Thomas Mühlematter

**User API and Examples**

**Overview**

The goal of this project is to have a simple interface implementing time series and ways to manipulate them. It must be easy to use without being an expert at programming. To have an easy interface, we chose implement a system based on commands. The alternative would have been to give the user the ability to input commands resembling natural language but this would have been more complex for the user when using complex requests.

The inspiration for this is the Matlab interface. The user calls functions and gets results that can be assigned to variables. The user can also choose which server to contact or where and how to store data on the computer.

The user interacts with the system using commands. Each command is followed by the reply from the system. This reply can be a message of success or error or the result of a certain function.

More precisely, the main features of this system are:

* Create, store and load time series
* Apply transformations on single time series
* Compare two time series
* Connect to a Spark server or use local data to store the time series
* Ability to choose an indexing scheme and to select different clustering schemes

**High-level API**

Console

It handles the communications between the user and the system. More precisely, it checks that the inputs are correct and send the user’s commands to the system. It also shows him the messages (success/error,result) sent back by the system.

Operations

Receives the command of the user sent by the console and process them. It communicates with Timeseries,Console,Transformations and Storage.

Timeseries

It regroups all informations pertaining to the timeseries and stores the current temporary data fetched from file or online.

Transformations

It contains all the transformations that can be applied on the time series. It is directly called by the class Operations.

Storage

The I/O is handled by this class. This is the class that is called by operations each team data need to be fetched or stored. This is also the class that handles indexing, clustering and replication.

Indexing

The indexing scheme’s choice will be left to the user. He will be able to choose how to segment the time series and which model to choose to store each segment. There are two main categories of schemes to index the time series: online and offline.

**Command Line (1. matlab style, 2. natural language)**

Each command is followed by the reply from the system. This reply can be a message of success or error or the result of a certain function.

a) User will be able to save value or result in a variable

**1.** timeserie = load(‘file.txt’);

**2.** past load file ‘file.txt’ to timeserie

b) User will be able to use function from PAST API

**1.** DFT(timeserie, 10, 100)

**2.** past execute DFT of timeserie between 10 and 100

c) User will be able to use standard math operation

**1.2.** 1 + 1

d) user will be albe to use condiational operation (if, for, while)

**Examples (C:command,R:reply)**

Find range of a certain time series

C: T=Timeseries(“id of the timeseries to load”);

R: T created successfully

C: a=range(1,10000);

R: a=5486

C: exit;

R: Byebye

Loading a non-existing time series

C: T=Timeseries(“id of a non-existing  ts”);

R: The Timeseries “id of a non-existing ts” does not exist

C: exit;

R: Byebye

Storing time series

C: T=Timeseries(“id of the timeseries to load”);

R: T created successfully

C: movingAverageSmoother(1,10000,2);

R: moving average smoother applied

C: store(T)

R: T successfully stored

C: exit;

R: Byebye

**Application Example**

An application of time series that is very useful in trading is the forecasting of stock prices. We will concentrate on the evolution of the price of a single stock.

As an example of what our system can do we will try to predict the price of a stock one week from now.

To do this, we need sufficient real or synthetic data for the price of a certain stock. We will assume that the price of a certain stock follows a certain pattern and that we have data for a sufficiently long time.

The input for the prediction is the stock prices for the current week (for instance, the last five days). Then we search in the data the week or weeks that were the most similar to the current week.

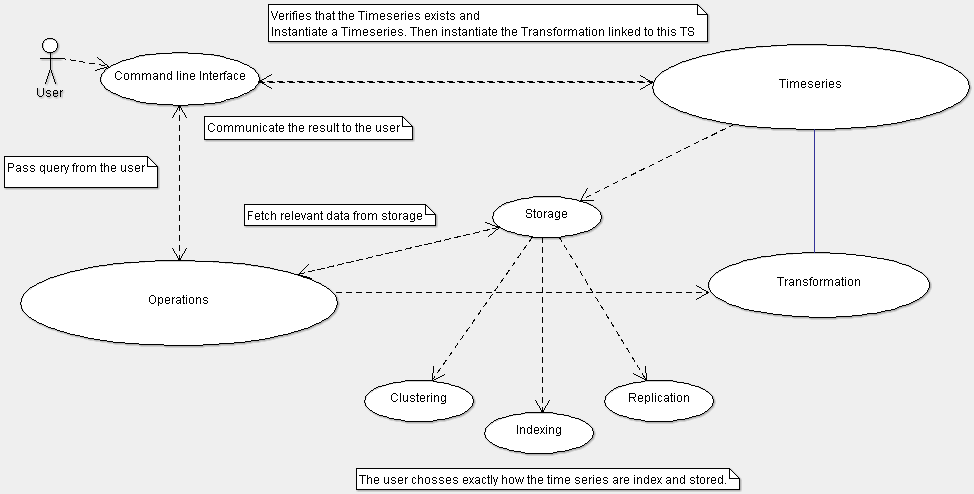
Then, we compute the average of the projections (the following weeks of the result) to find the future behavior of the time series. As an option, we could also return the average stock price for the following week.

The drawback for this application is that it assumes that the price of the stock follows a certain pattern without taking into account the punctual effects on the price like market crashes, merger, etc and the overall market evolution.

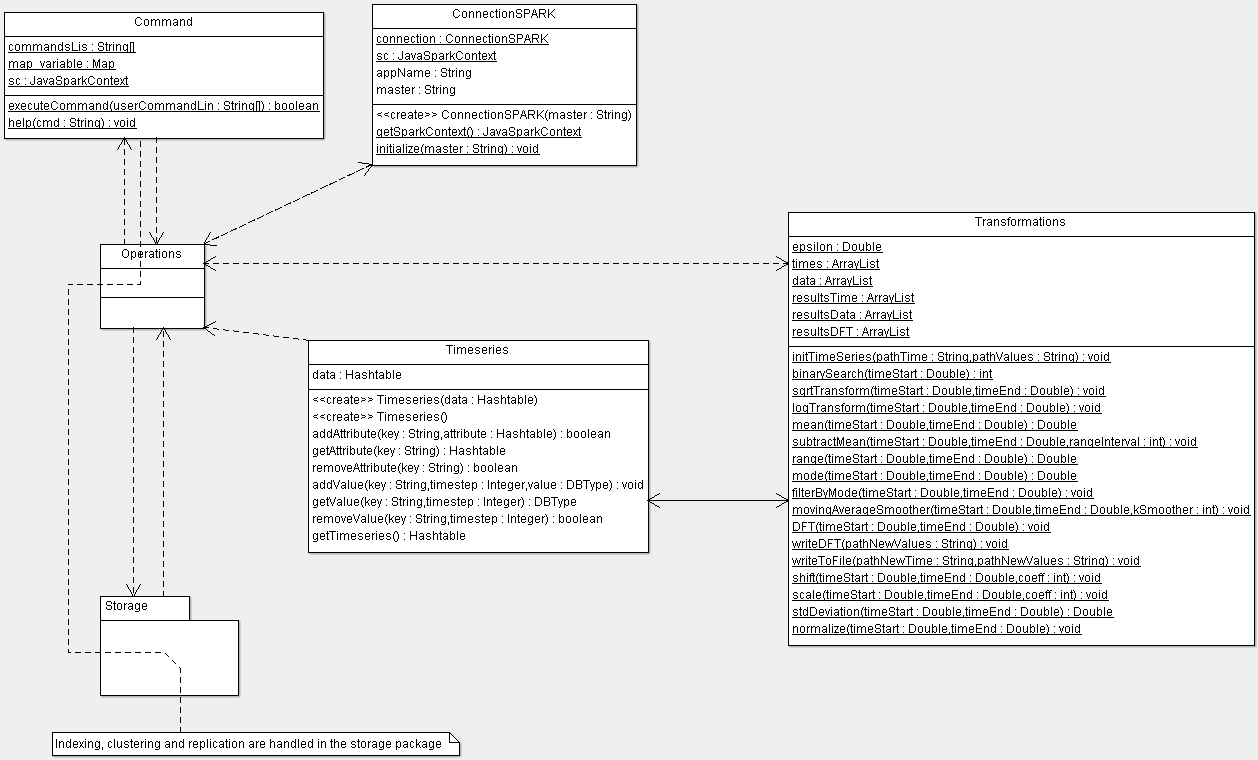
**Another possible application**

We could also devise an application where we do not forecast stock prices but  some weather characteristics like earthquakes or rain. This could be interesting as the data for weather may be easier to find.

**UML Usage diagram**



**UML Class diagram**

****