High Performance XML/XSLT Transformation Server

Design document

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

CS 461 | CS Senior Capstone | Group #40

Fall 2016

November 30, 2016

Abstract

A design document for the XML/XSLT document transformer “XZES40-Transformer”, the UI, web-api, and installation packages.

CONTENTS

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to outline the entirety of the design of the XZES40 application for the purposes of reference during development and for communication with the project sponsor(s).

1.2 Scope

The scope of this document is to outline in broad and specific terms the means by which the XZES40-Transformer application will be developed.

1.3 Development Time-line

The following Gantt chart outlines the projected time-line for development. This starts at the first week of January (winter-term week 1), and goes through project completion in June (end of spring term).

1.4 Summary

XZES40-Transformer is an application for transforming one XML with one XSLT document for the purposes of data transformation, similar to data transformation in a spreadsheet program. The uses for this rage widely from business to scientific to personal. Most importantly, any XML/XSLT document transformer needs to be able to handle a high volume of requests by a wide variety of clients. To address these needs the XZES40 team is building an Open Source XML/XSLT document transformer with key optimization built in to improve the document transformation process; helping businesses, institutions, and individuals get more done in a day. [3] [7]

1.5 Issuing organization

This document has been issued by Oregon State University and the Apache Software Foundation through the OSU 2016-2017 CS Capstone class.

1.6 Change history

2 References

References

[1] *About the Unicode Standard*. http://www.unicode.org/standard/standard.html.

[2] *Apache Xalan*. "http://xalan.apache.org: Apache Software Foundation.

[3] *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. https://www.w3.org/TR/2008/REC-xml-20081126/: W3C Organization.

[4] *HTTP Server Project*. https://httpd.apache.org/.

[5] *International Components of Unicode*. http://icu-project.org: The Unicode Consortium.

[6] *The Document Object Model*. https://xerces.apache.org/xerces2-j/dom.html: The Apache Software Foundation.

[7] *XSL Transformations (XSLT) Version 1.0*. https://www.w3.org/TR/1999/REC-xslt-19991116: W3C Organization.

3 Glossary

Apache An application for sending and receiving HTTP requests on a remote host.. 11, 12, 18

API Application Programming Interface. Connects the Apache server to upload or return the files.. 3, 4, 11, 12, 16, 17

BSD Berkeley Software Distribution. A popular UNIX operating system.. 16–18

CentOS A popular Linux operating system based on Red Hat Enterprise Linux.. 16, 18

CGI Common Gateway Interface. A way for a script on a local host to be run remotely via a web-server like Apache.. 11, 12, 18

CLI Command Line Interface. A user interface to a computer's operating system.. 11, 14, 17, 18

Debian A popular Linux operating system.. 16, 18

DOM A cross-platform and language-independent application programming interface that treats XML document as a tree structure.. 3–11

FOSS Free and Open Source Software. Software which is developed and maintained for free and by a community.. 18

FPM Effing Package Management. A software package creator which targets multiple Unix-like operating systems package managers.. 16, 18

GUI Graphical Interface. A user interface to a computer application.. 13

Hash-map Hash map based implementation of the Map interface.. 6

HTTP Hypertext Transfer Protocol. XZES40-Transformer interact with remote clients.. 11, 12, 14

ICU International Components of Unicode. A library process UTF-8 character format document.. 5, 11

Linux A popular UNIX-like operating system.. 16–18

MD5 A complex hashing algorithm used to securely verify data is consistent.. 8

OS Operating System. Software which runs other software.. 16

OS-API Operating System API. It will be the program's interface with the operating system.. 5, 11, 16, 17

Python A program scripting language used for a wide variety of purposes from scientific applications to dynamic websites.. 11–14, 18

Struct A complex data type declaration.. 7

UI User Interface. Exposes some functionality, usually referring to one on a computer, to the user through some way of interaction either mouse, keyboard, or combination of the two.. 11

Unicode Provides an unique number for every character.. 6

UNIX A family of Operating Systems encompassing Linux, BSD, and MacOS.. 11, 14, 18

URI Universal Resource Indicator. Also called a url, this is the path a client needs to use to identify a web-api endpoint.. 12

UTF-8 Unicode Text Format. The international standard for encoding test-based data.. 5, 10, 11

Web-API Web API. Connects web application by Apache server to upload or return the files.. 10–13, 18

Web-UI Web UI. A user interface which uses a browser to render its content.. 13

Windows A very popular non-unix operating system.. 16, 17

WIX Windows Installer XML. A Windows software packaging program.. 16

Xalan C++ A library used to XML transformation in C/C++.. 4, 17

Xerces C++ A library used to perform XML transformation in C/C++.. 6, 17, 18

XML Extensable Markup Language. The human-readable data format used and processed by our application.. 3–6, 12–14, 17

XSLT Extensable Stylesheet Language. The human readable format used to transform documents in our application.. 3–6, 12–14, 17

4 Document Transformer

This section outlines the views and viewpoints related to the Document Transformer. These components relate to core functionality of the application; taking input XML and XSLT documents, parsing them into DOM objects, transforming those into a new XML document, and then returning the final product to a user. This section does not necessarily outline how users interact with the application, for that one should see the User Interface section (5).

4.1 Transformer

As the name suggests, the Transformer is the core of the XZES40 Document Transformer, and the application as a whole. At a high level this component takes two documents, an XML and XSLT document, and returns a transformed XML document. The rest of the application is built around this component and all other parts of the application depend on or work toward this feature.

4.1.1 Context

The Transformer provides the core functionality of the application, transforming input documents into output documents.

All users of the application use the Transformer indirectly by way of using it to transform their input documents into output documents. The Transformer component will not be directly exposed in the final application, however it will be accessible via the Web API (5.1).

4.1.2 Composition

The Transformer's functionality is outlined in the following steps:

1) Receive parsed DOM object.
2) Check to see if the specified documents have already been transformed by checking for the transformed file in the in-memory cache (4.4).

   The input to this component is a retrieved cache object. By hashing these input objects together a unique key is generated. This key is used to check if the input documents have already been transformed, and is where the new document is stored if it has not already been transformed.
3) If the documents have already been transformed the object in the cache is **returned**; otherwise proceed.
4) Call the Xalan-C++ transformation library functions for XML transformation. [2]
5) Store the newly parsed document into the in-memory cache.
6) **Return** the newly parsed document.

4.1.3 Dependencies

The Transformer directly depends on the following internal application components for the following reasons:

- The Document Parser (4.3) is used to create a transformable DOM object from an input file. [6]
- The Document Cache (4.4) is used to store and retrieve parsed and transformed documents. While this cache isn't strictly necessary for document transformation, it is used to speed up the process drastically.

4.1.4 State Dynamics

The Transformer deals heavily with state dynamics in two ways: transforming documents and storing/retrieving cached documents.

4.1.4.1 Transforming documents

The Transformer does not directly handle transforming documents, this task is delegated to the Xalan C++ library. [2] The transformer takes two or more documents as input: an XML DOM object and list of XSLT DOM objects. The input documents are transformed by the Xalan C++ library and the resulting document is eventually returned by the writer (4.6) via the API (5.1).

4.1.4.2 Cached documents

The Transformer also operates with the cache (4.4). Input files are references to locations in the cache, and output documents are stored in the cache to avoid duplicate computations later on.

4.1.5 Interactions

Few components directly call the Transformer, however the transformer depends heavily on communicating with the cache. This is done by using a "cache" object. Documents are retrieved by using a "get" method and added to the cache with a "set" method. More information can be found in the Cache section of this document (4.4).

4.1.6 Interfaces

The transformer has one programmer-facing interface, the "transform()" function. This can be used for testing, mocking, or implementation purposes.

4.1.6.1 DOM object transform(hash-map * cache, DOM-object-reference xml_document, [DOM-object-reference xslt_document_1, ...])

- This function takes as argument the object cache, the XML object, and a list of references to XSLT documents.
- As output it returns a new DOM object.

4.2 Reader

The Reader takes input documents of varying file encoding, converts them to the UTF-8 file encoding, and prepares them to be parsed (4.3). [1] [5]

4.2.1 Composition

The Reader performs the simple but necessary operation of taking an input file and normalizing it for the parser to handle its contents. This is done with the following steps:

1) A file pointer is passed to the Reader.
2) This file is processed into a string. If necessary the character encoding of the file is determined.
3) The file is read into a temporary file.
4) If the file is of the Unicode encoding it is kept so; otherwise it is passed to the ICU library for proper encoding. [5]
5) The file's contents are saved locally for the parser to handle later.

4.2.2 Dependencies

The Reader depends internally on the OS-API (6.2) to read an input file. It is also a decency of the parser as non-unicode files will not be accepted by the Parser (4.3).

4.2.3 State Dynamics

The Reader takes in an input file and transforms it to a UTF-8 encoded document to be passed to the parser. The following conditions and types of events are of interest:

1) If the input file is completely mangled, or not a recognizable human-readable file the program will return an error the user.
2) If the input document is human readable but is not UTF-8 encoded then the ICU library will transform the document to the correct encoding for further processing.
3) If the input document is human readable and UTF-8 encoded then nothing is done to the file and it is saved for the parser.

It should be noted that the Reader does not handle checking if the document is *well formatted* (i.e., an XML 1.0 formatted document [3]), just that it is a UTF-8 document.

4.2.4 Resources

The Reader externally depends on the ICU library to decode the input document and encode it in Unicode for the Parser (4.3). [5]

The Reader also utilizes the OS-API (4.3) to perform reading input files.

4.2.5 Interface

The Reader has one interface to be used for testing, mocking, or implementation:

4.2.5.1 file reader(file-pointer input_document)

The function takes as input an open file pointer (integer) and returns a parsed file (string) or panics and causes and application exit.

4.3 Parser

The Parser gets the input XML and XSLT files and transforms them to DOM objects for the Transformer to use in the transformation process.

4.3.1 Context

The Parser is a major component of the application. The Parser receives as input an XML and a XSLT file and then outputs the DOM representation of those files. The user will not use this function directly, but it executed by the Reader (4.2).

4.3.2 Dependencies

The Parser depends on the Reader to format the input file correctly as Unicode. The Parser also depends on the Cache to store and retrieve parsed objects.

4.3.3 Interactions

The parser function is the middle step of the transformer function.

1) The Parser obtains a Unicode formatted XML or XSLT file from the Reader.
2) The Parser checks if the file has been parsed and stored in the Cache, and does not re-parse the file if it is in the Cache.
3) The Parser generates a DOM object via the Xerces C++ library.
4) The Parser stores the DOM object to the Cache.

4.3.4 Resources

The Xerces C++ library will be used during the parsing to generate a DOM object from the input file.

4.3.5 Interfaces

The following function declaration will be used for the parser method:

4.3.5.1 dom_object document_parser(file)

receives an XML file or a XSLT file, and then parses it into a DOM object which is stored in the Cache.

4.4 Cache

The Cache is a core feature of the XZES40 application. The Cache speeds up document transformation by storing and retrieving previously parsed documents.

In practice the Cache will operate much the same as a Hash-map does, storing data at a location given a key which can also be used to retrieve the data.

4.4.1 Context

The program heavily depends on the Cache. The Cache can store, delete, and retriever the DOM object from the in-memory cache

4.4.1.1 Storing data

is the major element of the Cache. Cache stores data from the user, primarily parsed DOM data to avoid re-compiling files again.

4.4.1.2 Deleting data

provides the ability to remove items from the cache for whatever reason.

4.4.1.3 Retrieving data

allows users to fetch information stored in the cache given the object's key.

4.4.2 Composition

Below are few of the components making up the Cache.

- The Cache stores input data in a Struct along with with the last time it was accessed and they key used to access the data.
- The Cache can retrieve the data via searching the associated key.
- The Cache can delete data corresponding with a key.
- The Cache can dump its contents to disk periodically.
- The Cache can load its data form an input file. This along with the previous item is used to recover from a system failure.

4.4.3 Logical

XZES40 handles object as a special structure. The Cache saves the data as the following information.

4.4.3.1 Key

This is a hash value associated with a cached object. This value is for retrieving the data from the Cache.

4.4.3.2 Content

The content is the parsed or transformed DOM object.

```
struct node {
        dom data;
    string key;
    date last_used;
}
```

4.4.4 Information

If the object is not in the Cache, the set Cache stores the DOM object into the in-memory cache. The way to remove the object in the cache is the delete method.

4.4.5 State Dynamics

The Cache handles the following state changes in the following ways.

- When the Cache starts it allocates a block of memory for storage. If a previously dumped cache contents exist on disk this data populates the cache contents.
- If an item is being set the cache ignores previously existing data. It is the developers duty to ensure important data is not being overwritten.
- When data is being read the state of that item in cache is assumed not to change. If the item is not found it returns an empty object.
- When an item is deleted it returns a SUCCESS status if the object existed and is not deleted, and a FAILURE status if the object did not exist in the cache before the call.
- Periodically the Cache dumps it's contents in memory to a local storage. This is used to recover from a system failure and is loaded on startup.

4.4.6 Interactions

The parser sends a parsed DOM object to the Cache. The Cache will search the flies if it is exist in the memory. If the file exists in the memory, the Cache retrieves the DOM object from the Cache and return to the Parser. If the file does not exist in the memory, the Parser continues to process the file, and stores parsed file into the Cache. The Transformer does a similar thing.

4.4.7 Algorithms

The Cache can be divided into five major function. The First function is **set**, second is **get**, third is **dump**, fourth is **load**, and fifth is **delete**.

4.4.7.1 Set

stores an object at a location in the allocated memory block based on the an MD5 hash of the "key" parameter given.

4.4.7.2 Get

This function retrieves the DOM object from the cache by the key of the parsed file.

1) If there is key exist in the cache, **return** the DOM object at that location.
2) If the key does not exist in the cache, **return** an empty storage struct object.

4.4.7.3 Dump

This function saves the contents of the Cache to disk.

- The Dump saves all data from the cache.
- The Dump outputs data to disk.

4.4.7.4 Load

This function loads the cache to memory.

- The load read all data from the disk.
- The load allocate the memory size for storing data.
- The load write data to memory.

4.4.7.5 Delete

This function delete the DOM object from the cache by the key of the parsed file.

1) The Delete receives a key to delete the data it's located at.
2) If there is key exist in the cache, the Delete deletes this DOM object in the cache, and **return** that there is successfully delete.
3) If the keys is not key in the cache, **return** is no DOM object inside of cache.

4.4.8 Resources

The Cache needs a large memory allocation to function well. The Cache needs a storage device to dump data to so it can save its contents.

4.4.9 Dependencies

The Cache depends on the Parser and the Transformer to populate it. The Parser sends a file to the Cache for checking if file exists in the Cache. The Parser also can send a parsed DOM object for storing into the Cache if file does not exist in the Cache. The Cache may either return a DOM object to the Transformer, or the cache return null if the DOM object is not in the Cache.

4.4.10 Interfaces

Below are the major interfaces for the cache component of XZES40-Transformer.

4.4.10.1 void document_cache(dom_object file)

This checks the DOM object if it exist in the Cache.

- If it's not in the cache, return fail.
- If it is in the cache, return true.

4.4.10.2 dom_object store_cache(dom_object file)

This receives object file and store it into the memory.

4.4.10.3 dom_object delete_cache(dom_object file)

This deletes data from the Cache.

- If the DOM object exists in the cache, return true.
- If the DOM object does not exist in the Cache, return error.

4.4.10.4 dom_object read_cache(dom_object file)

This retrieves data from the cache.

- If the DOM object exists in the cache, return the DOM object.
- If the DOM object does not exist in the Cache, return error.

4.4.10.5 dom_object dump_cache(dom_object file)

Periodically the Cache dumps data out from memory to disk, because we don't want lose any data. Loading data from memory ensures continuous speed improvements as the cache grows.

4.4.10.6 dom_object load_cache(dom_object file)

When the application crashed, the Cache can load the older data from the disk.

4.5 Parallel Computation

In addition to the Cache component of XZES40-Transformer (4.4), the application will also carry out certain computations in parallel to further leverage the computing resources available to it and compile documents even faster.

4.5.1 Context

The Parallel Computation component of the application carries out the following operations in parallel to speed up documentation transformations.

4.5.1.1 Document Parsing

will be carried out in parallel. These operations are logically independent so they can be carried out simultaneously without affecting data integrity.

4.5.1.2 User Requests

will also be carried out in Parallel, handled by Apache.

4.5.2 Dependencies

The Parallel Computation component of the application will be carried out at the high level by Apache delegating parsing jobs, see the Web-API section for more information 5.1. the application will also carry out parallel computation internally (C++) using the MPI library.

4.5.3 Interactions

Our MPI-based thread computing will operate mostly autonomously, except when putting data into and fetching data from the application Cache (4.4). With the exception of interacting with the Cache each parsing thread will not interact with other internal components of the application.

4.5.4 Algorithms

One major concern with handling a cache by multiple threads and processes in parallel is avoiding data corruption. This is not a problem we have yet solved and further revisions of this document will elaborate on how we will handle this dilemma.

4.5.5 Interfaces

When documents are being parsed each parsed document will spawn it's own thread. This functionality is in the Parser component of the application in section 4.3.

4.6 Writer

The Writer takes a DOM object and writes it to disk at a specified location on disk.

4.6.1 Composition

The Writer performs a straight-forward task by the following steps:

1) The function gets passed a DOM object to write to disk.
2) The DOM object gets translated to a string.
3) That string is converted to UTF-8.
4) The UTF-8 string is written to disk at the specified location on disk.

4.6.2 Dependencies

The Writer depends internally on the Cache (4.4) to retrieve documents which have been transformed to disk.

4.6.3 State Dynamics

The only state that the application works with is encoding documents and writing them to disk. This entails the following processes:

- A DOM object is un-parsed into a human-readable string.
- That human readable string is encoded as UTF-8.
- That UTF-8 string is written to a specified location on disk relative to the application's storage directory root.

4.6.4 Resources

The writer depends externally on the ICU library to encode a string in the Unicode character encoding. [5] It also depends on the OS-API 6.2 to perform system operations like writing files.

4.6.5 Interfaces

The writer will have one interface:

4.6.5.1 status-code write(DOM-object document, path location)

Where the inputs are a DOM object and a path. The path is relative to the configured basepath of the application, this defaults on UNIX systems to "/tmp/xzes/".

The output is a status code for reporting to the user interface if the document was written correctly.

5 User Interface

The Document Transformer is fine and great, but without a user interface it's not useful. The following two sections, the Web-API (5.1), Website interface (5.2), and CLI (5.3) together outline the ways users will will interact with the system described in the previous section.

5.1 Web-API

The Web-API is the standardized interface between the user interfaces and an instance of the application running on a host, communicating over HTTP.

5.1.1 Context

The users for the Web-API are those who use the application from the CLI (5.3) and (Website 5.2). Both interfaces interact with the XZES40-Transformer host via standard HTTP request methods. But nobody should use the API directly as a UI is much easier than crafting an HTTP POST request.

5.1.2 Composition

The Web-APIis composed of the following components:

1) An Apache runs on the remote host.[4]
2) The server manages a Python CGI script which handles accepting requests and sending responses.
3) The Python script calls the XZES40-Transformer application locally, passing input documents from a POST request and sending response files via the CGI interface.

5.1.3 Dependencies

The Web-API internally depends on an XZES40-Transformer binary which accepts an input XML file, and input XSLT file, and writes a transformed file to disk in a predictable location. The binary should also exit with predictable exit codes to communicate any errors or successes.

5.1.4 Interfaces

The Web-API communicates with remote clients via a standard HTTP API. Below are the requests a client can send and the possible responses:

5.1.4.1 POST /api/

This is a request sent to the API endpoint (/api/) with the intention of getting two input files transformed into a new document. This can respond in the following ways.

200 OK Means the transformation was successful. This response includes a body containing the transformed file and a URI to re-download the response file.

400 USER ERROR Means that the user sent malformed documents. This can include a document which does not follow the XML/XSLT standard to a document which does not have a readable character encoding. This response includes a body containing an appropriately specific error.

500 SERVER ERROR Means that the server experienced an internal error while processing the request. This includes fatal XZES40-Transformer errors. This response includes a body containing an appropriately specific error.

The POST request is a request with a Form containing input documents in fields titled XML and XSLT for an XML 1.0 formatted document and an XSLT 1.0 formatted document respectively.

5.1.4.2 GET /api/

This is a request sent to the API endpoint (/api/) with the intention of getting a status of the server. This can respond in the following ways:

200 OK Means the Web-API endpoint is active and functioning correctly.

404 NOT FOUND Means that the Web-API endpoint is not configured correctly or the user is accessing a page which is not available.

503 SERVICE UNAVAILABLE Means that the Web-API is setup correctly, but the application on the remote host is not operating correctly.

The GET request is an empty GET request to the servers "/api/" endpoint.

5.1.4.3 Python Interfaces

The following interfaces are used for implementing the above HTTP interfaces.

process_request(http-request req)

This method is used to receive an HTTP API request. It does this by reading the request header and deciding based on that to carry out one of the above responses. In the case of a POST request it uses an "exec" call to the local XZES40-Transformer application binary and responds with the error / output file of that application.

5.1.5 Resources

The Web-API has the following external dependencies:

- The **Apache** is used to process HTTP requests by running the Python CGI script.

- **Python 2.7+** is the programming language the Web-API is written in.
- **cgi** is a Python library for processing web requests in Python.
- **cgitb** is a Python library for developing with the cgi library.
- **mod_wsgi** is an Apache module for interacting with Python.
- **mod_python** is another Apache module for interacting with Python.

They should be installed via a system package manager or the Python package manager (whichever is appropriate) for the Web-API to operate correctly.

5.2 Website

The website interface is the major user interface for the XZES40 application. Website interface is a GUI interface for user.

5.2.1 Context

The Website interface is common interface for application. User can uses website to complete request with few of instructions. User can upload the XML/XSLT files that they want, and the website will give the feed back.

5.2.2 Composition

The web interface to access the application in a browser. Here is components of the Web-UI.

- An upload field for the **XML** file.
- An upload field for the **XSLT** file.
- A button for user send the request to server via **POST** method.
- After user sends a request to server the website will respond with a message to user. This message may be error or successfully upload or generate download link.

5.2.3 State Dynamics

There are few of states for website.

- If user upload bad files the website gives feedback that the files were bad.
- If user upload files and server transforming the documents the website gives the feedback that user upload files successfully.
- If user uploads files and server is down the website gives the warning that transformation service is not working.
- If user uploads good files and the server successfully transforms the documents the user is given a download link to the new file.

5.2.4 Interactions

The initialization status of website is waiting for user upload files. After user select flies that they want upload and click the submit button. Website will send the request to server via the Web-API. There will be many possible feedback as following.

- If the files uploaded are bad the website pops up a warning about the malformed files.
- If the user uploads both an XML and XSLT file the user is alerted that the request is good to go and that the website will generate a download link for them.
- If the connection is broken the website will give them a warning that user should check the connection between client and server.

5.2.5 Interfaces

The Web interface is XZES40 main interface for user. The web interface asks user to upload XML files and XSLT files, and gives the feedback of result. Here is the diagram for website interface.

5.3 CLI

The CLI is used to interact with the system via a text-based terminal/shell interface.

5.3.1 Context

The users of this interface are individuals who either prefer the CLI over a web interface or for testing purposes as automating tasks with the CLI is very common.

5.3.2 Composition

The CLI, written in Python, and will be composed of the following components.

- The main function parses command-line arguments specified below in the “Interfaces” part of this section.
- A query is built for the server including the XMLand XSLT documents to the specified server.
- The query is sent to the server in a POST request.
- When the response is received either an error message is displayed to the user or a file is saved locally.

5.3.3 Resources

The CLI depends on the following system dependencies:

- Python 2.7+ is the programming language it will be implemented in, so a Python runtime will be necessary for the application to run.
- Requests is the standard Python library for interacting with servers over HTTP.
- A terminal and UNIX compliant shell will also be necessary to access the application via the CLI.

5.3.4 Interfaces

The CLI has the following options available at the command-line:

5.4 Example Use-cases

The following are examples of the CLI in use:

“Un-happy path” with bad host.

6 System Requirements

The XZES40-Transformer will be required to work on the Debian Linux system. Once development on Debian is completed the application will be ported to other OS platforms. Our team will release Debian, CentOS, BSD and Windows packages.

6.1 Installation Packages

For installation convenience the XZES40 project will provide a Linux, BSD, and Windows installation packages.

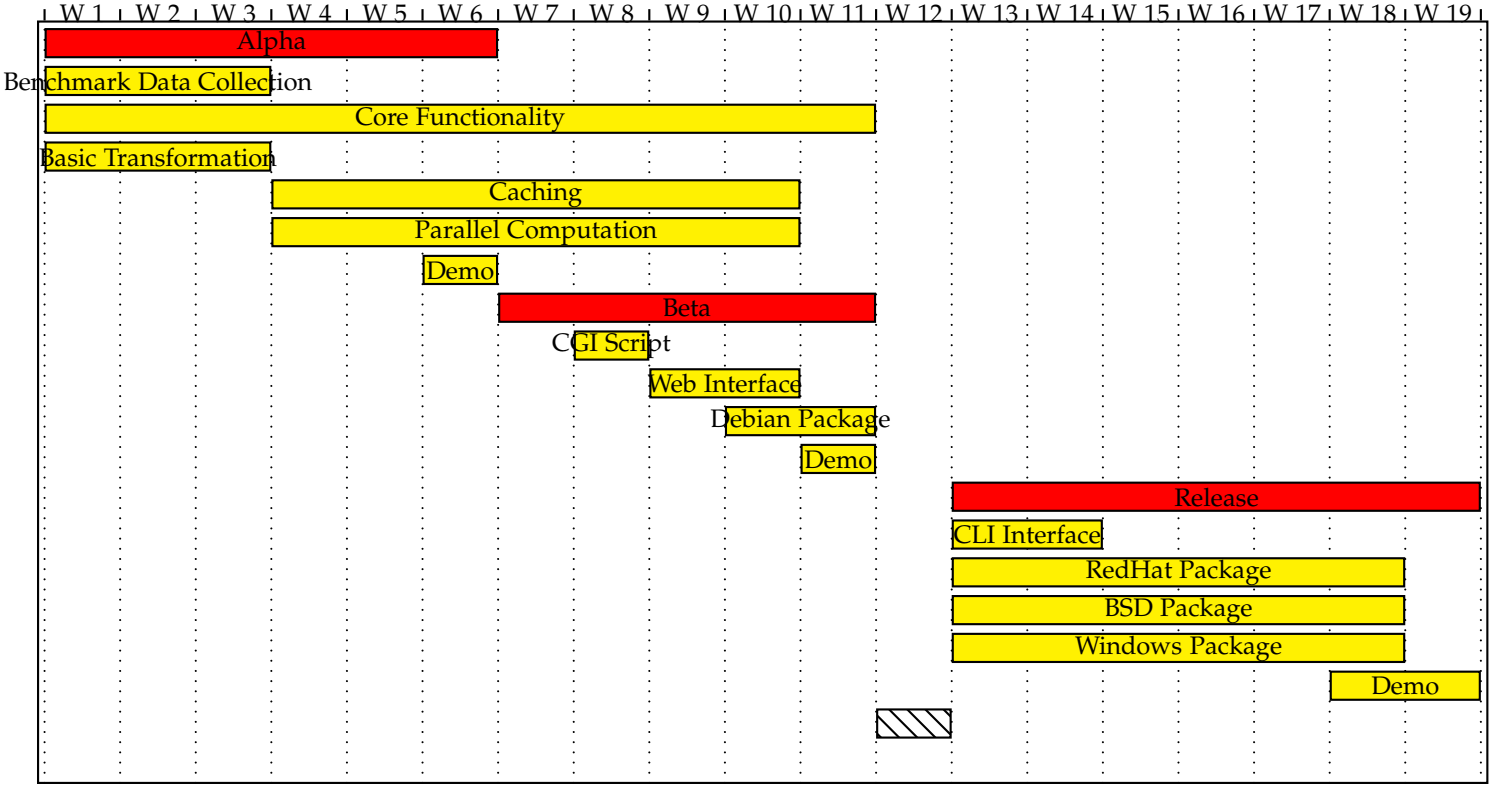

Figure 1: Development Gantt Chart Timeline.

Revision	Date
Working Draft	2016-11-17

Figure 2: Change History Table

```
$ xzes40
    --server=<server-url[:port]> # API endpoint and port to be used.
    --xml=<input-file>.xml       # Input XML file
    --xslt=<style-sheet>.xslt    # Input XSLT file
```

Figure 3: CLI Flags. All fields encased in chevrons symbols are required. All fields with square brackets are optional.

```
$ xzes40 --server=http://xzes40.example.com:8080 \
         --xml=input-file.xml \
         --xslt=style.xslt
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Downloading transformed document to ./xzes40-transformer-2016-11-22.xml
```

Figure 4: “Happy path” use-case without specifying return file name.

```
$ xzes40 --server=http://xzes40.example.com:8080 \
             --xml=input-file.xml \
         --xslt=style.xslt \
         --output=transformed-doc.xml
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Downloading transformed document to ./transformed-doc.xml
```

Figure 5: “Happy path” use-case with return file name.

```
$ xzes40 --server=http://xzes40.example.com:8080 \
         --xml=input-file.xml \
         --xslt=style.xslt \
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Server responded with error. One of your documents is malformed.
```

Figure 6: “Un-happy path” with malformed document.

6.1.1 Context

The user can directly download installation packages through the internet and install the XZES40-Transformer on their local systems. We will create the Debian package initially. After this, we will create the CentOS, BSD, and Windows packages. We will upload those packages to the internet and the users can directly download the packages in their operation system.

6.1.2 Resources

The following tools will be used to create the installation packages.

6.2 OS-API

The OS-API will be the programs interface with the OS. It will be created to make porting the application easier.

```
$ xzes40 --server=http://fakesite.com:8080 \
         --xml=input-file.xml --xslt=style.xslt \
Sending input-file.xml and style.xslt to the http://xzes40.example.com:8080
Unable to reach server. Is the host/port correct?
```

Figure 7: “Un-happy path” with bad host.

```
$ xzes40  --xml=input-file.xml--xslt=style.xslt
Please specify a host
```

Figure 8: “Un-happy path” with no host.

```
$ xzes40  --server=http://xzes40.example.com:8080
Please specify an input xml and/or xslt document.
```

Figure 9: “Un-happy path” with missing input document.

Packages	Tools	Description
Linux & BSD Packages	• FPM	• Translates packages from one format to another • Allows re-use of other system’s packages
Windows Packages	• WIX	• It is a open source project. • It is more stable and security than other tools. • It has steed

Figure 10: Installation Packages Resources

6.2.1 Logical

Any operating system specific operation will be wrapped by the API.

6.2.2 Interactions

The application will interact with the host system via an OS-API. This means that all operating system specific operations (e.g., read, write, seek, etc) will be done via an API. When the application is compiled on a new target platform (e.g., Linux, BSD, Windows) a new platform API must be created for compatibility.

6.2.3 Interfaces

The OS-API interface is for the developer. This interface performs operations by asking the OS-API to carry out the task and that request is translated to the system-specific system-call.

6.3 Performance Benchmark

6.3.1 Context

We will run test against similar application to determine how fast XZES40 should be. Throughout development we will put our application through the same paces and compare which is faster.

6.3.2 Resources

We will be bench-marking these applications specifically for the following reasons.

Technology	Description
Xalan C++ CLI	• Xalan C++ uses Xerces C++ to parse XML documents and XSLT. • The project provides an open source CLI program to test the project libraries. • Free and Open Source
Altova	• To meet industry demands for an ultra-fast processor. • It offers powerful, flexible options for developers including cml, python. • Superior error reporting capabilities include reporting of multiple errors, detailed error descriptions.

Figure 11: Bench-marking Resources

7 DESIGN RATIONALE

This section explains why certain design decisions were made and "connects the pieces" of the application.

7.1 Cache Decisions

The reason the Cache uses an in-memory system rather than a database or a file-based cache is purely for performance reasons. Using an in-memory cache over a file-based one should yeild faster performance. As a compromise the cache is dumped to a file periodically to save the state of the cache in case the application daemon (which holds the cache) is restarted.

This may cause problems as the service may run out of memory. In an attempt to mitigate this a Garbage Collector may be built which is triggered when a certain percentage of the application's allocated memory is used, or on certain time intervals.

7.2 Web-API Decisions

The Web-API connects the application to the outside world in an ideally simple to implement and quick to operate fashion. The application uses Apache to handle incoming requests, these are passed to a CGI script which could be written in anything, we chose Python because it is easy to write, maintain, and is well supported.

Python is called by Apache which then in turn, based on the request, either returns an application status or calls the XZES40 application locally. In this way Python with Apache is a simple, well supported set of tools which expose the application to the outside world.

These tools were not necessarily chosen for their speed, and so a redesign may be necessary if they create a bottle-neck in the request pipeline.

7.3 Packaging Decisions

This document outlines the use of FPM to create its Linux and UNIX packages. This decision was made for convenience and to allow for quick iteration on the package. In doing research there *are* other tools which can be used to create packages on CentOS, Debian, and BSD but they are very system specific and so would make iterating on the package very difficult, and updates to the software a pain to package. Using FPM we can even automate the build of packages on our UNIX-like systems.

7.4 Bench-marking decisions

To demonstrate the competitiveness of the XZES40-Transformer application we chose to compare the performance of this application against an FOSS tool, the Xerces C++ CLI, and a Closed Source tool, the Altova application.

This feels like a fair and balanced comparison as we hope to be competitive with Closed Source alternatives but need to ensure we are at least better than a close FOSS alternative.

Elijah C. Voigt

Signature

Date

Zixun Lu

Signature

Date

Shuai Peng

Signature

Date

Steven Hathaway

Signature

Date