**CS673S16 Software Engineering**

**Team 5 - Project Ukubuka**

**Software Design Document**

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role(s) | Signature | Date |
| Rohit Agrawal | Team Leader/ Implementation Leader | *Rohit Agrawal* | 10/16/2017 |
| Yashvardhan Nanavati | Backup Team Leader/ Configuration Leader/ Implementation Leader | *Yashvardhan Nanavati* | 10/16/2017 |
| Wenjun Shen | Requirement Leader/Environment and Integration Leader | *Wenjun Shen* | 10/17/2017 |
| Tianqi Xu | Requirement Leader/ Security Leader | *Tianqi Xu* | 10/17/2017 |
| Anirvan Maiti | Environment & Integration Leader  Design Leader | *Anirvan Maiti* | 10/18/2017 |
| Aswin Vasudevan | Implementation, Design and Security Leader | *Aswin Vasudevan* | 10/18/2017 |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Date** | **Change** |
| **1.0** | **Yashvardhan Nanavati** | **10/16/2017** | **Initial Draft** |
| **2.0** | **Rohit Agrawal** | **10/25/2017** | **Minor Review Changes** |
| **3.0** | **Yashvardhan Nanavati** | **11/07/2017** | **Added Software Architecture and its Description** |

[Introduction](#_gjdgxs)

[Software Architecture](#_30j0zll)

[Design Patterns](#_1fob9te)

[Key Algorithms](#_3znysh7)

[Classes and Methods](#_2et92p0)

[References](#_tyjcwt)

[Glossary](#_3dy6vkm)

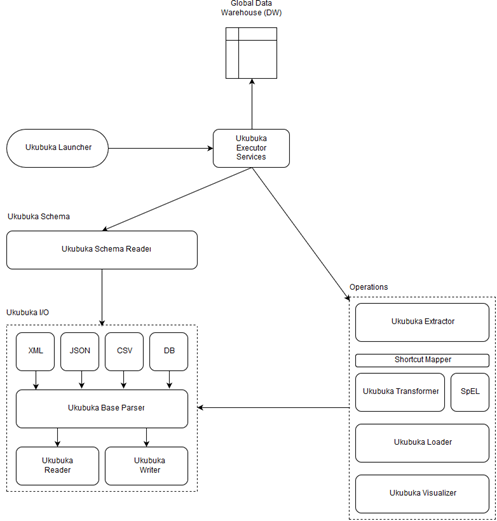
# Introduction

Motivation for this project comes mainly from the inability of the existing visualization tools like ‘Kibana’ to parse multi-level data in the form of JSON and their outdated, lack-lustered visualizations. Although powerful, Kibana lacks to produce interactive visualizations defeating its own purpose.

The name Ukubuka is inspired from a unique language spoken in South Africa and roughly translates to *"View"* or *"Visualization"* in Zulu which is one of the official languages of South Africa and is spoken by about 9 million people mainly in Zululand.

Ukubuka aims at producing high-quality interactive visualizations. It is powerful, lightweight and easy to use. It allows the users to analyze, manipulate and visualize their datasets in a novel and creative ways with no serious programming.

# Software Architecture



* Ukubuka Launcher is the main entry point for the user.
* The Ukubuka Executor service can be considered as the heart of the engine. It is responsible to call and manage the communication with Ukubuka I/O services and Operations class.
* The Global Data Warehouse will work as a huge storage of data where all the files will be stored while the Ukubuka Engine works on them.
* **Ukubuka I/O services:**

Ukubuka Base Parser sits on top of the Ukubuka Reader and Ukubuka Writer classes. The Reader and Writer classes are only responsible for reading the input and writing the input. They are called by the base parser when the schema arrives and by the Extractor when the file fetched from the user specified location arrives.

The Reader class then reads the file and hands it over to their respective parsers such as CSV, JSON, etc. The Writer class is responsible for writing the string data in the JSON format.

* **Ukubuka Operations:**

Ukubuka Operations houses the ETLV (Extract, Transform, Load and Visualize processes) which act as the core of the Ukubuka Engine. It also interacts with the I/O services while processing a particular file.

The Ukubuka Extractor is responsible to fetch the files from the remote location mentioned by the user or the Global Data Warehouse. The Shortcut Mapper is responsible for importing the required java libraries for transformations to be performed on the data. This is done in order to hide the complexity from the user.

The Ukubuka Transformer Class holds the crux of all the transformations to be performed on the data. It implements the logic for those transformations and evaluates the mathematical expressions using SpEL (Spring Expression Language). This allows Ukubuka Engine to practically perform any transformation that can be done using Java.

The Extractor class is responsible to join multiple datasets standardized by the Ukubuka Transformer class and create a single conglomerate of data which can then be used by the Ukubuka Visualizer class for spitting out crafty visualizations using the user specfied parameters and data.

# Design Patterns

In this section, you shall describe any design patterns used in your software system.

We have used the Java **Singleton** Design pattern for our project which comes under the Creational Design pattern category. As its definition states, this design pattern restricts the instantiation of a class and ensures that there exists only one instance of the class in the JVM (Java Virtual Machine).

We also plan to implement **Factory** and **Builder** design patterns in our project. The Factory method can be seen as a simplified version of the Builder method.

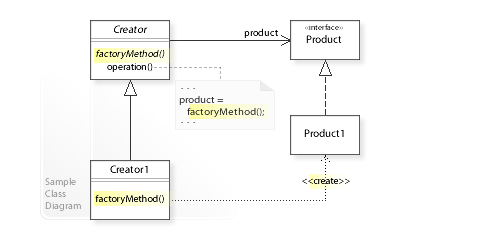
In the Factory design pattern, the factory is in charge of creating various subtypes of an object depending on the needs of a project. However, it does not know the exact subtype of the object. Rather it’s a high level view.

An example of Factory method could be if there is a method named “produceCars”, it might return a “BMW” or “Audi” typed object with it being unaware of the specific subtypes of cars in those objects.

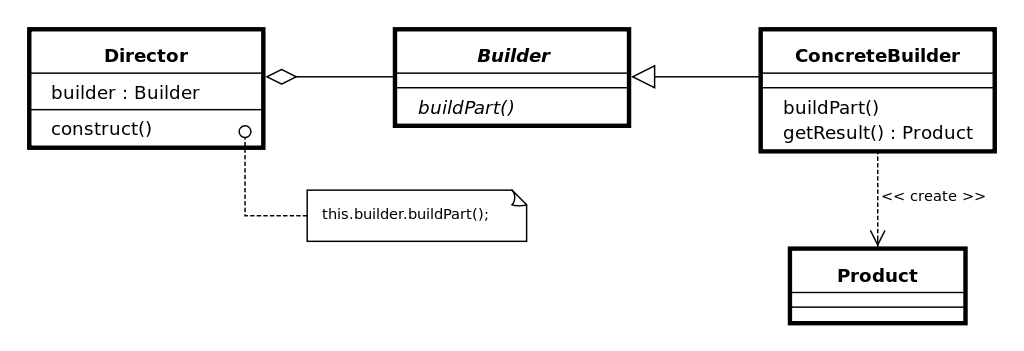
In the Builder design method, different subtypes are also created by it. But the composition of the objects might differ in the same subclass.

As an extension of the example given in the Factory method, we might have a “produceCars” method which creates objects of type “BMW” typed object with a 4 cylinder engine, or a “BMW” typed object with 6 cylinders. The builder pattern thus allows for this type of detailing.

**Factory Pattern:**

****

**Builder Pattern:**



# Key Algorithms

In this section, you shall describe any key algorithms used in your software system, either in terms of pseudocode or flowchart.

**Ukubuka Base Parser:**

This is a key algorithm that we have implemented. The parser acts as an interface between the reader/writer classes and the classes which handle parsing of csv/json/xml files. It is responsible to effectively communicate between the above mentioned modules. It takes the file read by the reader and passes it to a appropriate class depending upon the appropriate type of the file. It also handles if the delimiter of the file is something other than “,” and appropriately passes it to the respective parser. Also, missing headers in a data file are filled up using default column names such as column\_1, column\_2 etc. At the end, the files are consumed as a list of strings by the Ukubuka Engine.

**Example:** “A;B;C;D” is converted to “A,B,C,D”

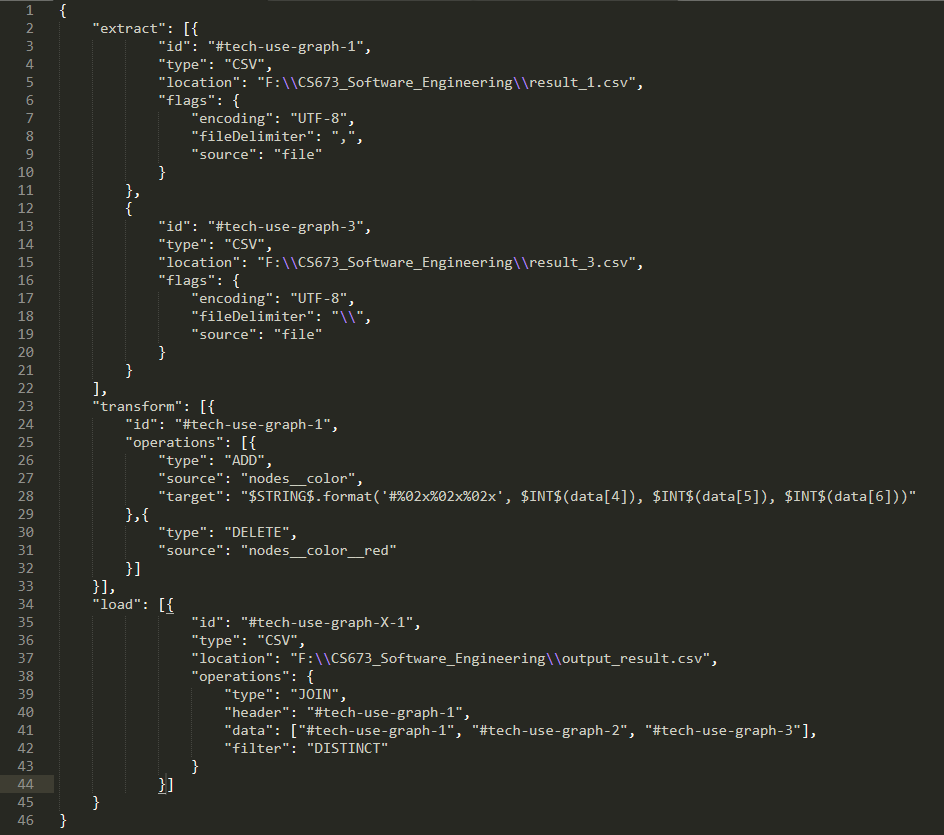
**Ukubuka Transformation:**

This algorithm uses SpEL (Spring Expression Language) to perform transformations on the data in the files specified by the user. The Spring Expression Language (SpEL for short) is a powerful expression language that supports querying and manipulating an object graph at runtime. The language syntax is similar to Unified EL but offers additional features, most notably method invocation and basic string templating functionality. This Algorithm is extremely powerful and can potentially solve any kind of mathematical computational problems including complex trigonometric, and statistical conundrums. For user-friendliness we have created a supporting class called “Shortcuts” which Imports the corresponding libraries in Java with abstraction to the user. The user just enters normal names of the operations and the algorithm takes care of importing the corresponding classes and interfaces.

**Ukubuka Schema:**

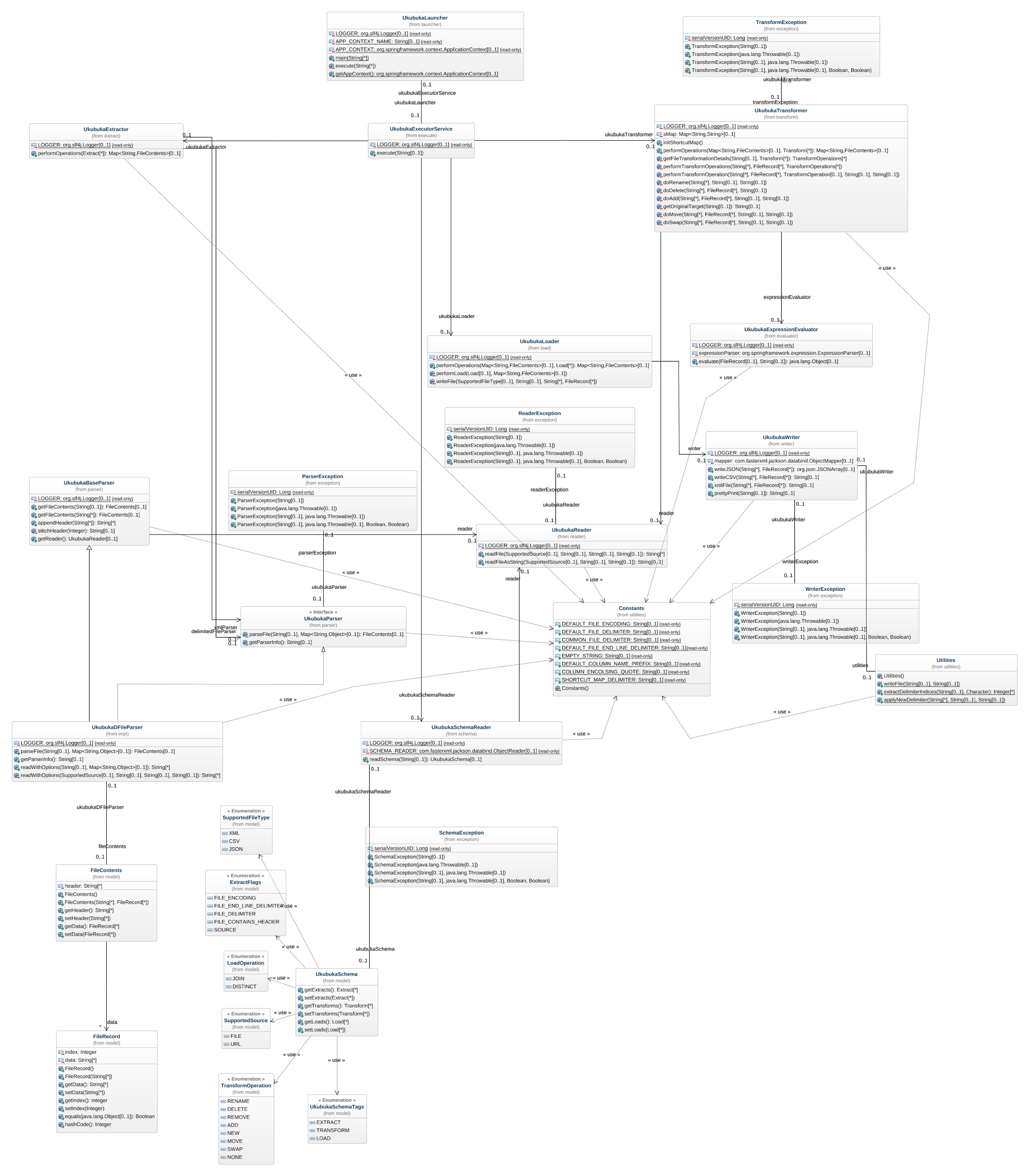
Another novel structure we came up with is the Ukubuka Schema in which the user has been given pre-defined tags to specify the manipulations he/she wants to do on the data set. Also, multiple loads, transformations and extractions can be done using a single schema file. This allows the user to consolidate all manipulations into a single structure seamlessly performs the specified operations without much hassle.

**Example Schema:**



# Classes & Methods

This part can be a reference to automatic generated document for all classes and methods.



Zoomed Version : [Ukubuka-core-class-diagram](https://drive.google.com/a/bu.edu/file/d/0B-YuiXUg094GOE01RWQ2OV9RUDA/view?usp=sharing)

# References

* + Kibana(<https://www.elastic.co/products/kibana>): Kibana gives you the freedom to select the way you give shape to your data. And you don’t always have to know what you're looking for. With its interactive visualizations, start with one question and see where it leads you.
  + <http://www.en.wikipedia.org/>
  + <http://stackoverflow.com>

# Glossary