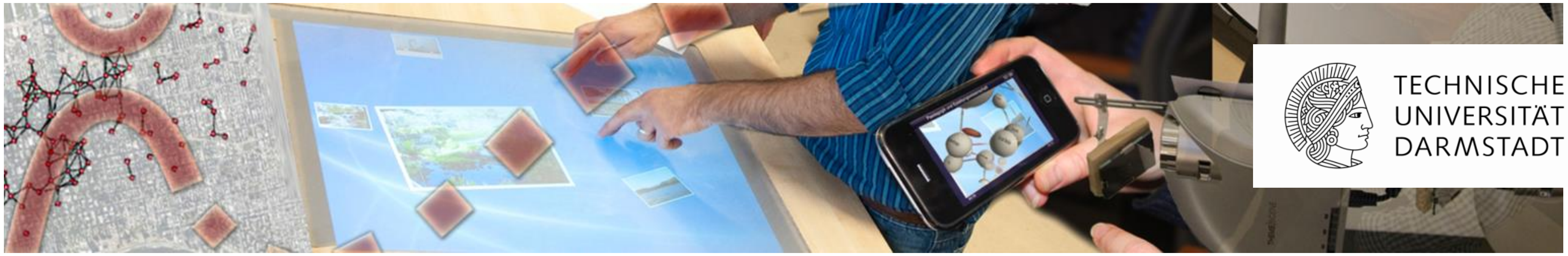
TECHNISCHE UNIVERSITÄT DARMSTADT

OpenDiabetesVault: Data Gathering and Data Slicing Algorithms

MSc. Thesis Presentation by
Ankush Chikhale

Supervisor - Jens Heuschkel

2 February 2017



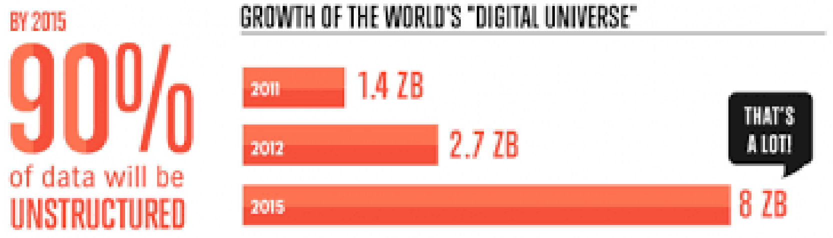
CONTENT

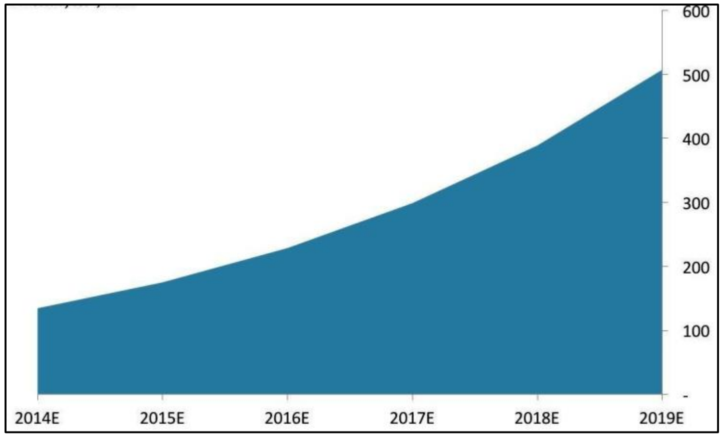
Introduction

▪Volume of data being created is growing every year

▪According to Forbes, by the year 2020 the amount of generated data will be 44 trillion gigabytes.

▪One of the biggest challenges is to handle the huge amount of data and gain information from them.





Introduction

Motivation:

- Researchers at TK have undertaken a project for analyzing medical data.

- By analyzing medical data, researchers wish to generate patterns/ Trends.

- Basic steps of Data mining/analysis is to have a big set of data.

Introduction

Motivation:

- Where do we get this huge data from?
    - Crawler Carelink Website
    - Google Maps Timeline
    - Smartband App
    - Glucosio Plugin

- During this thesis work, we concentrated on data extraction from the Crawler Carelink Website only.

- Once data extraction is done, This data has to be filtered/sliced to the requirement of the problem statement.

- Parallel to developing new algorithms, Unit testing of each module has been taking place.

Introduction

Goal and Contribution:

- To develop algorithms for extracting bulk amount of data from Carelink website.
- To develop algorithm for cleaning huge data set concerning the expected data.
- To build unit test cases for data extraction and data cleaning algorithms.

Crawler Algorithm

Proof of Concept- Crawler

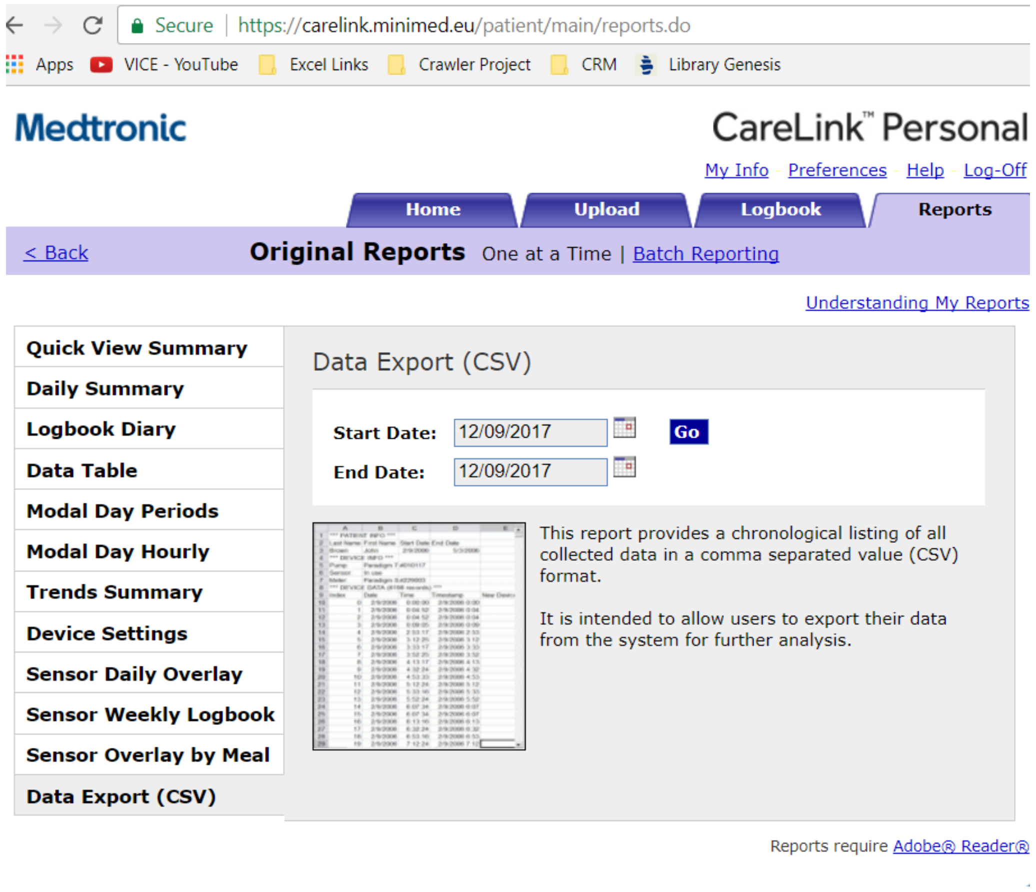
Carelink Website:

▪This Web-based system is designed to help you take information from all of your diabetes management tools – your insulin pump, continuous glucose monitor, blood glucose meter(s), and logbook – and organize it into easy-to-read charts, graphs and tables.

▪These reports can help you and your healthcare provider discover trends and other information that can lead to improved therapy management for greater control.

https://carelink.minimed.eu/

Proof of Concept- Crawler



Secure | https://carelink.minimed.eu/patient/main/reports.do

Apps | VICE - YouTube | Excel Links | Crawler Project | CRM | Library Genesis

Medtronic — CareLink™ Personal

My Info | Preferences | Help | Log-Off

Home | Upload | Logbook | Reports

< Back — **Original Reports** One at a Time | Batch Reporting

Understanding My Reports

- Quick View Summary
- Daily Summary
- Logbook Diary
- Data Table
- Modal Day Periods
- Modal Day Hourly
- Trends Summary
- Device Settings
- Sensor Daily Overlay
- Sensor Weekly Logbook
- Sensor Overlay by Meal
- Data Export (CSV)

Data Export (CSV)

Start Date:	12/09/2017
End Date:	12/09/2017

Go

This report provides a chronological listing of all collected data in a comma separated value (CSV) format.

It is intended to allow users to export their data from the system for further analysis.

Reports require Adobe® Reader®

- Report page in browser from where report has to be fetched is shown in screenshot.
- To reach to this Page, User has to Login into Website.
- We need Program logic to bypass the User Agent which will use Cookies to keep Session and download in bulk.

https://carelink.minimed.eu/

Crawler – Technologies used

Programming Language and Libraries:

- Programming Language used is Java
- As programming language is decided, which all libraries in Java will help with Crawling

Library	Short Description
Jsoup	Jsoup is a Java library working with real-world HTML
Jaunt API	Jaunt is a new, free, Java library for web-scraping & web-automation, including JSON querying.
HtmlCleaner	HtmlCleaner is an open source HTML parser written in Java.
Jtidy	JTidy is a Java port of HTML Tidy, a HTML syntax checker and pretty printer. Like its non-Java cousin, JTidy can be used as a tool for cleaning up malformed and faulty HTML.
NekoHTML	NekoHTML is a simple HTML scanner and tag balancer that enables application programmers to parse HTML
TagSoup	TagSoup is a library for parsing HTML/XML

Crawler – Jsoup

Jsoup:

- Jsoup is a Java library working with real-world HTML.
- It provides a very convenient API for extracting and manipulating data.
- Jsoup implements the WhatWG HTML5 specification.
- It parses the HTML to the same DOM as modern browsers do.
    - scrape and parse HTML from a URL, file, or string
    - Find_and extract data, using DOM traversal or CSS selectors
    - Manipulate the HTML elements, attributes, and text
    - clean user-submitted content against a safe white-list to prevent XSS attacks
    - output tidy HTML
- It does have good online community support.

https://jsoup.org/

Proof of Concept- Crawler

Login Functionality:

- First step is to login programmatically

```
Connection.Response res =
Jsoup.connect("https://carelink.minimed.eu/patient/j_security_check").data("j_username",
username).data("j_password",password).method(Connection.Method.POST).execute();
 loginCookies = res.cookies();
```

- j_username and j_password are the field names deducted from inspecting Dom object.
- With the help of these filed names and Jsoup login is possible.
- Cookies will be saved which will further used to manipulate dates for Csv report.

Proof of Concept- Crawler

Date validation for CSV:

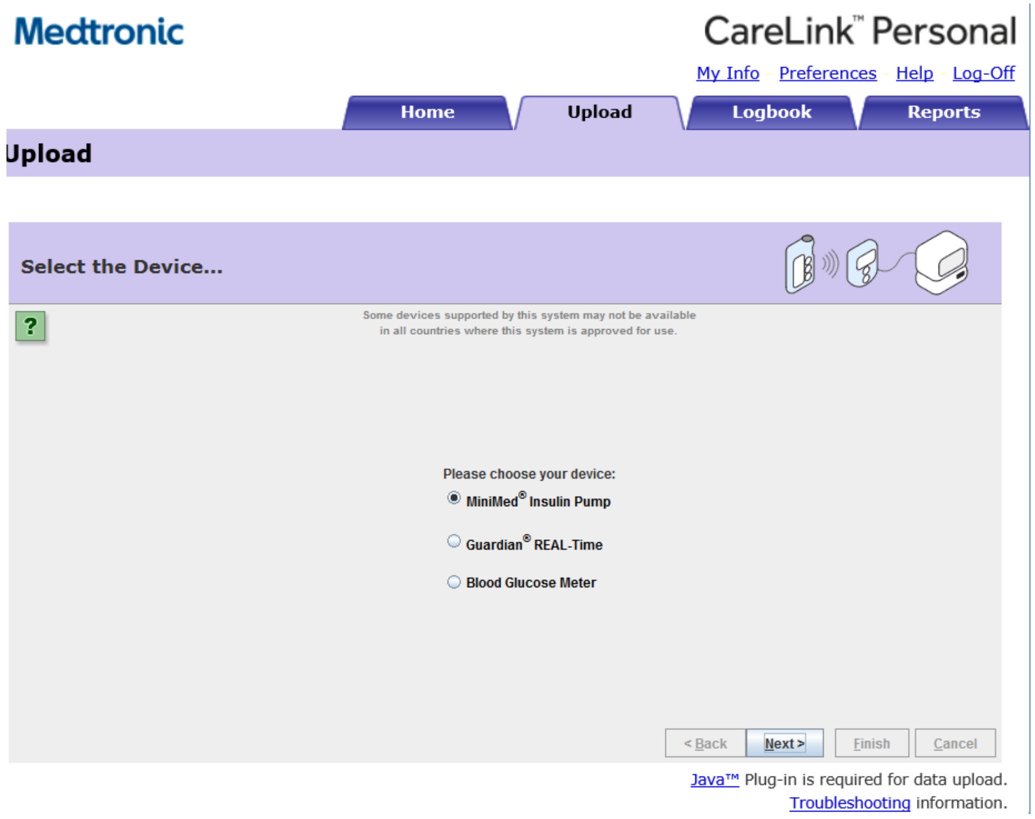
- Date format should be DD/MM/YYYY for user with English as base language and DD.MM.YYYY as German a base language.
- In our current work we have restricted our support to users using only English and German language.
- Start date and end date should not be before 01/01/1998 (01.01.1998)
- The start date should not be greater than end date.
- Start date and end date shall not be greater than Today's date.

```
Connection.Response ReportDocument =
Jsoup.connect("https://carelink.minimed.eu/patient/main/selectCSV.do?t=11?t=11?t=11?t=11")
.timeout(60000).cookies(loginCookies).data("report", "11").data("listSeparator", ",")
.data("datePicker2", startDate) // start date
.data("datePicker1", endDate) // End date
.method(Connection.Method.GET).execute();
```

Applet Wrapper Algorithm

Applet Wrapper

- After successful Poc of CSV download, Next step was to upload data to Crawler website.



Medtronic CareLink™ Personal

My Info Preferences Help Log-Off

Home | Upload | Logbook | Reports

Upload

Select the Device...

Some devices supported by this system may not be available in all countries where this system is approved for use.

Please choose your device:

[✓ MiniMed® Insulin Pump]

Guardian® REAL-Time

Blood Glucose Meter

< Back | Next > | Finish | Cancel

Java™ Plug-in is required for data upload.
Troubleshooting information.

- Upload section in Carelink website has been implemented with Applet.
- Intital Idea of this task was to take the applet out of browser, i.e. to run applet with login cookies generated with Jsoup.
- But We faced technical difficulties here, using Cookies for Applet.
- The applet uses its own cookies generated for browser.
- This is technical roadblock and hence we came up with idea of simulation

https://docs.oracle.com/javase/tutorial/deployment/applet/index.html

Applet Wrapper-Browser automation Tools

- Moving forward We implemented the Concept of browser automation.
- There are numerous browser automation tools available such as
  - Kantu
  - QF-Test
  - Sahi
  - SOAtest:
  - iMacros
  - Selenium

Applet Wrapper-Selenium

- Selenium supports Ruby, Java, NodeJS, PHP, Perl, Python, C#, Groovy as the scripting language.
- The web driver provided by Selenium for Internet Explorer is very reliable, sophisticated .
- It is Open source.
- One most important constraint is with recent browser upgrades except Internet Explorer, none other supports Applet.
- This was also a major motivation to choose Selenium.

Proof of concept-Applet Wrapper

- To run Selenium in IE browser, Selenium IE web drivers need to be installed on machines.
- This created an added dependency of installing Webdriver prior hand.
- We bypassed this dependency with adding Iewebdriver.exe into our project folder.
- If a user runs our Java program as a jar file, it is added into Jar file as well.
- Hence this will reduce one step of dependency.

```
System.setProperty("webdriver.ie.driver", fileWhereIEDriverislocated.getAbsolutePath());
driver = new InternetExplorerDriver(capabilities);
driver.manage().window().maximize();
driver.get("https://carelink.minimed.eu/patient/entry.jsp?bhcp=1");
```

Proof of concept-Applet Wrapper

Login:

- Using selenium and it‘s driver properties login is performed and theron login page appears
- During each operation with selenium we have an waiting time of 2-3 seconds because sometimes it takes time to load a page.
- Below is the code snippet for entering login details and

```
driver.findElement(By.id("j_username")).sendKeys(loginName);
driver.findElement(By.id("j_password")).sendKeys(loginPassword);
driver.findElement(By.id("j_password")).sendKeys(Keys.ENTER);
```

- In a similar fashion upload section is clicked with Selenium

```
driver.findElement(By.id("upload")).sendKeys(Keys.ENTER);
```

Proof of concept-Applet Wrapper

Upload:

- Once the upload page is clicked, Applet will appear.
- DoM object manipulation is not possible here as it appears static.
- Hence we user Robot class in Java to simulate user kepypressed such as ALT+N(Next) Alt+B(back) and so on.
- Which Key combination to press depends on user language chosen.
- Depending on what logic needs to be performed, buttons are clicked and theron upload functionality can be completed.

Applet Wrapper- Validating options

- What all options are available:
  - #Device for pump communication. Available options are: stick, bgdevice
  - #Select Pump Type. Available options are: Minimed, Paradigm
  - #Serial number of the pump you upload data from. 10 characters for Minimed 600 pumps. 6 characters for Paradgim pumps
- Depending on which device selected, combination of Keys to simulate user changes as well.
- Validation also needs to be done for serial number entered.
- All the three options are needed to run Applet wrapper.

Combining Crawler and Wrapper

Combining Crawler and Wrapper

- Two independent modules were successfully created, but how to club them in a framework
- We came up with an Idea of using Config file.
- What will config file contain:
  - User name and password (Ofcourse hashed ☺)
  - Path to save CSV data (optional)
  - Different options for Applet wrapper

Options to run a Java program

- Let us first look at how one would run this Java program with different combinations together
- We have used Commons CLI (Apache Commons) external libraries.

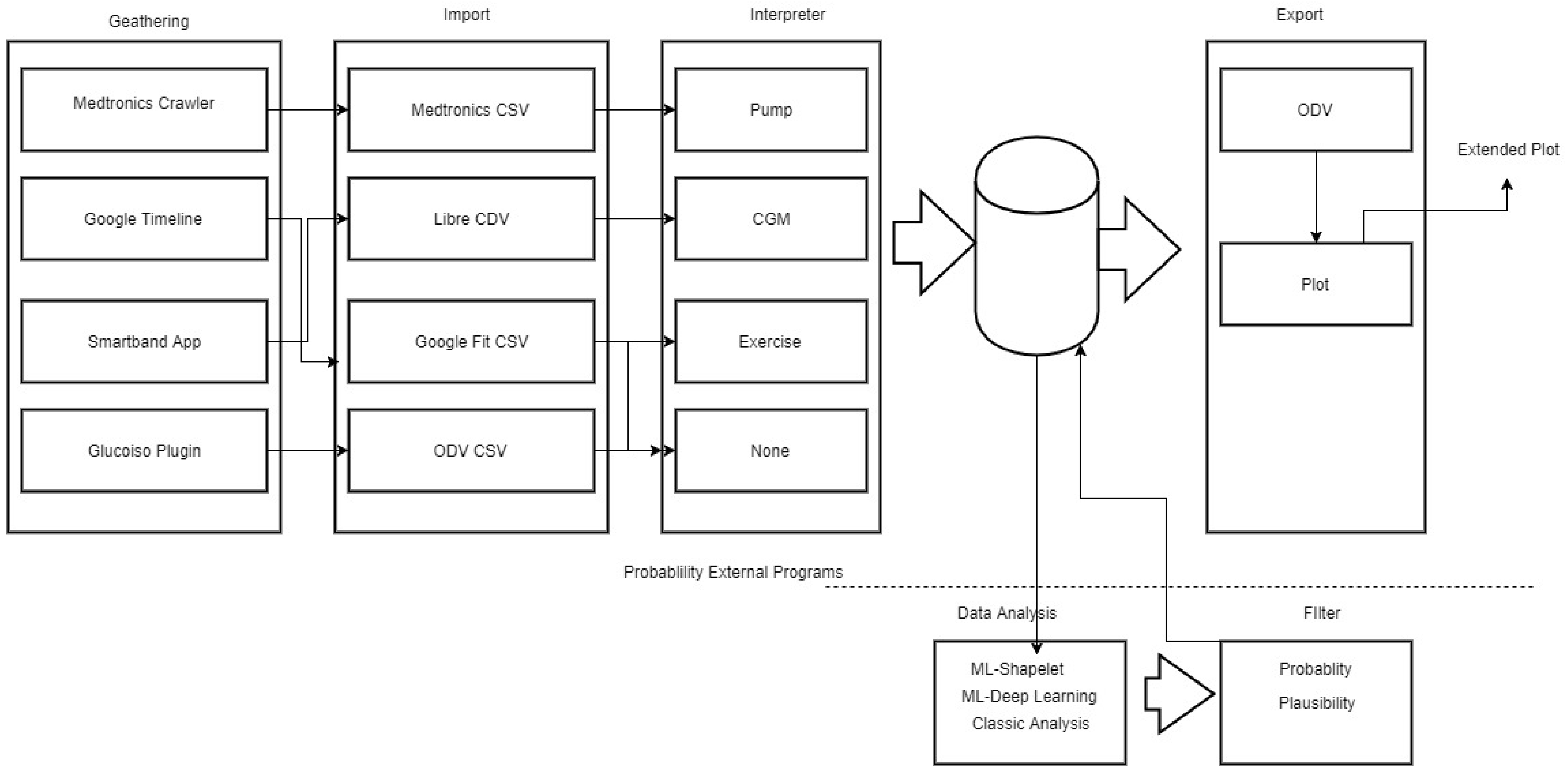
```
Version: Carelink Crawler v0.1
Options:   -v,--version      show program's version number and exit
  -h, --help        show this help message and exit
  -i, --init FILE initializes a new config file at the given path.
  -c, --config FILE defines the used config file.
  -o,--output-path FILE defines output path (Default is ./).
  -crawler starts in crawler mode. -from and -to is required.
  -from defines start time point for the dataset.
  -to defines end time point for the dataset.
  -u, --upload starts in upload mode.
  Combination to initilize a program : java -jar XYZ.jar -i,--init
  Combination to run upload program : java -jar XYZ.jar -c or --config completepathwithfilename  -u,--upload
  Combination to run crawler program : java -jar XYZ.jar -c,--config completepathwithfilename -crawler -from 15/05/2017 -to
20/06/2017 --output outputfolderpath
Date Format:
English: DD/MM/YYYY
German: DD.MM.YYYY
```

Config file Example

```
#Username for access on http://carelink.minimed.eu
UserName: chikhalehero
#Password for access on http://carelink.minimed.eu
Password: qdoKzKHZndaYFc3O1/Z7/g==:M4h2mVYzeivqJMwaU9xeaw==
#Device for pump communication. Available options are: stick, bgdevice
Device: stick
#Select Pump Type. Available options are: Minimed, Paradigm
Pump: Minimed
#Serial number of the pump you upload data from. 10 characters for Minimed 600 pumps. 6
characters for Paradgim pumps
SN: 1234567890                    #SN Number should be of 10 characters (alpha numeric only)
```

Data slicer Algorithm

OpenVault Architecture



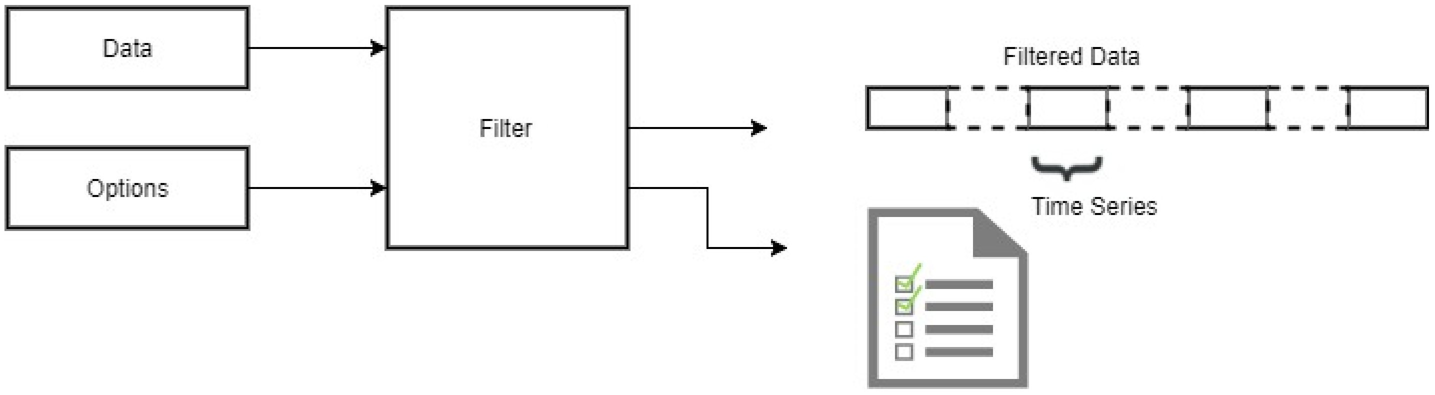
https://github.com/OpenDiabetes/OpenDiabetesVault-engine/wiki/Architecture

Data Filtering

- For data filtering we provide the Filter interface.

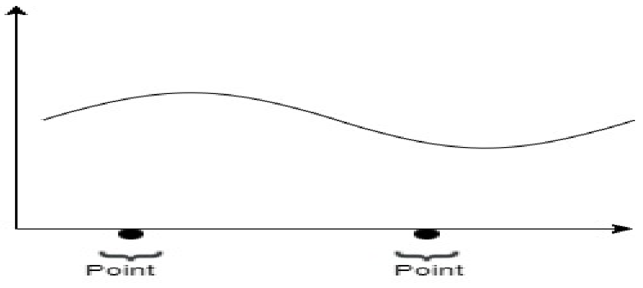
- Filters can be registered at processing services like the DataSlicer and have to implement the Filter interface.

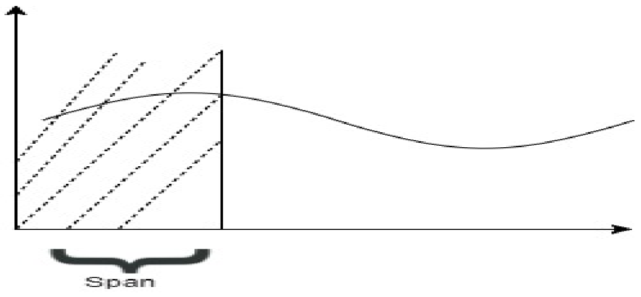
- Multiple filters are combined with the logical "and" paradigm.



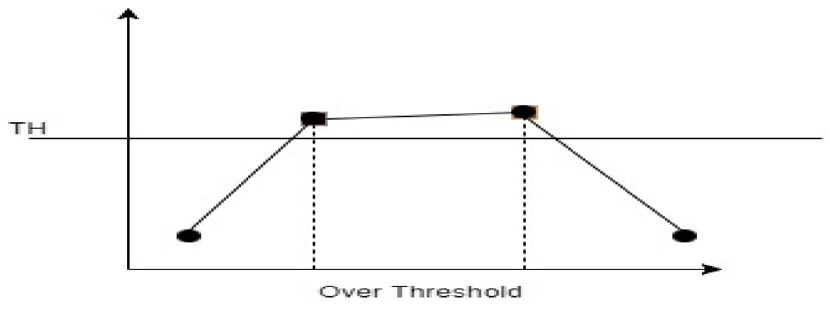
Data Filtering- Types

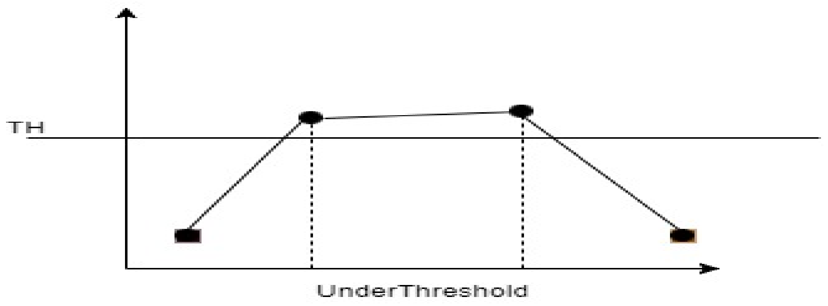
- **Time Related Filter:** This kind of filters is used to find a time point or a time span where some kind of event happened or some kind of data is available or not available. For instance, if you want to check the basal insulin rate you search for time spans where no bolus is given. Filters of the time point type should provide a margin option enable data observation around the





- **Threshold-Related Filter:** This kind of filter are used to find data over or under a certain threshold.





Data Filtering- Types

- The data set used for Filtering data was in below format

```
vaultEntries.add(new VaultEntry(VaultEntryType.GLUCOSE_BG, TimestampUtils.createCleanTimestamp("2017.06.29-04:53", "yyyy.MM.dd-HH:mm"), 109.0, tmpAnnotations));
vaultEntries.add(new VaultEntry(VaultEntryType.STRESS, TimestampUtils.createCleanTimestamp("2017.06.29-04:56", "yyyy.MM.dd-HH:mm"), 36.25));
vaultEntries.add(new VaultEntry(VaultEntryType.HEART_RATE, TimestampUtils.createCleanTimestamp("2017.06.29-04:56", "yyyy.MM.dd-HH:mm"), 72.0));
vaultEntries.add(new VaultEntry(VaultEntryType.SLEEP_LIGHT, TimestampUtils.createCleanTimestamp("2017.06.29-04:58", "yyyy.MM.dd-HH:mm"), 24.0));
vaultEntries.add(new VaultEntry(VaultEntryType.BASAL_PROFILE, TimestampUtils.createCleanTimestamp("2017.06.29-05:00", "yyyy.MM.dd-HH:mm"), 1.05));
```

- For TimeSpan Filter, Program will take start time and end time as parameter, with dataset and will produce all the entires matching the criteria.

```
LocalTime startTime = LocalTime.parse("04:46:00");
LocalTime endTime = LocalTime.parse("09:46:00");
Filter filter = new TimeSpanFilter(startTime, endTime);
```

https://github.com/OpenDiabetes/OpenDiabetesVault-engine/wiki/Data-Filtering

Data Filtering- Types

- Similarly for timepoint a particular timepoint is given with marginInMinutes, which will create a time frame of few points and will return timepoints matching those timepoints.

```
LocalTime startTime = LocalTime.parse("04:46:00");
 Filter filter = new TimePointFilter(startTime, 12);

public TimePointFilter(LocalTime timePoint, int marginInMinutes) {
      LocalTime startTime = timePoint.minus(marginInMinutes, ChronoUnit.MINUTES);
      LocalTime endTime = timePoint.plus(marginInMinutes, ChronoUnit.MINUTES);
      filter = new TimeSpanFilter(startTime, endTime);
   }
```

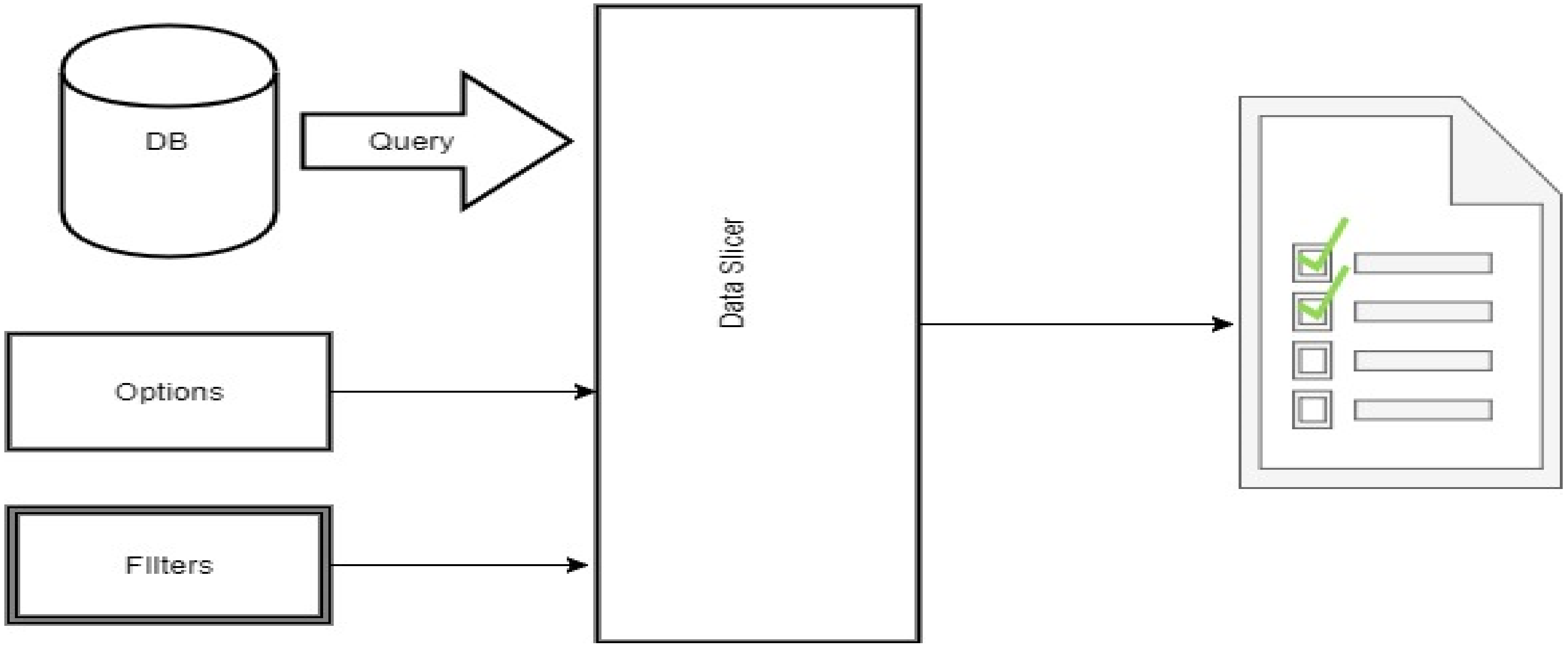
- For Overhead and underhead Threshold, Program expects inupt as below

```
Filter filter = new OverThresholdFilter(VaultEntryType.BASAL_PROFILE, 1.00, FilterType.BASAL_AVAILABLE,
FilterType.BASAL_TH);
```

Data Slicer

TECHNISCHE UNIVERSITÄT DARMSTADT

- For most experiments we have a certain focus on what kind of data we need and how the quality has to be. To get this focused data we provide the DataSlicer.



https://github.com/OpenDiabetes/OpenDiabetesVault-engine/wiki/Data-Slicing

Data Slicer- Implementation

- Data slicer expects result in option value with Timestamp from a bunch of a filtered data.
- To get the sliced data, program would expect parameters as option and Which Timestamp is needed from a bunch of results i.e. First of series or Mid of series of End of series.

```
 Filter filter = new TimePointFilter(startTime, 12);
DataSlicerOptions options = new DataSlicerOptions(60, FIRST_OF_SERIES);
DataSlicer instance = new DataSlicer(options);
        instance.registerFilter(filter);
switch (options.outputFilter) {
 case FIRST_OF_SERIES: tmpTimestamp = item.getKey();
  break;
  case MID_OF_SERIES: tmpTimestamp = generateMidPoint(lastResult, new Date((item.getValue().getTime() + item.getKey().getTime()) / 2));
break;
case END_OF_SERIES:  tmpTimestamp = item.getValue();
break;
default:
throw new AssertionError("Unknown output filter");
}
```

Unit Testing

Unit Testing

- With respect to thesis work, The entire project is divided in three main modules as Crawler, Uploader and Data slicer.
- The plan for setting unit testing is bifurcated with respect to module.
- Adding unit testing has not only helped us understanding modules more in depth but also significantly improved the code quality.
- Especially in case of Data slicing algorithm, we were not exactly sure before had how we need the output to appear. But using sample data set and writing unit test cases for it helped to visualize, what output we would achieve.

Conclusion

- The framework in this thesis is basically built to solve problem of generating huge amount of medical data at bulk without repetitive human efforts.
- Generating data from website posses challenges such as parsing HTML, checking login user and user's preferred language.
- Depending on individual parameters, logic has been implemented.
- The generated data posses challenge to extract exact information needed using different options and parameters hence Data slicers and filters are also implemented.
- The current Thesis work focuses on extracting and uploading data from/to one particular website. In future if the need to generate data from even more multiple sources, this thesis work can be very well expanded to encapsulate different such sources.

Questions?