**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** |
| **Expert:** | **Jonas Chapuis** |
| **Academic year:** | 2021-2022 |

Yverdon-les-Bains on Monday, May 16, 2022

**Table of contents**

[1 Preamble 6](#_Toc109411452)

[2 Introduction 7](#_Toc109411453)

[2.1 What is therapeutic drug monitoring 7](#_Toc109411454)

[2.2 Tucuxi 7](#_Toc109411455)

[2.3 Health situation in Tanzania 8](#_Toc109411456)

[2.4 Goal of this work 9](#_Toc109411457)

[3 Works to do 10](#_Toc109411458)

[3.1 Specification 10](#_Toc109411459)

[3.1.1 Context 10](#_Toc109411460)

[3.1.2 Objective 10](#_Toc109411461)

[3.1.3 Features 10](#_Toc109411462)

[3.2 Requirements 11](#_Toc109411463)

[3.2.1 Functional 11](#_Toc109411464)

[3.2.2 Nonfunctional 11](#_Toc109411465)

[3.3 Testing 11](#_Toc109411466)

[3.4 Architecture 11](#_Toc109411467)

[4 Analysis 12](#_Toc109411468)

[4.1 Technologies 12](#_Toc109411469)

[4.1.1 Development language 12](#_Toc109411470)

[4.1.2 Integrated Development Environment 12](#_Toc109411471)

[4.1.3 Compiler 12](#_Toc109411472)

[4.2 Input – TuberXpert query file 12](#_Toc109411473)

[4.2.1 Global 13](#_Toc109411474)

[4.2.2 Admin 13](#_Toc109411475)

[4.2.3 Covariates 17](#_Toc109411476)

[4.2.4 Drugs 17](#_Toc109411477)

[4.2.5 Treatment 18](#_Toc109411478)

[4.2.6 Samples 18](#_Toc109411479)

[4.2.7 Targets 19](#_Toc109411480)

[4.2.8 XpertRequests 20](#_Toc109411481)

[4.3 Global application overview 23](#_Toc109411482)

[4.4 Program execution flow 24](#_Toc109411483)

[4.4.1 Get best drug model and validate the covariates 25](#_Toc109411484)

[4.4.2 Assess the doses 26](#_Toc109411485)

[4.4.3 Assess the samples 27](#_Toc109411486)

[4.4.4 Assess the targets 28](#_Toc109411487)

[4.4.5 Create adjustment trait 29](#_Toc109411488)

[4.5 Report 32](#_Toc109411489)

[4.5.1 Header 32](#_Toc109411490)

[4.5.2 Admin 33](#_Toc109411491)

[4.5.3 Covariates and checks 34](#_Toc109411492)

[4.5.4 Treatment and checks 35](#_Toc109411493)

[4.5.5 Samples and check 37](#_Toc109411494)

[4.5.6 Best adjustments and suggested adjustment 38](#_Toc109411495)

[4.5.7 Computation facts 43](#_Toc109411496)

[5 Implementation 46](#_Toc109411497)

[5.1 Run TuberXpert 46](#_Toc109411498)

[5.2 Query import 47](#_Toc109411499)

[5.2.1 Administrative data 47](#_Toc109411500)

[5.2.2 Xpert request data 48](#_Toc109411501)

[5.2.3 Xpert query data 48](#_Toc109411502)

[5.2.4 Xpert query import 49](#_Toc109411503)

[5.2.5 Simplified import overview 49](#_Toc109411504)

[5.3 Create the results holders 50](#_Toc109411505)

[5.3.1 Xpert query result and xpert request result 50](#_Toc109411506)

[5.3.2 Covariates, doses and samples validation results 51](#_Toc109411507)

[5.4 Select the correct flow steps 52](#_Toc109411508)

[5.4.1 Abstract xpert flow step and abstract xpert flow step provider 53](#_Toc109411509)

[5.4.2 Simplified overview of the flow step provider selection 53](#_Toc109411510)

[5.5 Process an XpertRequestResult 54](#_Toc109411511)

[5.5.1 Load a translations file 54](#_Toc109411512)

[5.5.2 Validate the covariates and select the best drug model 55](#_Toc109411513)

[5.5.3 Validate the doses 56](#_Toc109411514)

[5.5.4 Validate the samples 56](#_Toc109411515)

[5.5.5 Validate the targets 57](#_Toc109411516)

[5.5.6 Create the adjustment trait 57](#_Toc109411517)

[5.5.7 Execute the requests 58](#_Toc109411518)

[5.5.8 Print the report 59](#_Toc109411519)

[5.5.9 Error management 61](#_Toc109411520)

[5.6 Termination 61](#_Toc109411521)

[6 Tests 62](#_Toc109411522)

[6.1 Unit and integration tests 62](#_Toc109411523)

[7 Conclusion 62](#_Toc109411524)

[8 Bibliography 63](#_Toc109411525)

[9 Authentication 64](#_Toc109411526)

[10 List of abbreviations 65](#_Toc109411527)

[11 List of figures 66](#_Toc109411528)

[12 Planification 67](#_Toc109411529)

[12.1 Initial 67](#_Toc109411530)

[12.2 Intermediate 68](#_Toc109411531)

[12.3 Final 69](#_Toc109411532)

**Department:** TIC

**Faculty:** Computer science

**Specialization:** Software

**Student:** Melvyn Herzig

**Responsible teacher:** Prof. Yann Thoma

**Expert:** Jonas Chapuis

Bachelor’s thesis 2021-2022

TuberXpert

**Publishable summary**

Tanzania is a country with a high incidence of tuberculosis. Every year, 120'000 people contract the disease. However, only half of them receive treatment and most of the time it is a failure. In one to two-thirds of patients with tuberculosis, the peak concentration of the antitubercular drug is below the reference range. In addition, one-third of people with tuberculosis must take antiretroviral drugs that further lower the concentration of the antitubercular drug. The risk of treatment failure is real.

Taking a not suitable treatment is dangerous. The patient may become drug resistant, intoxicated or even die. In consequences, the Tanzanian government has expressed its desire to end tuberculosis by 2035.

Fortunately, the fight is not lost. The main ally is therapeutic drug monitoring, which is a specialized practice in measuring drug levels in the blood to a treatment. However the whole process is slow. In addition, it takes an experienced pharmacologist to interpret the samples and recommend an effective adjustment. To facilitate this part, Professor Yann Thoma, in partnership with the CHUV, has developed a software program called Tucuxi. However, it still needs to be handled by an experienced practitioner.

In view of the current situation, TuberXpert is a project between Switzerland and Tanzania that aims to create a clinical decision support system to democratize access to therapeutic drug monitoring. Based on the Tucuxi computation core, this bachelor thesis consists in developing a first version of the software.

Indeed, by obtaining information on the patient, TuberXpert is able to evaluate the plausibility of the received data and to propose a first adapted treatment or an adjustment by providing a complete report in English or French.

Since the software was developed with a basic knowledge of therapeutic drug monitoring, the results are very encouraging. In its final form, with more knowledge of pharmacology, TuberXpert should be a real success.

# Preamble

This Bachelor's thesis 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 Bachelor's thesis 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 has been developed. TDM is a precision medicine that prescribes a personalized dosage to each patient based on the monitoring of the evolution and the prediction of the drug concentration in the blood.



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

After the administration of a drug, the prediction of the evolution of blood concentrations depends on :

* The administration methods (formulation, route of administration and dosages)
* The pharmacokinetic parameters which describe the behavior of the drug in the body. These parameters are influenced by the physical characteristics of the patient called covariates.

## Tucuxi

Currently, Professor Yann Thoma and his team have developed [Tucuxi](http://www.tucuxi.ch/). 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 (samples). The pharmacokinetic parameters (PK) are:
  + Typical patient. If the default covariates are used to calculate them.
  + A priori. If they exist, the patient covariates replace the default values to calculate them.
  + A posteriori. If they exist, the patient covariates replace the default covariates. In addition, the patient samples are considered for the calculation.
* Suggestion of dosage adjustments to reach an optimal drug concentration state.
* Generation of reports.
* Integration with Electronics Health Record systems.

Tucuxi is available in 3 formats:

* [A web application](http://webdemo.tucuxi.ch/).
* A graphical user interface application for desktop.
* A command line interface application for desktop

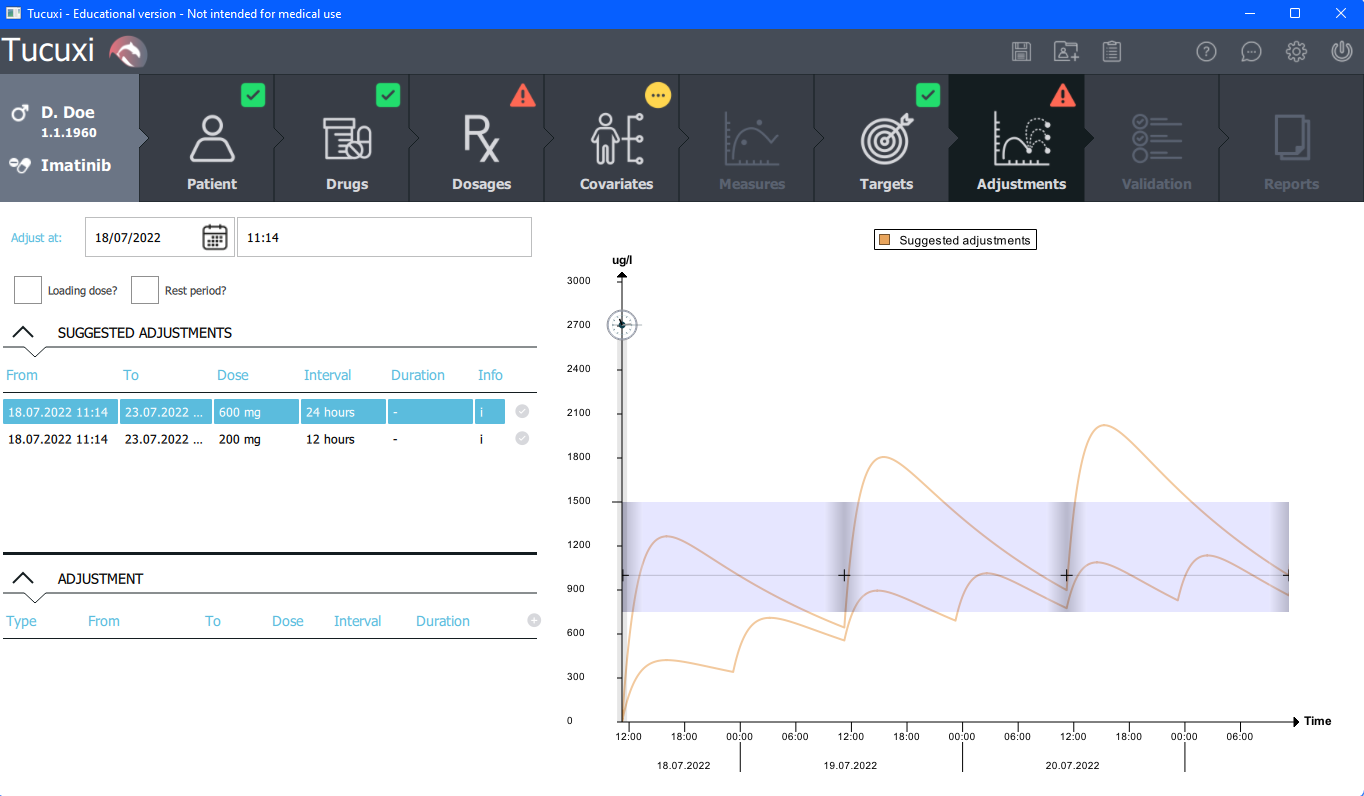


Figure 2 Tucuxi graphical user interface with suggested first dosages

In order to work, Tucuxi needs 2 types of information:

**Patient information:**

As explained, TDM aims to provide treatments tailored to the patient. In order to best fit the patient's needs, Tucuxi needs as much information as possible such as physical characteristics called covariates, dosages, blood samples and ideal concentrations to be achieved called targets.

**Drug information:**

In order to calculate drug concentration predictions, Tucuxi needs to know the technical characteristics of the drug. All this information is available in what is called a drug model. It indicates which covariates have an influence, formulas to calculate pharmacokinetic parameters, supported formulations and routes of administration, some generic targets and the composition of the drug (what are the active moieties and analytes).

It is important to know that there is more than one drug model per drug. In general, drug models can differ in almost everything they contain: they can support another set of covariates, another formulation and route of administration, or another set of pharmacokinetic parameters.

The drug model must be chosen according to the information we have about the patient.

## Health situation in Tanzania

Tanzania has a high burden of tuberculosis. 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.

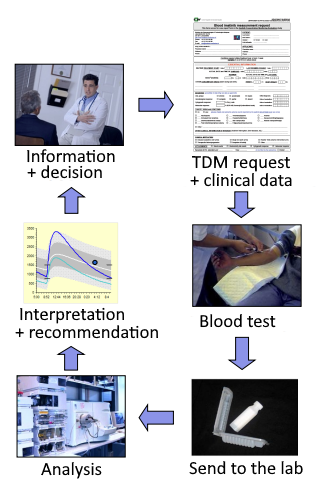
However, the problem is that only 60’000 – 75’000 patients are notified and receive a treatment. In addition, studies have reported that rifampicin dosages are insufficient most of the time. For example, investigations by Heysell et al. (2011) and Tostmann et al. (2013) in the Kilimanjaro region showed 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 the last dose intake.

Moreover, a considerable proportion of individuals with TB are co-infected with HIV. It represents 25 to 40% of the monitored people. Administration of antiretrovirals with first-line anti-tuberculosis drugs further reduces the concentration of rifampicin.

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 has shown that individuals with TB and DM are five times more likely to die than those without diabetes.

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, because both the drug measurement and the interpretation of concentration results are demanding. In addition, the number of experienced pharmacologists is not sufficient to provide well-established interpretation and recommendation everywhere their expertise is needed.

Although Tucuxi is an important player in the democratization of TDM, it still needs to be handled by experienced pharmacologists.

This is where TuberXpert comes into play. It comes at the beginning of a large project between Switzerland and Tanzania led by Prof. Thoma Yann, Prof. Mpagama Stellah and Prof. Guidi Monia. The objective is to develop a clinical decision support system (CDSS) to fight tuberculosis. To simplify the development process and reuse what already exists, TuberXpert will be a software that integrates the Tucuxi computation core. By receiving complete information from a patient, the system will assess the relevance of the data provided and then will determine 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 3 -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 of the whole project. It will then be taken over by three Ph.D. students in charge of the development and the concrete application of the project.

# Works to do

This chapter presents the specification of this Bachelor’s thesis. Then, it describes the requirements: how the system must behave and what are its properties. Finally, some comments on the tests to be performed and the possible architecture to adopt.

## Specification

### Context

This work takes place in the domain of therapeutic drug monitoring (TDM). Currently, Tanzania is experiencing a high amount of tuberculosis 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

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 information about the patient and his treatment trough an XML file similar to those used with Tucuxi CLI.
  + The XML structure may be extended with new useful elements if needed.
* The program will analyze and verify the relevance of the data.

**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 the 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 the patient.
  + Dosage adjustment.
  + …

**Multi-language**

* The program must support various languages.
* At least, the English version 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.

## 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, his treatments and some requests to provide a few computation instructions. Then, through an output report, the program displays the received data and highlights the suspicious ones, suggests an initial dosage or an adjustment of the drugs and provides any additional useful information for the therapeutic drug monitoring. The output contains readable sentences, and the report displays some graphs of the adjustment.

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

As input, TuberXpert needs an XML file containing patient and treatment information as well as requests that contain computation instruction. The structure of the file is the same as that of Tucuxi CLI. However, it could extend the basic structure by adding some new elements, at least, a new type of request dedicated to TuberXpert control and probably some administrative information

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 the 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 computation 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 some full sentences, the evaluation of the data, the adjustment graphs for the HTML and PDF format or the adjustment data to generate a graph for the XML format. The report can be generated in several languages.

### Nonfunctional

This project is most likely a proof of concept. For this work, it is not necessary to develop a graphical user interface. 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 to extract the information.

Although this work is made in the context of tuberculosis, it is developed without an extensive knowledge of rifampicin or any other drug. Consequently, the development is generic in considering all drugs. 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 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 a third-party report generator to fill some fields of a report template.

# Analysis

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

## 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 project collection, TuberXpert will also be developed in C++. This language is once again very advantageous. As the software could 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 it will use the same development environment. Qt Creator is a very advantageous and rich integrated development environment (IDE). It easily allows cross-platform development, test management, language management and many other frameworks. Qt Creator 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 Creator 6.0.2 community

### Compiler

As this project should be easy to use on as many computers as possible, I have chosen two classic and widely used compilers.

Version:

* (Windows) MinGW-W64 6.3.0
* (Ubuntu) GCC 11.2.0

## Input – TuberXpert query file

Like the Tucuxi CLI, TuberXpert receives all the useful information about the patient, his treatments and the adjustments to be performed through an xml query file. Most of its structure will be essentially the same as Tucuxi CLI[[1]](#footnote-1).

Although, this section reviews the TuberXpert query structure, it does not explain how to form the common elements with Tucuxi CLI but what they represent and, if necessary, what will be checked.

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

### Global

Down below is the overall structure of the input. It consists of the date of computation and the administrative data (admin). Then, there is information about the patient’s covariates followed by the drugs he is taking. Finally, comes the requests element. It contains the new xpertRequest elements used to tell TuberXpert which drug should be adjusted and how the adjustment report should be.

**Example of the global structure of the query**:

<?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">

    <date>2018-07-11T13:45:30</date>

**<admin>[…]</admin>**

    <drugTreatment>

        <patient>

            <covariates>[…]</covariates>

        </patient>

        <drugs>[…]</drugs>

    </drugTreatment>

    <requests>  
      **<xpertRequest>[…]</xpertRequest>**

      […]

    </requests>

</query>

*In the Tucuxi CLI query, there are the queryId, clientId and language elements. In TuberXpert, these elements are not necessary. So, they are not necessarily present. Although their value is currently unused, they are still retrieved if they are present.*

The date will be used to fix the “present” time. It will be particularly useful to calculate an age from a date of birth in a test and getting the same result today and in 10 years. However, the general practice will be to put the local time in this element.

The noticeable addition is the admin element and the xpertRequest element.

### Admin

The structure of the admin element is inspired by the one from the bachelor’s thesis of Nadir Benallal. It is flexible and contains every needed field to store contact information of the patient and the mandator as well as clinical data of any kind

This administrative information will be used to find out who the patient is, who the adjustment mandator is and how they can be contacted. It should be displayed at the beginning of the report generated to know who is involved.

**Checks:**

At this stage, no data validation is performed.

<admin>

    <mandator>

        <person>[…]</person>

        <institute>[…]</institute>

    </mandator>

    <patient>

        <person>[…]</person>

        <institute>[…]</institute>

     </patient>

     <clinicalData>[…]</clinicalData>

</admin>

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <admin> |  | 0:1 | The administrative data. |
| \_<mandator> |  | 0:1 | The mandator of the adjustment. |
| \_\_<person> | Person | 1:1 | Personal contact information of the mandator. |
| \_\_<institute> | Institute | 0:1 | Institute contact information of the mandator. |
| \_<patient> |  | 0:1 | The patient that will follow the adjustment. |
| \_\_<person> | Person | 1:1 | Personal contact information of the patient. |
| \_\_<institute> | Institute | 0:1 | Institute contact information of the patient. |
| \_<clinicalDatas> | clinicalData | 0:1 | Contains the clinical data. |
| \_\_<clinicalData> | String | 0: ∞ | Any additional data. |

*The clinicalData element has an attribute called “key”. The content of this attribute is used as key to retrieve the value. For example: <clinicalData key=”roomNumber”>25</clinicalData>*

*It is recommended to write the key in camelCase so that it is correctly rendered in the HTML/PDF report.*

**Person element**

A person element contains the personal contact information of the patient or the mandator.

<person>

    <id>asdf</id>

    <title>Dr.</title>

    <firstName>John</firstName>

    <lastName>Doe</lastName>

    <address>[…]</address>

    <phone>[…]</phone>

    <email>[…]</email>

</person>

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <person> |  | 1:1 | Personal contact information |
| \_<id> | string | 0:1 | Identifier of the person |
| \_<title> | string | 0:1 | Title of the person |
| \_<firstName> | string | 1:1 | The first name of the person |
| \_<lastName> | string | 1:1 | The last name of the person |
| \_<address> | Address | 0:1 | Address of the person |
| \_<phone> | Phone | 0:1 | Phone number of the person |
| \_<email> | Email | 0:1 | Email of the person |

**Institute element**

An institute element contains the contact information of an institute.

<institute>

    <id>456789</id>

    <name>CHUV</title>

    <address>[…]</address>

    <phone>[…]</phone>

    <email>[…]</email>

</institute>

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <institute> |  | 0:1 | Institute contact information |
| \_<id> | string | 0:1 | Identifier of the institute |
| \_<name> | string | 1:1 | Name of the institute |
| \_<address> | Address | 0:1 | Address of the institute |
| \_<phone> | Phone | 0:1 | Phone number of the institute |
| \_<email> | Email | 0:1 | Email of the institute |

**Address element**

An address element contains the address information of a person or an institute.

<address>  
 <street>Av. de l'Ours 1</street>  
 <postalCode>1010</postCode>  
 <city>Lausanne</city>  
 <state>Vaud</state>  
 <country>Suisse</country>  
</address>

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <address> |  | 0:1 | Address of an institute or a person |
| \_<street> | string | 1:1 | Street of the address |
| \_<postalCode> | string | 1:1 | Postal code of the address |
| \_<city> | string | 1:1 | City of the address |
| \_<state> | string | 0:1 | State of the address |
| \_<country> | string | 0:1 | Country of the address |

**Phone element**

A phone element contains a number and a type of phone number for a person or an institute.

<phone>  
 <number>0213140001</number>  
 <type>private</type>  
</phone>

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <phone> |  | 0:1 | Phone number of a person or an institute |
| \_<number> | string | 1:1 | Phone number |
| \_<type> | string | 0:1 | Phone type |

*The <type> tag is a string enumeration. It can be “private” or “professional”.*

**Email element**

An email element contains an email address and his type for a person or an institute.

<email>  
 <address>anemail@email.mail</address>  
 <type>professional</type>  
</email>

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <email> |  | 0:1 | Email of a person or an institute |
| \_<address> | string | 1:1 | Email address |
| \_<type> | string | 0:1 | Email type |

*The <type> tag is a string enumeration. It can be “private” or “professional”.*

### 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.

<covariates>

    <covariate>

        <covariateId>bodyweight</covariateId>

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

        <value>40</value>

        <unit>kg</unit>

        <dataType>double</dataType>

    </covariate>

    […]

</covariates>

**Checks:**

The covariate value is checked using the covariate definition validation of the drug file.

*In drug models, each required covariate has a covariate definition. It provides a default value and a validation formula that indicates whether the corresponding patient covariate value is fit or not. If this validation domain is not met, the drug model may not fit. If the value of a covariate is not expected, the influenced pharmacokinetics may have unverified values that may lead to implausible predictions.*

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

### Drugs

The drugs element contains some 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, the patient’s treatment, the patient’s blood samples and the targets the patient must reach.

<drugs>

    <drug>

        <drugId>rifampicin</drugId>

        <activePrinciple>something</activePrinciple>

        <brandName>somebrand</brandName>

        <atc>something</atc>

        <treatment>[…]</treatment>

        <samples>[…]</samples>

        <targets>[…]</targets>

    </drug>

    […]  
</drugs>

**Checks:**

The identifier of the drug for which an adjustment is requested must match at least one drug file.

*If there are no matching drug files, the adjustment for that drug will be dropped.*

### Treatment

The dosages are 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 describing 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 allow the dosage validation.

*<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 a dose reaches the minimum or 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 multiple analytes. Therefore, a concentration contains its associated analyte identifier (analyteId), a value and a unit.

<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 check the samples, the program will compute an a priori estimation, i.e., with the patient’s covariates instead of typical patient characteristics . Then, it will check if a sample is below or above a 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.*

### Targets

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 a corresponding active moiety (activeMoietyId), a type (targetType), a unit and some thresholds: minimum to reach (min), maximum to reach max, best to reach, inefficiency limit (inefficacyAlarm) and toxicity limit (toxicityAlarm).

<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 as the drug file, but it must not have an identical active moiety id and target type to another personalized target.

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

### XpertRequests

The requests element contains some xpertRequest elements which is the way the user tells TuberXpert what to do. This is where the last change from the original query format occurs.

<requests>

**<requestXpert>[…]</requestXpert>**

    […]

</requests>

*In the Tucuxi CLI query, the requests element contains some request elements. It is the user's way of controlling the Tucuxi computation core.. It is a highly configurable element that tells the core whether it should calculate a prediction, an adjustment or percentiles. TuberXpert does not need this element because the requests for Tucuxi computation core will be created by TuberXpert itself. Consequently, this element is not necessary. So, it is not necessarily present. Although its value is currently unused, it is still parsed if it is present.*

An xpertRequest contains two types of information:

How to generate the report?

* In English? In French? …
* XML? HTML? PDF?

What drug to adjust and how?

* The identifier of the drug to be adjusted.
* If we know when, the date of adjustment.

And for more advanced users:

* Is a loading dose/rest period allowed to reach the target faster at the beginning of the treatment?
* What type of target to use: from the query? From the drug model? What to do when there are both?
* What formulation and administration route should be used for the adjusted treatment?

**Checks:**

No particular check is made, except that the adjustment date or the different enumeration values are correct with a good format.

<requestXpert>

    <drugId>rifampicin</drugId>

    <output>

        <format>XML</format>

        <language>en</language>

    </output>

    <adjustmentDate>2018-07-06T08:00:00</adjustmentDate>

    <options>

        <loadingOption>noLoadingDose</loadingOption>

        <restPeriodOption>noRestPeriod</restPeriodOption>

        <targetExtractionOption>populationValues</targetExtractionOption>

        <formulationAndRouteSelectionOption>lastFormulationAndRoute  
                                                </formulationAndRouteSelectionOption>

    </options>

</requestXpert>

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <requestXpert> |  | 1: ∞ | The request for TuberXpert. |
| \_<drugId> | string | 1:1 | The identifier of the drug to adjust. |
| \_<output> |  | 1:1 | Specifications about the output. |
| \_\_<format> | string | 1:1 | Output format. |
| \_\_<language> | string | 1:1 | Output language. |
| \_<adjustmentDate> | date | 0:1 | Date of adjustment. |
| \_<option> |  | 0:1 | Options specifications. |
| \_\_<loadingOption> | string | 0:1 | Allow loading dose or not. |
| \_\_<restPeriodOption> | string | 0:1 | Allow rest period or not. |
| \_\_<targetExtractionOption> | string | 0:1 | Extraction option for targets. |
| \_\_<formulationAndRouteSelectionOption> | string | 0:1 | Selection of the potential formulations and routes. |

*The <format> tag is a string enumeration that allows choosing the output format. It can be “html”, “xml” or “pdf”.*

*The <language> tag is a string enumeration that allows choosing the output language. It can be “en” or “fr”.*

*The <loadingOption> tag is a string enumeration. It can be “noLoadingDose” or “loadingDoseAllowed”.*

*From Tucuxi CLI Software Usability Specification*:

* *“noLoadingDose: No loading dose can be added to the new dosage”*
* *“loadingDoseAllowed: If the current dosage is under the target, a loading dose can be added at the beginning of the new dosage to more rapidly reach the optimum.”*

*If the tag is not present, the recommendation from the drug model is used.*

*The <restPeriodOption> tag is a string enumeration. It can be “noRestPeriod” or* “*restPeriodAllowed”.*

*From Tucuxi CLI Software Usability Specification:*

* *“noRestPeriod: No rest period can be added to the new dosage”*
* *“restPeriodAllowed: If the current dosage is over the target, a rest period can be added at the beginning of the new dosage to more rapidly reach the optimum.”*

*If the tag is not present, the recommendation from the drug model is used.*

*The <targetExtractionOption> tag is a string enumeration. It can be “populationValues”, “aprioriValues”, “individualTargets”, “individualTargetsIfDefinitionExists”, “definitionIfNoIndividualTarget” or “individualTargetsIfDefinitionExistsAndDefinitionIfNoIndividualTarget”.*

*From Tucuxi CLI Software Usability Specification:*

* *“populationValues: Forces the population values to be used”*
* *“aprioriValues: Forces the a priori values to be calculated and used”*
* *“individualTargets: Only use the individual targets”*
* *“individualTargetsIfDefinitionExists: Only use the individual targets if a target definition exists”*
* *“definitionIfNoIndividualTarget Use the individual target, and if for an active moiety and a target type no individual target exists, then use the definition”*
* *“individualTargetsIfDefinitionExistsAndDefinitionIfNoIndividualTarget: Use the individual target if a target definition exists, and if for an active moiety and a target type no individual target exists, then use the definition”*

*If the tag is not present, the value “definitionIfNoIndividualTarget” is used.*

*The <formulationAndRouteSelectionOption> tag is a string enumeration.* It can be “lastFormulationAndRoute”, “defaultFormulationAndRoute” or “allFormulationAndRoutes”.

*From Tucuxi CLI Software Usability Specification:*

* *“lastFormulationAndRoute: Use only the last formulation and route used in the current treatment. If the treatment is empty, then use the default formulation and route of the drug model.”*
* *“defaultFormulationAndRoute: Use only the default formulation and route of the drug model”*
* *“allFormulationAndRoutes: Use all available formulation and routes of the drug model”*

*If the tag is not present, the value “lastFormulationAndRoute” is used.*

## Global application overview

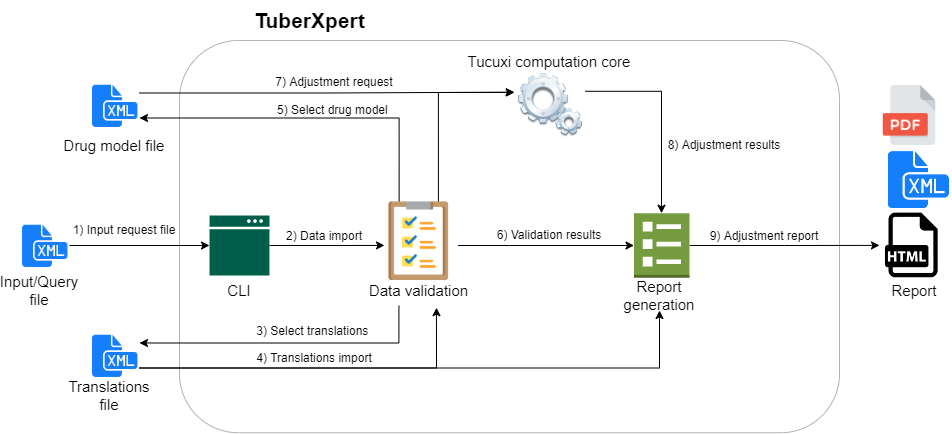


Figure 4 Global application overview and components

1. The program receives as input the patient and his treatments information in XML format.
2. The program loads the data from the query file.

This is where the validation phase begins.

1. The TuberXpert specific request is analyzed and determines the translations file to be loaded according to the required output language.
2. The translations file is loaded.
3. A drug model is selected that matches the patient information.
4. Data validation is complete and the results are ready for the report generation.
5. Based on the data validation, an adjustment request is made for Tucuxi computation core.
6. The adjustment data are provided to the report generator.
7. The adjustment report is generated according to the required output format.

The steps 3 to 9 are repeated for each drug TuberXpert request required by the user. This allows each adjustment to be processed independently ensuring that some that fail do not impact others that might succeed (e.g., a missing translation file).

## 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.

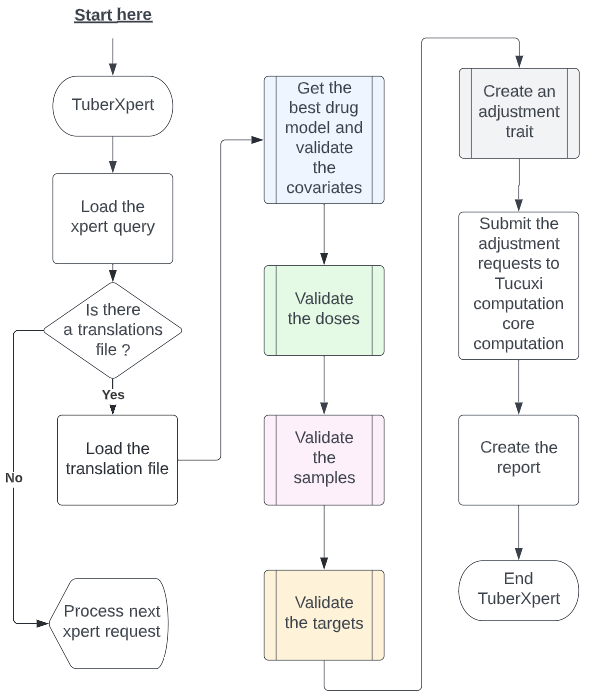


Figure 5 Program global execution

The first step is to load the query for processing. This is the only step performed once. The next flow steps are done for each TuberXpert request:

* 1st flow step) Check if there is a translation file for the required output language and load it.
* 2nd flow step) Try to get the best drug model and to validate the covariates.
* 3rd flow step) Try to validate the doses.
* 4th flow step) Try to validate the samples.
* 5th flow step) Try to validate the targets.
* 6th flow step) Try to create the adjustment trait used by Tucuxi computation core to calculate an adjustment.
* 7th flow step) Based on the trait from the last step, we are able to perform the adjustment computations to obtain: the adjustment data, the pharmacokinetics parameters of different types (typical patient, “a priori” and eventually “a posteriori”) and extrapolated steady state statistics.
* 8th flow step) Try to print the report.

Any of these steps may fail. In this case, the TuberXpert request being processed is abandoned and the next request is processed.

### Get best drug model and validate the covariates

One task of the system is to choose a drug file to use. Since the drug model selection requires knowing which covariates are present and valid, we take the opportunity to do covariate validation at the same time.

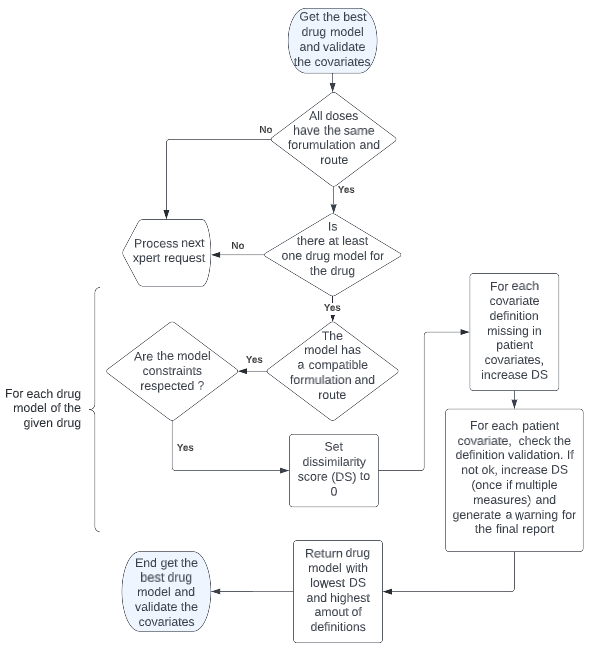


Figure 6 Process of drug model selection and covariate validation

Firstly, we will verify that all the doses of the patient’s treatment have the same formulation and administration route. Then, for each drug model available for the drug of interest, we check that the formulation and route of administration of the treatment are supported, otherwise the model is dropped. After that, we check that the constraints are respected. A drug model can say “If you have a GIST, your body weight must be bigger than 40 kg”. In other words, constraints are preliminary conditions that define if we can use the model or not. Next, we calculate a dissimilarity score DS based on the model covariate definitions.

*DS = +*

The model with the lowest dissimilarity score is chosen. In case of a tie, the model with the most covariate definitions is chosen. The method is not optimal, but it is a good starting point. What happens if two models tie perfectly, but one may be fitting better? This type of question is not considered by this algorithm. In the future, a close collaboration with pharmacologists would be necessary to determine for each drug, “how to effectively choose the drug file that fit the most to the patient for a given drug”.

*Drug model constraints and covariate definition validations do almost the same thing but have very different effects:*

* *- A constraint has the power to say that the drug model should not be used while the covariate validation simply says whether the covariate value is expected or not, without much consequence.*
* *- A constraint can consider several covariates whereas a covariate definition is more likely to consider only the value of the covariate to which it is related.*

*At this point, we can say:*

*If the input doses have the same formulation and administration route:*

* *Yes: Search for the best drug model.*
* *No: Return an error for this TuberXpert request and process the next one.*

*Is there a drug model that matches the patient’s information?*

* *No: Return an error for this TuberXpert request and process the next one.*

*In regards of the selected drug model, are some covariate definitions missing from the patient’s covariates:*

* *Yes: Use their default values.*

*In regards of the selected drug model, are the patient’s covariates supported by the covariate definitions of the drug model:*

* *Yes: Generate a warning for those covariates in the report. Maybe a double check is required.*

### Assess the doses

To adjust a dose, it is important that the doses used to perform the adjustment computation are relevant. Therefore, after loading the best drug model, the CDSS will check that every dose of the request matches the recommended dose range from the drug model.

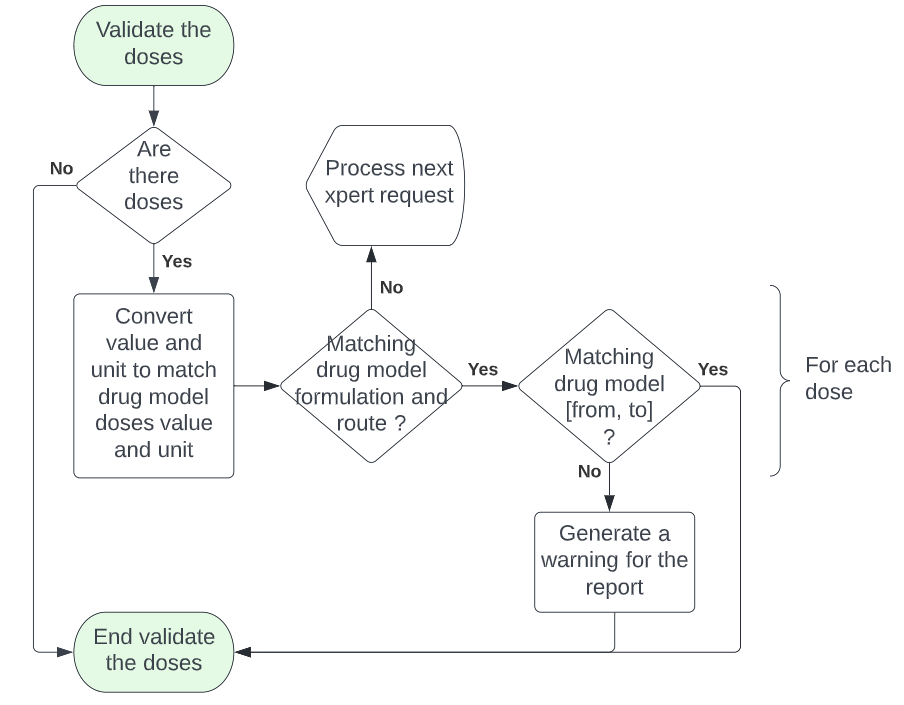


Figure 7 Process of doses validation

Every drug file has an “availableDoses”element that has a unit and a range [from, to]. This information is used to validate the doses of the treatment. The Tucuxi computation core “UnitManager” class can convert a value from a base unit into a target unit. Thus, the CDSS will be able to convert each dose and verify that they are within the range of the formulation and route of administration of the corresponding drug file.

*At this point, we can say, if the doses are normal*, *i.e., within the normal range:*

* *No: Generate a warning in the report. Maybe a double check is required.*

### 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 response of the body to the treatment.

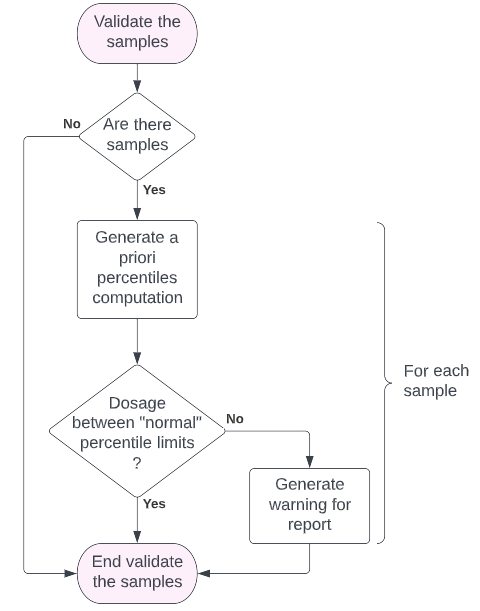


Figure 8 Process of samples validation

Using Tucuxi computation core, the system will generate an “a priori” percentile request. Considering the patient’s covariates, 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 critical warning.
* Below 10 or above 90: a normal warning.

*At this point, we can say:*

*If the samples are normal, i.e., above and below a certain percentile:*

* *Yes: No problem.*
* *No: Generate a warning in the report. Maybe a double check is required.*

### Assess the targets

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

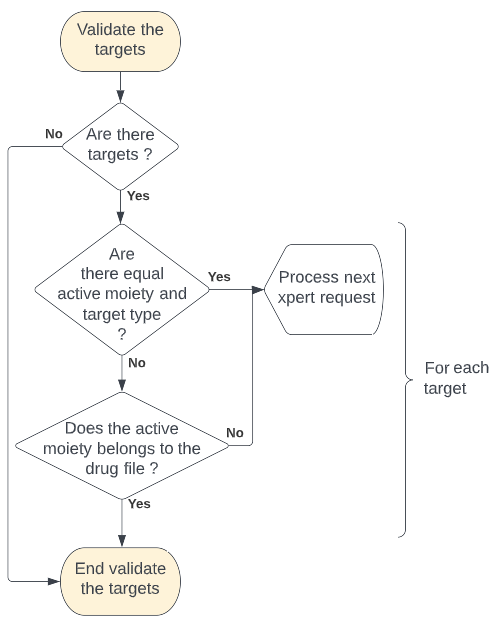


Figure 9 Process of targets validation

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 file.

*At this point, we can say:*

*If the custom targets are normal, i.e., non-redundant and using a good active moiety of the related drug model:*

* *Yes: Keep them.*
* *No: Return an error for this TuberXpert request and process the next one.*

*The possibility to write custom targets is normally intended for experienced practitioners.*

*There is no absolute rule that is easy to implement to determine what constitutes a relevant target for every 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 trait

The last step before launching an adjustment computation is to take the last decisions to prepare the adjustment trait that will be used by Tucuxi computation core to perform the adjustment.

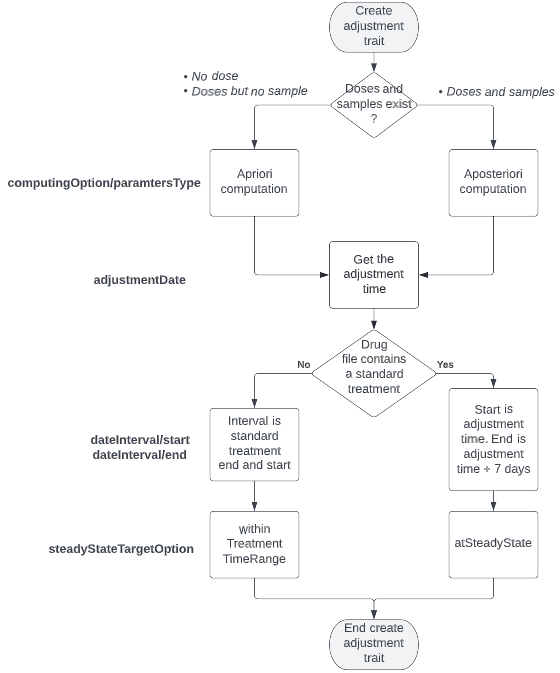


Figure 10 Process of adjustment trait creation

At that point, we need four more information to create the adjustment trait:

* What is the parameters type of the computing option?
  + If the patient has no dosage or sample, the parameters type is “a priori”.
  + If the patient has dosages and samples, the parameters type is ”a posteriori”.
* When is the adjustment?
  + If the value is set in the TuberXpert request, use it.
  + Otherwise
    - If there is a treatment in progress, use the next intake time.
    - If there is a completed treatment, from the last intake time, adds 2 \* half-life[[2]](#footnote-2) of the drug until the computation time is reached. The resulting time used.
    - If there is no treatment, it uses the computing time plus 1 hour.
* What is the interval to display?
  + If the drug model does not specify a standard treatment, the interval starts at the adjustment time and ends 7 days after the start date.
  + If the drug model specifies a standard treatment, the interval starts and ends accordingly 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 will be able to create a request with this adjustment trait. Some elements are forced by TuberXpert, but some others could be tweaked by the user using the TuberXpert custom request.

For example, here is a complete adjustment request for Tucuxi computation core (in TuberXpert, we will only use the equivalent object in C++):

<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 |
| requestId | Constant. For example, “adjustment\_” + < associated drug ID>. |
| drugId | Extracted from TuberXpert request element. |
| drugModelId | Retrieved by the TuberXpert drug model selection. |
| parametersType | From 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 | From decisions presented previously. |
| end | From decisions presented previously. |
| adjustmentDate | From decisions presented previously.. |
| bestCandidatesOption | Always *bestDosagePerInterval.* |
| loadingOption | By default, follow drug model recommendation or retrieved from the TuberXpert request element. |
| restPeriodOption | By default, follow drug model recommendation or retrieved from the TuberXpert request element. |
| steadyStateTargetOption | From decisions presented previously. |
| targetExtractionOption | By default, definitionIfNoIndividualTarget or retrieved from the TuberXpert request element. |
| formulationAndRouteSelectionOption | By default, lastFormulationAndRoute or retrieved from the TuberXpert request element. |

## Report

This chapter discusses the forms that the output will take. It is expected to be in the form of an XML document, an HTML page or a PDF document. 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 to emphasis what information is necessary and deduce what will be inserted in the XML document.

Let’s assume that the root element of the XML report is *tuberxpertResult*. All the following XML structures will be inserted as direct children.

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <tuberxpertResult> |  | 1:1 | The data in the XML report. |

### Header

This first part contains the date of computation as well as general facts about the drug concerned, such as its identifier, the last dose administered, and the drug model selected for this adjustment.

**HTML representation:**

Une image contenant table

Description générée automatiquement

**XML organization:**

**<computationTime>2022-08-04T07:00:00</computationTime >  
<language>en</language>  
<drug>  
 <drugId>rifampicin</drugId>  
 <lastDose>  
 <value>800</value>  
 <unit>mg</unit>  
 </lastDose>  
 <drugModelId>ch.tucuxi.rifampicin.svensson2017</drugModelId>  
</drug>**

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <computationTime> | date | 1:1 | The simulated present time. |
| <language> | string | 1:1 | The language of the translated elements. |
| <drug> |  | 1:1 | Minimum drug information. |
| \_<drugId> | string | 1:1 | The drug identifier. |
| \_<lastDose> |  | 1:1 | Last dose information. |
| \_\_<value> | double | 0:1 | The value of the last dose. |
| \_\_<unit> | string | 0:1 | The unit of the last dose. |
| \_<drugModelId> | string | 1:1 | The selected drug model identifier. |

The computationTime has the same value as the date element from the TuberXpert query.

*The language is an enumeration. The values are “en” or “fr”.*

Even if it is not present in the HTML version of the report. The desired output language will be included in the XML version with the language element. Unlike the HTML/PDF versions which are not likely to be reprocessed again, the XML version will probably be reprocessed again by another program. It is therefore a good idea to add any information needed for further processing. Without the language, we would not be able to know the language of the sentences and we would not be able to effectively add certain changes.

Another consequence of the fact that the XML version can be reprocessed is that not all standardized values will be translated. One can easily think of formulations, administration paths or target types.

If we turn the problem the other way around, the only things we want to translate are the warning messages because those are the only non-normalized data we have, the covariate names, and the covariate descriptions because we don't want to incorporate all the available translations for the latter two values.

### Admin

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

**HTML representation:**

Une image contenant table

Description générée automatiquement

**XML organization:**

The XML output should stick to the structure of the admin element from the TuberXpert query.

**<admin>  
 <mandator>   
 <person>[…]</person>  
 <institute>[…]</institute>  
 </mandator>  
 <patient>  
 <person>[…]</person>  
 <institute>[…]</institute>  
 </patient>  
 <clinicalDatas>[…]</clinicalDatas>  
</admin>**

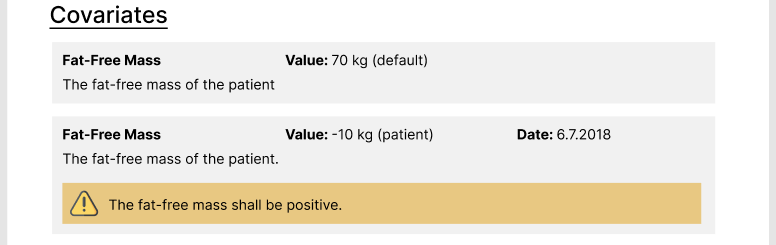
The detailed description of this element is available [here](#_Admin).

### 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 from the drug model.

If the covariate is from the patient, it includes the measurement date. Also, if it does not respect the drug model validation, a warning is displayed.

**HTML representation:**



Here, there is an example of a default “fat-free mass” covariate. There is an alternative representation with a warning if the value from the patient does not meet the requirements and the measurement date.

**XML organization:**

**<covariates>  
 <covariate>  
 <covariateId>ffm</covariateId>  
 <name>Fat-Free Mass</name>  
 <value>-10</value>  
 <unit>kg</unit>  
 <dataType>kg</dataType>  
 <desc>The fat-free mass of the patient</desc>  
 <source>default / patient</source>  
 <date>Error message</ date >   
 <warning level=’normal’>Error message</warning>  </covariate>  
 […]  
</covariates>**

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <covariates> |  | 1:1 | Covariates specification. |
| \_<covariate> |  | 1: ∞ | Description of a covariate. |
| \_\_<covariateId> |  | 1:1 | The unique identifier of the covariate. |
| \_\_<date> | date | 0:1 | If from the patient, the date of measure. |
| \_\_<name> | string | 1:1 | The translated name of the covariate. |
| \_\_<value> | string | 1:1 | The value of the covariate. |
| \_\_<unit> | string | 1:1 | The unit of the covariate. |
| \_\_<dataType> | string | 1:1 | The covariate value data type. |
| \_\_<desc> | string | 1:1 | The translated description of the covariate. |
| \_\_<source> | string | 1:1 | The source of the covariate (“patient” or “default”) |
| \_\_<warning> | string | 0:1 | The translated warning message if the validation is not respected. |

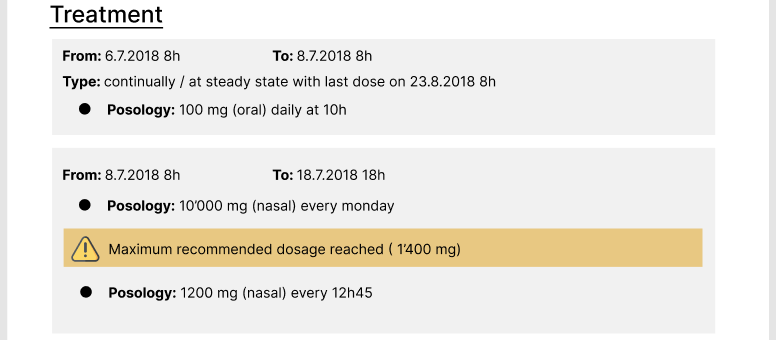
*The warning element always has a "level" attribute with the value "normal".*

*The data type is an enumeration. The values are “int”, “double”, “bool” or “date”.*

### Treatment and checks

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

**HTML representation:**



The challenge is to translate the treatment of the TuberXpert query into something visual.

The base dosage of a dosage time range element can be a dosage loop or a dosage at steady state. If this is the case, the “Type” indication is displayed under the time range dates.

* Loop: continually
* At steady state: at steady state with last dose on

For the doses, the main idea is to display an indication of the posology near each dose according to their type:

* Lasting: Every + <interval>
* Daily: Daily at + <time>
* Weekly: Every <day> + at + <time>
* Repeat: <number of repetitions> time(s)

Since dosages can be nested, each achieved dosage will be added to the posology of the final dose. For example, if a daily dose is nested within a dosage repeat, the posology may be:

* 100g (oral) daily at 8h15, 4 time(s)

**XML organization:**

The output will return the treatment node as it entered with a small difference. Each suspicious dose element in lasting/daily/weekly dosage node will receive an optional warning element with an error value.

For example, with a lastingDosage, the following situation could be possible:

*As already mentioned, the treatment is a complex element. This element follows the same structure as the element defined by Tucuxi. If you are interested in more details, see the file "tiersdoc/Tucuxi\_CLI\_Usability\_Specification.pdf". The only element that changes is the dose.*

**<treatment>  
 […]  
 <lastingDosage>   
 […]  
 <dose>  
 <value>400</value>  
 <unit>mg</unit>  
 <infusionTimeInMinutes>60</infusionTimeInMinutes>  
 <warning level=’normal’>Error message</warning>   
 </dose>  
 […]  
 </lastingDosage>  
</treatment>**

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <dose> |  | 1:1 | Description of a dose. |
| \_<value> | decimal | 1:1 | The value of the dose. |
| \_<unit> | string | 1:1 | The unit of the dose. |
| \_<infusionTimeInMinutes> | decimal | 1:1 | The infusion time (min) of the dose. |
| \_<warning> | string | 0:1 | Warning that describes the problem of the dose value. |

*The warning element always has a "level" attribute with the value "normal".*

### 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 it 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:**



*For now, the Tucuxi computation core works with a single analyte. The day when the multiple analytes feature will be on rail, we must consider sorting the samples by analytes.*

**XML organization:**

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

For example, the following situation could be possible:

**<samples>  
 <sample>  
 <sampleId>sample\_1</sampleId>  
 <sampleDate>2018-07-06T08:00:00</sampleDate>  
 <concentrations>  
 <concentration>  
 <analyteId>rifampicin</analyteId> <value>7</value>  
 <unit>mg/l</unit>  
 <percentile>50</percentile>  
 <warning level=’normal/critical’>Error message</warning>   
 <!-- Optionally--> <concentration>  
 […]  
 </concentrations>  
 </sample>  
 […]  
</ samples >**

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <samples> |  | 1:1 | The samples of the drug. |
| \_<sample> |  | 0: ∞ | Description of a sample. |
| \_\_<sampleId> | int | 1:1 | The unique identifier of the sample. |
| \_\_<sampleDate> | date | 1:1 | The date of the sample. |
| \_\_<concentrations> |  | 1:1 | Concentrations. |
| \_\_\_<concentration> |  | 1:∞ | Concentration specification. |
| \_\_\_\_<analyteId> | string | 1:1 | The unique identifier of the analyte. |
| \_\_\_\_<percentile> | int | 1:1 | The group number over the 99 percentiles. |
| \_\_\_\_<value> | decimal | 1:1 | The value of the concentration. |
| \_\_\_\_<unit> | string | 1:1 | The unit of the value. |
| \_\_\_\_<warning> | string | 0:1 | Warning of the sample percentile. |

*The warning element always has a "level" attribute:*

* *“critical”: If the sample is below 5 or above 95.*
* *“normal”: Otherwise.*

### Best adjustments and suggested adjustment

This section is divided into two main parts:

* The best adjustments per interval.
* The suggested adjustment.

Each of these parts begins the same way:

* A brief introduction.
* A graph that displays the adjustment predictions.
* The treatments that match to the displayed predictions.

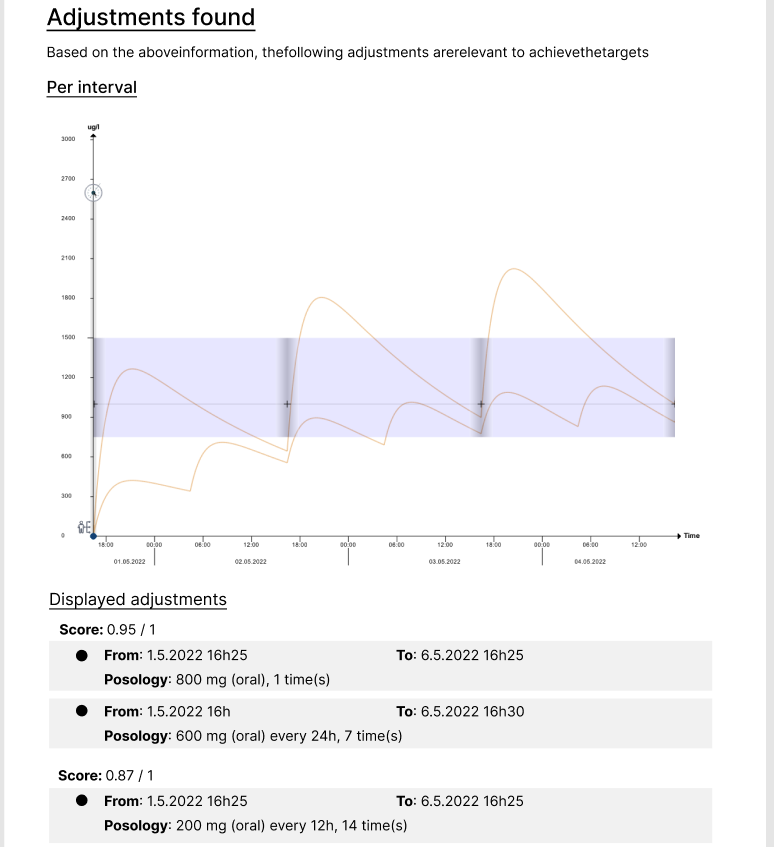
Each adjustment receives a score assigned by Tucuxi computation core. Per adjustment, the targets used get a score according to the following formula:

*“The idea is that this score reflects the squared relative departure of the predicted concentration from the target value at the corresponding time, expressed in the scale of the relative radius of the therapeutic interval.”*

In other words, if the prediction tends towards the target, the score will tend towards 1, otherwise towards 0.The higher the score, the better.

Finally, when each target has a score, the adjustment score is obtained from the average of all target scores.

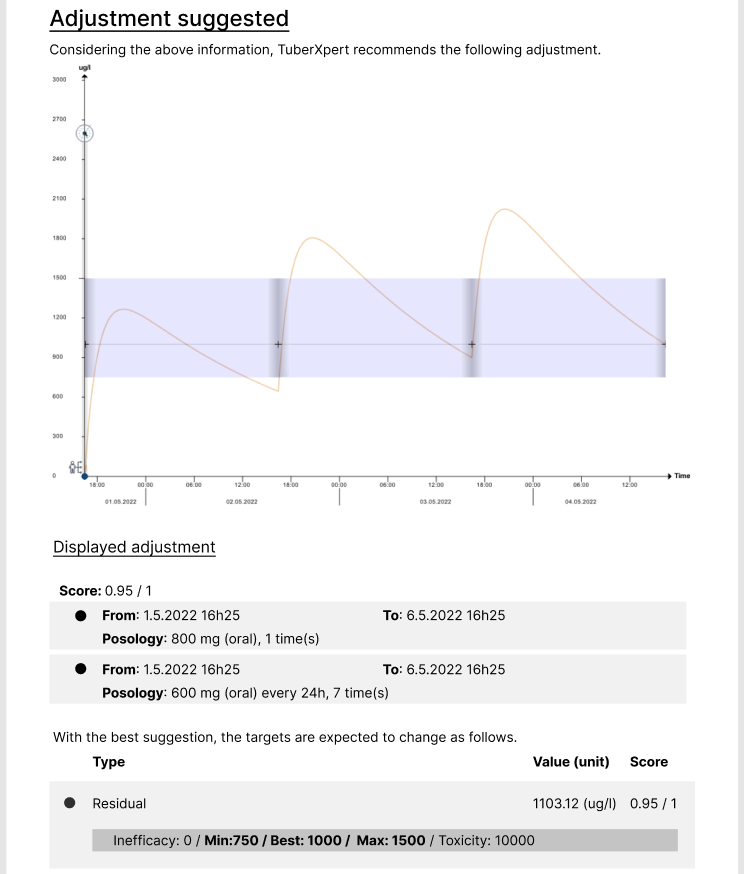
**HTML representation of the best adjustments:**



We can see the score at the top of the adjustment treatments. The way the adjusted treatments are displayed is the same as the patient treatment is displayed.

It the Tucuxi computation core finds only one adjustment, this part of the HTML will be dropped because it will strictly display the same graph and the same adjusted treatment as the next part.

**HTML representation of the suggested adjustments:**

****

This second part highlights the adjustment with the highest score already displayed in the best adjustments per interval.

Additionally, it displays the achievement of the targets. For each target, it lists its type, its predicted value, its score and the inefficacy and toxicity limits.

**XML organization:**

The XML content that contains information for the previous two parts should be approximately the same as the dataAdjustmentelement of the Tucuxi computation response. For example, the following XML structure might be possible:

**<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>**

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

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <dataAdjustment> |  | 1:1 | The data of the adjustments found. |
| \_<analyteIds> |  | 1:1 | Analyte id container. |
| \_\_<analyteId> | string | 0:∞ | The unique identifier of an analyte. |
| \_<adjustments> |  | 1:1 | Adjustments container. |
| \_\_<adjustment> |  | 1:∞ | Adjustment description. |
| \_\_\_<score> | double | 1:1 | The score of the adjustment. |
| \_\_\_<targetEvaluations> | TargetEvaluations | 1:1 | Target Evaluations. |
| \_\_\_<dosageHistory> | DosageHistory | 1:1 | Dosage history. |
| \_\_\_<cycleDatas> | CycleDatas | 1:1 | Cycle datas. |

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <targetEvaluation> |  | 1:∞ | TargetEvaluation specification. |
| \_<targetType> | string | 1:1 | The type of the target. |
| \_<analyteId> | string | 1:1 | The unique identifier of an analyte. |
| \_<unit> | string | 1:1 | The unit of the value. |
| \_<value> | double | 1:1 | The value of the target. |
| \_<score> | double | 1:1 | The score of the target. |
| \_<min> | double | 1:1 | The minimum value to reach. |
| \_<best> | double | 1:1 | The ideal value. |
| \_<max> | double | 1:1 | The maximum value to not reach |
| \_<inefficacyAlarm> | double | 1:1 | The inefficacy limit. |
| \_<toxicityAlarm> | double | 1:1 | The toxicity limit. |

*The target type is an enumeration. The values are “peak”, “residual”, “mean” or “auc”, “aucOverMic”, “timeOverMic”, “aucDividedByMic” or “peakDividedByMic”.*

The dosage history element contains the list of dosage to be followed for the adjustment. Its structure is the same as the one in the query dosage history.

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

The predictions for a given adjustment are contained in cycleData elements in the cycleDatas element.

*This element follows the same structure as the element defined by Tucuxi. If you are interested in more details, see the file "tiersdoc/Tucuxi\_CLI\_Usability\_Specification.pdf".*

A cycleData represents the predictions between two intakes. It contains a **start** and **end** date, the **unit** of the values, a times element that contains a comma-separated list of times, in hours, starting from zero and a values element that contains a comma-separated list of concentration values. The number of times matches the number of values.

**<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>**

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <cycleData> |  | 0:∞ | TargetEvaluation specification. |
| \_<start> | date | 1:1 | The start date of the cycle data. |
| \_<end> | date | 1:1 | The end date of the cycle data. |
| \_<unit> | string | 1:1 | The unit of the concentration values. |
| \_<times> | double | 1:1 | A list of time. |
| \_<values> | double | 1:1 | A list of concentration values. |

*The field values contains a comma-separated list of concentration values. The number of values matches the number of times.*

*of times.*

*The field times contains a comma-separated list of times, in hours, starting from 0. To get the real time of a concentration, this time should be added to start.*

### Computation facts

This is the last section of the report. It contains general facts about the computation, such as the pharmacokinetics parameters, some steady-state predictions and the covariates used by the computation core. This second list of covariates is useful because it allows to double check the covariates. In addition, some covariates can be calculated on running time based on the patient’s other covariates. Thus, it is possible to see if there are any calculated covariates.

**HTML representation:**

Une image contenant table

Description générée automatiquement

**XML organization:**

<parameters>  
 <typical>  
 <parameter>  
 <id>Ka</id>  
 <value>0.609</value>  
 </parameter>  
 […]  
 </typical>  
 <apriori>  
 <parameter>[…]</parameter>  
 […]  
 </apriori>  
 <aposteriori>  
 <parameter>[…]</parameter>  
 […]  
 </aposteriori>  
</parameters>  
<statistics>  
 <auc24>36863.6</auc24>  
 <peak>2023.58</peak>  
 <residual>999.701</residual>  
</statistics>  
<computationCovariates>  
 <computationCovariate>  
 <id>bodyweight</id>  
 <value>40.000000</value>  
 </computationCovariate >  
 […]  
</computationCovariates>

The parameters element contains a listing of each pharmacokinetics parameter for each computation type: typical, apriori and aposteriori. Depending on the parameters type of the request, additional adjustment requests are required to obtain all parameters. For example, if the current adjustment is “a posteriori”, additional adjustment requests are made to get the “typical*”* and “a priori”parameters.

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <parameters> |  | 1:1 | The pharmacokinetic parameters. |
| \_<typical> |  | 1:1 | The parameters of the typical patient. |
| \_\_<parameter> |  | 1:∞ | A pharmacokinetic parameter. |
| \_\_\_<id> | string | 1:1 | The identifier of the parameter. |
| \_\_\_<value> | string | 1:1 | The value of the parameter. |
| \_<apriori> |  | 1:1 | The parameters “a priori”. |
| \_\_<parameter> |  | 1:∞ | A pharmacokinetic parameter. |
| \_\_\_<id> | string | 1:1 | The identifier of the parameter. |
| \_\_\_<value> | string | 1:1 | The value of the parameter. |

|  |  |  |  |
| --- | --- | --- | --- |
| \_<aposteriori> |  | 0:1 | The parameters “a posteriori”. |
| \_\_<parameter> |  | 1:∞ | A pharmacokinetic parameter. |
| \_\_\_<id> | string | 1:1 | The identifier of the parameter. |
| \_\_\_<value> | string | 1:1 | The value of the parameter. |

The statistics element contains the predictions at steady state. The steady state can be approximated using the formula:

The half-life and the multiplier are temporal considerations located in the drug model. The half-life is the time it takes for the amount of a drug’s active substance to reduce by half.

These statistics are computed in each “cycleData” returned by the Tucuxi computation core. It is thus necessary to make another adjustment request, but in a light way, to obtain them.

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <statistics> |  | 1:1 | The pharmacokinetic parameters. |
| \_<auc24> | double | 1:1 | Area under the curve on 24h. |
| \_<peak> | double | 1:1 | Peak concentration. |
| \_<residual> | double | 1:1 | Residual concentration. |

The computationCovariates element lists all the covariates used during the computation, represented by a computationCovariate element which contains a value and the id of covariate. These values can be found in any adjustment data returned by Tucuxi computation core.

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| <computationCovariates> |  | 1:1 | The covariates during computation. |
| \_<computationCovariate> |  | 1:∞ | A computation covariate. |
| \_\_<id> | string | 1:1 | The identifier of the covariate. |
| \_\_<value> | double | 1:1 | The value of the covariate. |

# Implementation

With a walk-trough of an execution, this chapter presents the classes in TuberXpert and their utility.  
Sometimes there are UML diagrams to better visualize the interactions. However, these diagrams are not meant to fully present the content of the classes. They will only contain what is necessary to understand the purpose of the class.

*For a complete description of the classes, there is an automatically generated code documentation available in   
“public/tuberxpert\_code\_documentation”.*

The TuberXpert implementation is available in “*dev/tucuxi-tuberxpert*”. Inside this directory is the file *“tucuxi-tuberxpert.pro”* which allows to open the project in Qt Creator.

The organization of the project is as follows:

* It includes the source code of Tucuxi CLI.
* The source code of TuberXpert is accessible by the “tuberxpert” inclusion.
* The master program is in a single file “tuberxpert.cpp.” It contains the main function that drives the execution.
* The “Other files” folder contains some XML translations files and some XML validation files.

Figure 11 TuberXpert project in Qt Creator

At the beginning of each chapter, there is a table that summarizes the important files. To lighten the annotations, we assume that all paths are relative to *“dev/tucuxi-tuberxpert”*.

## Run TuberXpert

|  |
| --- |
| Concerned files |
| /src/tuberxpert.cpp |
| /src/tuberxpert/computer.h |

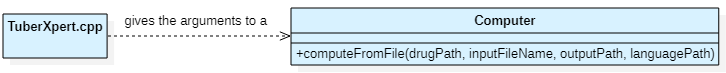


Figure 12 Starting class diagram of the TuberXpert program

TuberXpert works as CLI that expects the following arguments:

**-d <path/to/drug/models>** to indicate where the drug model files are. There is a basic collection in *“dev/tucuxi-drugs/drugfiles”*.

**-i </path/to/xml/query>** to indicate the location of the TuberXpert query to process. There are a few examples in *“dev/tucuxi-tuberxpert/xml/query”*.

**-o </path/to/directory>** to indicate where the result reports should be printed.

**-l </path/to/translations/files>** to indicate where the translations files are. Actually, TuberXpert supports English and French in *“dev/tucuxi-tuberxpert/language/”*.

The main file "tuberxpert.cpp" analyses the arguments and transmits them to the TuberXpert computer which launches the TuberXpert flow.

*The class Computer also provides a “computeFromString” method that accepts the contents of a TuberXpert query as a string instead of the file name. Therefore, it could be easily implemented on web server.*

## Query import

|  |
| --- |
| Concerned files |
| /src/tuberxpert/query/admindata.h |
| /src/tuberxpert/query/xpertquerydata.h |
| /src/tuberxpert/query/xpertqueryimport.h |
| /src/tuberxpert/query/xpertrequestdata.h |

The first task of the TuberXpert computer is to import the TuberXpert query. For a description of the query file, see Input – TuberXpert query file. But first, let's look at all the classes involved during the import.

*The structure of the query can be validated with the file "tuberxpert\_computing\_query.xsd" in "dev/tucuci-tuberxpert/xml/query".*

### Administrative data

The AdminData class provides all the information of the admin element of the TuberXpert query.

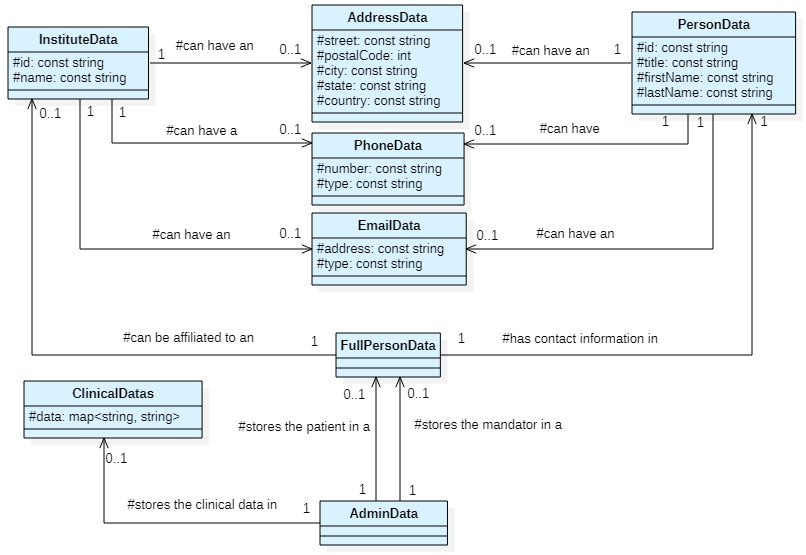


Figure 13 Class diagram of administrative data

The class modeling follows the same structure as the query file. The admin element is represented by the AdminData class. The ClinicalDatas class contains a map for each clinicalData found in the clinicalDatas element. The attribute “key” is used as the input key of the map and the value of clinicalData as the input value of the map. The FullPersonData class is used for the mendator and the patient elements. The PersonData and InstituteData classes contain data from the person and institute elements. The AddressData, the PhoneData and EmailData classes encapsulate the information contained in the address, phone and email elements.

### Xpert request data

The XpertRequestData class provides all the information of an xpertRequest element of the TuberXpert query.

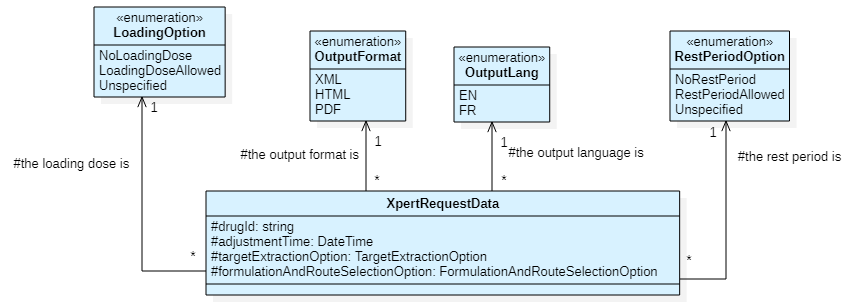


Figure 14 Class diagram of the requestXpert data

The xpertRequest element is represented by the XpertRequestData class. The output format and language are represented by the enumerations OutputFormat and OutputLang. The loadingOption and restPeriodOption values are translated into the LoadingOption and the RestPeriodOption enumerations. The value “Unspecified” is used when the user does not explicitly allow or disallow these options. This means that the recommendations of the drug model should be used. Finally, the targetExtractionOption and the formulationAndRouteSelectionOption use the enumerations implemented in Tucuxi computing core. Thus, the values are not displayed on this UML.

### Xpert query data

The XpertQueryData class is the C++ equivalent of the TuberXpert xml query.

Since the TuberXpert query is a specialization of the Tucuxi query, the TuberXpert query class inherits from the Tucuxi query class. Therefore, all the structure for the common elements is ready, we just need the AdminData and XpertRequestData class to store the additional elements

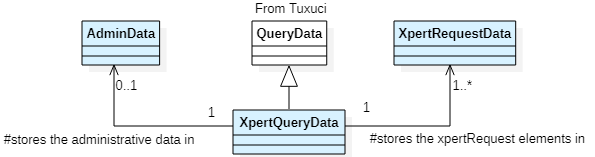


Figure 15 Class diagram of the TuberXpert query

### Xpert query import

The XpertQueryImport class is responsible for importing the TuberXpert query. In the same way that the TuberXpert query class inherits from the Tucuxi query class, the importer inherits from the Tucuxi importer.

In this way, we can use the existing implementation for the common elements.

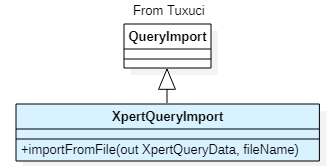


Figure 16 Class diagram of the TuberXpert query importer

*In response to the “computeFromString” method of the Computer class, the class XpertQueryImport also provides an “importFromString” method that accepts the contents of a TuberXpert query as a string instead of the file name. Therefore, it could be easily implemented on web server.*

*Additionally, the content of the TuberXpert query could be validated with the file “dev/tucuxi-tuberxpert/xml/query/tuberxpert\_computing\_query.xsd”.*

### Simplified import overview

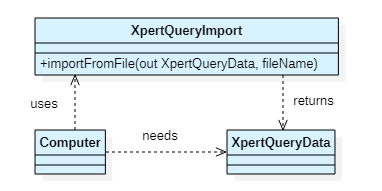


Figure 17 TuberXpert query import simplified class diagram

The TuberXpert computer must import the TuberXpert query as an XpertQueryData which is the C++ equivalent of the TuberXpert query. To do this, the TuberXpert computer uses the XpertQueryImport class, specially designed for the import of XpertQueryData.

## Create the results holders

|  |
| --- |
| Concerned files |
| /src/tuberxpert/result/abstractvalidationresult.h |
| /src/tuberxpert/result /covariatevalidationresult.h |
| /src/tuberxpert/result /dosevalidationresult.h |
| /src/tuberxpert/result /samplevalidationresult.h |
| /src/tuberxpert/result /xpertqueryresult.h |
| /src/tuberxpert/result /xpertrequestresult.h |
| /src/tuberxpert/query /xpertquerytocoreextractor.h |

Before we begin any execution of the flow steps, we need a way to store the information needed to print the reports. Let's explore this implementation from the macro to the micro level.

### Xpert query result and xpert request result

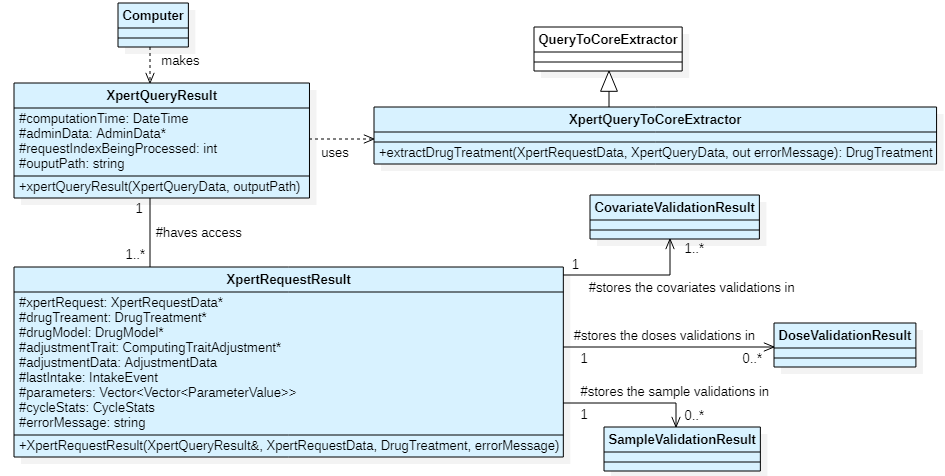


Figure 18 Xpert query result and xpert request results class diagram

When the TuberXpert computer has successfully imported the query in an XpertQueryData, it creates the objects that will be used during the whole process: from the drug model selection until the report generation.

**XpertQueryToCoreExtractor**

This class is in charge of creating a DrugTreatment from an XpertQueryData based on a drug identifier in an XpertRequestData. In fact, this extraction phase is necessary to have objects that can be processed by the Tucuxi calculation core. A drug treatment contains the patient's DosageHistory, PatientCovariate objects, Targets objects and Sample objects.

We have inherited QueryToCoreExtractor to add some checks before extracting the treatment. For example, if we try to extract a treatment from a drug that does not exist or exists multiple times in the XpertQueryData, there will be a specific error message that will be logged.

**XpertQueryResult**

The idea behind this class is to store the common data that are needed by all the XpertRequestResult objects.

At creation, for each xpertRequest, the XpertQueryResult extracts the related processing, creates an XpertRequestResult and places the potential error returned by XpertQueryToCoreExtractor.

**XpertRequestResult**

The XpertRequestResult class is the key class of TuberXpert. This class will go through each step of the flow, provide all the necessary information and gather their results to finally generate the adjustment report of an xpertRequest.

When it is created, it only contains the reference to the XpertRequestResult that created it and its drug treatment.

### Covariates, doses and samples validation results

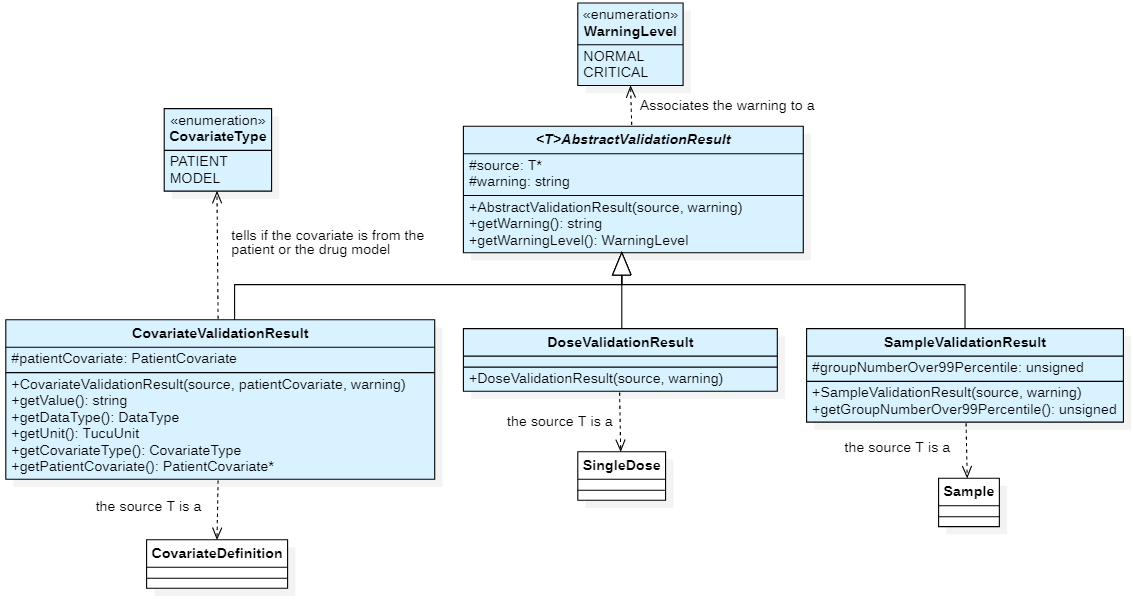


Figure 19 Covariates, doses and samples validation results class diagram

If we remember correctly, we need to provide a validation to the [covariates](#_Get_best_drug), the [doses](#_Assess_the_doses) and the [samples](#_Assess_the_samples). This is the purpose of these three classes. The pattern is to have classes that provides additional information to already existing objects.

*A good question to ask is: why are we using the composition instead of the inheritance ? For two reasons:*

1. *As we will see later in this section, there is phase that extracts the XpertQueryData into Treatment object that contain PatientCovariate, SingleDose and Sample objects usable by Tucuxi computation core. We do not have control over this phase and it was not necessary to rewrite it for our needs.*
2. *For the CovariateDefinitionResult objects, we need to know which drug model we are using with the given treatment. This not possible to know before extracting the treatment from the XpertQueryData.*

*So, with the composition we have more flexibility.*

**Covariate validation result**

One important thing to understand is that when a prediction is made, the Tucuxi computation core always needs at least one value for each associated drug model covariate definition. However, the source of the covariate makes no difference, whether it comes from the patient or from the default values of the drug model definition. So, we have to do it ourselves. Once we select our drug model, this class will allow us to make the difference while having the ability to attach warning messages if necessary.

There is going to be one object per covariate definition that is not defined in the patient covariates. In this case, the PatientCovariate is null pointer, there is no error message, the getCovariateType method will return "model" and the other methods will return the default values provided by the covariate definition.

There is going to be one object per patient covariate that corresponds to a drug model covariate definition. In this case, the PatientCovariate is not null pointer, there could be an error message if the covariate value does not meet the validation of the covariate definition. The getCovariateType method will return "patient" and the other methods will return the values provided by the patient covariate.

In any case, the warning level is always “normal”.

This object allows to be very flexible with the covariate , whether or not there is a corresponding patient covariate. We can get the type of a covariate, its value and its warning message.

**Dose validation result**

This class simply allows to link a warning message to a dose.

In any case, the warning level is always “normal”.

**Sample validation result**

For each patient blood sample, this class stores the position of the sample relative to the 100 groups formed by the 99 possible percentiles from 1 to 100. For example, if the sample is before the first percentile, its position will be 1. If its position is 100, the sample is above the 99 percentiles.

In addition, there may be a warning message. Depending on the position of the sample, the warning level is either "normal" or "critical":

* "critical": if the position is <= 5 or > 95.
* "normal": otherwise.

*Do you remember? This validation results class looks suspiciously like a class that provides all the information we need for the report. Take a look at the possible XML output of* [*covariates*](#_Covariates_and_checks)*,* [*doses*](#_Treatment_and_checks) *and* [*samples*](#_Samples)*.*

## Select the correct flow steps

|  |
| --- |
| Concerned files |
| /src/tuberxpert/flow/abstract/abstractxpertflowstep.h |
| /src/tuberxpert/flow/abstract/abstractxpertflowstepprovider.h |
| /src/tuberxpert/flow/general/generalxpertflowstepprovider.h |

At that time, the TuberXpert computer created the holders of the results that do not contain much :

* XpertQueryResult: the administrative data, the computation time, the output path and the index of the XpertRequestResult being processed.
* XpertRequestResult: The TuberXpert request for which this object contains results, the drug treatment for the TuberXpert request.

Now, thanks to an abstract factory system, we are going to get a provider of the flow steps that we are going to execute for a given XpertRequestResult.

*Do you remember? TuberXpert is developed in a general way without targeting the behavior of a specific drug. In the future, with this abstract factory system, we can easily imagine having a custom flow step provider that offers a more specific implementation than the general system.*

### Abstract xpert flow step and abstract xpert flow step provider

Une image contenant texte

Description générée automatiquement

Figure 20 Abstract factory class diagram

**AbstractXpertFlowStep**

The idea is that all the steps in the flow have the same interface to be easily replaced or extended. Thanks to the XpertRequestResult which has access to all the data to be validated and the results of the validation, this is possible. Therefore, a flow step to be executed needs to inherit from AbstractXpertFlowStep.

**AbstractXpertFlowStepProvider**

This is the class we need to inherit from to be a step provider. As we can see, there is one getter method per flow step we plan to execute.

*Do you remember? It seems that the AbstractXpertFlowStepProvider has the methods that return what we need to execute the steps that we have planned in the section* [*Program execution flow*](#_Program_execution_flow)*.*

*You get the idea, to fully process an xpertRequest, we have to go through all these steps.*

### Simplified overview of the flow step provider selection

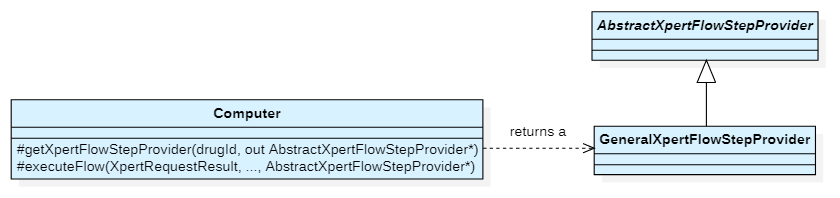


Figure 21 Simplified class diagram of the TuberXpert computer retrieving the flow steps

After creating the results holders, TuberXpert starts processing each XpertRequestResult with the executeFlow method, one after the other. Before, it needs to select the correct XpertFlowStepProvider that fit the most to the related drug. The getXpertFlowStepProvider method is in charge to return a concrete instance of an AbstractXpertFlowStepProvider that correspond to the given drug identifier.

*In fact, the "getXpertFlowStepProvider" method returns a GeneralXpertFlowStepProvider in all cases. But in the future, if a drug needs a flow that differs a little bit, everything has been provided to make this possible.*

## Process an XpertRequestResult

|  |
| --- |
| Concerned files |
| /src/tuberxpert/language/\* |
| /src/tuberxpert/flow/general/\* |
| /src/tuberxpert/exporter/\* |

At the very beginning of the executeFlow method of the TuberXpert computer, before executing the flow steps (drug model selection with covariate validation until report generation), the first thing to do is to load the translations into a singleton that can be used anywhere and at any time. This is especially useful for translating dose and sample warnings and HTML/PDF report content. First, let's see what a translation file is.

### Load a translations file

A translations file is an XML that contains a list of translation element. Each element has an attribute that stores a key, and a value that is the actual translation.

**<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<translations xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”  
 xsi:noNamespaceSchemaLocation=”translations\_file.xsd”>**

<translation key=”A\_key” >A translation</entry>  
 […]

**</translations>**

|  |  |  |  |
| --- | --- | --- | --- |
| Tag name | Format | Occ. | Description |
| < translations > |  | 1:1 | Root element |
| \_< translation > | string | 0: ∞ | A translation to be retrieved thanks to the key attribute |

**Language manager**

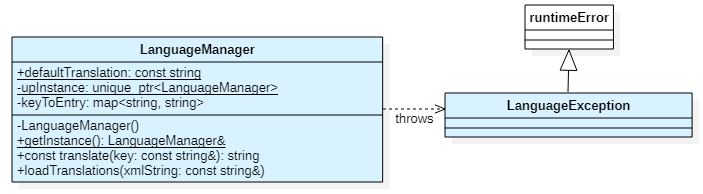


Figure 22 Class diagram of the language manager

The language manager is a singleton class that is responsible for maintaining the translations during the processing of an XpertRequest during the entire flow. The loadTranslations method allows to load a translations file as a string. The translate method is used to retrieve the translation for a given key. If the key does not exist, a default translation string is returned. If the language manager fails to load the translations, it returns a LanguageException.

**Simplified overview**

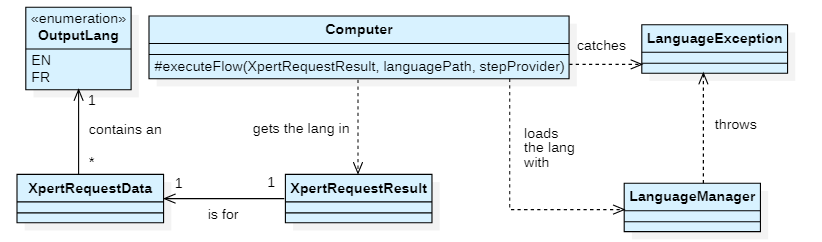
****

Figure 23 Simplified class diagram of the TuberXpert computer loading a language

The TuberXpert computer gets the language to be loaded through the XpertRequestResult it is processing. It converts the output language into a string. Then it tries to read the corresponding translation file in the directory pointed to by the languagePath argument and loads the contents of the resulting string with the LanguageManager. If it catches a LanguageException, the processing of the XpertRequestResult is aborted.

*Additionally, the content of the translations file could be validated with the file “dev/tucuxi-tuberxpert/language/translations\_file.xsd”.*

*You may ask: How do I add a new language? Check the document "tuberxpert\_add\_language.pdf" in annex.*

### Validate the covariates and select the best drug model

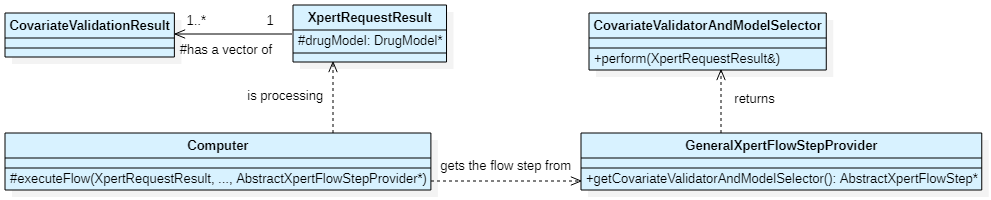


Figure 24 Covariates validation and drug model selection class diagram

The [validation of the covariate and the drug model selection](#_Get_best_drug) involves all these classes. If the execution of the CovariateValidatorAndModelSelector class is successful, it places a vector of CovariatesValidationResult and the selected drug model in the XpertRequestResult.

*Therefore, we know:*

* *Which drug model best fits the patient's covariates.*
* *Which covariates the patient does not define. The default value for the definition will be used.*
* *Which of the patient's covariates are useful for the selected drug model and whether they meet the validation of the corresponding definition.*

The warning message of a CovariateValidationResult is taken directly from the error message of a covariate definition validation.

When we generate the final report, to get the covariate information, we will only need to iterate over the CovariatesResults vector.

*What about the translation of the name, description and validation error of the covariate? Tucuxi is here to help us. All this information is contained in the covariate definition accessible in each CovariateValidationResult. When the best drug model is found, we check that each of these values is available in the required language or at least in English. If not, the processing of the XpertRequestResult is dropped.*

### Validate the doses

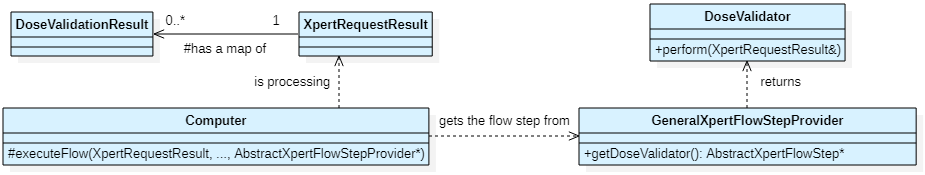


Figure 25 Doses validation class diagram

The [validation of the doses](#_Assess_the_doses) involves all these classes. If the execution of the DoseValidator class is successful, it places a map of DoseValidationResult in the XpertRequestResult.

*Why a map and not a vector? Because we don't need a validation result for every dose. We only need a validation for doses that exceed the recommended values. Moreover, to generate the report, it is easier to iterate on the doses of the drug treatment rather than on the validation results. Indeed, in a drug treatment, the doses are not organized in a linear way but in a tree. It is therefore more efficient to prepare and access the validation results.*

The dose warning message is realized by prepending the exceeded dose to a sentence from the LanguageManager.

For example: “Maximum recommended dosage reached (400.00 mg)”.

*Therefore, we know which doses need to be verified.*

### Validate the samples

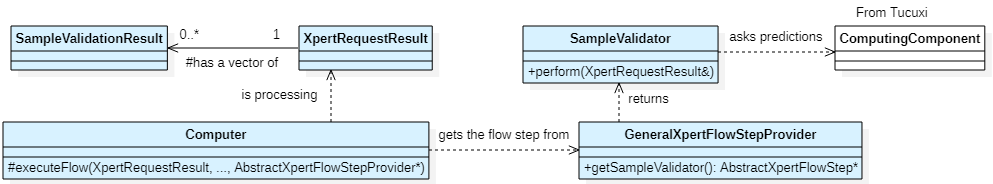


Figure 26 Samples validation class diagram

The [validation of the samples](#_Assess_the_samples) involves all these classes. If the execution of the SampleValidator class is successful, it places a vector of DoseValidationResult in the XpertRequestResult.

The only difference with analysis is that one percentile calculation request is executed per sample. With only one percentile calculation request, if the first and the last sample are really spaced in time, the Tucuxi computation core does not accept to do the calculation because there are too many points to calculate.

*We therefore know the position of the samples on the 99 percentiles and whether any need to be checked.*

The sample warning message is realized by prepending the percentage of the population that is below or above the suspicious sample percentile to a sentence from the LanguageManager.

For example: If the sample percentile is 5, “95% of the population is above this measure”.

### Validate the targets

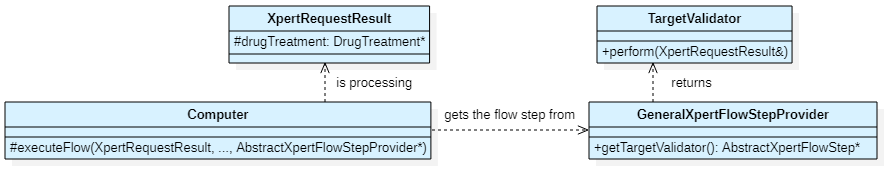


Figure 27 Targets validation class diagram

The [validation of the targets](#_Assess_the_targets) involves all these classes. The TargetValidator checks is the targets of the drug treatment in the XpertRequestResult are valid. It does not store any results because either the targets are valid and it proceeds to the next flow step, or they are not valid and the processing is abandoned.

### Create the adjustment trait

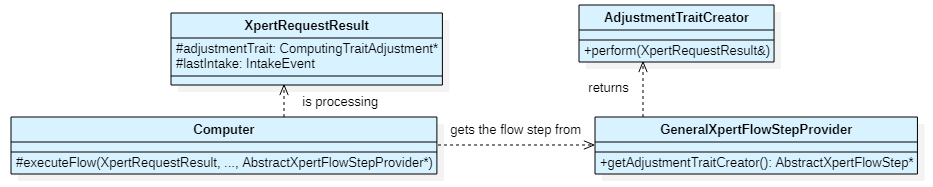


Figure 28 Adjustment trait creation class diagram

The [creation of the adjustment trait](#_Create_adjustment_trait) involves all these classes. If the execution of the AdjustmentTraitCreator class is successful, it places a ComputingAdjustmentTrait in the XpertRequestResult. The trait system is what is used to tell the Tucuxi computation core what to predict.

When creating the adjustment trait, the AdjustmentTraitCreator should retrieve the list of intakes from the patient's dosing history. We take this opportunity to retrieve the last catch if there is one. This way, we don't need to do the extraction again later.

*At that point, we have everything we need to calculate the patient's adjustment or first dose.*

### Execute the requests

Une image contenant table

Description générée automatiquement

Figure 29 Request execution class diagram

The RequestExecutor class submits 3 or 4 adjustment predictions to the Tucuxi computation core:

1. The first prediction directly uses the adjustment trait created in the previous step. It aims to obtain the adjustmentData which contains for each adjustment found:
   * The predicted concentrations
   * The times offset
   * The dosages linked to the predictions
   * The expected values for each target

Depending on the type of computation, the pharmacokinetics parameters “a priori” or “a posteriori” of the best adjustment.

1. The second prediction is based on the trait of the first prediction but the prediction end time is made to be on a steady state and the adjustment data are only for the best adjustment. From this resulting adjustment data, we extract the steady state statistics, such as:
   * The peak concentration
   * The residual concentration
   * The mean concentration over 24 hours
2. The third prediction is made only if the first one used “a posteriori” pharmacokinetics parameters. It is also based on the trait of the first prediction but requires “a priori” parameters , the minimum number of points per hour and the adjustment data is only for the best adjustment. From this prediction we get the “a priori” pharmacokinetics parameters.¨
3. The last prediction trait is similar to the third one but it uses “typical patient” pharmacokinetics parameters to obtain these parameters.

*In the current state of the Tucuxi computation core, we are forced to make these additional predictions to get all the information we need. However, these predictions are as light as possible for the computation.*

*At that point, we have everything we need to generate the report.*

### Print the report

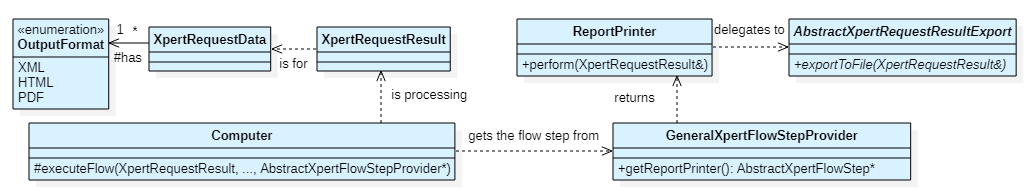


Figure 30 Report generation class diagram

The report generation involves all these classes. The ReportPrinter simply creates a concrete instance of an AbstractXpertRequestResultExport for the given OutputFormat of the XpertRequestData in the XpertRequestResult. When the corresponding export is created, it simply starts the report generation with the "exportToFileMethod". If the execution of the ReportPrinter is successful, the processing of the XpertRequestResult is finished. If there are any, the computer can start with a new one.

**AbstractRequestResultExport**

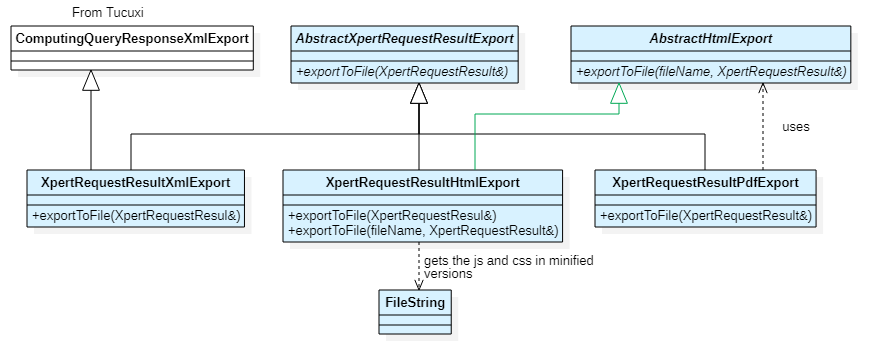


Figure 31 Xpert request result exporter class diagram

Let's introduce each of these exporters.

**XpertRequestResultXmlExport**

This exporter inherits the Tucuxi exporter because it reuses some of its methods to export the dosage history. Apart from the export of TuberXpert data, this does not add any new logic to what was already there.

The structure of the xml is the same as the one presented [here](#_Report).

*The structure of the report can be validated with the file "tuberxpert\_computing\_response.xsd" in "dev/tucuci-tuberxpert/xml/response".*

**XpertRequestResultHtmlExport**

With this export, I did the PDF export at the same time. I've been looking for a way to convert HTML files to PDF files in C++. There are a few solutions but most of them are overloaded, too uncertain for me to risk during this project, not cross-platform or not open source. The best solution I found is to use a project called wkhtmltopdf. It is a C library that uses the Qt WebKit renderer. It is free, open source, cross-platform and always maintained. So, what's the trick? The problem is that even if it is still maintained, it doesn't support the latest versions of CSS and JS. So, I had to adapt my html renderer to be convertible with wkhtmltopdf 0.12.6.

*The library does dynamic linking. The QMake file in "dev/tucuci-tuberxpert/src/tuberxpet/tuberxpert.pri" automatically switch from Windows « .dll » files to Ubuntu « .a » files.*

The points of attention are as follows:

* CSS: Many layout options are not supported, such as Flexbox. The HTML layout of TuberXpert has been done only with tables, margins and paddings. However, this is not a problem since we are only trying to make an A4 page (at least the width) in HMTL. So, we can get rid of the responsiveness so that the layout is very simplified.
* JS: JavaScript is only supported in EcmaScript 5 and not in EcmaScript 6. However, it is now possible to write JavaScript and transpile it into older versions.

*I made a Babel project with Node.js to transpile JS with EcmaScript 6 to EcmaScript 5. The project contains a TuberXpert HTML example and a readme.md that explains how to use it. The project is available in "/dev/tucuxi-tuberxpert/html". This is the playground where I designed the report I wanted to have after generating an HTML report with TuberXpert.*

Now that I have a solution to create the PDF report from HTML, I need to find a solution to create an HTML report. To do this, I used a library called [Inja](https://github.com/pantor/inja) 3.3.0 which allows string templating from JSON data. It allows loops, conditions, defaults and it is a header only library.

*Both libraries are located in the folder "dev/tucuxi-tuberxpert/libs". Both libraries support Windows 10/11, Unbuntu 22.04 and they are opensource.*

Basically, the HTML exporter defines a template string, transforms the XpertRequestResult data into JSON and that's it. The only thing that differs from the playground project is that the file created is an all-in-one file to be easily manipulated by the user. All the necessary JS and CSS files are minified and placed in static attributes of the FileString class.

*Currently the graphics are drawn by JS scripts provided by Prof. Yann Thoma each time the document is opened. In a future version, it may be more suitable to generate an image, convert it to base 64 and insert it in the final report.*

*The only JS file I created myself is "dev/tucuxi-tuberxpert/html/src/asset/js/tuberxpert.js". This file is responsible for translating the data inserted by the HTML exporter into JS objects used by the graphics library.*

**XpertRequestResultPdfExport**

Thanks to the previous efforts, the PDF exporter is straightforward. It is inspired by this [example](https://github.com/wkhtmltopdf/wkhtmltopdf/blob/master/examples/pdf_c_api.c) of the official wkhtmltopdf GitHub. We are just setting the margin and disabling the smart shrinking which is messing the final render.

As we expected, the PDF exporter creates a temporary HTML file. To do this, we had to make the HTML exporter inherit from AbstractHtmlExport to be able to pass a special file name that contains a temporary hint. Normally the exporters create file names automatically from the XpertRequestResult. This new inheritance allows the PDF exporter to give the HTML file a name that indicates that the file is temporary.

*Although, the PDF exporter tries to delete the temporary HTML file, it might be blocked by very aggressive anti-virus software. This way, if deletion is not possible, the user knows that it is not an important file.*

### Error management

Une image contenant texte

Description générée automatiquement

Figure 32 Xpert request result with error handling class diagram

This section is not a flow step. It is just there to explain how errors are handled between each flow step.

There are many reasons that can cause a step to fail, but when this happens, the reason detected is logged as an error message in the XpertRequestResult and the current process step ends immediately. The TuberXpert computer checks after each flow step if the XpertRequestResult can continue with the "shouldContinueProcessing" method. If the method says "yes", the XpertRequestResult goes to the next step, otherwise the error message is logged and the XpertRequestResult is abandoned.

## Termination

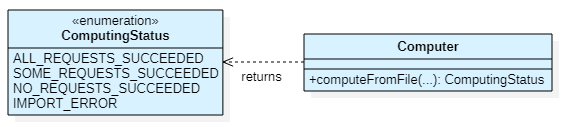


Figure 33 Ending class diagram of the TuberXpert computer

When all the XpertRequestResult have been processed, the TuberXpert computer returns a computing status that depends on the number of XpertRequestResult that were successfully processed and if the TuberXpert query could be imported. The program exit code depends on the Computing status received.

|  |  |  |
| --- | --- | --- |
| Exit code | Return status | Reason |
| -2 | / | Program arguments are missing. |
| -1 | IMPORT\_ERROR | The TuberXpert query could not be imported. |
| 0 | ALL\_REQUESTS\_SUCCEEDED | All TuberXpert requests have been fully processed. |
| 1 | SOME\_REQUESTS\_SUCCEEDED | Not all TuberXpert requests have been fully processed. |
| 2 | NO\_REQUESTS\_SUCCEEDED | No TuberXpert request has been fully processed. |

*This concludes this "Implementation" chapter. Again, this chapter is designed to show who does what and when. It is not an in-depth explanation of the code. I think I've made a big effort to document it, so feel free to take a look at the code documentation available in “public/tuberxpert\_code\_documentation”.*

# Tests

This chapter presents the tests that were performed to validate the behavior of the system.

## Unit and integration tests

A test program has been set up to verify that TuberXpert works properly during the flow steps, except for the report generation. To do this, it uses the framework Fructose[[3]](#footnote-3). This framework has been chosen because it is the one already used by Tucuxi.

*The test program is an independent Qt project available in « dev/tucuxi-tuberxpert/test ».*

The subjects tested are:

* The TuberXpert utils methods.
* The language manager.
* The TuberXpert query import.
* The TuberXpert query to core extraction.
* The *XpertQueryResult* creation.
* The covariate validator and the drug model selector.
* The dose validator.
* The sample validator.
* The target validator.
* The adjustment trait creator.
* The request executor.

*For a complete description of the tests, there is an automatically generated code documentation available in   
“public/tuberxpert\_test\_documentation”.*

The organization is simple to understand. There is a test class for each component to test and an additional *TestUtils* class that groups the common elements between the tests:

* Some drug models.
* Date format.
* Unit.
* Translations.
* Function that set up the environment of execution (2 to 5 of the following points).

In fact, most test cases follow the same pattern:

1. Prepare a query.
2. Import the query.
3. Import some drug models.
4. Import the translation.
5. Create the results holders.
6. Execute the business logic.
7. Compare.

**Tests results:**

Passing

# Conclusion

<coming soon>

# 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>
* 2022, cppreference.com, <https://en.cppreference.com/w/>

# 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 Sunday, May 15, 2022

Une image contenant texte

Description générée automatiquement Melvyn Herzig

# List of abbreviations

**ATC** *anatomical therapeutic chemical.*

**CDSS** *clinical decision support system.*

**DM** *diabetes mellitus.*

**EHR** *electronics health record.*

**HTML** hypertext markup language, a file format for web page.

**PDF** *portable document format, a file format.*

**PK** *pharmacokinetics.*

**TB** *tuberculosis.*

**TDM** *therapeutic drug monitoring.*

**UML** *unified modeling language, a way to make diagrams.*

**XML** *extensible markup language, a file format.*

# List of figures

[Figure 1 - Dosage Scheme (BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document]) 7](#_Toc109411533)

[Figure 2 Tucuxi graphical user interface with suggested first dosages 8](#_Toc109411534)

[Figure 3 -TDM process (BUCLIN Thierry, 2022, Les Bases de la pharmacocinétique clinique [PDF document]) 9](file:///C:\Users\melvy\Desktop\Dev\2022-herzig\publi\tb_Herzig_Melvyn_2022.docx#_Toc109411535)

[Figure 4 Global application overview and components 23](#_Toc109411536)

[Figure 5 Program global execution 24](#_Toc109411537)

[Figure 6 Process of drug model selection and covariate validation 25](#_Toc109411538)

[Figure 7 Process of doses validation 26](#_Toc109411539)

[Figure 8 Process of samples validation 27](#_Toc109411540)

[Figure 9 Process of targets validation 28](#_Toc109411541)

[Figure 10 Process of adjustment trait creation 29](#_Toc109411542)

[Figure 11 TuberXpert project in Qt Creator 46](file:///C:\Users\melvy\Desktop\Dev\2022-herzig\publi\tb_Herzig_Melvyn_2022.docx#_Toc109411543)

[Figure 12 Starting class diagram of the TuberXpert program 46](#_Toc109411544)

[Figure 13 Class diagram of administrative data 47](#_Toc109411545)

[Figure 14 Class diagram of the requestXpert data 48](#_Toc109411546)

[Figure 15 Class diagram of the TuberXpert query 48](#_Toc109411547)

[Figure 16 Class diagram of the TuberXpert query importer 49](#_Toc109411548)

[Figure 17 TuberXpert query import simplified class diagram 49](#_Toc109411549)

[Figure 18 Xpert query result and xpert request results class diagram 50](#_Toc109411550)

[Figure 19 Covariates, doses and samples validation results class diagram 51](#_Toc109411551)

[Figure 20 Abstract factory class diagram 53](#_Toc109411552)

[Figure 21 Simplified class diagram of the TuberXpert computer retrieving the flow steps 53](#_Toc109411553)

[Figure 22 Class diagram of the language manager 54](#_Toc109411554)

[Figure 23 Simplified class diagram of the TuberXpert computer loading a language 55](#_Toc109411555)

[Figure 24 Covariates validation and drug model selection class diagram 55](#_Toc109411556)

[Figure 25 Doses validation class diagram 56](#_Toc109411557)

[Figure 26 Samples validation class diagram 56](#_Toc109411558)

[Figure 27 Targets validation class diagram 57](#_Toc109411559)

[Figure 28 Adjustment trait creation class diagram 57](#_Toc109411560)

[Figure 29 Request execution class diagram 58](#_Toc109411561)

[Figure 30 Report generation class diagram 59](#_Toc109411562)

[Figure 31 Xpert request result exporter class diagram 59](#_Toc109411563)

[Figure 32 Xpert request result with error handling class diagram 61](#_Toc109411564)

[Figure 33 Ending class diagram of the TuberXpert computer 61](#_Toc109411565)

[Figure 34 Gantt chart of the initial planning 67](#_Toc109411566)

[Figure 35 Gantt chart of the intermediate planning 68](#_Toc109411567)

# 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 34 Gantt chart of the initial planning

## Intermediate



Figure 35 Gantt chart of the intermediate planning

## Final

<coming soon>

1. For a complete query specification, see the file “*Tucuxi CLI Software Usability Specification*”. If you have access to TuberXpert repository, this file is available in “*/tiersdoc/Tucuxi\_CLI\_Usability\_Specification.pdf”.* [↑](#footnote-ref-1)
2. The half-life is the time it takes for the amount of a drug’s active substance to reduce by half. It is the drug model. [↑](#footnote-ref-2)
3. <http://www.andrewpetermarlow.co.uk/software/fructose.html> [↑](#footnote-ref-3)