant box includes the following:

- Creates a Debian Jessie virtual machine.
- Mounts the repo to the `/xzes` directory (which is symlinked to `/home/vagrant/xzes`).
- Installs the `xerces` and `xalan` libraries.
- Sets up the Apache `xzes.py` script and VirtualHost.
- Enables and starts the `xzesd` systemd service.

This is designed to a minimal virtualbox only consisting of what is
necessary for the XZES application to run.
Nothing more, nothing less.

Usage

1. Install Vagrant and a virtual machine runner (VirtualBox for instance).
2. In a shell navigate to this directory.
3. Run `vagrant up` to start the virtual machine.
   If everything works correctly this command will output debug information and exit successfully.
   READ THE OUTPUT OF THIS COMMAND
4. Run `vagrant ssh` to login to the virtual machine.
5. On your host OS, run `vagrant suspend` to put the machine in hibernation.
6. On your host OS, run `vagrant destroy` to destroy the machine.
   This will not delete any data on your host machine, only data stored exclusively on the machine.

NOTE: You may need to install the vagrant plugin `vagrant-vbguest` if you get an error about
    mounting directories:

```
$ vagrant plugin install vagrant-vbguest
```

Once the vagrant machine is up, i.e., just running `vagrant up` successfully, you should be able to go
    to `http://192.168.33.22:8000` in your web-browser and use an instance of the application for
    development.

Debugging

If you are using this to develop the application, you should use `vagrant ssh` to login to the machine
    and run the following commands to figure out what is/is not working:

```
Is xzesd running?
$ sudo systemctl status xzesd
Is apache running?
$ sudo systemctl status apache2
What is the output of xzesd? (logs)
$ sudo journalctl -xe
Print all apache logs as they are generated.
Go to your browser and try using the website.
```

```
Only works if Apache is running (above)
$ sudo tail -f /var/log/apache2/*.log
```

About the setup script

The setup script in this directory is used to directly provision a stock Debian box.
If you intend to deploy this application, start with that script and work from there.
If you are on a Debian system and have checked out this code repository, you should be able to use the `setup.sh` script with limited changes to install the application from source.

Variables

The following scripts should be edited in the `setup.sh` script to correspond with the appropriate change in behavior.

```bash
# Ideally used in automation...
${TRAVIS_BUILD_DIR:="/xzes"}
# Specifies the directory where the repository is checked out
${XZES_SRC_DIR:=$TRAVIS_BUILD_DIR}
# Specifies where to install the application to
XZES_BIN="/usr/local/bin/"
# Specifies where to push cgi scripts
XZES_WWW_BIN="/var/www/cgi-bin"
# Where to put html/css/js files
XZES_WWW="/var/www/xzes"
# If set to true, copies files instead of sym-linking them
XZES_INSTALL=false
# set this to false if you do not want a symlink to the code in your homedirectory
# otherwise set it to your user name
DEV_USER="vagrant"
```

The most important would be `XZES_INSTALL` which is a boolean to either install or not install the application, and `XZES_SRC_DIR` which specifices where the checkout of XZES40-Transformer is.

[vbox]: https://www.virtualbox.org/wiki/Downloads
[vagrant]: https://www.vagrantup.com/docs/getting-started/

7 Learning: Technical

8 Learning: Personal

8.1 Zixun Lu

8.2 Shuai Peng

8.3 Elijah C. Voigt

9 Expo poster

A color copy of the poster presented at expo can be found on the following page.

Why we made XZES40 Transformer.

Useful, Challenging, and Open Source

This project was chosen because it provided students the opportunity to create a real-world impact, solve an interesting problem, and contribute to the Free, Libre, and Open Source Community (FLOSS).

The software may be used by individual, organization, and enterprise users to XML perform document transformations. This is a necessary and time consuming task which can now be done faster and *in the cloud*. It proved technically challenging in that it required an understanding of C programming, in-memory caching, parallel computation, application networking, and extensive library usage. The final product was shared with the FLOSS community, giving both our client, the Apache Software Foundation, and any the rest of the world a tool which is useful, usable, and free (*in more than one way*).

Applications and uses

XML is a machine readable data markup language used to store any well-formed topic. XML Stylesheets are used to express transformations of a given XML document. These are used together to store large amounts of data, and "ask" the data a question like "How many employees in this XML file took less than three days of vacation, and make more than $50,000 per year?"

XML doument transformation is used for largely utilitarian purposes in a wide veriety of fields. Police departments, corporate offices, academic institutions, and countless other organizations need and use XML documents and transformations.

The XZES40 application provides an easy to use and access platform for transforming such documents over the web.

CS CAPSTONE PROJECT: XZES40 TRANSFORMER

High Performance XML/XSLT Transformation Server

Optimized for Speed and Convenience

The XZES40 Transformer takes two files as input, an XML document and XSLT stylesheet, and a variable number of custom build parameters or dependency files, and generates a new XML document. While this type of application is not new the concept has been improved with key optimizations and made accessable over the internet via a Web interface.

The strengths of this system are are that it does not need to be installed locally on a user's system, but is still able to perform it's tasks in a timely manner.

The application uses the Apache Xalan and Apache Xerces C++ libraries to perform XML document transformations and a Python CGI script running through an Apache HTTPD server to respond to incomming requests. The transformer daemon is optimized to perform fast transformations by caching previously encountered XML documents and performing document transformations in parallel when possible. The Python CGI script communicates with the Daemon over a local network socket; multiple requests are sent to the daemon and each one is serviced in a POSIX thread. Once the transformation is complete it is sent back to the user.

The web service runs on a remote server atop existing techologies so it can be run, managed, and modified by anybody with Linux and Apache HTTPD experience.

Conclusions/Outcomes

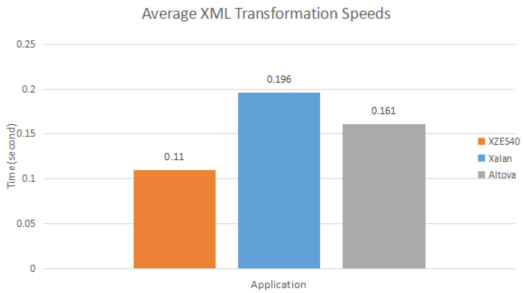
The developed application can generate a new XML file given an XML input, Stylesheet input, and custom build parameters via a Web interface, all without error.

The minimium requirements were completed, like transforming XML documents, as well as many of the optimizations and benefits we set out to complete. The most directl benefit of these being a web interface to access the transformer over the internet. In addition to this we successfully completed a simple in-memory caching system and parallel transformation of the documents. In the end we gave users an application fast enough to be competitive with enterprise software, while still being convenient to usable via the browser.

Results

The following is a table which represents the benchmark results of transforming 19 files 3 times each on the Altova enterprise XML transformer, Xalan Open Soruce transformer, and our XZES40 web Transformer. The files tested came from the Xalan project source code repository.

filename	XZES40	Xalan	Altova
builddocs.xml	0.109s	0.204s	0.187s
buildlibs.xml	0.109s	0.214s	0.193s
charter.xml	0.110s	0.221s	0.190s
ommandline.xml	0.110s	0.201s	0.178s
download.xml	0.113s	0.205s	0.169s
extensions.xml	0.108s	0.196s	0.177s
faq.xml	0.108s	0.188s	0.165s
getstarted.xml	0.108s	0.193s	0.162s
index.xml	0.107s	0.189s	0.151s
Install_save.xml	0.108s	0.212s	0.162s
install.xml	0.110s	0.208s	0.147s
overview.xml	0.110s	0.179s	0.155s
programming.xml	0.108s	0.169s	0.143s
samples.xml	0.106s	0.177s	0.135s
secureweb.xml	0.105s	0.199s	0.139s
test-faqs.xml	0.109s	0.185s	0.136s
usagepatterns.xml	0.121s	0.205s	0.147s
average	0.110s	0.196s	0.161s



The team behind XZES40 Transformer.

Sponsor

Steven Hathaway
The Apache Software Foundation
Email: shathaway@apache.org

Team Members

Elijah C. Voigt
Email: voigte@oregonstate.edu

Shuai Peng
Email: pengs@oregonstate.edu

Zixun Lu
Email: luzi@oregonstate.edu

About the Team

Elijah, Shuai, and Zixun are are fourth year Computer Science students at Oregon State University. Elijah is an Applied CS student focusing on mathematics and computer security, hoping for a career in systems engineering and games development. Shuai is a Systems CS student and is passionate about understanding and designing complex systems, looking for a career in software engineering. Zixun is a double major in CS and Business, intending to be an entrepreneur.

The team also thanks and appreciates Steven Hathaway who gave much of his time and patience to the project.

10 APPENDIX 1

11 APPENDIX 2