**Bachelor’s thesis**

TuberXpert

****

|  |  |
| --- | --- |
| **Student :** | **Melvyn Herzig** |
| **Work proposed by :** | Professor Yann Thoma  REDS  Route de Chesaux 1  1401 Yverdon-les-Bains |
| **Responsible teacher :** | **Professor Yann Thoma** |
| **Academic year :** | 2021-2022 |

Yverdon-les-Bains on Sunday, February 24, 2022

**TuberXpert specifications**

**Context**

This work takes place in the domain of therapeutic drug monitoring (TDM). Currently, Tanzania is experiencing a high amount of tuberculosis (TB) cases. The country has expressed its commitment to end TB by 2035. The main tool to achieve this goal is TDM. However, the problem is that it is a long and difficult process. A software called Tucuxi has already been developed and is part of the solution. Nevertheless, it does not solve the fact that the tool needs TDM professionals to be used efficiently. On this perspective, the purpose of this work is to develop a clinical decision support system (CDSS) on top of Tucuxi computation core to automate TDM decisions making.

**Objective**

By the 29nth of July, an extensible CDSS must be developed, tested, and documented. The CDSS will use a local version of Tucuxi computation core for dosage prediction and adjustment computations. The system will be a command line interface that will produce a dosage adjustment report based on the received inputs.

**Features**

**Input validation**

* The program will receive an XML file similar to Tucuxi computation core.
  + The XML structure may be extended with new useful elements if needed.
* The program will analyze, and check data relevance.

**Drug file selection**

* The program must be able to select a relevant drug file for each drug in input.

**Dosage adjustment**

* The program must be able to understand the current state of a treatment and suggest an adjustment.

**Output**

* The output of the decisions must be an XML file that can be used by various templates for report generation.

**Report generation**

* The program must summarize all useful information in a well formatted report.
  + Suspicious covariates.
  + Drug file selected.
  + Graph (A priori or A posteriori, depending on patient).
  + Dosage adjustment.
  + …

**Multi language**

* The program must support various language.
* At least, English must be available.
  + It should be easy to add a translation and use it.

**Testing**

* The program behavior must be tested with various inputs.
  + Since it is difficult to predict all cases, obvious cases testing is sufficient.

**Deadlines**

|  |  |
| --- | --- |
| When | What |
| 16.05.2022 | Intermediate report. |
| 29.07.2022 12:00 | Upload final report, poster, and publishable summary on Gaps. |
| 29.07.2022 12:00 | Notify responsible and secretariat by email when previous point is done. |
| 22.08.2022 – 16.09.2022 | Bachelor thesis defense |

**Table of contents**

[1 Preamble 6](#_Toc103354750)

[2 Introduction 7](#_Toc103354751)

[2.1 What is therapeutic drug monitoring 7](#_Toc103354752)

[2.2 Current situation in Tanzania 7](#_Toc103354753)

[2.3 Goal of this work 8](#_Toc103354754)

[3 Software needs 9](#_Toc103354755)

[3.1 Requirements 9](#_Toc103354756)

[3.1.1 Functional 9](#_Toc103354757)

[3.1.2 Nonfunctional 9](#_Toc103354758)

[3.2 Testing 9](#_Toc103354759)

[3.3 Architecture 9](#_Toc103354760)

[4 Analysis 10](#_Toc103354761)

[4.1 Global application overview 10](#_Toc103354762)

[4.2 Technologies 10](#_Toc103354763)

[4.2.1 Development language 10](#_Toc103354764)

[4.2.2 Integrated Development Environment 10](#_Toc103354765)

[4.2.3 Compiler 11](#_Toc103354766)

[4.3 Input 11](#_Toc103354767)

[4.3.1 Global 11](#_Toc103354768)

[4.3.2 Covariates 12](#_Toc103354769)

[4.3.3 Drugs 12](#_Toc103354770)

[4.3.4 Dosage 13](#_Toc103354771)

[4.3.5 Samples 14](#_Toc103354772)

[4.3.6 Target 14](#_Toc103354773)

[4.3.7 Request 15](#_Toc103354774)

[4.4 Program execution flow 16](#_Toc103354775)

[4.4.1 Get best drug file 17](#_Toc103354776)

[4.4.2 Assess the dosages 18](#_Toc103354777)

[4.4.3 Assess the samples 19](#_Toc103354778)

[4.4.4 Assess the targets 20](#_Toc103354779)

[4.4.5 Create adjustment request 21](#_Toc103354780)

[4.5 Output 23](#_Toc103354781)

[4.5.1 Header 23](#_Toc103354782)

[4.5.2 Administrative 24](#_Toc103354783)

[4.5.3 Covariates and checks 25](#_Toc103354784)

[4.5.4 Treatment and checks 26](#_Toc103354785)

[4.5.5 Samples and check 27](#_Toc103354786)

[4.5.6 Adjustments 28](#_Toc103354787)

[4.5.7 Computation facts 32](#_Toc103354788)

[5 Tests 32](#_Toc103354789)

[5.1 Unit Tests 32](#_Toc103354790)

[5.1.1 LanguageManager 32](#_Toc103354791)

[5.1.2 XpertQueryImort 32](#_Toc103354792)

[6 Conclusion 35](#_Toc103354793)

[7 Bibliography 36](#_Toc103354794)

[8 Authentication 37](#_Toc103354795)

[9 Planification 38](#_Toc103354796)

[9.1 Initial 38](#_Toc103354797)

[9.2 Intermediate 39](#_Toc103354798)

[9.3 Final 40](#_Toc103354799)

**Department:** TIC

**Faculty:** Computer science

**Specialization:** Software

**Student:** Melvyn Herzig

**Responsible teacher:** Prof. Yann Thoma

Bachelor’s thesis 2021-2022

TuberXpert

**Publishable summary**

TODO

|  |  |  |
| --- | --- | --- |
| Student :  Herzig Melvyn | Date and time :  …………………………………… | Signature :  …………………………………… |
| Responsible teacher :  Thoma Yann | Date and time :  …………………………………… | Signature :  …………………………………… |

# Preamble

This Bachelor's thesis (hereinafter referred to as "TB") is produced at the end of the course of study, with a view to obtaining the title of Bachelor of Science HES-SO in Engineering.

As an academic work, its content, without prejudging its value, does not engage the responsibility of the author, nor that of the jury of the Bachelor's thesis and of the school.

Any use, even partial, of this TB must be made in compliance with the copyright.

HEIG-VD

The head of the department

Yverdon-les-Bains on Friday, March 4, 2022

# Introduction

## What is therapeutic drug monitoring

Nowadays, many drugs or antibiotics are used to treat diseases such as tuberculosis and HIV. Usually, the doctors prescribe generic doses that are suitable for the general population. Unfortunately, everyone’s metabolism reacts differently which makes generic dosages often ineffective.

Some people will have insufficient circulating drug exposure caused by an underdose. Thus, the treatment will be ineffective, and the patient may become drug resistant. Conversely, an overdose may result in intoxication. This would force an interruption of the treatment in order not to worsen the patient's health.

To avoid such situations, therapeutic drug monitoring (TDM) has been developed. TDM is a precision medicine that prescribes a personalized dosage to each patient based on the monitoring of the evolution of the drug concentration in the blood.



Figure 1 - Dosage Scheme (BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document])

The purpose of TDM is to measure the concentration of the drug regularly and accurately in order to know its evolution. After examination, an expert in pharmacology can determine how to adjust the dosage to be above the threshold of inefficiency and below the threshold of toxicity.

## Current situation in Tanzania

Tanzania has a high burden of Tuberculosis (TB). Over the last decade, Tanzanian health authorities estimate an incidence of 120’000 – 150’000 patients per year for TB. The global community, through the END TB strategy, has declared its willingness to end TB by 2035, and a vital component of the arsenal for this includes resorting to the correct use of anti-TB drugs, in particular the first-line agents: isoniazid, rifampicin, ethambutol, and pyrazinamide.

However, the problem is that only 60’000 – 75’000 patients are notified and receive a treatment. In addition, studies have reported that rifampicin dosages are insufficient most of the time. For example, investigations by Heysell et al. (2011) and Tostmann et al. (2013) in the Kilimanjaro region showed that one to two thirds of uncomplicated TB patients had maximum concentrations below the reference range of 8-24 mg/L, defined two hours after last dose intake.

Moreover, a considerable proportion of individuals with TB are coinfected with HIV. It represents 25 to 40% of the monitored people. Administering antiretroviral drugs with first line antitubercular drugs lower furthermore the rifampicin concentration.

On top of that, TB patients have an increased risk to get affected by diabetes mellitus (DM). It represents 4-16% of the TB population. Unluckily, DM may alter the pharmacokinetics (PK) of various drugs which include antitubercular. Mtabho et al. observed that DM predicted low levels of rifampicin in TB Tanzanian patients. Sadly, evidence have shown that individuals with TB and DM have a 5-fold risk of death compared to those without DM.

At this point, we easily understand that the risk of treatment failure or unfavorable outcome is real if the dosages stay unsuitable.

## Goal of this work

The need to end tuberculosis is real and urgent in Tanzania. Unfortunately, TDM is a long and complicated process. In addition, the number of experienced pharmacologists is not sufficient to provide well established interpretation and recommendation everywhere their expertise is needed.

Currently, Professor Yann Thoma and his team have developed Tucuxi. It is a software intended for the practice of TDM. Already developed for several years, the software offers many features:

* Drug concentration predictions based on population and patient data (covariates) as well as on previous measurements.
* Suggestion of dosage adjustments to reach an optimal drug concentration state.
* Generation of printable reports.
* Integration with Electronics Health Record (EHR) systems.

Although it simplifies the TDM process, Tucuxi is intended for experienced pharmacologists.

TuberXpert comes at the beginning of a large project between Switzerland and Tanzania led by Prof. Thoma Yann, Prof. Mpagama Stellah, and Prof. Mpagama Stellah. The aim is to develop a Clinical Decision Support System (CDSS) to fight tuberculosis. Thus, TuberXpert is a software layer that adds to the existing Tucuxi computing core. By receiving complete information from a patient, the system assesses the relevance of the data provided and then determines whether an adjustment of dosage is necessary. All interpretations made by the program will then be provided to the user in the form of a simplified report compared to the original software. The main purpose of TuberXpert is to simplify the “Interpretation and recommendation” phase of TDM for non-experts

Figure 2 -TDM process (BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document])

In other words, TuberXpert is a turnkey solution for TDM. The software developed during this Bachelor's thesis is a first step. It will then be taken over by three PhD students in charge of the development and the concrete application of the project.

# Software needs

This chapter presents a specification of what is to be implemented. It describes how the system should behave and what its properties are.

## Requirements

### Functional

**From the user’s point of view**: TuberXpert is an automated clinical decision support system.

It takes as input a query: some information about a patient and a request. Then, thanks to an output report, the program suggests an initial dosage or an adjustment of the drugs and any useful information for the therapeutic drug monitoring. The output contains readable sentences, and the report displays graphs of the adjustment.

**From the developer’s point of view**: TuberXpert is an additional software layer on top of Tucuxi computing core.

As input, the TuberXpert needs an XML file containing the patient’s information and the request to be executed. The structure of the file is the same as that of Tucuxi computing core. However, it could extend the basic structure by adding some new elements, at least, a new type of request dedicated to TuberXpert control.

The first step in the execution is to obtain a relevant drug file. TuberXpert does not ask the user to specify which drug model to use. It chooses to most appropriate one based on the available patient information

Then, the program performs a validity check on the data provided. It assesses whether the values received are plausible. Are they normal ? Did the practitioner make a typing error ?

After the validation step, TuberXpert prepares an adjustment request that it submits to Tucuxi computing core.

Finally, it generates an output report corresponding to a format requested by the user. This can be XML, HTML or PDF. The report contains full sentences and the adjustment graphs for HTML and PDF or the adjustment data to generate a graph for XML. The report can be generated in several languages.

### Nonfunctional

This project is most likely a proof of concept. For the purpose of this work, a graphical user interface is not be developed. The program will be run in a command line interface.

As long as it is relevant, the generated report must be good looking and user friendly. In other words, it should not be painful to read and extract the information.

Although this work is made in the context of tuberculosis, it is developed without extensive knowledge of rifampicin. Consequently, the development is generic in considering all drug. Therefore, it should be easy to edit the parts specific to the adaptation of a drug

## Testing

The program behavior must be evaluated with various inputs. Since it is difficult to predict all cases, obvious cases testing is enough.

## Architecture

In terms of software architecture, the clinical decision support system (CDSS) may be separated from the report generation. The CDSS must offer a standardized output (most probably in XML format). This output may be used by some report generator to fill some fields of a report template.

# Analysis

This chapter presents the analysis that done prior to the implementation. The objective is to anticipate and take some decisions based on business and technical analysis.

## Global application overview

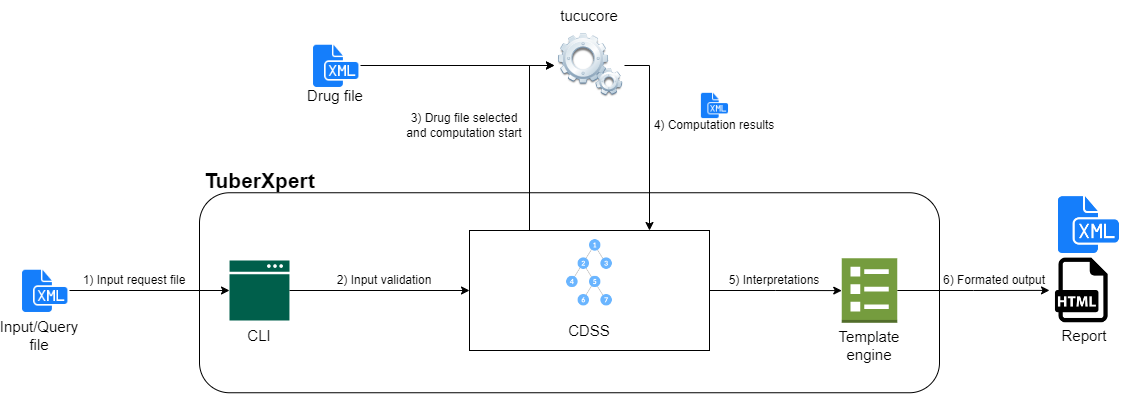


Figure 3 Global application overview and components

1. The program receives as input a query in XML format. For simplification purpose, the XML format will be the same used by Tucuxi computing core. However, it will be extended with a specific request and administrative elements
2. The program loads the data from the query file and transmit them to the CDSS that will determine if they are consistent and relevant.
3. The CDSS will determine the best drug model to be used and starts the computation.
4. The computations results are transmitted as XML to the CDSS which analyses them.
5. The decisions are passed to a template engine that format the information into the requested format.
6. The report is returned.

## Technologies

### Development language

Initially, Tucuxi was developed in C++.This language was chosen for its superior performance since the software requires a lot of computing performance. Then, other projects were added. Most of them were also developed in C++ for the same reasons.

Thus, to preserve the homogeneity of the projects, TuberXpert will also be developed in C++. This language is once again very advantageous. As the software will be used on low-powered computers in Tanzania, the performance will be optimized by using a low-level language.

Version: C++ 17

### Integrated Development Environment

Once again, it will walk in the steps of Tucuxi and will use the same development environment . QT is a very advantageous and rich IDE. It easily allows cross-platform development, test management, language management and many other frameworks. QT is very versatile. It also makes it easier to work with existing projects at the same time. De facto, it is the IDE that will be used to develop this project.

Version: QT 6.0.2 community

### Compiler

To simplify the development environment installation, this project is compiled with the compiler integrated with QT.

Version (Windows): MinGW-W64-builds-5.0.0

## Input

The TuberXpert query XML file is responsible for providing all the useful information about the patient, his treatments, and the computation to be performed. Most of its structure is essentially the same as Tucuxi computing core[[1]](#footnote-1).

Although, this section briefly goes through the input structure, it does not explain how to form the common elements with Tucuxi computing core but what they represent and, if necessary, what is checked.

When a pertinent spot it reached, it explains what additional element needs to be added to fit TuberXpert needs such as administrative data and the custom request.

### Global

Down below is the overall structure of the input. It consists of information about the query: queryId, clientId, date and language. Then there is information about the patient’s covariates followed by the drugs he is taking. Finally, comes the requests element. It contains request elements used to tell Tucuxi computing core what calculation to perform from what data, but it is ignored by TuberXpert since it will create these requests itself. However, it will include a new type: the new TuberXpert request requestXpert.

**Example of the global structure of the input**:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<query version="1.0"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:noNamespaceSchemaLocation="computing\_query.xsd">

    <queryId>imatinib\_2</queryId>

    <clientId>124568</clientId>

    <date>2018-07-11T13:45:30</date>

    <language>en</language>

**<****admin> […] </admin>**

    <drugTreatment>

        <patient>

            <covariates> […] </covariates>

        </patient>

        <drugs> […] </drugs>

    </drugTreatment>

    <requests> […]</requests>

</query>

At this point, the noticeable addition is the admin element. This new element should contain all administrative information such as:

* Patient and adjustment mandator:
  + Title, first name, last name.
  + Address.
  + Phone and email.
* Institute of the patient and the adjustment mandator:
  + Name.
  + Address.
  + Phone and email.
* Miscellaneous clinical data

This administrative information will be used to find out who the patient is, who the adjustment mandator is and how can we contact them. Finally, it will be displayed in the report generated at the beginning to know who is involved.

**Checks:**

At this stage, no checks are performed.

### Covariates

The covariates element contains the list of the patient’s known covariates defined by an identifier covariateId, a date of measure, a value, a value type (datatype) and a unit.

**Example of covariate:**

<covariates>

    <covariate>

        <covariateId>bodyweight</covariateId>

        <date>2018-07-11T10:45:30</date>

        <value>40</value>

        <unit>kg</unit>

        <dataType>double</dataType>

    </covariate>

    […]

</covariates>

**Checks:**

Covariates values are checked using drug files covariates domain.

*After a drug file is selected, each patient covariate that does not meet the domain of the same covariate in the drug file will generate a warning in the final report.*

### Drugs

The drugs element contains drug elements. It represents the treatments the patient is undergoing. A drug element typically contains: the associated drug identifier (drugId), the name of the active principle (activePrinciple), the manufacturer’s brand name (brandName), the drug atc (Anatomical Therapeutic Chemical), the patient’s treatment, the patient’s blood samples and the targets the patient is trying to achieve.

**Example of drug:**

<drugs>

    <drug>

        <drugId>imatinib</drugId>

        <activePrinciple>something</activePrinciple>

        <brandName>somebrand</brandName>

        <atc>something</atc>

        <treatment></treatment>

        <samples></samples>

        <targets></targets>

    </drug>

    […]  
</drugs>

**Checks:**

The drug identifier must match to at least one drug file.

*If no drug gets at least one drug file, the execution is aborted, and an error is returned.  
If one or more drugs get at least one drug file, the execution continues for those drugs only.*

### Dosage

The dosages are located in the treatment element. It contains the patient’s dosage history for a given drug. A dosage has a start date and an end date. It shows what the patient takes, when and on what basis.

The dosage element is complex but flexible. It allows to describe dosages such that “take a drug at 8:00 every day except on Sunday.” The main point is that it will always contain a dose element that will allow the dosage validation.

**Example of dosage:**

*<treatment>*

*<dosageHistory>*

*<dosageTimeRange>*

*<start>2018-07-06T08:00:00</start>*

*<end>2018-07-08T08:00:00</end>*

*<dosage>*

*[…]*

                <dose>

                    <value>400</value>

                    <unit>mg</unit>

                    <infusionTimeInMinutes>60</infusionTimeInMinutes>

                </dose>

*[…]*

*</dosage>*

*</dosageTimeRange>*

*</dosageHistory>*

    […]

*</treatment>*

**Checks:**

Each dose will be converted to match the unit of the available doses of the drug file. Then, each value will be compared to the domain of the available doses from the drug file.

*If dose reaches the minimum and maximum bounds from the drug file, a warning will be printed in the final report.*

### Samples

The samples element contains the patient’s blood samples. In other words, a list of drug concentration measurements. A sample is defined by an identifier (sampleId), a date of measure (sampleDate) and some concentrations. There are multiple concentrations when the drug contains multiples analytes. Therefore a concentration contains its associated analyte identifier (analyteId), a value, and a unit.

**Example of sample:**

<samples>

    <sample>

        <sampleId>123456</sampleId>

        <sampleDate>2018-07-07T06:00:00</sampleDate>

        <concentrations>

            <concentration>

                <analyteId>imatinib</analyteId>

                <value>0.7</value>

                <unit>mg/l</unit>

            </concentration>

        </concentrations>

    </sample>

    […]

</samples>

**Checks:**

In order to checks the samples, the program will compute aposteriori estimation. Then, it will check if a sample is below or above certain percentile.

*If a sample is below the percentile X or above the percentile Y, the program will print a warning in the final report.*

### Target

The adaptation engine uses targets to adapt the dosage to the patient’s needs. The drug files provide such targets, but they correspond to the typical patient. In other words, those targets are generic and not patient specific. Therefore, it is possible to replace them by providing some in the query file. A target contains the corresponding active moiety (activeMoietyId), a type (targetType), a unit and the thresholds: minimum to reach (min), max to reach, maximum to reach max, inefficiency limit (inefficacyAlarm), and toxicity limit (toxicityAlarm).

**Example of target:**

<targets>

    <target>

        <activeMoietyId>imatinib</activeMoietyId>

        <targetType>residual</targetType>

        <unit>mg/l</unit>

        <min>20</min>

        <best>25</best>

        <max>30</max>

        <inefficacyAlarm>15</inefficacyAlarm>

        <toxicityAlarm>50</toxicityAlarm>

    </target>

    […]

</targets>

**Checks:**

For a personalized target to be valid, it must have the same active moiety id and target type as the drug file, but not identical to another personalized target.

*If a target is invalid, the program will abort and return a specific error.*

### Request

The request element represents the computation that the core must perform. It is the way the user can control Tucuxi computing core. It contains an identifier (requestId) used in the response to identify the answered request, the identifier (drugId) of the associated drug, a drug model to use (drugModelId), and a trait that tells that tells it to compute a prediction, an adjustment, or percentiles.

**Example of request:**

<requests>

**<requestXpert> […] </requestXpert>**

    <request>

        <requestId>aposteriori</requestId>

        <drugId>imatinib</drugId>

        <drugModelId>ch.tucuxi.imatinib.gotta2012</drugModelId>

        <COMPUTING TRAIT COMES HERE>

        </COMPUTING TRAIT COMES HERE>

    </request>

*[…]*

</requests

The last change from the original query format occurs here. TuberXpert introduces its own request element under the name requestXpert. It is not possible to extend the original request with a custom trait because it contains some mandatory fields that should not be filled in by the users of TuberXpert. In the same way that a request controls Tucuxi computing core, a requestXpert controls TuberXpert.

At least, it must contain the following information:

* The identifier of the drug to adjust.
* The computing options like the computing trait of request.
* The type of output
  + Language (EN, FR …)
  + Type (XML, HTML …)
* The type of computation
  + Local
  + Distant (Currently unsupported but in anticipation of future updates)

**Checks:**

On request: no verification is performed. The system ignores them because it will make its own requests.

On requestXpert: it is supposed to respect the XML schema definition. Nevertheless, it will check that the type of output is supported before continuing the execution.

## Program execution flow

This chapter presents the execution steps when running the program. The colors represent the main steps that are detailed in specific subchapters.

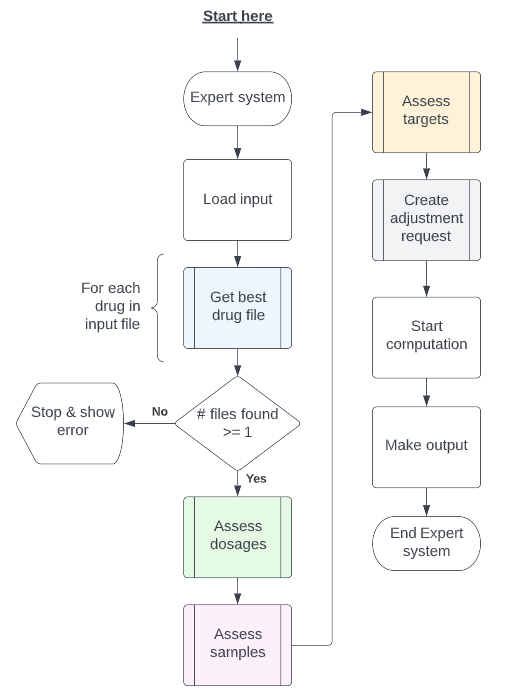


Figure 4 Program global execution

The first step is to load the query input file in order to manipulate it. Then, foreach drug found, it tries to get the best drug file. If the program does not find a drug file for any drug, the execution cannot continue. Otherwise, if a drug file is found for some drugs, it assesses the dosages, samples, and targets. After that, it creates an adjustment request for Tucuxi computing core. When the computation is done, it retrieves an XML response file from the computation core. Finally, it formats all the useful information into the required output.

### Get best drug file

One task of the system is to choose a drug file to use. To do this, a simple heuristic will be implemented.

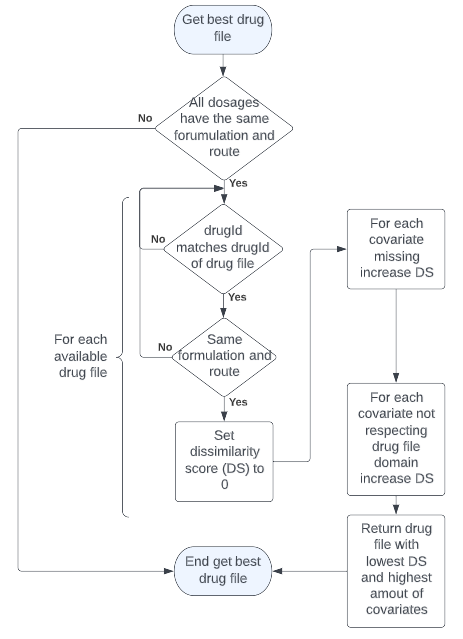


Figure 5 Process of drug files evaluation

**Heuristic behavior**

For each available model of a given drug that correspond to the formulation and route of the dosages, we calculate a dissimilarity score S based on the covariates of the model.

*S = +*

The model with the lowest dissimilarity score is chosen. In case of a tied, the model with the most covariates is chosen.

The method is not optimal, but it is a good starting point. What happens if a covariate is mandatory or if two models tied perfectly, but one may be fitting better? This type of question is not considered by this algorithm. In the future, a close collaboration with doctors will be necessary to determine for each drug “how to choose the drug file that fit the most to the patient”.

*We use the drug file name to find the drug id without loading the entire file. So, at this point, we can say:*

*If the input dosages have the same formulation and route:*

* *Yes: Look for the best drug file*
* *No: Return an error or a warning in the report*

*In regards of the selected drug file, are the covariates supported:*

* *Yes: No problem*
* *No: Generate a warning in the report*

*If there are some potential drug files for each drug of the query:*

* *Yes: Load them*
* *No: Return an error if there are none for each drug, else just an error in the final report.*

### Assess the dosages

To adjust a dosage, it is important that the dosages used to perform the computation are relevant. Thus, after loading the best trug file, the CDSS will check that every dosage of the request matches the dosage domain in the drug file.



Figure 6 Process of dosages assessment

Every drug file has an *availableDoses* element that has a unit and a range [from, to]. This information is used to assess the dosages of the input. The Tucuxi computing core *UnitManager* class converts a value and a unit with a target unit. Thus, the CDSS will be able to convert each dosage and checks if they are in the domain of the drug file.

*At this point, we can say:   
If the dosages are normal, that is, within the normal range:*

* *Yes: No problem*
* *No: Generate a warning in the report*

### Assess the samples

The samples need to be representative of what was really measured. If the samples are wrong, the computation core will compute a wrong adjustment because the samples are not representative of the body’s response to the treatment.

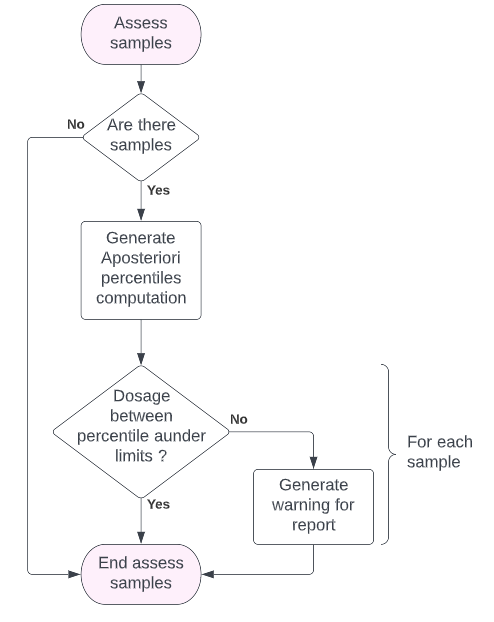


Figure 7 Process of samples assessment

Using the computation core, the system will generate an aposteriori percentile request. Considering the patient covariates, dosages, and samples, it will be possible to determine which percentile the patient is in.

For example, we will use 4 percentiles that will change the level of warning:

* Below 5 or above 95: a major warning.
* Below 10 or above 90: a normal warning.

*At this point, we can say:*

*If the samples are normal, that is, above and below certain percentile:*

* *Yes: No problem*
* *No: Generate a warning in the report*

### Assess the targets

In the input XML file, it is possible to create custom target that override the default targets of the drug file.



Figure 8 Process of targets assessment

Firstly, it checks if two custom targets have the same active moiety and the same target type. In this case, the targets are redundant. Since we cannot choose between these targets, we display an error and stop the adjustment for the relevant drug. Then it checks that the active moiety of the custom target is an active moiety of the drug using the drug file.

*At this point, we can say:*

*If the custom targets are normal, that is, non-redundant and using a good active moiety of the related drug:*

* *Yes: Keep them*
* *No: Stop the relevant drug computation. If some drugs are valid, write an error in report else return an error code.*

*The possibility to write custom target is normally intended for experienced practitioner.*

*There is no absolute rule that is easy to implement to determine what constitutes a relevant target for each drug. As a result, this version of the CDSS does not check for unit, min, max, best, and alarms values.*

*In future versions, specific rules should be implemented for each drug.*

### Create adjustment request

The last step before launching adjustment computation is to take the last decisions to prepare the request for Tucuxi computing core.

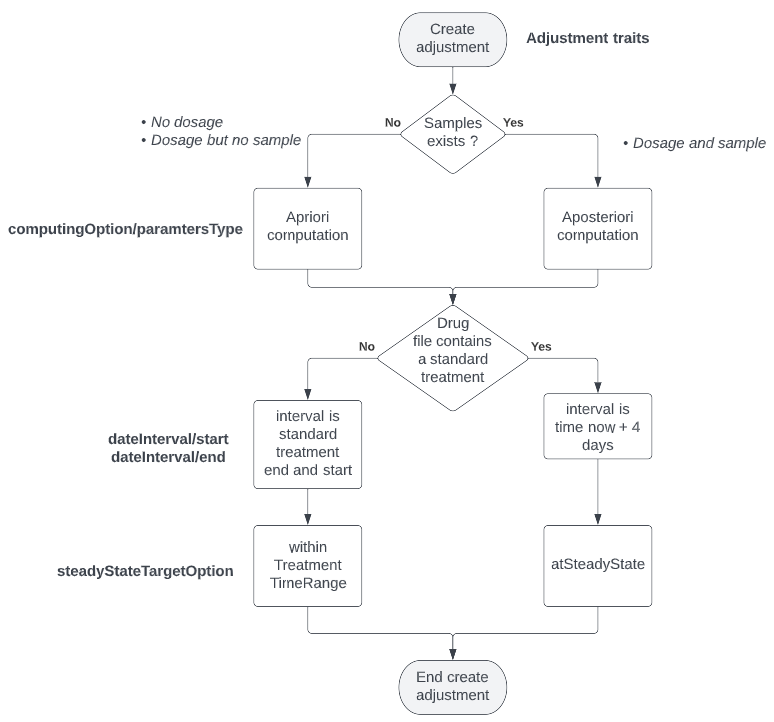


Figure 9 Decisions for adjustment request.

We need to define 3 information:

* What are the parameters of the computing option?
  + If the patient has no dosage or sample, the parameters type is Apriori.
  + If the patient has a dosage and a sample, the parameters type is Aposteriori.
* What is the interval to display?
  + If the drug model does not specify a standard treatment, the interval starts today and ends in 4 days.
  + If the drug model specifies a standard treatment, the interval starts and ends according to the standard treatment interval.
* What is the steady state target option?
  + If the drug model does not specify a standard treatment, the option is *atSteadyState.*
  + If the drug model specifies a standard treatment, the option is *withinTreatmentTimeRange.*

Once the program gets these answers, it can create a request with an adjustment trait. Some elements are forced by TuberXpert, and some others can be tweaked by the user using the TuberXpert custom request.

For example, here is an example of an adjustment request for Tucuxi computing core:

<request>

    <requestId>adjustment</requestId>

    <drugId>rifampicin</drugId>

    <drugModelId>ch.tucuxi.rifampicin.svensson2017.tdd</drugModelId>

    <adjustmentTraits>

        <computingOption>

            <parametersType>aposteriori</parametersType>

            <compartmentOption>allActiveMoieties</compartmentOption>

            <retrieveStatistics>true</retrieveStatistics>

            <retrieveParameters>true</retrieveParameters>

            <retrieveCovariates>true</retrieveCovariates>

        </computingOption>

        <nbPointsPerHour>20</nbPointsPerHour>

        <dateInterval>

            <start>2018-01-12T07:00:00</start>

            <end>2018-03-15T12:59:00</end>

        </dateInterval>

        <adjustmentDate>2018-01-12T07:00:00</adjustmentDate>

        <options>

            <bestCandidatesOption>bestDosagePerInterval</bestCandidatesOption>

            <loadingOption>noLoadingDose</loadingOption>

            <restPeriodOption>noRestPeriod</restPeriodOption>

            <steadyStateTargetOption>atSteadyState</steadyStateTargetOption>

            <targetExtractionOption>definitionIfNoIndividualTarget

                                                              </targetExtractionOption>

            <formulationAndRouteSelectionOption>lastFormulationAndRoute

                                                  </formulationAndRouteSelectionOption>

        </options>

    </adjustmentTraits>

</request>

The values are defined as follows:

|  |  |
| --- | --- |
| Element | Value inside |
| request | Constant. For example: “adjustment\_” + < associated drug ID>. |
| drugId | Extracted from TuberXpert request element. |
| drugModelId | Retrieved by TuberXpert drug model selection heuristic. |
| parametersType | Decisions presented previously. |
| compartmentOption | Always *allActiveMoieties*. |
| retrieveStatistics | Always *true.* To be displayed in the final report. |
| retrieveParameters | Always *true.* To be displayed in the final report. |
| retrieveCovariates | Always *true.* To be displayed in the final report. |
| nbPointsPerHour | Always *20*. |
| start | Decisions presented previously. |
| end | Decisions presented previously. |
| adjustmentDate | By default, adjust at the next interval or retrieved from TuberXpert request element. |
| bestCandidatesOption | By default, *bestDosagePerInterval* or retrieved from TuberXpert request element. |
| loadingOption | By default, follow drug model recommendation or retrieved from TuberXpert request element. |
| restPeriodOption | By default, follow drug model recommendation or retrieved from TuberXpert request element. |
| steadyStateTargetOption | Decisions presented previously. |
| targetExtractionOption | By default, definitionIfNoIndividualTarget or retrieved from TuberXpert request element. |
| formulationAndRouteSelectionOption | By default, lastFormulationAndRoute or retrieved from TuberXpert request element. |

## Output

This chapter discusses the forms that the output will take. It is expected to be in the form of an XML document or an HTML page. As a first approach to understanding what information need to be displayed, I have produced a first draft of the HTML report page on Figma. From that point, it is possible emphasis what information is necessary and deduce what will be inserted in the XML document.

### Header

This first part contains the generation date as well as the general facts about the drug concerned, such as its identifier and the last dose administered.

**HTML representation:**

Une image contenant table

Description générée automatiquement

**XML possible organization:**

**<generationDate> (date) </generationDate>  
<drug>  
 <drugId> (string) </drugId>  
 <lastDose>  
 <value> (decimal) </value>  
 <unit> (unitType) </unit>  
 </lastDose>  
</drug>**

### Administrative

It contains all administrative data of the mandator, the patient as well as the clinical information. In fact, this part displays every administrative data found in admin element of the input.

**HTML representation:**

Une image contenant table

Description générée automatiquement

**XML possible organization:**

The XML output will follow the same structure as admin element from the input.

For more information on its structure, see the previous chapter “Input”. As a reminder, it is organized as follows:

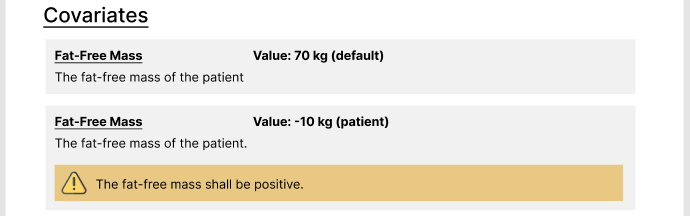
**<admin>  
 <mandator>   
 <person> … </person>  
 <institute> … </institute>  
 </mandator>  
 <patient>  
 <person> … </person>  
 <institute> … </institute>  
 </patient>  
 <clinicalData> … </clinicalData>  
</admin>**

### Covariates and checks

This section lists all the covariates that are needed for the adjustment computation. It indicates the value and the unit that will be used and the source of the covariate, whether it is from the patient or the drug model.

In addition, if a covariate does not respect the drug model checks, a warning is displayed.

**HTML representation:**



Here, we have an example with a “fat-free mass” covariate. We can see the representation of the warning if the value from the patient does not meet the requirements.

**XML possible organization:**

**<covariates>  
 <covariate **warning=’Error message’**>  
 <name> (string) </name>  
 <value> (string) </value>  
 <unit> (unitType) </unit>  
 <desc> (string) </desc>  
 <source> default / patient </source>  
 </covariate>**

**… Rest of covariates …**

**</covariates>**

The output will return the list of the covariates used, with the following information:

* Name of the covariate
* Value entered to computation
* Unit entered to computation
* Desc of the covariate
* Source of the covariate (drug model or patient query)

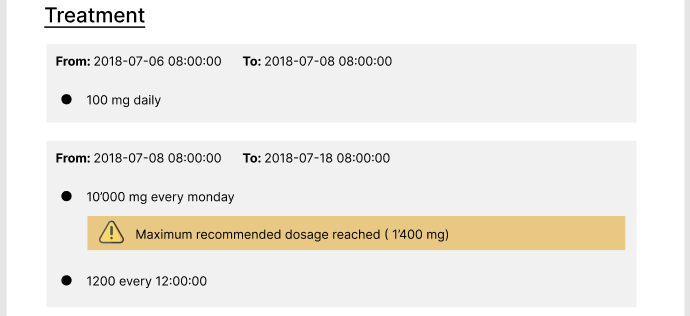
There is an optional attribute that may or may not be present depending on whether the covariate is within the boundaries of the drug model.

* If the covariate is out of bound, it will receive the “warning” attribute with an error value.
* Otherwise, the attribute is not added

### Treatment and checks

This section lists the dosages from the patient’s dosage history. It shows each dosage within a dosage time range. It displays a warning for a dosage if the dose recommended from the drug model is reached.

**HTML representation:**



The main idea is to display an indication of the posology near each dosage depending on their type:

* Lasting: Every + <interval>
* Daily: Daily at + <time>
* Weekly: Every <day> + at + <time>

If the dosage is inside a repeated dosage, it may be prefixed by the number of intakes. For example:

* 4 x Every 12 hours

**XML possible organization:**

The output will return the treatment node as it entered with a small difference. Each suspicious lasting/daily/weekly dosage node will receive “warning” attribute with an error value.

For example, the following situation could be possible:

**<treatment>  
 <dosageHistory>  
 <dosageTimeRange>  
 <start> … </start>  
 <end> … </end>  
 <dosage>  
 <dosageLoop>  
 <lastingDosage **warning=’Error message’**>  
 …  
 </lastingDosage>  
 </dosageLoop>  
 </dosage>  
 </dosageTimeRange>  
 </dosageHistory>  
</treatment>**

### Samples and check

This section lists the patient’s samples. It shows the date of the sample, its measure, and the percentile to which it belongs. It displays a warning for a sample if the sample reaches some given threshold:

* Red warning if the percentile is below 5 or above 95
* Yellow warning if the percentile is below 10 or above 90

**HTML representation:**



**XML possible organization:**

The output will return the samples node as it entered with two differences. Each suspicious concentration will receive “warning” attribute with an error value, and each concentration will receive a new “percentile” element

For example, the following situation could be possible:

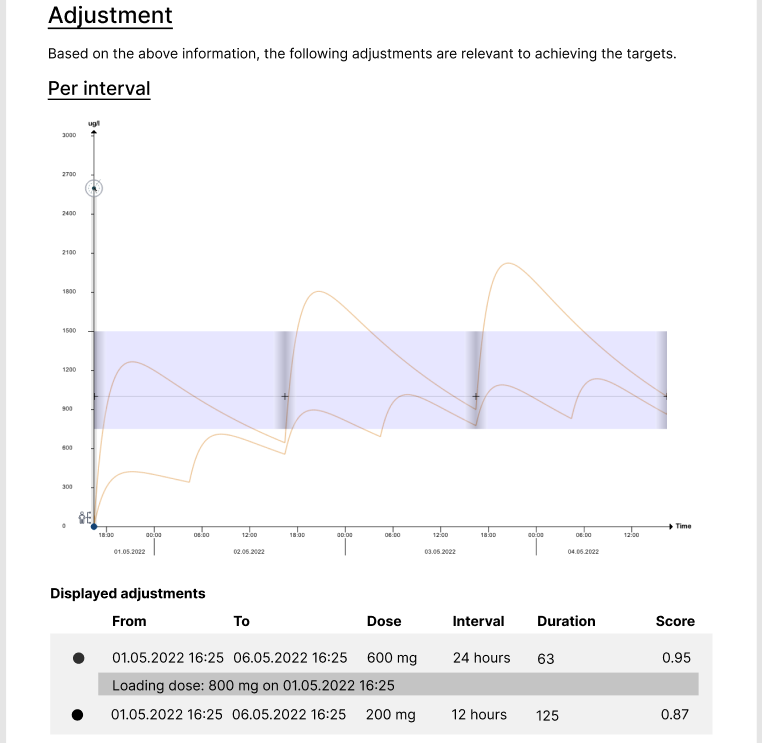
**<samples>  
 <sample>  
 <sampleId> … </sampleId>  
 <sampleDate> … </sampleDate>  
 <concentrations>  
 <concentration **warning=’Error message’**>  
 <analyteId> … </analyteId>  
 **<percentile> X </percentile>** <value> … </value>  
 <unit> … </unit>  
 <concentration>  
 …  
 </concentrations>  
 </sample>  
 …  
</ samples >**

The “warning” attribute is optional. So, it won’t be used for expected concentration.

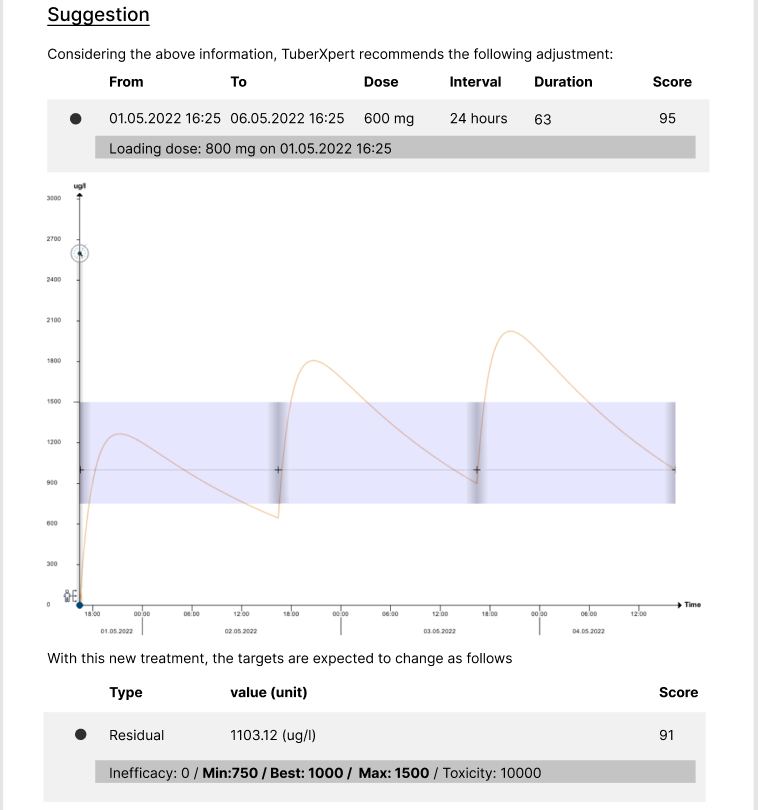
### Adjustments

This section shows the possible adjustments and the one that is judged the best by Tucuxi computation core.

**HTML representation:**



This first part of the adjustment section displays a short introductory sentence followed by a graph of all the adjustments found. Below the graph, a table lists all the displayed dosages. It indicates the time range, the dose, the intake interval, the number of intakes and the attributed score of adjustment quality by Tucuxi computation. For a complete specification scores, see the document “Tucuxi System Description “. The higher the score, the better. In addition, if necessary, it indicates the required loading dose.

****

This second part highlights the adjustment with the highest score. It shows its time range, the dose, the intake interval, the number of intakes and the score assigned by Tucuxi computation. Additionally, it displays the achievement of the targets. For each target, it lists its type, its predicted value, its score, and the thresholds. The target score is the same as the query when there is only one target, but in reality the adjustment score is the average of all target score.

**XML possible organization:**

The XML content will roughly be the same as the *“DataAdjustment”* element from the Tucuxi computation response. For example, the following XML structure could be possible:

**<dataAdjustment>  
 <analyteIds>  
 <analyteId> […] </analyteId>  
 </analyteIds>  
 <adjustments>  
 <adjustment>  
 <score> 0.985 </score>  
 <targetEvaluations> [See below] </targetEvaluations>  
 <dosageHistory> [See below] </dosageHistory>  
 <cycleDatas> [See below] </cycleDatas>  
 </adjustment>**

**…  
 </adjustments>  
</dataAdjustment>**

The **analyteIds** is the list of all the analyte identifiers involved in the **cycleDatas**. It is only used when a drug contains more than one analyte.

**<targetEvaluations>  
 <targetEvaluation>  
 <targetType> residual </targetType>  
 <unit >ug/l </unit>  
 <value> 1103.122367 </value>  
 <score> 0.919806 </score>  
 <min> 750 </min>  
 <best> 1000 </best>  
 <max> 1500 </max>  
 <inefficacyAlarm> 0 </inefficacyAlarm>  
 <toxicityAlarm> 10000 </toxicityAlarm>  
 </targetEvaluation>**

**…  
</targetEvaluations>**

The **targetEvaluation** element is much like a target element, but with a **value** and an achievement **score**.

**<dosageHistory>  
 <dosageTimeRange>  
 <start>2018-01-12T07:00:00</start>  
 <end>2018-03-15T12:59:00</end>  
 <dosage>[…]</dosage>  
 </dosageTimeRange>  
 …  
</dosageHistory>**

The dosage history element contains the list of dosage to be followed for adjustment. Its structure is the same as that of the input dosage history.

When there are multiple **dosageTimeRange**, if the first one contains a repetition of one iteration, it is a loading dose.

**<cycleDatas>** **<cycleData>** **<start>2018-01-12T07:00:00</start>** **<end>2018-01-12T19:00:00</end>** **<unit>ug/l</unit>** <times> […] </times>  
 <values> […] </values>  
 </cycleData>  
**</cycleDatas>**

### Computation facts

# Tests

## Unit Tests

A test program has been set up to perform unit testing. To do this, it uses the framework Fructose[[2]](#footnote-2).

### LanguageManager

The language/*languagemanager* class is tested by the *test\_languagemanager* class.

|  |  |
| --- | --- |
| Name:  Retrieve dictionary | Result: Success |
| Description:  The test tries to perform loadDictionary with various xml string.. |
| Validation:   * When a bad string is used, a LanguageException is raised. * When a valid string is used, no exception is thrown. |

|  |  |
| --- | --- |
| Name:  Word translation | Result: Success |
| Description:  The test tries to perform *translate* method calls with an invalid key and then with a valid key. |
| Validation:   * When the bad key is used, it gets the string  Tucuxi::Language::LanguageManager:: defaultTranslation. * When the valid key is used, the corresponding value from dictionary xml file is retrieved. |

### XpertQueryImort

The query/*xpertqueryimport* class is tested by the *test\_xpertqueryimport* class.

|  |  |
| --- | --- |
| Name:  Retrieve complete admin. | Result: Success |
| Description:  The test tries to load a query file that contains the most complete admin element possible. It includes a mandator with his institute, a patient with his institute and non-empty clinical data element. Then, it checks if all the information from the file is correctly returned. |
| Validation:   * The import status is Status::Ok. * All information from file is correctly returned. |

|  |  |
| --- | --- |
| Name:  Retrieve no admin. | Result: Success |
| Description:  The test tries to load a query file without any admin element inside. |
| Validation:   * The import status is Status::Ok. * When getting the std::optional<std::reference\_wrapper<const AdministrativeData>>, it has no value. |

|  |  |
| --- | --- |
| Name:  Retrieve empty admin. | Result: Success |
| Description:  The test tries to load a query file without an admin element that is empty. |
| Validation:   * The import status is Status::Ok. * When getting the std::optional<std::reference\_wrapper<const AdministrativeData>>, it has a value. * When using getters of the received administrative data, the optional values returned have no value. |

|  |  |
| --- | --- |
| Name:  Retrieve minimal person | Result: Success |
| Description:  The test tries to load a query file with minimal patient and mandator. It means that they only have a first name and a last name. |
| Validation:   * The import status is Status::Ok. * When getting their identifier and name, the getters return an empty string. * When getting their address, phone, email and institute, the getters return optional without value. |

|  |  |
| --- | --- |
| Name:  Retrieve minimal institute | Result: Success |
| Description:  The test tries to load a query file with minimal patient and mandator institute. |
| Validation:   * The import status is Status::Ok. * When getting their identifier and name, the getters return an empty string. * When getting their address, phone, and emai, the getters return optional without value. |

|  |  |
| --- | --- |
| Name:  Retrieve minimal coordinates | Result: Success |
| Description:  The test tries to load a query file with minimal address, phone and email in persons and institutes. |
| Validation:   * The import status is Status::Ok. * When getting their missing values, it returns an empty string. |

|  |  |
| --- | --- |
| Name:  Error when missing mandatory in mandator person | Result: Success |
| Description:  The test tries to load a query file with missing mandatory values in mandator person. |
| Validation:   * The import status is Status::Error. * All the mandatory nodes appear in error message. |

|  |  |
| --- | --- |
| Name:  Error when missing mandatory in mandator institute | Result: Success |
| Description:  The test tries to load a query file with missing mandatory values in mandator institute. |
| Validation:   * The import status is Status::Error. * All the mandatory nodes appear in error message. |

|  |  |
| --- | --- |
| Name:  Error when missing mandatory in a patient person | Result: Success |
| Description:  The test tries to load a query file with missing mandatory values in patient person. |
| Validation:   * The import status is Status::Error. * All the mandatory nodes appear in error message. |

|  |  |
| --- | --- |
| Name:  Error when missing mandatory in a patient institute | Result: Success |
| Description:  The test tries to load a query file with missing mandatory values in patient institute. |
| Validation:   * The import status is Status::Error. * All the mandatory nodes appear in error message. |

|  |  |
| --- | --- |
| Name:  Retrieve complete requestXpert | Result: Success |
| Description:  The test tries to load completely formed requestXpert. |
| Validation:   * The import status is Status::Ok. * All the values are correctly returned by the getters. |

|  |  |
| --- | --- |
| Name:  Retrieve default requestXpert | Result: Success |
| Description:  The test tries to load a requestXpert that has not filled values that are optional. |
| Validation:   * The import status is Status::Ok. * When using getter on the missing optional values, the corresponding default value is returned. |

|  |  |
| --- | --- |
| Name:  Error when no requestXpert | Result: Success |
| Description:  The test tries to load file that has no requestXpert. |
| Validation:   * The import status is Status::Error. * The requestXpert vector got from XpertQueryData has 0 length. * The importer error message indicates "No requestXpert found". |

|  |  |
| --- | --- |
| Name:  Error when missing mandatory requestXpert | Result: Success |
| Description:  The test tries to load a requestXpert that has missing mandatory values. |
| Validation:   * The import status is Status::Error. * All the mandatory nodes appear in error message. |

# Conclusion

# Bibliography

* Prof. THOMA Yann, prof. MPAGAMA Stellah, GUIDI Monia, 2022, TuberXpert: Clinical Decision Support System for antituberculosis medical drugs [PDF document]
* BENALLAL Nadir, 2018, Expert System for drug dosage adaptation [PDF document], Yverdon-les-Bains : Haute école d’ingénierie et de gestion du canton de Vaud. Bachelor’s thesis
* CADUFF Max, 2020, Expert System for drug dosage adjustments [PDF document], Yverdon-les-Bains : Haute école d’ingénierie et de gestion du canton de Vaud. Bachelor’s thesis
* BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document]
* TUCUXI dev team, 2022, Tucuxi CLI Software Usability Specification [PDF document]
* TUCUXI dev team, 2022, Tucuxi Drug Model File Specification [PDF document]
* TUCUXI dev team, 2022, Tucuxi System Description [PDF document]
* DATAPHARM, 2022, EMC, 21 Feb 2022 [24 Mar 2022], <https://www.medicines.org.uk/emc/product/8788/smpc>

# Authentication

The undersigned, Melvyn Herzig, hereby certifies that he alone conducted this work and did not use any other source than those expressly mentioned.

Yverdon, the Friday, March 4, 2022

Melvyn Herzig

# Planification

This chapter presents the Gantt charts of the planning. The initial diagram was made after a couple of hours of work, the intermediate diagram after 150 hours, and the final diagram after 450 hours.

## Initial



Figure 10 Gantt chart of the initial planning

## Intermediate



Figure 11 Gantt chart of the intermediate planning

## Final

1. For a complete input specification, see the file “Tucuxi CLI Software Usability Specification” [↑](#footnote-ref-1)
2. <http://www.andrewpetermarlow.co.uk/software/fructose.html> [↑](#footnote-ref-2)