

Integration of Fast Healthcare Interoperability Resources (HL7 FHIR) into the openMIS open source Health Insurance Management System

Bachelor thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Databases and Information Systems Group
<https://dbis.dmi.unibas.ch/>

Examiner: Prof. Dr. Heiko Schuldt
Supervisor: MSc ETH Claudia Saupper (Swiss TPH)

Faris Ahmetasevic
faris.ahmetasevic@stud.unibas.ch
2015-059-538

20.07.2020

Acknowledgments

I would like to thank Prof. Dr Schuldt for giving me the opportunity to write my Bachelor thesis on the openIMIS Initiative with the team of the Swiss TPH.

Further thanks are due to the entire staff of Swiss TPH, who welcomed me warmly and gave me the feeling to be a full member of the team. Especially my supervisor Claudia Saupper and Dragos Dobre, who were available at any time to support me, give feedback and answer any questions without hesitation. Also, I want to thank Michel Borer, who as well wrote his Bachelor thesis with the Swiss TPH, for his support and for being a good team member, as well as Patrick Delcroix, who sacrificed two days to set up the modular version of openIMIS together with us. The possibility to hold weekly meetings was very helpful and I appreciate that Claudia and Dragos took the time to do so.

To close, I would like to thank my family and friends who supported me during the whole four months and relieved me in other areas, so that I have the time to work on my Bachelor thesis.

Abstract

The Swiss Tropical and Public Health Institute (Swiss TPH) is part of the openIMIS Initiative, which consists of organizations from all over the world that have joined forces to set up a health insurance system. The goal of the open source Insurance Management Information System (openIMIS) is to provide insurers with a low cost solution for managing health insurance schemes and to give policyholders a way to administer their schemes with their mobile phones.

However, the difficulty is that healthcare data must be accessible anytime and anywhere to interact with other hospital management systems. Health Level 7 Fast Healthcare Interoperability Resources (HL7 FHIR) is used to ensure this exchange of data.

Since openIMIS uses FHIR version 3 interface and therefore many relevant attributes for the administration of insurance are missing, the objective of this bachelor thesis is to extend these attributes to FHIR version 4.

Table of Contents

Acknowledgments	ii
Abstract	iii
1 Introduction	1
1.1 Swiss TPH	1
1.2 openIMIS	1
1.2.1 Objective	2
1.2.2 How to achieve that?	2
1.2.3 History	2
1.2.4 System Description	4
1.3 HL7 FHIR	5
1.3.1 Resources	5
1.3.1.1 Structure	5
1.3.1.2 JSON Representation	6
1.3.2 Codes	8
1.3.3 Extensions	8
1.4 Process of the project	8
2 Preparation	10
2.1 Requirements	10
2.1.1 Operating System	10
2.1.2 Microsoft SQL Server 2017	10
2.2 Setting up openIMIS	10
3 Development	13
3.1 Architecture of the FHIR version 3 module	13
3.1.1 Configurations	14
3.1.2 paginations.py	14
3.1.3 Models	14
3.1.4 Converters	15
3.1.5 Serializer	16
3.1.6 permissions.py	16
3.1.7 views.py	16

3.1.8	urls.py	16
3.2	Access to the FHIR API	17
3.3	Integration of FHIR R4 into openIMIS	17
3.4	Mapping	17
3.4.1	Mapping process	21
3.5	Data Migration Tool	22
3.5.1	Database Management System	22
3.5.2	JSON File Creator	23
3.5.3	Creating Database Tables	25
3.5.4	Merging Both Tools	27
3.5.5	How to use it?	27
3.6	Publishing the openIMIS FHIR R4 Module	28
4	Conclusion	30
4.1	Conclusion	30
4.2	Future Work	30
	Bibliography	32
	Appendix A Appendix	35
A.1	List of Abbreviations	35
A.2	List of used Modules	36
A.3	List of Mapped Resources	36
A.4	End Product	36
A.5	Mapping Tables	37
A.5.1	Overview Table	37
A.5.2	Claim Table	38
A.5.3	ClaimResponse Table	39
A.5.4	Coverage Table	40
A.5.5	Patient Table	41
A.5.6	Practitioner Table	42
A.5.7	PractitionerRole Table	42
A.5.8	Location Table	43
A.5.9	CoverageEligibilityRequest Table	43
A.5.10	CoverageEligibilityResponse Table	44
A.5.11	Communicationrequest Table	44
A.5.12	Condition Table	45
A.5.13	Medication Table	45
A.5.14	ActivityDefinition Table	46
A.5.15	HealthcareService Table	46
	Declaration on Scientific Integrity	47

1

Introduction

Like almost everything else in the world, healthcare financial systems are being digitised. We, who live in digitally advanced countries, are not aware of the problems that a task cannot be completed efficiently and reliable at the touch of a button. The openIMIS Initiative has therefore set itself the task of developing a free alternative that allows low-income countries to manage their health care finances quickly and efficiently.

1.1 Swiss TPH

Prior to discussing openIMIS, I would like to explain briefly what the Swiss TPH is.

The Swiss Tropical and Public Health Institute¹ (Swiss TPH) is one of the world's most recognized research institutes associated with the University of Basel and is active in the field of global health.

From the identification of medical problems, to the development of the necessary measures, the development of remedies as well as the monitoring and evaluation of treatment campaigns, all activities are carried out by the more than 850 employees².



1.2 openIMIS

openIMIS³ is an open source system for managing health insurance schemes by offering an easily understandable and user-friendly interface.

¹ Official website of the Swiss TPH [10]

² German Wikipedia page of the Swiss TPH [11]

³ Official openIMIS website [6]

1.2.1 Objective

"We have around 400 million people without access to a complete set of essential health services." [47]

With this described problem the openIMIS Initiative has set itself the goal to achieve an universal health coverage and an universal social protection by fulfilling the Sustainable Development Goals⁴⁵ (SDGs). These are 17 global goals described by the General Assembly of the United Nations, which are to be achieved by the year 2030. The openIMIS Initiative focuses especially on SDG 1, which aims to end poverty in all its forms worldwide, and SDG 3, which is intended to ensure a healthy life for all people of all ages. The subcategories SDG 1.3 and SDG 3.8 are thereby mainly addressed.

SDG 1.3 tackles to implement nationally appropriate social protection systems and measures for all, including floors, and by 2030 achieve substantial coverage of the poor and the vulnerable. On the other hand, SDG 3.8 wants to achieve universal health coverage, including financial risk protection, access to quality essential health-care services and access to safe, effective, quality and affordable essential medicines and vaccines for all.

1.2.2 How to achieve that?

The openIMIS Initiative is a rapidly growing developer community, which ensures a continuous improvement of the system as well as a possibility to adapt to the needs of different countries by customizing specific scheme types. Because of that, openIMIS is designed to be compatible with other IT-services to have a better data exchange. The last and most important part is, that it is an open source system, so anyone can download it for free and modify the code according to their preferences, which will directly lead to a feedback for the community and again to the continuous improvement.

1.2.3 History

At the beginning, in 2012, the Insurance Management Information System (IMIS) was launched with the help of the Swiss Development Cooperation⁶ (SDC) and developed among others by SwissTPH to be used in Tanzania. Due to the success of the openIMIS predecessor, which was not an open source system at that time, other countries such as Cameroon and Nepal were willing to use this system.

⁴ Wikipedia page of SDG [9]

⁵ Official list of SDG indicators [8]

⁶ Official website of the SDC [7]

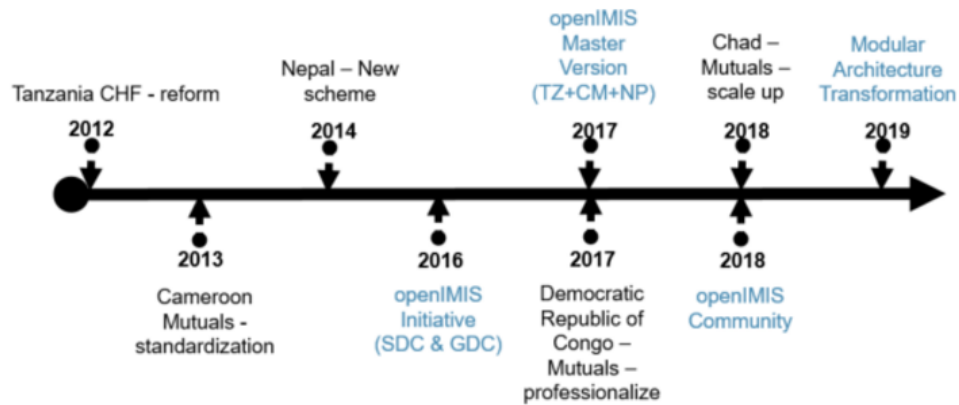


Figure 1.1: From IMIS to openIMIS [48]

Eventually, in 2016, in cooperation with the German Development Cooperation⁷ (GDC) and under the coordination of the Gesellschaft für Internationale Zusammenarbeit⁸ (GIZ), the openIMIS initiative was founded, which licensed IMIS as an open source software, openIMIS was born. Shortly after, further countries joined the community.

Due to the rapid growth and success of openIMIS, it was decided in 2019 to design the architecture in a modular way, in order to be able to adapt the requirements of the schemes of the different countries faster and more efficiently.



Figure 1.2: Current countries using openIMIS [49]

⁷ Official website of the BMZ [1]

⁸ Official website of the GIZ [5]

1.2.4 System Description

The system works through a centralized server where people with administrator rights can log in to get access to the various functions.



Figure 1.3: openIMIS web application

For example, to create an insurance scheme, follow the functions in Figure 1.3 step by step, starting with the locations. However, a detailed tutorial⁹ is given on the official YouTube channel of the openIMIS Initiative, which explains the system in practice.

Once one has gone through these steps, the generated data must now be sent to other systems, such as various health facilities. This data exchange is guaranteed by HL7 FHIR.

⁹ openIMIS Demo video on YouTube [46]



1.3 HL7 FHIR

Fast Healthcare Interoperability Resources¹⁰ (FHIR) is, as already mentioned, an open interoperability standard framework for the exchange of data between healthcare organizations and their computer systems. It was published by Health Level Seven International (HL7), a non-profit organization, in the year 2014¹¹. The current version FHIR R4 (Release #4) was released on the 30th October of 2019.

1.3.1 Resources

FHIR uses different modular components, called "resources", for health record classification with which it transmits information in small fragments. To understand what resources are, you can think of them as tables in SQL databases that are linked to other tables, that is, other resources.

Currently there are 146 resources, divided into different categories, such as financial or clinical, but which all have the same structure.

1.3.1.1 Structure

The resources are structured like folders, in which there are fields as well as composite fields. Each field and composite field has a cardinality that describes how often they can occur.

There are four different cardinalities:

- (0..1) - a field can occur once or not at all
- (0..*) - a field is a list that can have several values or none at all
- (1..1) - a field must have a single value
- (1..*) - a field is a list which must have one or more values

In addition, each field and composite field has a type, which is either a primitive or complex type. The primitive types are those that have only one value and no additional elements as children, for instance integers or strings. Complex types, on the other hand, have the same folder-like structure as the resources and also include fields that consist only of primitive types.

¹⁰ Official FHIR website [3]

¹¹ Wikipedia page of FHIR [4]

Name	Flags	Card.	Type
Location	TU		DomainResource
identifier	Σ	0..*	Identifier
status	?! Σ	0..1	code
operationalStatus	Σ	0..1	Coding
name	Σ	0..1	string
alias		0..*	string
description	Σ	0..1	string
mode	Σ	0..1	code
type	Σ	0..*	CodeableConcept
telecom		0..*	ContactPoint
address		0..1	Address
physicalType	Σ	0..1	CodeableConcept
position		0..1	BackboneElement
longitude		1..1	decimal
latitude		1..1	decimal
altitude		0..1	decimal
managingOrganization	Σ	0..1	Reference(Organization)
partOf		0..1	Reference(Location)
hoursOfOperation		0..*	BackboneElement
daysOfWeek		0..*	code
allDay		0..1	boolean
openingTime		0..1	time
closingTime		0..1	time
availabilityExceptions		0..1	string
endpoint		0..*	Reference(Endpoint)

Figure 1.4: Structure of the Location resource [2]

Each of these types has its own type, which is the Element and represents one of the two highest instances. The Element is also the type of the BackboneElement, the types of the composite fields. Besides the Element, the second structure, which has neither cardinality nor type, is the Resource type. However, it should not be confused with the resources mentioned above. This base Resource is the basis of all other resources and is the type of the DomainResource.

1.3.1.2 JSON Representation

To read the resources in human understandable languages they can be accessed by requesting information from an FHIR server using a pull method. When data needs to be displayed, a request for a specific resource is sent to the FHIR server. This resource can be returned

in two different formats on the web browser, XML (Extensible Markup Language) or JSON (JavaScript Object Notation), where openIMIS supports the JSON representation. Depending on the implementation you can perform different actions when connecting to the RESTful API (Representational State Transfer Application Programming Interface) server. Example given, you can filter resources by their unique identifier with the URL in this form [baseURL]/[resource]/[id], which is a field in the resource structure.

This is an example of a JSON representation of the Location resource from Figure 1.4:

```
{
  "resourceType": "Location",
  "id": "8ACF51CF-EB6D-44DB-AED5-75412408E791",
  "identifier": [
    {
      "type": {
        "coding": [
          {
            "code": "UUID",
            "system": "https://hl7.org/fhir/valueset-identifier-type.html"
          }
        ]
      },
      "use": "usual",
      "value": "8ACF51CF-EB6D-44DB-AED5-75412408E791"
    },
    {
      "type": {
        "coding": [
          {
            "code": "LC",
            "system": "https://hl7.org/fhir/valueset-identifier-type.html"
          }
        ]
      },
      "use": "usual",
      "value": "R1"
    }
  ],
  "name": "Ultha",
  "physicalType": {
    "coding": [
      {
        "code": "R",
        "system": "http://terminology.hl7.org/CodeSystem/location-physical-type.html"
      }
    ]
  },
  "text": "region"
}
```

Figure 1.5: Json representation of the Location resource from openIMIS

In this case we have a URL which looks like this: [baseURL]/Location/8ACF51CF-EB6D-44DB-AED5-75412408E791

1.3.2 Codes

If codes are given in a resource, the types Code, Coding or CodeableConcept are used. These contain the code, which is represented by a pair of "system" and "code". The "code" is the actual value of the code, where the "system" is a URL that defines the Code System in which the code value is located. Within the Code System there is the Value Set. This indicates which possible code values are contained in the Code System.

All codes from system <http://terminology.hl7.org/CodeSystem/location-physical-type>

Code	Display	Definition
si	Site	A collection of buildings or other locations such as a site or a campus.
bu	Building	Any Building or structure. This may contain rooms, corridors, wings, etc. It might not have walls, or a roof, but is considered a defined/allocated space.
wi	Wing	A Wing within a Building, this often contains levels, rooms and corridors.
wa	Ward	A Ward is a section of a medical facility that may contain rooms and other types of location.
lvl	Level	A Level in a multi-level Building/Structure.
co	Corridor	Any corridor within a Building, that may connect rooms.
ro	Room	A space that is allocated as a room, it may have walls/roof etc., but does not require these.
bd	Bed	A space that is allocated for sleeping/laying on. This is not the physical bed/trolley that may be moved about, but the space it may occupy.
ve	Vehicle	A means of transportation.
ho	House	A residential dwelling. Usually used to reference a location that a person/patient may reside.
ca	Cabinet	A container that can store goods, equipment, medications or other items.
rd	Road	A defined path to travel between 2 points that has a known name.
area	Area	A defined physical boundary of something, such as a flood risk zone, region, postcode
jdk	Jurisdiction	A wide scope that covers a conceptual domain, such as a Nation (Country wide community or Federal Government - e.g. Ministry of Health), Province or State (Community or Government), Business (throughout the enterprise), Nation with a business scope of an agency (e.g. CDC, FDA etc.) or a Business segment (UK Pharmacy), not just an physical boundary

Figure 1.6: Value Set Location Physical Type [14]

This value set is shown in Figure 1.5 under physicalType.

1.3.3 Extensions

In case you need a field which does not originally appear in the resource, you can use extensions.

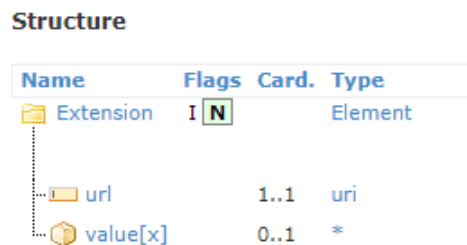


Figure 1.7: Structure of an Extension [13]

Extensions form a separate composite field which can contain several complex or primitive types. Each of these types is defined by an URL and a value. The name of the type is appended to the value, so you can see which type was used.

1.4 Process of the project

Now that we have all the information about the different components of this work, we come to the actual procedure of my project.

This project has four steps, which I had to achieve. These are:

1. Become familiar with the existing openIMIS implementation and HL7 FHIR (versions 3 and 4).
2. Provide a mapping from relevant fields of the most popular and relevant hospital management systems and openIMIS, with focus on the fields and attributes that are so far not yet supported.
3. Integrate support for HL7 FHIR version 4 into openIMIS (back-end implementation in python).
4. Develop a tool for the transfer of data from the openIMIS database to a FHIR database through the new developed FHIR version 4.

The first point can be seen as an introduction to the project, where I should first of all familiarize myself with the code for the integration of FHIR version 3 and get a feeling for the differences between the two FHIR versions. This point also includes the installation of all required programs.

The next two points can be taken together, since mapping resources requires FHIR R4 to be already integrated.

Last but not least, a program should be written which transfers the data from the openIMIS database to a new database which has the structure of the FHIR resources.

2

Preparation

Before you can start with the actual work, the mapping of the resources and the transfer of the database, you have to fulfill several requirements to install all needed programs and start openIMIS.

2.1 Requirements

2.1.1 Operating System

Because the current implementation of openIMIS uses the Microsoft SQL Server¹², the system can currently only run on Windows devices. Of course, a possibility is to install the server on Mac OS or Linux with the Docker¹³ program, but then important applications like the SQL Server Management Studio or the SQL Server Configuration Manager are missing. It is also recommended to use an administrator account, because for many files the user needs access rights.

2.1.2 Microsoft SQL Server 2017

To be on the safe side it is recommended to download the 2017 version of the Microsoft SQL Server, because openIMIS was developed and tested especially with this version. The reason for this is explained in the next section. Since recently the developer version of Microsoft SQL Server 2017 can be downloaded directly from the openIMIS wiki page¹⁴.

2.2 Setting up openIMIS

The installation of the two openIMIS versions, the legacy and the modular version, was probably the most tedious work of the whole project, because many problems occurred in the process which took a lot of time to fix.

¹² Official website of the Microsoft SQL Server [15]

¹³ Official website of Docker [12]

¹⁴ openIMIS Database Installation wiki page [16]

For the installation I made use of the openIMIS wiki¹⁵ page, which contains a guide that goes through every step.

Since I generally use Mac OS and I am most familiar with this operating system, I wanted to download openIMIS on an Apple computer.

I started with the modular version. The first step was to download all the required repositories, which can be found on the openIMIS GitHub¹⁶ site and directly linked on the installation guide for the modular version on the openIMIS wiki page¹⁷, as well as installing the openIMIS demo database which is needed to have any entries in the webapp.

Downloading the Microsoft SQL Server was not a challenge at the beginning because I created a container with Docker and started the server on the terminal with a virtual environment. When the gateway, which connects the frontend with the backend server, caused problems by not being able to create a running container, I finally switched to a Windows device, where I hoped to get the setup done quickly.

In one of our weekly meetings it turned out that the modular version is dependent on the legacy version and should therefore be installed first. So I installed the legacy version first. The installation and configuration of the SQL server, the openIMIS demo database¹⁸, and the IIS (Microsoft Internet Information Services) was very quick and easy. When I finally changed the permissions in the event logs and start the web application, I could not connect to the server. I found an error message on my web page that denied me access because I allegedly had no rights, although I followed exactly the points from the manual. After some searching I found out that in the Registry Editor not only the EventLog folder and the Security key it contains must have administrator rights, but also the State key. After this change the legacy version worked fine.

Now it was the turn of the modular version, which only required the download of six modules, an adaptation of the .env file by system variables that formed a link to the database and a few terminal commands.

But this did not work as easily as expected, because the modular version has to be installed individually to each user. Fortunately Patrick Delcroix found time to work with me on this problem. It took us half an afternoon to fix the error messages Patrick Delcroix had not seen before and to establish a connection.

The disillusionment followed one day later, when all of a sudden error messages appeared again and nothing worked anymore. It appeared that the database drivers were designed for the SQL Server 2017 version and I was running the latest Microsoft SQL Server 2019. Once the SQL Server is on the device, it is very difficult to uninstall. It took me a whole week to get the 2017 version up and running, because the protocols for the server could not

¹⁵ Installation Manuals for the modular and legacy version of openIMIS [17]

¹⁶ Official openIMIS GitHub page [18]

¹⁷ Installation guide for the modular version of openIMIS [19]

¹⁸ Repository for the openIMIS databases [45]

be started. I then decided to reset my whole computer, otherwise it would not have worked.

After a successful reinstallation of the server, but now the SQL Server 2017 version, I went through the whole procedure from the beginning and came back to the point, that the installation of the modular version causes problems.

Since Michel Borer, as before with the legacy version, had to struggle with the same problems, Dragos Dobre and Patrick Delcroix decided to install the modular version together with us, so that we could finally start with the actual work. For Michel the installation worked, but for me the time was not enough, because I had to deal with error messages once again.

In the meantime Dragos Dobre developed a development tool¹⁹ which directly downloads and installs the necessary modules from PyPI²⁰ (Python Package Index) I need for my work, like the openIMIS backend and the FHIR modules. After some initial problems we sat down together and finally managed to get the modular version running. We initialized the system variables as local variables on Windows, but we did not consider the port, because it was probably in conflict with another port.

After a long and tedious setup, both versions were running perfectly, so the next and main part, the programming, could begin.

¹⁹ GitHub link to the development tool repository [20]

²⁰ Official website of PyPI [33]

3

Development

The development is divided into three parts, the first part being the integration of FHIR R4 into the openIMIS system. This is the logical step, as the mapping of resources requires a base to start with. Further, I will explain the mapping process and finally the implementation of the migration tool.

3.1 Architecture of the FHIR version 3 module

To fully understand how the FHIR API module works, it is important to first understand how it is constructed.

The module `api_fhir`²¹ contains eight directories and seven files, which are all connected to each other, and whereby I explain the most important ones in a more precise way.

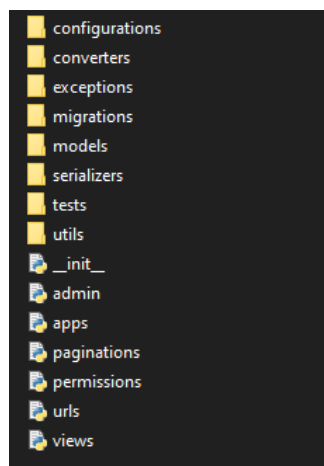


Figure 3.1: Structure of the `api_fhir` module

²¹ Link to the `openimis-be-api_fhir_py` repository [22]

3.1.1 Configurations

In the Configurations folder²² there are various configuration files, which are used to implement the required codes for the individual resources. These codes either originate from the value sets of FHIR or are self-generated codes that are used specifically for openIMIS. These configuration sets are passed on in the apps.py²³ script as a default list to the moduleConfiguration.py²⁴ script, which then creates the configurations.

3.1.2 paginations.py

From the previously mentioned configurations the generalConfiguration.py²⁵ script, which contains default values for the API's, is passed to the paginations.py²⁶ script. This script then creates the bundle resource in which it inserts the data of the mapped resources. It also describes the layout of the webpage with the values it got from the generalConfiguration.py script. For example how many entries there are on a page and how to edit them.

3.1.3 Models

The files in this folder are the interface between the various FHIR resources and the mapping of those just mentioned. In each of these files a one-to-one representation of the folder-like structure of the resources is created.

```
class LocationPosition(BaseElement):
    altitude = Property('altitude', float)
    latitude = Property('latitude', float)
    longitude = Property('longitude', float)

class Location(DomainResource):

    identifier = Property('identifier', 'Identifier', count_max=*)
    status = Property('status', str) # LocationStatus
    operationalStatus = Property('operationalStatus', 'Coding')
    name = Property('name', str)
    alias = Property('alias', str, count_max=*)
    description = Property('description', str)
    mode = Property('mode', str) # LocationMode
    type = Property('type', 'CodeableConcept')
    telecom = Property('telecom', 'ContactPoint', count_max=*)
    address = Property('address', 'Address')
    physicalType = Property('physicalType', 'CodeableConcept')
    position = Property('position', 'LocationPosition')
    managingOrganization = Property('managingOrganization', 'Reference') # referencing `Organization`
    partOf = Property('partOf', 'Reference') # referencing `Location`
    endpoint = Property('endpoint', 'Reference', count_max=*) # referencing `Endpoint`
```

Figure 3.2: Implemented structure of the Location resource

²² Link to the configurations directory in GitHub [24]

²³ Link to the apps.py script in GitHub [23]

²⁴ Link to the moduleConfiguration.py script in GitHub [28]

²⁵ Link to the generalConfiguration.py script in GitHub [26]

²⁶ Link to the paginations.py script in GitHub [29]

Name	Flags	Card.	Type
Location			DomainResource
identifier	Σ	0..*	Identifier
status	?! Σ	0..1	code
operationalStatus	Σ	0..1	Coding
name	Σ	0..1	string
alias		0..*	string
description	Σ	0..1	string
mode	?! Σ	0..1	code
type	Σ	0..1	CodeableConcept
telecom		0..*	ContactPoint
address		0..1	Address
physicalType	Σ	0..1	CodeableConcept
position		0..1	BackboneElement
longitude		1..1	decimal
latitude		1..1	decimal
altitude		0..1	decimal
managingOrganization	Σ	0..1	Reference(Organization)
partOf		0..1	Reference(Location)
endpoint		0..*	Reference(Endpoint)

Figure 3.3: FHIR structure of the Location resource [21]

Each field is defined as a property that contains the name of the field, the types and, depending on which one is defined, the cardinality. The cardinality (0..1) is ignored, where cardinality (0..*) is set as `count_max='*'`, cardinality (1..1) as `required=True` and the cardinality (1..*) as `count_max='*', required=True` together.

The BackboneElements are declared in their own field in the resource, where this field points to the generated BackboneElement, which was created as a separate class. Thus, for every resource, every element and every complex type that is required, a representation in the code is mandatory.

3.1.4 Converters

The converters are the main files when it comes to mapping the resources. Thereby each resource has its own converter which maps the data from the openIMIS database to the corresponding field of the FHIR resource and vice versa.

The database data is defined in separate modules, such as the `openimis-be-location.py`²⁷ module, which itself has a `model.py` file where the columns of the database are initialized in tables. These tables can then be imported into the converter and the values in them accessed. But it is essential that the types of the database columns correspond to the types of the FHIR field.

I will go into more detail about the converters in the mapping section.

²⁷ Link to the GitHub repository of `openimis-be-location.py` [27]

3.1.5 Serializer

The FHIR, as well as the openIMIS conversions that were previously created, are then used in the serializer to map the fields and their data to the API. Here, each FHIR field with values is then added to an instance list which is returned to the views.py script.

3.1.6 permissions.py

In this module there are permissions for five HTTP methods to interact with the FHIR API. These are the GET, POST, PUT, PATCH and DELETE methods, although not all resources have permission to use each method.

- GET: A resource is requested from the server
- POST: Resources or data are sent to the server
- PUT: A resource is uploaded to the server using a target address
- PATCH: A set of instructions on how to modify resources
- DELETE: A resource is deleted from the server

In the permissions.py²⁸ script a set of permissions is created for each resource that will be displayed in the FHIR API. Which method is allowed is defined in the configuration in the module of the respective resource, which then imports the permission.py script and returns it to the view.py script, too.

3.1.7 views.py

The views.py²⁹ script receives the serializer and permissions of all resources and creates a view set for each of them. A look up field is defined, which usually contains the UUID, a unique identification number, with which the individual resources, respectively the rows of the database, can be filtered.

3.1.8 urls.py

With the urls.py³⁰ script a framework router is used to create the URL's, which are stored as a register. In this register are the names of the resources on the URL link, the view sets imported by the views.py script, and a basename. This basename is used to ensure that the URL is unique and does not lead to another resource that might have the same name.

²⁸ Link to the permissions.py script in GitHub [30]

²⁹ Link to the views.py script in GitHub [32]

³⁰ Link to the urls.py script in GitHub [31]

3.2 Access to the FHIR API

To get access to the FHIR API a superuser for the backend server had to be created first. For this, change to the openIMIS folder of the backend module, where the `manage.py`³¹ script is located and activate the virtual environment. If a user starts this script with the command `python manage.py createsuperuser`, he will be asked for a username and a password, which he creates himself. To run the server the command `python manage.py runserver` has to be launched.

Once the login credentials have been created and the server is running, the user has to log in at <http://localhost:8000/admin> to get the authorization to view the data in the API.

When everything is done, the FHIR API can be accessed via http://localhost:8000/api_fhir.

3.3 Integration of FHIR R4 into openIMIS

With the intention to create the FHIR R4 API and build on it, I duplicated the module `api_fhir` in the same directory and just renamed it to `api_fhir_r4`. In this new module I changed every name that had something to do with version 3 as well as the imports from `api_fhir` to the new module.

At the beginning, I only wanted to use the link http://localhost:8000/api_fhir_r4/ to get to a new page which belongs to the FHIR R4 API and which contains `api_fhir_r4` in the base URL's of the resources. I created the page, but every link of the API redirected me back to version 3.

With a bit of trial and error I finally found out that the basename from the `urls.py` script was the same as the one of the original module. With the renaming of the basenames I had a working API for version 4, which I could update step by step with the mapping of the new resources.

3.4 Mapping

Now that I have a base to work on, we come to the major part, the mapping.

In the beginning the goal was to adapt the resources already mapped from version 3 to the new version. Besides, I should document my progress on the openIMIS wiki page³². I took over the wiki documentation from FHIR v3 and updated it according to my tasks. On the wiki page is a table for each resource, which shows which FHIR field has to be connected to which openIMIS database field. These tables were created by Dragos Dobre so that I and other people know how the resources were mapped. They also served me as a guide and checklist.

The resources that had to be mapped at that time were Claim, ClaimResponse, Coverage, Patient, Practitioner, PractitionerRole, Location, CoverageEligibilityRequest, CoverageEli-

³¹ Path to the `manage.py` script: `openimis-be_py/openIMIS/manage.py`

³² FHIR R4 mapping wiki page [25]

gibilityResponse and CommunicationRequest.

But before I could start with the actual mapping, the first logical step was to check the resources from the models directory and if necessary remove fields that are not needed or add fields that do not exist yet. This was not very demanding work, as it was actually a straightforward approach. However, it helped me to better understand the structure of the resources, especially the composition of the different elements and complex types. This didn't take very long and once that was done, I was able to deal with the converters, since they are the foundation of the mapping.

In principle, each of these converters has the same design.

The structure is as follows:

```
class ExampleConverter(BaseFHIRConverter, ReferenceConverterMixin):

    @classmethod
    def to_fhir_obj(cls, imis_example):
        fhir_example = ExampleResource()
        cls.build_fhir_pk(fhir_example, imis_example.uuid)
        cls.buld_fhir_test_field(fhir_example, imis_example)
        return fhir_example

    @classmethod
    def to_imis_obj(cls, fhir_example, audit_user_id):
        errors = []
        imis_example = ExampleTable()
        cls.build_imis_test_column(imis_example, fhir_example, errors)
        cls.check_errors(errors)
        return imis_example

    @classmethod
    def get_reference_obj_id(cls, imis_example):
        return imis_example.uuid

    @classmethod
    def get_fhir_resource_type(cls):
        return ExampleResource

    @classmethod
    def get_imis_obj_by_fhir_reference(cls, reference, errors=None):
        example_uuid = cls.get_resource_id_from_reference(reference)
        return DbManagerUtils.get_object_or_none(ExampleTable,
            uuid=example_uuid)
```

The five methods shown in the example code are used in practically every converter. The first two, the *to_fhir_obj()* and the *to_imis_obj()*, methods are probably the most important. These are initialized by the BaseFHIRConverter and are used to collect fields of the resource or the columns of the database table and pass them on as a whole resource and whole table. In addition, the *to_fhir_obj()* method always contains the function *build_fhir_pk()*, which is also initialized by the BaseFHIRConverter and defines which column of the database table is the primary key.

As mentioned above, depending on the number of fields required in a resource, you can have multiple methods that map the values of a database table to a resource field. An example of a function found in the *to_fhir_obj()* method looks like this:

```
@classmethod
def build_fhir_test_field(cls, fhir_example, imis_example):
    if imis_example.patient_name is not None:
        fhir_example.name = imis_example.patient_name
```

Here the names of patients found in the ExampleTable are mapped to the name field of the ExampleResource. If the API allows a POST method for this resource, you must also map it the other way around. The same resource field and the same table column are being used.

```
@classmethod
def build_imis_test_column(cls, imis_example, fhir_example, errors):
    name = fhir_example.name
    if not cls.valid_condition(name is None,
                               gettext('Missing example `name` attribute'), errors):
        imis_example.patient_name = name
```

This is the way to map fields that only take the value of the database tables directly. At this point I would like to mention three special cases.

CodeableConcept

Of course, it is possible to map CodeableConcepts like all other fields, but there is a special function which automatically creates the structure of the CodeableConcepts.

```
@classmethod
def build_fhir_example_codeable_concept(cls, fhir_example, imis_example):
    fhir_example.code = cls.build_codeable_concept(code, system, text)
```

This function takes the three values "code", "system" and "text", where the code is the actual code from the database table. The system is a URL, which links the value set of the code. The text is in most cases either a description of the code or its full name.

Reference

Since we already know what a reference is, we also need to know how it is implemented. As with the CodeableConcepts there is a separate function for this.

```
@classmethod
def build_fhir_example_reference(cls, fhir_example, imis_example):
    fhir_example.location = LocationConverter.
        build_fhir_resource_reference(imis_example.location)
```

We know that almost every converter has the *get_reference_obj_id()* method which defines the key for the references. If we now create a reference to the location, as in the example, we need to import the LocationConverter and apply the *build_fhir_resource_reference()* function to it. By doing so, we get back a reference value, the value which the LocationConverter defines in its *get_reference_obj_id()* method.

The ExampleTable also needs a link to the LocationTable. In this case the ExampleTable has a foreign key which points to the LocationTable ID.

Extension

The extension is a list that contains fields that do not occur in the resource. First, an extension field is initialized with the desired type, as value plus the type name, whereby the type is also initialized if you want to specify the exact values. In addition, each extension has a URL, which can be a link to a page or just the name of it. Then all extensions created are appended to the FHIR resource.

```
@classmethod
def build_fhir_example_extension(cls, fhir_example, imis_example):
    extension = Extension()
    extension.url = "valueCodeableConcept"
    extension.valueCodeableConcept = cls.
        build_codeable_concept(code, system, text)
    fhir_example.extension.append(extension)
```

In this example I combine the extension with a CodeableConcept. Here the type is not initialized, because we have a function which directly creates a CodeableConcept.

The next three methods are all initialized by the ReferenceConverterMixin class. The *get_reference_obj_id()* method determines which value is used in case of a reference to this class. References have this form in openIMIS: `Resource Name/Reference Value`.

This means that if another resource needs the ExampleResource as a reference, the resource name is then ExampleResource and the reference value in this case is the UUID of the ExampleTable.

get_fhir_resource_type() merely indicates the type of this resource.

The last method, *get_imis_obj_by_fhir_reference()*, returns the database object that has the searched reference value of the resource if there exists one.

3.4.1 Mapping process

I started mapping the `CoverageEligibilityRequest` because Dragos Dobre warned me that this resource was previously only called `EligibilityRequest` and this name change had a big impact on the whole mapping of it. As it turned out this was not the best idea, because I still did not understand the code exactly and because this resource did not allow a `GET` method, which made it difficult to test and control the correctness, I should have started with a resource that is easier to map. Somehow, after some time, I managed to map it successfully, which in retrospect gave me a better understanding of the implementation.

While I was still very early in the mapping process, the first adjustments of the fields in the resources and three new resources to be mapped had already been made. So I had to map the `Condition`, `Medication` and `ActivityDefinition` resources, because they are very important for the `Claim` resource.

In the `Condition` resource the diagnoses of the patients were stored, in `Medication` the items like tablets or pills, which the patients have been prescribed, and in `ActivityDefinition` the services, i.e. surgery or other measures. These fields are specified for each patient in the claim resource as a list, which also has the reference to the specific resources.

Because HISP India³³ (Society for Health Information Systems Programmes) needed these resources for its work, the focus was now on these three resources. At the same time it was decided that I did not need to map the `Coverage` and the associated `Contract` resource because it was taken over by another organization.

With the `Medication` Resource finally came the breakthrough, with which I fully understood the principle of mapping.

With the completion of these three resources I had to create the view sets and permissions, so that you can reach them in the API with a URL. To do this, I first had to create the `create()` and `update()` methods in the serializer. I built them by copying a serializer script of an already mapped resource and modifying it with the respective values of the new resources. As with the serializers, I copied the permissions of other resources to create new ones. But to create these permissions, I had to import them from the `openimis-be-medical_py`³⁴ module, as they were defined there. Because they were not fully implemented, I could only specify the `GET` method.

Next I had to implement the view sets of the resources. At the example of the `Medication` Resource you can see how such a class looks like.

```
class MedicationViewSet(BaseFHIRView, mixins.RetrieveModelMixin,
                        mixins.ListModelMixin, GenericViewSet):
    lookup_field = 'uuid'
    serializer_class = MedicationSerializer
    permission_classes = (FHIRApiMedicationPermissions,)
```

³³ Official website of HISP India [36]

³⁴ Link to the GitHub repository of `openimis-be-medical_py` [39]

```
def get_queryset(self):  
    return Item.get_queryset(None, self.request.user)
```

The `get_queryset()` method is created in the models script of the corresponding modules. This query set is then returned with the data requested by the database user.

The structure of these classes is the same for each resource. Usually the UUID of a table is used as look up field to filter the resources, but this does not exist for the condition resource. In this case the condition ID is used.

The last resource to be recreated was the HealthcareServices resource. This takes the values of the HealthFacility table from the database. Because the HealthFacility table and the Location table were originally mapped together in the Location resource, both resources had to be rewritten with the individual tables. But since I already mapped a large number of fields, this was not a big work and therefore did not take much time.

Before I could start implementing the data migration tool, I had to map more necessary fields, because it was thought that they would not be needed, as well as test and fix bugs on the whole mapping.

3.5 Data Migration Tool

The objective of the migration tool is to transfer the data from the API into a new database, which, unlike the openIMIS database, has the structure of the FHIR resources.

3.5.1 Database Management System

Since the openIMIS initiative wants to move away from the expensive Microsoft SQL Server and Michel Borer in his Bachelor thesis "*Database Migration in the openIMIS open source Health Insurance Management System*"³⁵ had the task to migrate the openIMIS database to a PostgreSQL³⁶ database, I logically also used PostgreSQL for my migration tool.

So I downloaded PostgreSQL from the official website. The installation included the pgAdmin software, which provides a graphical user interface to create and manage PostgreSQL databases. In this one I created a new user, so I don't have to deal with the default settings.

The next step was then to figure out how to convert the resource data from the API so that I could use it in my code.

Dragos Dobre suggested fhirbase³⁷, an open source software based on PostgreSQL. After some experimentation, it turned out that fhirbase only supports FHIR version 3 and this tool did not meet the requirements we had for the database anyway, although it creates a database directly with the FHIR resources.

³⁵ openIMIS wiki page for Michel Borer's Bachelor thesis [40]

³⁶ Official PostgreSQL website [43]

³⁷ Official fhirbase website [34]

After searching in futile for alternatives, we decided not to use an existing FHIR database but to develop a database structure based on the mapping.

For this reason, Claudia Saupper suggested a package called `fhir.resources`³⁸ in one of our weekly meetings. This package allows you to create resource objects from JSON representations and also to save them in a JSON format.

To work with this package one must first install it with `pip install fhir.resources`. Firstly, I wanted to create a simple resource object and print the values in it. So, I wrote a test JSON file with a resource in it and loaded the data into the resource object. This worked very well, thus I wanted to create a database with a table in Python. Now in order to work with PostgreSQL in Python you have to import the "psycopg2" library and establish a connection with the created user to PostgreSQL.

The issue then was that when a database was created in a script, you could not connect to it in the same script. Because I couldn't find a solution to this problem and to save time, I manually created a database on pgAdmin. In the meantime Dragos Dobre asked me if it would be possible for me to implement a tool that stores the API data directly as a JSON file.

3.5.2 JSON File Creator

The tool to create JSON files needs a connection to the API to collect data from it with a GET method. To do this you import the library requests.

```
req = requests.get(url, auth=HTTPBasicAuth(user, password))
```

With the requests library you can then use the `get()` function, which is passed a URL and authentication credentials. In our case the URL is the one that leads to a resource in the API, for example `http://localhost:8000/api_fhir_r4/Location/` and the username and password are the ones you need to log in to `http://localhost:8000/admin/`. Then you can use the imported json library to assign the request as text to a variable.

```
data = json.loads(req.text)
```

But because every API consists of several pages you have to know how much data has to be inserted into the JSON files. For this I wrote a function which returns the maximum number of page numbers.

³⁸ `fhir.resources` package on PyPI [35]

It works like this:

```
size = 1
req = requests.get(url, auth=HTTPBasicAuth(user, password))
next_site_url = url + pageOffset

while req.status_code != 404:

    size += 1
    url = next_site_url + str(size)
    req = requests.get(url, auth=HTTPBasicAuth(user, password))
    continue

return size - 1
```

openIMIS has a variable in the API URL which indicates on which page you are currently located and is described as `page-offset=` with a number at the end. I first create a URL for the next pages using the URL that the function was passed to, then add the page offset. Then I run a while loop until it gets beyond the last page and receives an HTTP error code 404. Since in the loop a counter was added by 1 after each pass, we can return it subtracted by one, because we do not care on which page we got the error, but how many pages worked.

In the main function I run a loop which runs until the maximum number of pages is reached. Within this loop I make a GET request for each page in the API and store the data in a new variable. Because each page contains the data in a Bundle resource, we have to ignore it from the second page on, because we only want one Bundle with all entries of a resource. Since we use the data in JSON format, we can simply apply the `get()` function to our data. In this `get()` function we then determine what we want to have from the Bundle. With 'entry' in the function, we can filter everything in the entry list, which represents the list of the Bundle resource.

```
while page < last_page:
    page += 1
    url = next_site_url + str(page)
    req = requests.get(url, auth=HTTPBasicAuth(user, password))
    resources_next_page = json.loads(req.text)
    resources_next_page = resources_next_page.get('entry')

    for i in range(len(resources_next_page)):
        data['entry'].insert(EOF, resources_next_page[i])
```

In a for loop each index in the entry list is inserted at the end of the whole file until the data is returned.

The last step is then to create the data with a file writer and save them in a folder.

3.5.3 Creating Database Tables

As already explained, I could not create a database with python, so I went directly to the tables. With the also mentioned `psycopg2` library you connect to the database by passing the database name, username, password, host address and port as parameters. To execute queries, the connection creates a cursor.

```
connection = psycopg2.connect(dbname, user, password, host, port)
cursor = connection.cursor()
```

I then had to write a separate function for each resource and the subfolders in the mapped resources, because each field of a resource has to be defined individually to map it to the table. So you create a query for each resource by creating the table and initializing the columns, which is done with the cursor.

```
query = ("""DROP TABLE IF EXISTS Location;
          CREATE TABLE Location (
            location_id INT PRIMARY KEY NOT NULL,
            identifier VARCHAR(255) NOT NULL,
            name VARCHAR(100),
            physical_type VARCHAR(3),
            part_of VARCHAR(255) );""")

cursor.execute(query)
connection.commit()
```

The connection commits the changes to the database resulting in an empty table. To insert the values into the table, I read the JSON files I created with the first tool and put them into the Bundle of the imported `fhir.resource` package.

```
from fhir.resources.bundle import Bundle

bundle = Bundle(data)
```

In `bundle` is now the whole structure of the resources within the entry list. Over the length of this list the values of a resource can be looped to insert them into a row of the table. This is done with a for loop in which the actual resource is defined. Here it is the location resource.

The next step is to define the values to be inserted into the table. This is described with an SQL query, where I then have to filter the values from the resource and connect them to these values.

```
for i in range(len(bundle.entry)):
    location = bundle.entry[i].resource

insert = ("""INSERT INTO Location (location_id, identifier,
    name, physical_type, part_of)
    VALUES (%s,%s,%s,%s,%s) """)
```

Up to this point, the procedure in each function is exactly the same, except that you have different fields in the resource. From here on, you have to filter and initialize each field separately, paying attention to which fields are mandatory for the resource and which may or may not occur.

```
gender = patient.gender
address = None

if patient.address is not None:
    if addr := [x for x in patient.address if x.type == "physical"]:
        address = addr[0].text
```

In this example from the Patient Resource you can see that the gender field has a cardinality of (1..1) and must appear in the resource. Therefore, you can initialize it directly.

On the other hand, we have the address, which does not necessarily occur. First you have to check if this value occurs, if not you give the table a null value. If it occurs then we have to filter it, because most of the fields are in a list. Here fields like the "text" in the CodeableConcepts help us, because they describe the code but are not inserted into the tables.

Once this is done, you can insert the values into the table with the cursor and commit the changes with the connection.

```
data_to_insert = (gender, address)
cursor.execute(insert, data_to_insert)
connection.commit()
```

But what has to be considered is that if you add or remove fields to the API in the mapping step, you will have to integrate these fields into the database manually by going through these steps again. This happened to me a lot during the implementation of this tool, because some resources had mandatory fields that were not needed by openIMIS, but the fhir.resources package gives an error message because these fields are missing. That means I still had to deal with the mapping during the implementation of the migration tool, which was not a bad thing. With the fhir.resources package I was able to find and fix many small bugs in the mapping.

3.5.4 Merging Both Tools

Because I created the database tables with the locally generated JSON files, Dragos Dobre wanted me to read the data the same way I did in the JSON Creator Tool, via the API, and to put both tools into one script. First I cleaned up the code of the JSON Creator Tool and made it more compact by creating all files with one function. I did this by giving the function a global list of resource names and attaching them to the URL. I also added the functionality to automatically delete the files and the folder they are in, if it exists. That means you can create the current state of the data automatically without having to delete the data manually.

To get the data for creating the tables through the API, I used the same function I used to create the JSON.

Last but not least, I created two classes for the respective tools. In the first class there is a function, which creates all tables in order. Beside the function I created a directory with which I can choose in the main method which table I want to create individually. In the second class there are two functions which either creates the folder and writes the files or deletes them first and then rewrites them updated again.

And finally, to have a tool that is easy to use, I built a small dialog on the console that tells the user what to do to get the desired end product.

3.5.5 How to use it?

The Data Migration Tool can be downloaded directly from the `openimis-fhir-data-migration_py`³⁹ repository in GitHub. You can also find a detailed documentation on the openIMIS wiki⁴⁰ page of the tool. Since this tool depends on the FHIR R4 API, the backend server must be started before running.

At the beginning of the code you can adjust the global variables according to your settings, and specify the directory where you want to store the JSON files.

To run the tool on the console of e.g. PyCharm⁴¹, you have to go to the execution settings there. This can be reached by clicking on "Run" in the upper menu and selecting the option "Edit Configurations..." there. In the window which is now open, enable the "Emulate terminal in output console" checkbox. The reason for this is that the imported library `getpass` will not work on the console until you set this function. If you use the terminal it does not need any settings.

³⁹ `openimis-fhir-data-migration_py` repository link [41]

⁴⁰ openIMIS wiki page for the Data Migration Tool [42]

⁴¹ PyCharm page of the official JetBrains website [37]



```
> To create database tables press <1>
> To create json files press <2>
> To exit press <3>
> Your choice: █
```

```
> Database Name: FHIR_database
> Username: ImisUser
> Password:
> Host Address:
> Port Number:

Do you want to create all tables? (y/n) n

-----

Which table do you want to create?

- Activity_Definition: <1>
- Claim: <2>
- Claim_Response: <3>
- Communication_Request: <4>
- Condition: <5>
- Healthcare_Service: <6>
- Location: <7>
- Medication: <8>
- Patient: <9>
- Practitioner: <10>
- Practitioner_Role: <11>

> Your choice: █
```

(a) Start Dialogue

(b) After pressing 1

Figure 3.4: Dialogue on the Console

After starting the program the dialog from Figure 3.4 (a) appears, where you can decide what you want to do. If you enter 1, more instructions will appear (Figure 3.4 (b)). Here you first have to connect to the database. If you have a local database you can leave the host and port empty. Once connected you can decide if you want to create all tables at once or just one at a time. If you do not want to create all tables, a list of possible tables will be displayed.

If you choose to create the JSON files, they will be created immediately after you press 2. When a task is done, you will be taken back to the start dialog where you can do more things or end the program by pressing 3.

3.6 Publishing the openIMIS FHIR R4 Module

To finish my project, my last task was releasing my self-created module on PyPI.

For this, I first needed my own GitHub repository. I cloned the `openimis-be-api_fhir_r4_py`⁴² repository created by Dragos Dobre in the directory where all other modules were located and moved the FHIR R4 module from the previous version's folder. Because this module is now no longer recognized, I had to install it, as described in the README.md file in the `openimis-be_py` repository, by the command `pip install -e ../openimis-be-api_fhir_r4_py/`. The next step was to push a tag with the version number to GitHub. To complete the release you needed an account on PyPI and two packages to install.

⁴² `openimis-be-api_fhir_r4_py` repository link [38]

The first one is the wheel package which creates the module. Running `python setup.py bdist_wheel` will run the `setup.py` script that describes the module.

The second package is twine. As before you install it with `pip install twine`. To upload the module to the PyPI site, run `twine upload -r pypi dist/openimis_be_mymodule-1.2.3*`. Here you name your module and add the version number described in the tag.

Now you can easily install the `openimis-be-api-fhir-r4 1.0.0`⁴³ package via `pip`.

⁴³ Self created openIMIS FHIR R4 package [44]

4

Conclusion

4.1 Conclusion

The aim of this thesis was to develop an integration of the current FHIR R4 version to the openIMIS system for the Swiss Tropical and Public Health Institute (Swiss TPH), which can run alongside version 3. The objective was to map the most important fields of relevant hospital management systems and access the data through an API. In addition to this, a migration tool was developed, which besides creating database tables with the structure of the mapped resources, also writes the data of these resources as JSON files.

With the completion of the project, there are two products that can be used by the entire openIMIS developers community. Firstly, openIMIS is running with the latest version of the FHIR R4 framework and secondly, the data from this newly mapped version can be clearly displayed in a database or as JSON files using the Data Migration Tool.

This Bachelor thesis can be seen as a guideline for future mapping tasks and developments. It shows step by step the structure of the whole process.

Starting with the installation of the openIMIS system, it was also a test to see what difficulties inexperienced users have to start the system. With the discovered shortcomings the installation guide was immediately updated so that there will be no problems for future installations.

In addition, this thesis provides a detailed description of the FHIR structure, as well as the architecture of the `openimis-be-api_fhir_r4` module, to make it easier for developers to get started, as the code itself is not very well documented.

Overall, the project was completed successfully and satisfactorily, whereby it was an honour for me to work together with the Swiss TPH and the openIMIS Initiative.

4.2 Future Work

The first version of the module contains the most relevant resources for the moment, but due to the large number of resources and fields there are still many that can be mapped. Thereby the mapping of the fields in the code can be improved qualitatively with a compact

code style. I have agreed to be part of the openIMIS developers community and to continue working on the FHIR R4 integration, so that this points can be tackled.

If new fields are added to the mapping, they must also be added in the Data Migration Tool. A solution can be found to integrate the fields into the database tables more automatically, as well as finding a way to convert resources from the API directly into tables.

In addition, a way can be found to optimize the writing of the JSON files. Since there are over 35'000 claim entries in the real world usage, the memory is very strained, which means that writing takes a long time.

Bibliography

- [1] Official website of the bmz. <https://www.bmz.de/en/index.html>, 12.07.2020.
- [2] Structure of the location resource. <http://hl7.org/fhir/location.html>, 12.07.2020.
- [3] Official fhir website. <http://hl7.org/fhir/>, 12.07.2020.
- [4] Wikipedia page of fhir. https://en.wikipedia.org/wiki/Fast_Healthcare_Interoperability_Resources, 12.07.2020.
- [5] Official website of the giz. <https://www.giz.de/en/html/index.html>, 12.07.2020.
- [6] Official openimis website. <https://www.openimis.org/>, 12.07.2020.
- [7] Official website of the sdc. <https://www.eda.admin.ch/deza/en/home/sdc.html>, 12.07.2020.
- [8] Official list of sdg indicators. <https://unstats.un.org/sdgs/metadata/>, 12.07.2020.
- [9] Wikipedia page of sdg. https://en.wikipedia.org/wiki/Sustainable_Development_Goals, 12.07.2020.
- [10] Official website of the swiss tph. <https://www.swisstph.ch/en/>, 12.07.2020.
- [11] German wikipedia page of the swiss tph. https://de.wikipedia.org/wiki/Schweizerisches_Tropen-_und_Public-Health-Institut#Forschungsschwerpunkte_und_Struktur, 12.07.2020.
- [12] Official website of docker. <https://www.docker.com/>, 13.07.2020.
- [13] Structure of an extension. <https://www.hl7.org/fhir/extensibility.html>, 13.07.2020.
- [14] Value set location physical type. <http://hl7.org/fhir/valueset-location-physical-type.html>, 13.07.2020.
- [15] Official website of the microsoft sql server. <https://www.microsoft.com/de-de/sql-server/sql-server-downloads>, 13.07.2020.
- [16] openimis database installation wiki page. <https://openimis.atlassian.net/wiki/spaces/OP/pages/906592471/WA2.1+Database+installation>, 13.07.2020.
- [17] Installation manuals for the modular and legacy version of openimis. <https://openimis.atlassian.net/wiki/spaces/OP/pages/786104344/Installation+and+Country+Localisation>, 13.07.2020.

- [18] Official openimis github page. <https://github.com/openimis>, 13.07.2020.
- [19] Installation guide for the modular version of openimis. <https://openimis.atlassian.net/wiki/spaces/OP/pages/963182705/MO1.1+Install+the+modular+openIMIS+using+Docker>, 13.07.2020.
- [20] Github link to the development tool repository. <https://github.com/openimis/openimis-dev-tools/tree/developinitializing-modular-be-in-windows>, 14.07.2020.
- [21] Fhir v3 location resource page. <http://hl7.org/fhir/STU3/location.html>, 15.07.2020.
- [22] Link to the openimis-be-api_fhir_py repository. https://github.com/openimis/openimis-be-api_fhir_py, 15.07.2020.
- [23] Link to the apps.py script in github. https://github.com/openimis/openimis-be-api_fhir_py/blob/master/api_fhir/apps.py, 15.07.2020.
- [24] Link to the configurations directory in github. https://github.com/openimis/openimis-be-api_fhir_py/tree/master/api_fhir/configurations, 15.07.2020.
- [25] Fhir r4 mapping wiki page. <https://openimis.atlassian.net/wiki/spaces/OP/pages/1233649676/openIMIS+FHIR+R4+Overview+Page>, 15.07.2020.
- [26] Link to the generalconfiguration.py script in github. https://github.com/openimis/openimis-be-api_fhir_py/blob/master/api_fhir/configurations/generalConfiguration.py, 15.07.2020.
- [27] Link to the github repository of openimis-be-location.py. https://github.com/openimis/openimis-be-location_py, 15.07.2020.
- [28] Link to the moduleconfiguration.py script in github. https://github.com/openimis/openimis-be-api_fhir_py/blob/master/api_fhir/configurations/moduleConfiguration.py, 15.07.2020.
- [29] Link to the paginations.py script in github. https://github.com/openimis/openimis-be-api_fhir_py/blob/master/api_fhir/paginations.py, 15.07.2020.
- [30] Link to the permissions.py script in github. https://github.com/openimis/openimis-be-api_fhir_py/blob/master/api_fhir/permissions.py, 15.07.2020.
- [31] Link to the urls.py script in github. https://github.com/openimis/openimis-be-api_fhir_py/blob/master/api_fhir/urls.py, 15.07.2020.
- [32] Link to the views.py script in github. https://github.com/openimis/openimis-be-api_fhir_py/blob/master/api_fhir/views.py, 15.07.2020.
- [33] Official website of pypi. <https://pypi.org/>, 15.07.2020.
- [34] Official fhirbase website. <https://www.health-samurai.io/fhirbase>, 16.07.2020.
- [35] fhir.resources package on pypi. <https://pypi.org/project/fhir.resources/>, 16.07.2020.

- [36] Official website of hisp india. <https://hispindia.org/>, 16.07.2020.
- [37] Pycharm page of the official jetbrains website. <https://www.jetbrains.com/idea/pycharm/>, 16.07.2020.
- [38] openimis-be-api_fhir_r4_py repository link. https://github.com/openimis/openimis-be-api_fhir_r4_py, 16.07.2020.
- [39] Link to the github repository of openimis-be-medical_py. https://github.com/openimis/openimis-be-medical_py, 16.07.2020.
- [40] openimis wiki page for michel borer's bachelor thesis. <https://openimis.atlassian.net/wiki/spaces/OP/pages/1277231105/Database+migration+to+PostgreSQL+explorative+pilot>, 16.07.2020.
- [41] openimis-fhir-data-migration_py repository link. https://github.com/openimis/openimis-fhir-data-migration_py, 16.07.2020.
- [42] openimis wiki page for the data migration tool. <https://openimis.atlassian.net/wiki/spaces/OP/pages/1554448385/open-IMIS+FHIR+data+migration+tool>, 16.07.2020.
- [43] Official postgresql website. <https://www.postgresql.org/>, 16.07.2020.
- [44] Self created openimis fhir r4 package. <https://pypi.org/project/openimis-be-api-fhir-r4/>, 16.07.2020.
- [45] Repository of the openimis databases. https://github.com/openimis/database_ms_sqlserver, 17.07.2020.
- [46] openimis demo video on youtube. <https://www.youtube.com/watch?v=h3fj90penfUt=34s>, 17.07.2020.
- [47] Alexander Schulze. Introducing openimis – an open source solution for universal health coverage. <https://www.youtube.com/watch?v=6UnOnIUDXcY>, 12.07.2020.
- [48] SwissTPH. From imis to openimis. <https://www.swisstph.ch/de/ueberuns/scih/sysu/health-economics-and-financing/imis/>, 12.07.2020.
- [49] SwissTPH. Current openimis implementations. <http://openimis.org/>, 12.07.2020.



Appendix

A.1 List of Abbreviations

openIMIS	open source Insurance Management Information System
Swiss TPH	Swiss Tropical and Public Health Institute
FHIR R4	Fast Healthcare Interoperability Resources Release 4
HL7	Health Level Seven
SDG	Sustainable Development Goals
SDC	Swiss Development Cooperation
GDC	German Development Cooperation
GIZ	Gesellschaft für Internationale Zusammenarbeit
SQL	Structured Query Language
JSON	JavaScript Object Notation
XML	Extensible Markup Language
REST	Representational State Transfer
API	Application Programming Interface
URL	Uniform Resource Locator
PyPI	Python Package Index
HTTP	Hypertext Transfer Protocol
UUID	Universally Unique Identifier

A.2 List of used Modules

<code>openimis-be.py</code>	https://github.com/openimis/openimis-be.py
<code>openimis-be-api_fhir.py</code>	https://github.com/openimis/openimis-be-api_fhir.py
<code>openimis-be-claim_batch.py</code>	https://github.com/openimis/openimis-be-claim_batch.py
<code>openimis-be-claim.py</code>	https://github.com/openimis/openimis-be-claim.py
<code>openimis-be-contribution.py</code>	https://github.com/openimis/openimis-be-contribution.py
<code>openimis-be-core.py</code>	https://github.com/openimis/openimis-be-core.py
<code>openimis-be-insuree.py</code>	https://github.com/openimis/openimis-be-insuree.py
<code>openimis-be-location.py</code>	https://github.com/openimis/openimis-be-location.py
<code>openimis-be-medical_pricelist.py</code>	https://github.com/openimis/openimis-be-medical_pricelist.py
<code>openimis-be-medical.py</code>	https://github.com/openimis/openimis-be-medical.py
<code>openimis-be-policy.py</code>	https://github.com/openimis/openimis-be-policy.py
<code>openimis-be-product.py</code>	https://github.com/openimis/openimis-be-product.py
<code>openimis-be-report.py</code>	https://github.com/openimis/openimis-be-report.py

A.3 List of Mapped Resources

Claim	https://www.hl7.org/fhir/claim.html
ClaimResponse	https://www.hl7.org/fhir/claimresponse.html
Coverage	https://www.hl7.org/fhir/coverage.html
Patient	https://www.hl7.org/fhir/patient.html
Practitioner	https://www.hl7.org/fhir/practitioner.html
PractitionerRole	https://www.hl7.org/fhir/practitionerrole.html
Location	https://www.hl7.org/fhir/location.html
CoverageEligibilityRequest	https://www.hl7.org/fhir/coverageeligibilityrequest.html
CoverageEligibilityResponse	https://www.hl7.org/fhir/coverageeligibilityresponse.html
CommunicationRequest	https://www.hl7.org/fhir/communicationrequest.html
Condition	https://www.hl7.org/fhir/condition.html
Medication	https://www.hl7.org/fhir/medication.html
ActivityDefinition	https://www.hl7.org/fhir/activitydefinition.html
HealthcareService	https://www.hl7.org/fhir/healthcareservice.html

A.4 End Product

<code>openimis-be-api_fhir_r4.py</code>	https://github.com/openimis/openimis-be-api_fhir_r4.py
<code>openimis-fhir-data-migration.py</code>	https://github.com/openimis/openimis-fhir-data-migration.py
<code>openimis-be-api-fhir-r4 1.0.0</code>	https://pypi.org/project/openimis-be-api-fhir-r4/

A.5 Mapping Tables

A.5.1 Overview Table

FHIR R4 Resource	openMIS database tables	Notes	Status
Claim	<ul style="list-style-type: none"> tblClaim tblClaimItems tblClaimServices 	Request properties are mapped to Claim	mapped
ClaimResponse	<ul style="list-style-type: none"> tblClaim tblClaimItems tblClaimServices 	Response properties are mapped to Claim	mapped
Coverage	tblPolicy		mostly mapped
Patient	tblInsurees		mapped
Practitioner	tblClaimAdmin	used to represent base fields of Claim Administrator (without relation with health facility- FHIR R4 Location)	mapped
PractitionerRole	tblClaimAdmin	used to represent a relation between base ClaimAdmin (FHIR R4 Partitioner) and Health facility (FHIR R4 Location)	mapped
Location	tblLocation	tblHF now mapped as HealthcareService	mapped
CoverageEligibilityRequest	CoverageEligibilityRequest		mapped
CoverageEligibilityResponse	CoverageEligibilityResponse		mapped
CommunicationRequest	tblFeedback		mapped
Condition	tblICDCodes		mapped
Medication	tblItems		mapped
ActivityDefinition	tblServices		mapped
HealthcareService	tblHF		mapped

Figure A.1:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1233649676/openIMIS+FHIR+R4+Overview+Page>

A.5.2 Claim Table

FHIR R4 field	openMIS field	notes	mapping status
identifier	tblClaim.ClaimID / tblClaim.ClaimUUID / tblClaim.ClaimCode	Claim can have multiple identifiers. The most important one is mapped from ClaimUUID.	mapped
status	one of [entered, checked, reviewed, valuated] based on tblClaim.ClaimStatus	entered state by default	
patient	tblClaim.InsureeUUID	reference to Patient resource	mapped - mandatory
billablePeriod	tblClaim.DateFrom / tblClaim.DateTo	DateFrom and DateTo are mapped to a period of time between them	mapped - mandatory
diagnosis	tblClaim.ICDID / tblClaim.ICDID1 / tblClaim.ICDID2 / tblClaim.ICDID3 / tblClaim.ICDID4	reference to Condition resource There is an error with the POST for both versions. When the error is found complete the mapping.	mapped - mandatory from fhir to imis not mapped (diagnosis)
provider	tblClaim.Adjuster	to be validated	not mapped
total	tblClaim.Claimed	mapped as Money data type	mapped - mandatory
created	tblClaim.DateClaimed		mapped - mandatory
supportingInfo	tblClaim.Explanation / tblClaim.Guaranteed / tblClaimItems.Availability / tblClaimItems.Explanation / tblClaimServices.Explanation	information category is distinguishing the type of information renamed from information (STU3) to supportingInfo (R4)	mapped
facility	tblClaim.HFID	reference to HealthcareService resource FHIR specification: reference to Location ⚠	mapped - mandatory
enterer	tblClaim.ClaimAdminID	reference to Practitioner resource	mapped
type	tblClaim.VisitType		mapped - mandatory
item.productOrService	tblClaimItems.tblItem.ItemCode / tblClaimServices.tblServices.ServiceCode	actual Code value	mapped - mandatory
item.extension.productOrServiceReference	tblClaimItems.ItemUUID / tblClaimServices.ServiceUUID	reference to Medication resource if medical item reference to ActivityDefinition resource if medical service	mapped
item.quantity	tblClaimItems.QtyProvided / tblClaimServices.QtyProvided		mapped
item.unitPrice	tblClaimItems.PriceAsked / tblClaimServices.PriceAsked	price could differ from the price defined in medical item/service definition	mapped
item.category.text	"service" or "item"	distinguishing whether mapping is done from tblClaimServices or tblClaimItems	mapped - mandatory
provider	tblClaimAdmin.ClaimAdminUUID	reference to PractitionerRole resource	mapped for db tool reason
priority	code = "normal"	CodeableConcept	mapped for db tool reason
use	"claim"		mapped for db tool reason
status	"active"		mapped for db tool reason
claim.insurance.focal	True		mapped for db tool reason
claim.insurance.sequence	0		mapped for db tool reason
claim.insurance.coverage	tblPolicy.PolicyUUID	reference to Coverage resource	mapped for db tool reason

Figure A.2:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1389592619/FHIR+R4+-+Claim>

A.5.3 ClaimResponse Table

FHIR R4 field	OpenMIS field	notes	mapping status
identifier	tblClaim.ClaimID / tblClaim.ClaimUUID / tblClaim.ClaimCode	Claim can have multiple identifiers. The most important one is mapped from ClaimUUID.	mapped
patient	tblClaim.InsurerUUID	reference to Patient resource	mapped
outcome	tblClaim.ClaimStatus	changed from CodeableConcept to code	mapped
payment.adjustmentReason	tblClaim.Adjustment		mapped
total.category.code	based on the state: Approved , Valuated , Reinsured , Claimed	returning the benefit amount	mapped
total.amount	tblClaim.Approved / tblClaim.Valuated / tblClaim.Reinsured / tblClaim.Claimed	mapped as Money data type	mapped
communicationRequest	tblClaim.feedbackId	reference to CommunicationRequest resource decision if Feedback to be shared with Health Facilities ⚠	mapped
error.code.coding.code	tblClaim.rejectionReason		mapped
error.code.text	tblClaim.rejectionReason as primary language text		to be mapped
item.adjudication.reason	tblClaimItems.ClaimItemStatus / tblClaimItems.Justification / tblClaimItems.RejectionReason / tblClaimServices.ClaimServiceStatus / tblClaimServices.Justification / tblClaimServices.RejectionReason	adjudication.reason contains the rejection code (with adjudication.category = "rejected_reason")	mapped
item.adjudication.value	tblClaimItems.QtyProvided / tblClaimItems.QtyApproved / tblClaimServices.QtyProvided / tblClaimServices.QtyApproved		mapped
item.adjudication.amount	tblClaimItems.PriceAdjusted / tblClaimItems.PriceAdjusted / tblClaimItems.PriceApproved / tblClaimItems.PriceValuated / tblClaimServices.PriceAdjusted / tblClaimServices.PriceAdjusted / tblClaimServices.PriceApproved / tblClaimServices.PriceValuated	Monetary amount / value is taken from limitation value by default The openMIS field considered here depends on the status of the claims.	mapped
processNote.text	tblClaimItems.Justification / tblClaimItems.PriceOrigin / tblClaimServices.Justification / tblClaimServices.PriceOrigin	claimResponse.item.noteNumber can be used to join information about the mapped field and claim item	mapped
disposition			not mapped
created	TimeUtils.date	current date is taken on the moment of processing claim Should be mapped to ValidityTo for the date of last change?	mapped
request		reference to Claim	mapped
type	tblClaim.VisitType		mapped
status	tblClaim.ReviewStatus	status in [1: "Idle", 2: "Not Selected", 4: "Selected for Review", 8: "Reviewed", 16: "ByPassed"]	mapped
requestor	tblHF.HealthFacility	reference to HealthcareService resource	mapped
use	"claim"		mapped
insurer	"Organisation/openMIS"	"openMIS" from module configuration key	mapped

Figure A.3:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1389592652/FHIR+R4++ClaimResponse>

A.5.4 Coverage Table

FHIR R4 field	OpenIMIS field	notes	mapping status
Identifier	PolicyID / PolicyUUID		mapped
policyHolder	FamilyID		mapped
period	StartDate / ExpiryDate	start date and expiry date are mapped to a period of time between them	mapped
status	PolicyStatus		mapped
contract.term.asset.value dItem.net	PolicyValue	contract.valuedItem is now defined as contract.term.asset.value dItem	mapped
class	ProdID	renamed from grouping to class	mapped
contract.term.offer.party	OfficerID	contract.agent is now defined as contract.term.offer.party	mapped

Figure A.4:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1389297783/FHIR+R4+-+Coverage>

A.5.5 Patient Table

FHIR R4 field	openIMIS field	Notes	Mapping status
identifier	InsureeID / CHFID / passport / TypeOfId / InsureeUUID	there is a 0..* relation on identifier	InsureeID / CHFID / passport / InsureeUUID is mapped
name	LastName / OtherNames	name field contains values that are being mapped for both LastName and OtherNames fields	mapped
birthDate	DOB		mapped
gender	Gender	Link to gender_codes configuration key	mapped
maritalStatus	Marital	maritalStatus.text in [Married, Single, Divorced, Widowed, Not specified]	mapped
telecom	Phone / Email	telecom field contains values that are being mapped for both Phone and Email fields	mapped
photo.url	tblInsuree.PhotoID → tblPhotos.PhotoFolder + PhotoFileName	Should include the base URL	mapped
photo.creation	tblInsuree.PhotoID → tblPhotos.PhotoDate		mapped
generalPractitioner	HFID	reference/HealthcareService	mapped
address	CurrentAddress / GeoLocation	The patient can contain multiple addresses	mapped
link.other	link to tblInsuree.FamilyId → tblFamilies.Insureeid → tblInsurees.InsureeUUID	reference Patient/UUID UUID is head of the family	mapped
link.type	tblInsuree.relationship → tblRelations.relation		mapped
extension.isHead	tblInsuree.IsHead	url: isHead type: valueBoolean	mapped
extension.registrationDate	tblInsuree.ValidityFrom	url: registrationDate type: valueDateTime	mapped
extension.locationCode	link to tblInsuree.FamilyId → tblFamilies.LocationId → tblLocations.LocationUUID	url: locationCode type: valueReference	mapped
extension.educationCode.valueCoding.code extension.educationCode.valueCoding.display	tblInsuree.Education → tblEducations.EducationId tblInsuree.Education → tblEducations.Education	url: educationCode type: valueCoding	mapped
extension.professionCode.valueCoding.code extension.professionCode.valueCoding.display	tblInsuree.Profession → tblProfessions.ProfessionId tblInsuree.Profession → tblProfessions.Profession	url: professionCode type: valueCoding	mapped
extension.povertyStatus	tblInsuree → tblFamilies → Poverty	url: povertyStatus type: valueBoolean	mapped

Figure A.5:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1389133931/FHIR+R4+-+Patient>

A.5.6 Practitioner Table

FHIR R4 field	OpenIMIS field	notes	mapping status
identifier	claimAdminUUID / claimAdminCode	identifier field contains values that are being mapped for both claimAdminUUID and claimAdminCode fields	mapped
name	LastName / OtherNames	name field contains values that are being mapped for both LastName and OtherNames fields	mapped
birthDate	DOB		mapped
telecom	Phone / EmailId	telecom field contains values that are being mapped for both Phone and Email fields	mapped

Figure A.6:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1389592716/FHIR+R4+-+Practitioner>

A.5.7 PractitionerRole Table

FHIR R4 field	OpenIMIS field	notes	mapping status
practitioner - Reference(Practitioner)	Rest field of Claim Admin	example of reference (where "{ClaimAdminCode}" is the Claim Admin code): "Practitioner/{ClaimAdminCode}"	mapped
healthcareService - Reference(HealthcareService)	HFId	example of reference (where "{HFCode}" is the health facility code): "HealthcareService/{HFUID}"	mapped

Figure A.7:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1389592724/FHIR+R4+-+PractitionerRole>

A.5.8 Location Table

FHIR R4 field	openIMIS field	Notes	mapping status
identifier	tblLocations.LocationId / tblLocations.LocationCode / tblLocations.LocationUUID	Reference to FHIR Location from other resources is done through UUID field	mapped
name	tblLocations.LocationName		mapped
physicalType	tblLocations.LocationType / 'H'	type.coding contains one of ['R', 'D', 'W', 'V', 'H'] type.text is one of ['region', 'district', 'ward', 'village', 'hospital']	mapped
partOf	tblLocations.ParentLocationId → tblLocations.LocationUUID	Reference to the parent location. Regions don't have a parent. Composite structure: Region → District → Ward → Village	mapped

Figure A.8:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1389297887/FHIR+R4+-+Location>

A.5.9 CoverageEligibilityRequest Table

FHIR R4 field	OpenIMIS field	notes	mapping status
patient	CHFID		mapped
item.category	ServiceCode or ItemCode	Item or Service code as defined in openIMIS	mapped
item.productOrService	Medical Item or Service type	Specify if the item is a Service or an Item	mapped

Figure A.9:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1390182413/FHIR+R4+-+CoverageEligibilityRequest>

A.5.10 CoverageEligibilityResponse Table

FHIR R4 field	OpenMIS field	notes	mapping status
patient	eligibility_request		not mapped
insurance.item.benefit.allowedUnsignedInt	prod_id / total_admissions_left / total_visits_left / total- consultations_left / total_surgeries_left / total_deliveries_left / total_antenatals_left / service_left / item_left	item field contains all the information mapped to all listed fields benefitBalance is redefined as item and financial as item.benefit	mapped
insurance.item.benefit.allowedMoney	consultation_amount_left / surgery_amount_left / delivery_amount_left / hospitalization_amount_left / antenatal_amount_left	item field contains all the information mapped to all listed fields benefitBalance is redefined as item and financial as item.benefit	mapped
insurance.item.excluded	is_item_ok / is_service_ok	item field contains all the information mapped to all listed fields	mapped

Figure A.10:
<https://openimis.atlassian.net/wiki/spaces/OP/pages/1389297897/FHIR+R4+-+CoverageEligibilityResponse>

A.5.11 Communicationrequest Table

FHIR R4 field	OpenMIS field	notes	mapping status
identifier	feedbackId		mapped
reasonCode	CareRendered / PaymentAsked / DrugPrescribed / DrugReceived / Assessment	reasonCode field contains all the information mapped to all listed fields	mapped
occurrenceDateTime	feedBackDate		mapped
status	"active"		mapped

Figure A.11:
<https://openimis.atlassian.net/wiki/spaces/OP/pages/1389592873/FHIR+R4+-+CommunicationRequest>

A.5.12 Condition Table

FHIR R4 field	OpenMIS field	notes	mapping status
identifier	tblICDCodes.ICDID / tblICDCodes.ICDUUID / tblICDCodes.ICDCCode	tblICDCodes doesn't contains UUID identifier. This should be added in the next release.	mapped
code.coding	tblICDCodes.ICDCCode	Condition resource has a different field for code and is not part of the identifier.	mapped
code.text	tblICDCodes.ICDName	The name of the diagnosis.	mapped
recordedDate	tblICDCodes.ValidityFrom	date from when the diagnosis is valid.	mapped
subject	reference.type = "Patient"	reference to Patient resource	only mapped to be able to create a database

Figure A.12:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1399914531/FHIR+R4+-+Condition>

A.5.13 Medication Table

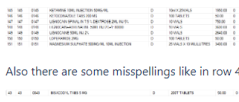
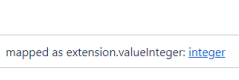
FHIR R4 field	openMIS field	notes	mapping status
identifier	tblItems.ItemID / tblItems.ItemUUID / tblItems.ItemCode		mapped
code.coding	tblItems.ItemCode	The code of the medication. Duplicated in identifier.	mapped
code.text	tblItems.ItemName	The name of the medication.	mapped
form	tblItems.ItemPackage	string part: "tables", "pieces", etc.	mapped as whole package
amount	tblItems.ItemPackage	integer part: 1000, 100, etc If not possible to split then keep only form field ItemPackage was not split because not all notations in the demo database are uniformly entered. For instance row 145 and 151.  Also there are some misspellings like in row 43. 	not mapped but possible see notes
frequency	tblItems.ItemFrequency	mapped as extension.valueInteger: integer	mapped
extension.unitPrice	tblItems.ItemPrice	mapped as Money openMIS extension to Medication url=unitPrice, valueMoney.value as tblItems.ItemPrice valueMoney.currency to be identified	mapped
extension.useContext.code	one of ["gender", "age", "venue"]	extension see UsageContext	mapped
extension.useContext.valueCodeableConcept.text	one of: <ul style="list-style-type: none"> tblItems.ItemPatCat decomposed in "gender" and "age" tblItems.ItemCareType for "venue" code 	if multiple values per code then duplicate the code for each value: ex. service available for man and women "gender" code "gender" and "age" codes can be found twice tblServices.ServPatCat is binary coded: Kids*8+Adults*4+Womens*2+Mens	mapped
extension.topic	<ul style="list-style-type: none"> tblItems.ItemType 	extension for DefinitionTopic	mapped

Figure A.13:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1400045588/FHIR+R4+-+Medication>

A.5.14 ActivityDefinition Table

FHIR R4 field	openMIS field	notes	mapping status
identifier	tblServices.ServiceID / tblServices.ServiceUUID / tblServices.ServCode		mapped
status	string "active"	mandatory => only valid services are managed	mapped
date	tblServices.ValidityFrom	the last date changed	mapped
name	tblServices.ServCode	the code of the medical service.	mapped
title	tblServices.ServName	the name of the medical service.	mapped
useContext.code	one of ["gender", "age", "workflow", "venue"]	see UsageContext	mapped
useContext.valueCodeableConcept.text	one of: <ul style="list-style-type: none"> tblServices.ServPatCat decomposed in "gender" and "age" tblServices.ServCategory for "workflow" code tblServices.ServCareType for "venue" code 	if multiple values per code then duplicate the code for each value: ex. service available for man and women "gender" code "gender" and "age" codes can be found twice tblServices.ServPatCat is binary coded: Kids*8+Adults*4+Womens*2+Mens	mapped
topic	tblServices.ServType	extension for DefinitionTopic	mapped
code	tblServices.ServCode and tblServices.ServName	can replace name and title	mapped
frequency	tblServices.ServFrequency	mapped as extension.valueInteger: integer	mapped
unitPrice	tblServices.ServPrice	mapped as extension.valueMoney: Money	mapped

Figure A.14:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1400012844/FHIR+R4+-+ActivityDefinition>

A.5.15 HealthcareService Table

FHIR R4 field	openMIS field	Notes	mapping status
identifier	tblHF.HFid / tblHF.HFCode / tblHF.HFUUID	Reference to FHIR HealthcareService from other resources is done through UUID field	mapped
name	tblHF.HFName		mapped
category	tblHF.HFLevel	type.coding contains one of ['H', 'C', 'D'] type.text is one of ['hospital', 'hospital center', 'dispensary']	mapped
type	tblHF.HFCareType	type.coding contains one of ['I', 'O', 'B'] type.text is one of ['in-patient', 'out-patient', 'both']	mapped
specialty	tblHF.HFSublevel	link to tblHFSublevel: <ul style="list-style-type: none"> coding.code as HFSublevel (id) text as HFSublevelDesc 	mapped
location	tblHF.LocationId	reference to Location/UUID	mapped
extraDetails	tblHF.HFAddress		mapped
telecom	tblHF.Phone / tblHF.Fax / tblHF.eMail	telecom field contains values that are being mapped for Phone, Fax and Email fields	mapped
program	tblHF.LegalForm	link to tblLegalForms: <ul style="list-style-type: none"> coding.code as LegalFormCode (id) text as LegalForms 	mapped
coverageArea	list of [reference tblLocation]	all tblLocations->tblHFCatchment->tblHF with ValidityTo=null	mapped

Figure A.15:

<https://openimis.atlassian.net/wiki/spaces/OP/pages/1517617153/FHIR+R4+-+HealthcareService>

Declaration on Scientific Integrity

Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor

Faris Ahmetasevic

Matriculation number — Matrikelnummer

2015-059-538

Title of work — Titel der Arbeit

Integration of Fast Healthcare Interoperability Resources (HL7 FHIR) into the openMIS open source Health Insurance Management System

Type of work — Typ der Arbeit

Bachelor thesis

Declaration — Erklärung

I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 20.07.2020



Signature — Unterschrift