Birzeit University

SWEN6301, SOFTWARE CONSTRUCTION

1st semester 2016/2017

**Multipart Downloader**

Software Design Document

This document describes the software design of **multipart downloader**, including UML diagrams, state machine and test cases

Contributors: Ahmad Bdair, Ahmad Qamhieh, Muhammad Abed Raboh.

Instructor: Dr. Ahmad Afaneh.

Modification History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Originator | Comments |
| 1 | 22.12.2016 | AhmadB, AhmadQ, Muhammad | Initial Draft. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Modification History 1](#_Toc470361893)

[Problem Definition 3](#_Toc470361894)

[Software Architecture 3](#_Toc470361895)

[Software Requirements 4](#_Toc470361896)

[UI 4](#_Toc470361897)

[Manifest file 4](#_Toc470361898)

[Manifest parser 4](#_Toc470361899)

[Tools 5](#_Toc470361900)

[License 5](#_Toc470361901)

[UML Class Diagram 5](#_Toc470361902)

[Finite State Machine 6](#_Toc470361903)

[Test Cases 7](#_Toc470361904)

[Invalid Content type URL 7](#_Toc470361905)

[Segments not separated by two stars 8](#_Toc470361906)

[Stream alternative 8](#_Toc470361907)

[Manifest to manifest linking 8](#_Toc470361908)

[No internet access / loss of the internet during the downloading 8](#_Toc470361909)

[Empty Manifest file 9](#_Toc470361910)

[Meetings Minutes 9](#_Toc470361911)

[Meeting #1 9](#_Toc470361912)

# Problem Definition

One way to handle the problem of storing large files is to break them into pieces that are placed on multiple disks or machines. At the same time, a way to download a data stream more quickly is to break it into parts and download the different parts from different peers. (similar to the peer-to-peer file sharing protocol BitTorrent). By downloading streams in multiple small parts, the chance of the entire download failing is reduced (since each part can be restarted individually) and load is spread more evenly across the network.

Multipart downloader will assemble a data stream from multiple, potentially endless, parts streaming individually from multiple machines. It allows the same part to be stored redundantly in multiple locations so it can be resilient to failures. Since the multipart streams you will be downloading may be unbounded and never end, your program will assemble the stream incrementally from its parts as they are downloaded, displaying the file or streamed sequence of files (an animated sequence of images, for example) as the download progresses.

# Software Architecture

The multipart downloader will consist of different layers; these layers are:

1. UI which will be based on java.
2. Manifest file: which contains references to segment/ segments, also it may contain references to other manifest files.
3. Segments which consists of URL’s, multiple segments will download the file from different locations.
4. Manifest parser which will parse the main manifest file.

# Software Requirements

In this section we will discuss the multipart downloader requirements.

## UI

The UI will be a java based, technology to be determent (either swing or Java FX), the layout of the UI is not yet determent.

## Manifest file

The manifest file will contain references to segments, the manifest itself might contain a reference to segments or other manifest file

## Manifest parser

The parser will parse the manifest based on these requirements.

1. The content type of the provided URL must contain text/segments-manifest or its URL ends with the suffix. Segments
2. The manifest lines are separated by two stars.
3. The manifest can provide alternative to the URL, if the first URL is not working, the downloader will try to access the second one, for example, the manifest might look like this.

http://machine1.birzeit.edu/picture.jpg-segment1

http://machine2.birzeit.edu/picture.jpg-segment1

\*\*

http://machine1.birzeit.edu/picture.jpg-segment2

http://machine2.birzeit.edu/picture.jpg-segment2

\*\*

http://machine1.birzeit.edu/picture.jpg-segment3

<http://machine2.birzeit.edu/picture.jpg-segment3>

1. The manifest file might contain URL to other manifest files, for example

http://hermachine.birzeit.edu/verse.txt

\*\*

http://machine2.birzeit.edu/chorus.txt

\*\*

<http://machine1.birzeit.edu/endless.txt.segments>

The last line in this manifest points to other manifest file.

# Tools

We will use the following tools to help us to develop the program

1. For source code, we will use github to store the source code and a copy of this document

URL: <https://github.com/AhmadBadir/Downloader>

1. We will use Bugzilla to track the developments of this program

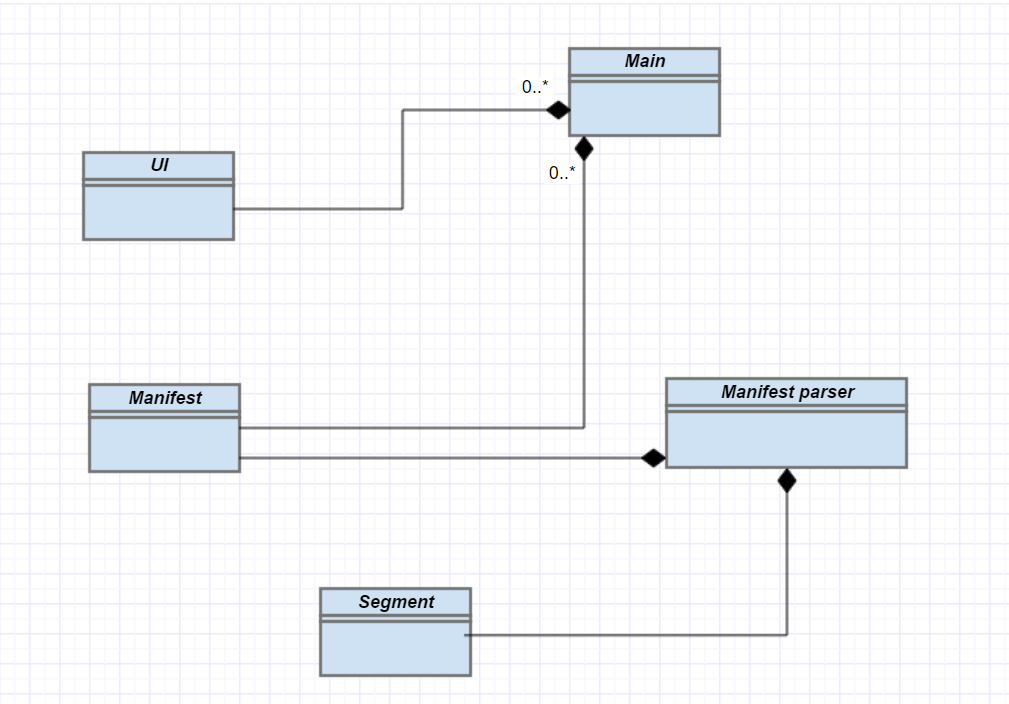
URL: TBD

1. We will use Eclipse to develop the program in java programming.
2. We will use Maven to export the executable Jar file.

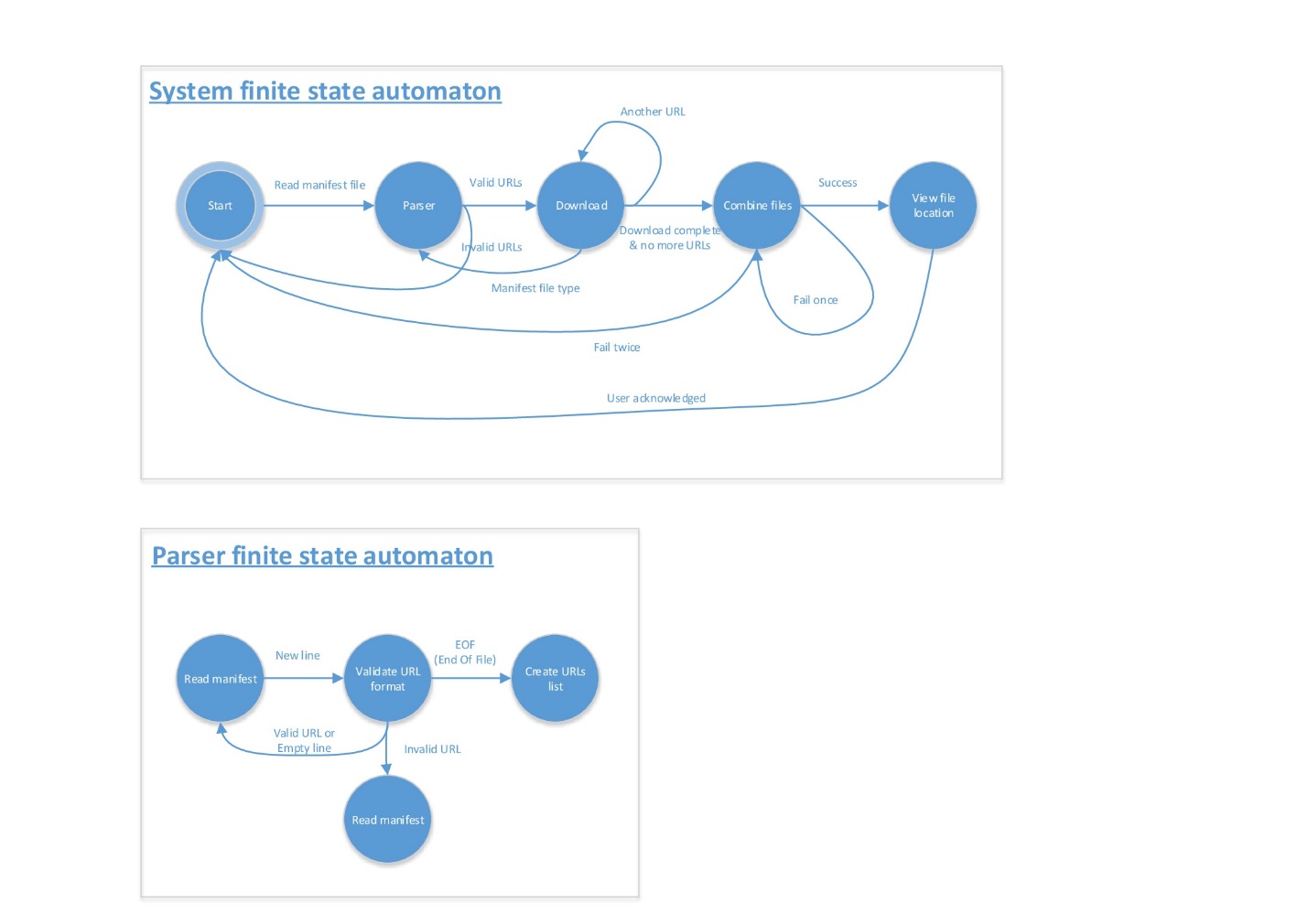
# License

The program will be under Apache 2.0 License.

# UML Class Diagram



# Finite State Machine



# Test Cases

## Invalid Content type URL

|  |  |  |
| --- | --- | --- |
| Test case Title | | Invalid Content type URL |
| Test case Procedure | | 1. Start the program. 2. Enter a URL to manifest file, the content type of the URL for a valid Manifest file must be text/segments-manifest or the URL ends with .segments 3. Now change the content type to texts/segment 4. Repeat the steps but this time change the URL to end with .seg while the content type is correct |
| Pass/ Fail criteria | 1. The program should reject the invalid URL. 2. The program should not crash. | |
| Automation Yes/ No | | Yes |

## Segments not separated by two stars

|  |  |
| --- | --- |
| Test case Title | Segments not separated by two stars. |
| Test case Procedure | 1. Start the program. 2. Enter a URL to manifest which its segments separated by one star (\*) or with different characters. |
| Pass/ Fail criteria | 1. The program should reject the separation of one star/ different characters. 2. The program should not crash. |
| Automation Yes/ No | Yes |

## Stream alternative

|  |  |
| --- | --- |
| Test case Title | Stream Alternative |
| Test case Procedure | 1. Start the program. 2. Enter a URL to manifest which contains stream with alternative 3. Make sure the first url isn’t working 4. The program should try to access the second url. |
| Pass/ Fail criteria | 1. The program should try the first url, if not accessible, it will move to the second alternative sream. 2. The program should not crash. |
| Automation Yes/ No | Yes |

## Manifest to manifest linking

|  |  |
| --- | --- |
| Test case Title | Manifest to manifest linking |
| Test case Procedure | 1. Start the program. 2. Enter a URL to manifest which contains link to other manifest. 3. The program should handle that case successfully |
| Pass/ Fail criteria | 1. The program should access the first manifest, parse the link to the second manifest and parse it successfully. 2. The program should not crash. |
| Automation Yes/ No | Yes |

## No internet access / loss of the internet during the downloading

|  |  |
| --- | --- |
| Test case Title | No internet access / loss of the internet during the downloading |
| Test case Procedure | 1. Start the program, make sure you don’t have internet access 2. Enter a URL to manifest 3. The program should generate error that no internet / link is not reachable. 4. Now restore the internet, enter the URL, wait for download to start, then cut down the internet |
| Pass/ Fail criteria | 1. The program should handle the case of internet loss successfully 2. The program should not crash. |
| Automation Yes/ No | No |

## Empty Manifest file

|  |  |
| --- | --- |
| Test case Title | Empty Manifest file |
| Test case Procedure | 1. Start the program, make sure you don’t have internet access 2. Enter a URL to empty manifest 3. The program should generate error the manifest has no content |
| Pass/ Fail criteria | 1. The program should generate clear error message telling that the manifest has no content. 2. The program should not crash. |
| Automation Yes/ No | Yes |

# Meetings Minutes

## Meeting #1

Attendance: AhmadB, AhmadQ, Muhammad Abed Rabboh

1. Set up the github URl – AI AhmadB
2. Set up the finite state machine – AI Muhammad
3. Setup the initial class diagram, we agreed on putting all relation to composition for now until the picture clears up – AI AhmadB.
4. Write down initial test cases – AI Ahmad Q.
5. Start writing the design document.

Further improvements on the UML, test cases are expected when start writing the code