**Bachelor’s thesis**

TuberXpert

Not confidential

****

|  |  |
| --- | --- |
| **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, the Sunday, February 24, 2022

Department TIC

Faculty Computer science

Specialization Software

Student Melvyn Herzig

Responsible teacher Prof. Yann Thoma

Bachelor’s thesis 2021-2022

Cliquez ici pour écrire votre titre de TB.

**Publishable summary**

Cliquez ou appuyez ici pour entrer du texte.

|  |  |  |
| --- | --- | --- |
| 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, the Friday, March 4, 2022

**Authentication**

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

Yverdon, the Friday, March 4, 2022

Melvyn Herzig

**Table of contents**

[1 Introduction 6](#_Toc102565045)

[1.1 What is therapeutic drug monitoring 6](#_Toc102565046)

[1.2 Actual situation in Tanzania 6](#_Toc102565047)

[1.3 Goal of this work 7](#_Toc102565048)

[2 Software needs 8](#_Toc102565049)

[2.1 Requirements 8](#_Toc102565050)

[2.1.1 Functional 8](#_Toc102565051)

[2.1.2 Nonfunctional 8](#_Toc102565052)

[2.2 Architecture 8](#_Toc102565053)

[3 Analysis 9](#_Toc102565054)

[3.1 Global application overview 9](#_Toc102565055)

[3.2 Technologies 9](#_Toc102565056)

[3.2.1 Development language 9](#_Toc102565057)

[3.2.2 Integrated Development Environment 9](#_Toc102565058)

[3.2.3 Compiler 10](#_Toc102565059)

[3.3 Input handling 10](#_Toc102565060)

[3.3.1 Global 10](#_Toc102565061)

[3.3.2 Covariates 11](#_Toc102565062)

[3.3.3 Drug 11](#_Toc102565063)

[3.3.4 Dosage 12](#_Toc102565064)

[3.3.5 Sample 12](#_Toc102565065)

[3.3.6 Target 13](#_Toc102565066)

[3.3.7 Request 13](#_Toc102565067)

[3.4 Program execution flow 14](#_Toc102565068)

[3.4.1 Get best drug file 15](#_Toc102565069)

[3.4.2 Assess the dosages 16](#_Toc102565070)

[3.4.3 Assess the samples 17](#_Toc102565071)

[3.4.4 Assess the targets 18](#_Toc102565072)

[3.5 Input 19](#_Toc102565073)

[3.6 Output 19](#_Toc102565074)

[3.6.1 Header 19](#_Toc102565075)

[3.6.2 Administrative 20](#_Toc102565076)

[3.6.3 Covariates and checks 21](#_Toc102565077)

[3.6.4 Treatment and checks 22](#_Toc102565078)

[3.6.5 Samples and check 23](#_Toc102565079)

[3.6.6 Adjustments 23](#_Toc102565080)

[3.6.7 Computation facts 23](#_Toc102565081)

[4 Tests 23](#_Toc102565082)

[4.1 Unit Tests 23](#_Toc102565083)

[4.1.1 LanguageManager 23](#_Toc102565084)

[4.1.2 XpertQueryImort 23](#_Toc102565085)

[5 Conclusion 26](#_Toc102565086)

# 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.

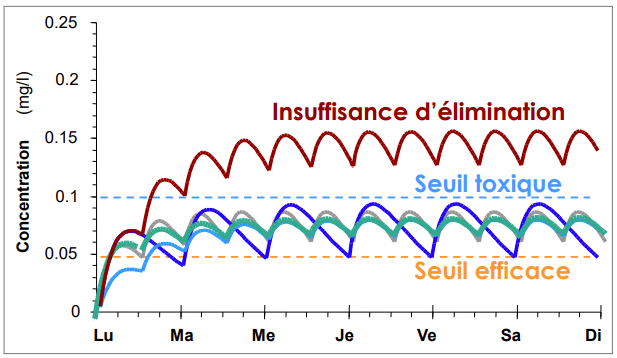


Figure 1 - Dosage Scheme (BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document])

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.

## Actual 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 central 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 significant 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 are kept 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 greatly 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 rifampicin 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 rifampicin 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 describes what TuberXpert needs to be and to do.

## Requirements

### Functional

* Assess the relevance of query data.
* Chose a drug model to be used accordingly to the query.
* Assess the expectedness of a drug concentration result, considering the patient’s characteristics.
* Assess the target attainment of the current dosage.
* Propose a dosage adjustment.
* Present clear and meaningful messages within the report to help the clinician with the decision-making process.
* Generate alerts when data seems suspicious or erroneous.
* It can support various languages.

### Nonfunctional

* There is no need to develop a graphical user interface. A CLI is sufficient.
* The generated report must be user friendly and good looking.
  + The report to be generated shall not only display useful values and graphs but shall also offer readable sentences.
* The solution must be developed keeping in mind that it must be scalable. In other words, it is easy to be modified for other drugs compatible with Tucuxi or to change the behavior of the system.

## Architecture

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

# Analysis

## Global application overview

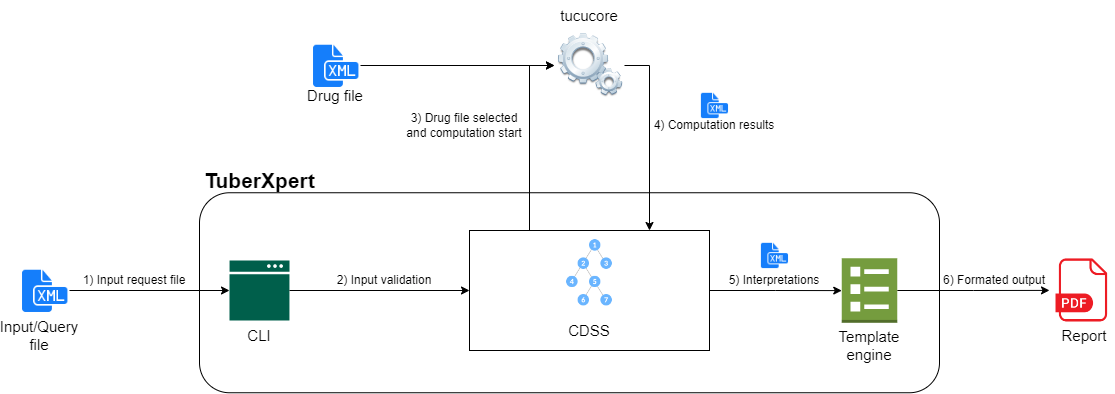


Figure 3 Global application overview and components

1. The program receives as input a query in XML format. For simplification purpose, the XML format will be the same used by tucucore. However, it will be extended with a specific request element
2. The program loads the data from the query file and transmit them to the CDSS that will determine if the data 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 as XML are passed to a template engine that format them into a readable format.
6. The report is saved into a PDF file.

## Technologies

### Development language

Initially, Tucuxi was developed in C++.This language was chosen for its good 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 probably be used on low-powered computers in Tanzania, we will optimize the performance by using a low-level language.

Version: C++ 17

### Integrated Development Environment

Once again, we will walk in the steps of Tucuxi. We 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 installation of the development environment, this project is compiled with the compiler integrated with QT.

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

## Input handling

The input will be an XML file with the same structure as tucucore. For a complete input specification, see the file “Tucuxi CLI Software Usability Specification”. Consequently, this chapter will not explain the expected structure but the checks that will be made by the system.

This chapter will parse the general structure of an input file and describe what is checked.

### Global

**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>

    <drugTreatment>

        <patient>

            <covariates></covariates>

        </patient>

        <drugs></drugs>

    </drugTreatment>

    <requests></requests>

</query>

**Checks :**

At this stage, none checks are performed.

### Covariates

The covariates element contains zero or many covariate elements .

**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 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.*

### Drug

The drugs element contains some drug elements about treatments the patient could be in.

**Example of drug:**

<drugs>

    <drug>

        <drugId>imatinib</drugId>

        <brandName>somebrand</brandName>

        <atc>something</atc>

        <treatment></treatment>

        <samples></samples>

        <targets></targets>

    </drug>  
</drugs>

**Checks:**

If the drug id corresponds 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 history may contain 0 or many dosages time range.

The dosage is a complex but flexible element. The main point is that it will always contain a dose element that will allow the dosage to be validated.

*<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 the minimum and maximum bounds from the drug file are reached, a warning will be printed in the final report.*

### Sample

The samples element contains some samples. In other words, a list of drug concentration measurements.

<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, a warning will be printed in the final report.*

### Target

The adaptation engine uses targets to fit the dosage to the needs of the patient. Drug files provides such targets, but it is possible to override them by providing one in the input file.

<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, it will be replaced by the default target from drug file and a warning will be printed in the final report.*

### Request

The request element represents the computation that the core is asked to perform.

<requests>

    <request>

        <requestId>aposteriori</requestId>

        <drugId>imatinib</drugId>

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

        <COMPUTING\_TRAIT>

        </COMPUTING\_TRAIT>

    </request>

</requests>

**Checks:**

Strictly speaking, no verification is performed. Original requests are ignored. The system will formulate its own queries.

*The system will implement a new type of request. Because it is not only intended to be executed locally but also on a web server, all parameter that could be program argument will be converted into properties of the custom request. This point will be discussed later on.*

## Program execution flow

This chapter presents the main steps when the program is executed. Each important step is colored and detailed in a subchapter.

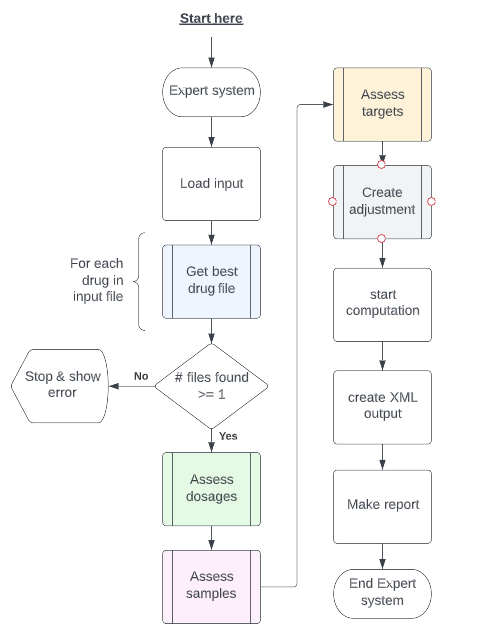


Figure 4 Program global execution

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, then the execution cannot continue. Otherwise, if a drug file is found for some drugs, it assesses the dosages, samples, and targets. After that, it forges an adjustment request for tucucore. When the computation is done, it creates an XML output file from the computation result and the input analysis. Finally, it creates the final report using the previously created XML output file.

### 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.

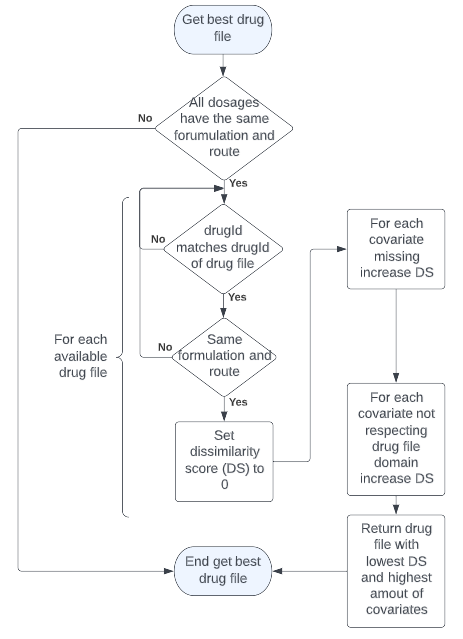


Figure 5 Process of drug files evaluation

**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 probably 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 in order 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

In order 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 tucucore *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.

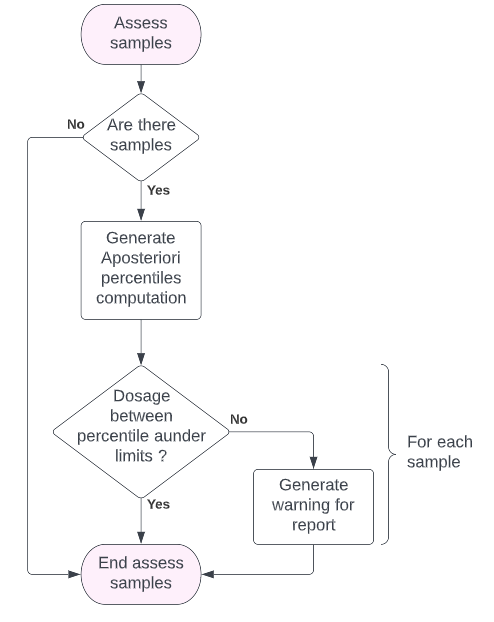


Figure 7 Process of samples assessment

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 in order 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.*

## Input

## 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. From that point, we can 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 facts about the drug such as:

* The identifier of the concerned drug
* The last dosage of the patient

**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:

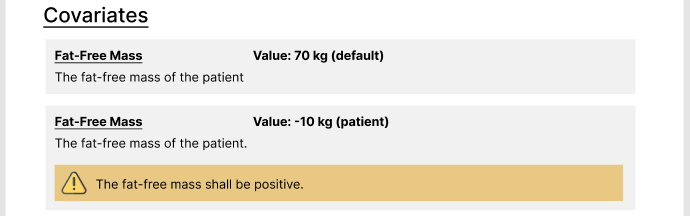
**<admin>  
 <mandator>   
 <person> … </person>  
 <institute> … </institute>  
 </mandator>  
 <patient>  
 <person> … </person>  
 <institute> … </institute>  
 </patient>  
 <clinicalData> … </clinicalData>  
</admin>**

### 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)

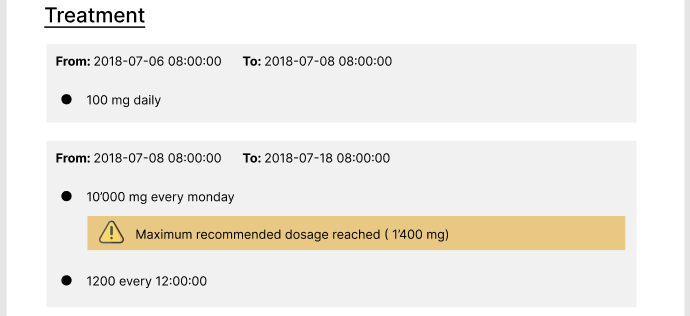
There is an optional attribute that may or may not be present depending on whether the covariate is within the boundaries of the drug model.

* 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

### Adjustments

### Computation facts

# Tests

## Unit Tests

A test program has been set up to perform unit testing. To do this, it uses the framework Fructose[[1]](#footnote-1).

### 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]
* DATAPHARM, 2022, EMC, 21 Feb 2022 [24 Mar 2022], https://www.medicines.org.uk/emc/product/8788/smpc

1. <http://www.andrewpetermarlow.co.uk/software/fructose.html> [↑](#footnote-ref-1)