DEAD

Final Report

Workgroup 102

Maryna Abitova (01468249)

Cordula Eggerth (00750881)

Veronika Lomasow (01200993)

Neli Petkova (01404258)

VU Advanced Software Engineering

June 2018

Table of Contents

1. Project Status Report

The Telemedicine System aims to consider the provision of various domain-specific functionalities, be they tailored to a certain user group or general for all users. Reflecting the different user groups' needs regarding the telemedicine services domain is the primary requirement. Therefore, it was necessary for us as a project team to determine the most important user groups of such a Telemedicine System. These user groups are visible as "actors" in the use case diagram. We decided to implement the roles of Patient, National Health Organization (NHO), Insurance, Pharmacy and Doctor, who have all distinct functionalities, which are reflected notably in the use cases (in the chapter on the "Scenarios view") plus the device, which can be a e.g. a sensor, a portable device, or a fixed installation. Some of the functionalities such as exporting transaction account data, the user management functionalities (such as registration, login or logout) and the like are common functionalities, which are relevant for more than one user group.

We identified the main use cases, which implied some constraints on what situations (or rather functionalities) are covered by our Telemedicine System and what not. A Patient is able to view his personal medical record items, and to order prescriptions listed in a specific medical record item. As an assumption, we agreed that a specific medical record item can contain only one prescription, which can be ordered from the pharmacy. Once the order is made, an invoice will be generated and will be visible for the Patient. If the Patient intends so, he can export his invoices and/or personal medical record. He can also to open the invoice details and also to choose to pay an invoice which is currently due. We assumed that once an invoice is paid, the boolean variable "paid" in the invoice is set as "true", which indicates that the invoice is settled, and that this is a transaction. Such transactions (i.e. paid invoices) will be listed in the transaction account, if a Doctor, a Pharmacy or an Insurance employee requests the list of transactions. Moreover, the Patient can create a Remote Patient Monitoring Item, which means that he can insert personal medical data into the system and submit this data to the Doctor via the system. The Doctor can then review the data and produce a medical record item based on the remote patient monitoring item, he reviewed. We assumed that a Doctor can only review remote patient monitoring item, which is data entered by the Patient from a remote place, but not change it. The Doctor can react in the way that he creates a medical Record item and for instance adds a prescription, which he deems necessary to treat the Patient's medical condition, as mentioned before. With regard to medical record items, the Doctor is allowed to change them, and to add new medical record items, if he examines a Patient. Upon completion of the medical record item data entry, an invoice is generated and added to the Patients invoice list. The Doctor, the Pharmacy, the Insurance as well as the Patient can generate and export documents, for which they have permission. The Patient can export his personal Medical Record Items as well as his personal Invoices. The Doctor and the Pharmacy can view and export Medical Record Items in general and can view their own

transactions, which means the invoice payments made to them. The Insurance can view and export all the related transactions. The NHO can gather (i.e. query) anonymized data on the general population for statistics purposes. Therefore, the personalized sensitive data of patients will be masked, before the report is shown to the NHO. The NHO has the permission to register new Doctors and new Pharmacies – this means that the latter cannot register by themselves but need the permission from the NHO. In addition to these use cases, the Telemedicine System performs automated notifications, for instance if an invoice from the Insurance is sent to the Patient to pay his monthly contribution, or if a Patient entered a serious condition during the Remote Patient Monitoring process, which needs immediate attention from a Doctor. These automated notifications are not explicitly shown as use cases, as we assumed that the system does them by itself based on pre-specified rules, and therefore there is no direct starting action by an actor necessary.

Considering the software architecture, we use the Layered Architecture Pattern, which consists of the Presentation & Application Layer, the Domain Layer (comprising the Domain Logic and the Domain Model), and the Persistence Layer. The Presentation & Application Layer deals with taking user input and showing the results of the lower layers to the user via the user interface. The Presentation & Application Layer interacts with the classes of the Domain Logic of the Domain Layer and Domain Model classes are used within the project at all times to implement the respective domain functionalities in the Telemedicine System. The Domain Logic implements the functionalities inherent to the domain of telemedicine, and which have been identified as main use cases in the "Scenarios view". To store data, retrieve data and perform changes on the data, the Persistence Layer is used. It deals with the data and is for instance charged with serialization and/or database operations. The Persistence Layer used DAO (data access object) interfaces, which are implemented by the specific type of persistent data operations class (e.g. serialization, database) used. The DAO interfaces were notably used for practical reasons as the EMF code-generation would overwrite serialization part, if this is not separate. Further details on our suggestions for the implementation of the Layers Architecture Pattern are given in the chapter "Logical view", more precisely in the class and package diagram.

Our model code was generated using Eclipse Modeling Framework (EMF). The Ecore class diagram can be found in the "Logical View" chapter, where the details are available. Overall, this diagram contains the model and specifies for instance necessary data types for the code generation (e.g. Calendar) and enums (e.g. MedicalCondition, RiskPerception) which are subsequently being used in the GUI and further implementation code as well as in the DSL.

We have also created XText Project to create a DSL, which aims to simplify the retrieval of patient data items by various systems. The purpose of this is to create a specific language which would allow to express a tailored solution to perform patient data queries more clearly. Our DSL Grammar is used to execute 3 functionalities: retrieval of medical record, patient

monitoring items or prescriptions; placing of these items in data source (inserting); and also removing of the particular datasets. The DSL queries has been chosen to be a bit similar to SQL and other query languages, so it is easy to understand and at least some of the stakeholders will already have experience using query language. Thus, they will not have troubles in using our DSL. Using this DSL Grammar enables us to create project using clearly defined syntax, which is understandable for all groups of stakeholders.

2. Division of Labor between Workgroup Participants

Team Member	Task
Marina Abitova	- Project Status Report - GUI screenshots presentation according to the real GUI interface - Update of "Explanatory Comments on the Class & Package Diagram" - Implementation of GUI windows related to Invoice Management / Transactions - Implementation of InvoiceMgmt in Domain Logic package - Implementation of DSL (query language for patient data) & creating instances - "dsl" documentation
Cordula Eggerth	- Project Status Report - EMF/Ecore Class Diagram - In Domain Model: EMF Generating of Code - User Interface implementation of basic layout according to the GUI mock-ups - User Interface implementation with Java WindowBuilder for UserMgmt, MedicalRecordMgmt, InsuranceMgmt, andPatientMgmt - In Domain Logic: UserMgmt, MedicalRecordMgmt, PatientMonitoringMgmt, DeviceMgmt, InsuranceMgmt - Data queries: Gather Anonymized

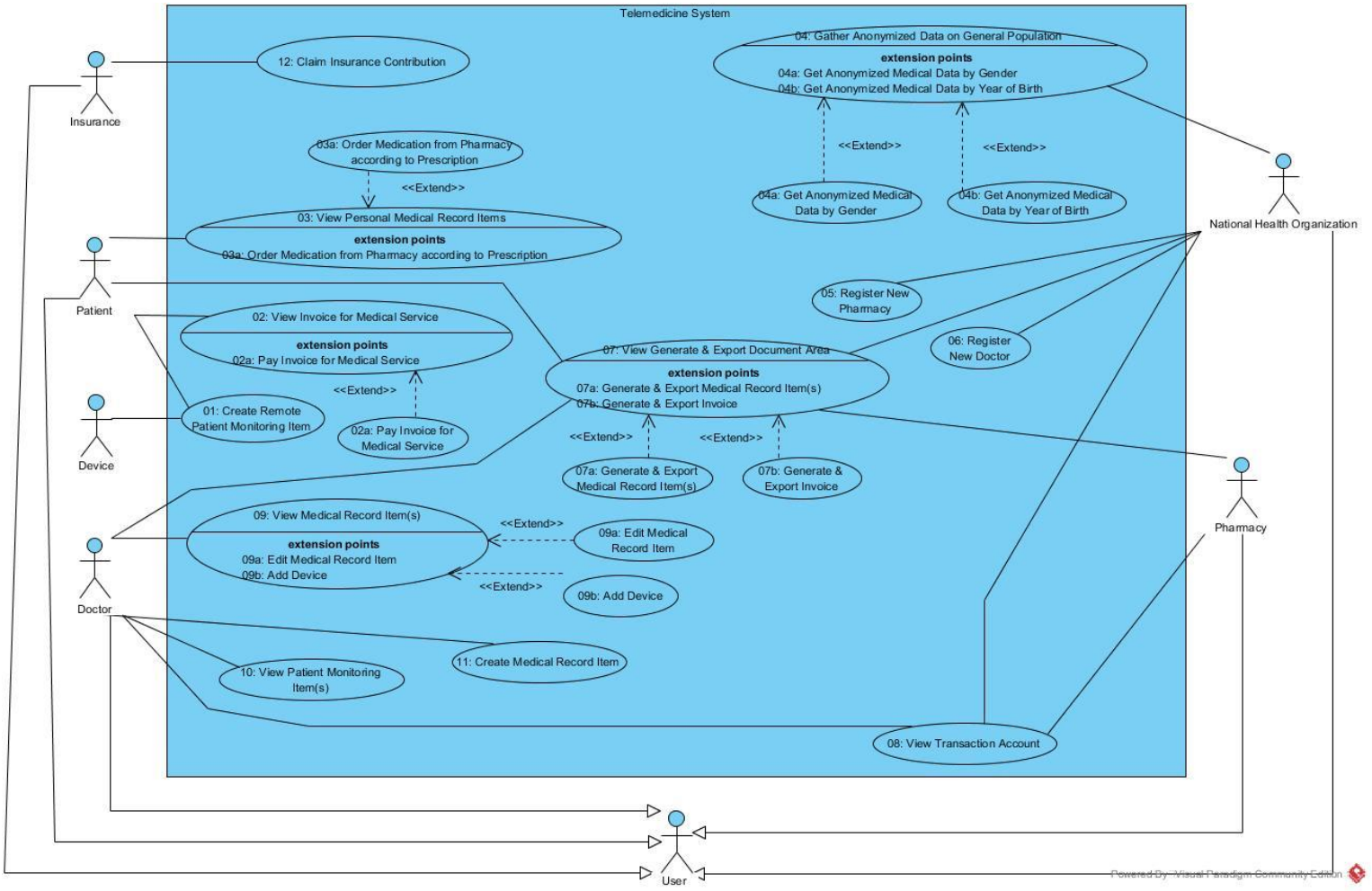
	Data Options (by Gender; by Year of Birth) - Use Case Diagram update (integrated new actors Insurance and Device, and the corresponding new use cases) - Class Diagram update (with regard to new use cases, actors and enums) - Deployment Diagram update (regarding SUPD Feedback comment on more detailed Physical View and distributed systems environment) - Javadoc Comments for the classes/package named above - Update of User Interface chapter with the real implemented GUI screens (see explanation in howto.pdf) - "howto" documentation - "dsl" documentation - Implementation of DSL (query language for patient data) & creating instances (using Xtext)
Veronika Lomasow	- Project Status Report - Sequence Diagram update (with regard to new use cases and actors, and SUPD feedback comments) - Activity Diagram update (with regard to new use cases and actors, and SUPD feedback comments) - Implementation of Notifications and Events, which are connected to the various management classes in Domain Logic package - GUI implementation for Notifications

	and Events consideration - Implementation of DSL (query language for patient data) & creating instances - "dsl" documentation
Neli Petkova	- Project Status Report - Component Diagram update (i.e. focusing on the main components and interfaces according to the SUPD feedback) - Implementation of ExportMgmt in Domain Logic package (for various different export formats) - GUI implementation for the functionality regarding use case "export" for all user types - Implementation of DSL (query language for patient data) & creating instances - "dsl" documentation

3. 4+1 Views Model of the Domain

3.1. Scenarios

Use Case Diagram:



Note:

The user-related processes of "login", "register", and "logout" are considered in the project, but should not be modelled elements of the use case diagram[1]. In this project, only the user can register herself/himself. A doctor or pharmacy needs to be "registered" (or respectively accredited by the national health organization.

In addition to this, processes which are automatically carried out by the Telemedicine System are not shown in the use case diagram, as they are triggered by the system itself. Such automated processes are for instance the scheduled events and notifications (in line with the guard-based examination and instrumentation).

[1] See Slides & References by Prof. Benkner (VO+UE Software Engineering).

Clarification with regard to the SUPD feedback:

What is meant by "Patient Monitoring Item"?

A Patient Monitoring Item is all information, which is remotely added via a Patient Account on a specific disease or condition. The information can either stem from a patient directly or from a device, which is tied to a patient.

What is meant by "Medical Record"?

A Medical Record Item is all information, which is produced directly by a Doctor. This means that it covers the results of an examination of the Patient by the Doctor for instance. Additionally, the Doctor can create a Medical Record Item when tackling a Patient Monitoring item (which was information entered remotely into the system). Then the Patient Monitoring item is marked as "treated" and the Medical Record as a reaction to the information entered remotely, is created.

If the difference between these two classes and the related use cases is still not clear, please contact the workgroup team for further discussion/explanation.

Use Case Descriptions:

Use Case:	**Create Remote Patient Monitoring Item**
Use Case ID:	01
Actor(s):	Patient; Device
Brief Description:	If the Patient uses remote medical services and devices, he can create a medical record item by himself, or a Device attached to the Patient's ID can enter Patient Monitoring Data. This occurs for instance if the Patient/Device enters measures of the blood sugar level in case of diabetes or if he is ill and measures the temperature or in case of blood pressure measurements for heart insufficiency conditions. For this, the data he enters is placed into a new Remote Patient Monitoring Item.
Pre-Conditions:	The Patient must be registered in and logged into the Telemedicine System. The Device must have been previously attached to a PatientId.

Post-Conditions:	The new Remote Patient Monitoring Item will be saved.
Main Success Scenario:	1. Patient clicks on “Create Patient Monitoring Item” 2. Patient fills in information required for the Patient Monitoring Item (in the respective form) 3. Patient saves the Patient Monitoring Item OR 1. Device adds Patient Monitoring Item by “create Patient Monitoring” procedure for devices. 2. Device saves the Patient Monitoring Item
Extensions:	--
Priority:	High
Performance Target:	System can handle several hundreds of Item creation requests on a daily basis.
Issues:	--

Use Case:	**View Invoice for Medical Service**
Use Case ID:	02
Actor(s):	Patient
Brief Description:	Patient view the Invoice for a Medical Service via the GUI
Pre-Conditions:	Patient must be logged into the system and must be in the invoices area
Post-Conditions:	Patient can decide if he wants to pay the Invoice now or later
Main Success Scenario:	1. Patient clicks on “View Invoice” 2. Patient reads Invoice
Extensions:	Patient can decide to pay the Invoice now (see Use Case 02a)
Priority:	High
Performance Target:	System can handle several hundreds of Invoice view requests

	daily.
Issues:	--

Use Case:	**Pay Invoice for Medical Service**
Use Case ID:	02a
Actor(s):	Patient
Brief Description:	Patient pays open Invoice for a medical service (e.g. for medication ordered from a Pharmacy, for contribution to National Health Organization, or for medical examination & treatment by Doctor)
Pre-Conditions:	Patient must be logged into the system and an open Invoice must be available
Post-Conditions:	Payment information (of the Invoice under concern) is transferred to the transaction account of the National Health Organization/Pharmacy/Doctor
Main Success Scenario:	1. Patient views an Invoice (which is still open for payment) 2. Patient clicks on "Pay Invoice" 3. Payment is confirmed 4. Payment information is entered into the respective transaction account of the creditor
Extensions:	--
Priority:	Medium-High
Performance Target:	System can handle several hundreds of payment requests per day
Issues:	Is invoice payment currency always euro?

Use Case:	**View Personal Medical Record Items**
Use Case ID:	03

Actor(s):	Patient
Brief Description:	The Patient can access the list of personal Medical Record Items, which have occurred in line with the Telemedicine System so far
Pre-Conditions:	Patient must be logged into the system. Access is restricted to the individual data entries for the Patient who is logged in.
Post-Conditions:	--
Main Success Scenario:	1. Patient goes to Personal Medical Records area 2. Patient views the list of his Medical Records associated with his user account
Extensions:	Order Medication from Pharmacy according to Prescription (see Use Case 03a)
Priority:	High
Performance Target:	System can handle several hundreds of view requests per day
Issues:	--

Use Case:	**Order Medication from Pharmacy according to Prescription**
Use Case ID:	03a
Actor(s):	Patient
Brief Description:	Patient orders medication prescribed by a Doctor from a Pharmacy
Pre-Conditions:	Patient must be logged into the system and must be in the “View Personal Medical Records” area
Post-Conditions:	Invoice is created so that the price of the medication can subsequently paid by the Patient into the Pharmacy’s transaction account
Main Success Scenario:	1. Patient is in the list of personal Medical Record Items 2. Patient clicks on “Order Medication” (from an available

	Pharmacy) where the Doctor entered a medication name in the prescription field 3. Medication order is confirmed
Extensions:	--
Priority:	Medium
Performance Target:	System can handle several hundreds of requests per day
Issues:	--

Use Case:	**Gather Anonymized Data on General Population**
Use Case ID:	04
Actor(s):	National Health Organization
Brief Description:	The National Health Organization can request medical record data on the general population in an anonymized form.
Pre-Conditions:	National Health Organization must be logged into the system
Post-Conditions:	--
Main Success Scenario:	1. National Health Organization clicks on "gather data on population" 2. Anonymized data is made available
Extensions:	- 04a Get Anonymized Medical Data by Gender - 04b Get Anonymized Medical Data by Year of Birth
Priority:	Medium
Performance Target:	Requests made by the National Health Organization can be served at any time
Issues:	Which export formats are offered for the data?

Use Case:	**Get Anonymized Medical Data by Gender**
Use Case ID:	04a

Actor(s):	National Health Organization (NHO)
Brief Description:	The National Health Organization can make a query for medical data by gender of the general population in an anonymized form.
Pre-Conditions:	National Health Organization employee must be logged into the system and must be in the area “Gather Anonymized Data on the General Population” in the GUI
Post-Conditions:	--
Main Success Scenario:	1. National Health Organization clicks on “Get Medical Data by Gender” 2. Anonymized data is made available and export options are available
Extensions:	--
Priority:	Medium
Performance Target:	Requests made by the National Health Organization can be served at any time
Issues:	What does the data presentation look like?

Use Case:	**Get Anonymized Medical Data by Year of Birth**
Use Case ID:	04b
Actor(s):	National Health Organization (NHO)
Brief Description:	The National Health Organization can make a query for medical data by year of birth of the general population in an anonymized form.
Pre-Conditions:	National Health Organization employee must be logged into the system and must be in the area “Gather Anonymized Data on the General Population” in the GUI
Post-Conditions:	--
Main Success Scenario:	1. National Health Organization clicks on “Get Medical

	Data by Year of Birth" 2. Anonymized data is made available and export options are available
Extensions:	--
Priority:	Medium
Performance Target:	Requests made by the National Health Organization can be served at any time
Issues:	What does the data presentation look like?

Use Case:	**Register New Pharmacy**
Use Case ID:	05
Actor(s):	National Health Organization
Brief Description:	The National Health Organization registers (or respectively accredits) a new Pharmacy in the Telemedicine System
Pre-Conditions:	National Health Organization must be logged into the system
Post-Conditions:	Registration process for the Pharmacy as a new user is completed
Main Success Scenario:	1. National Health Organization clicks on "Register New Pharmacy" 2. National Health Organization enters the registration data 3. National Health Organization confirms accreditation of new Pharmacy to the Telemedicine System
Extensions:	--
Priority:	Medium
Performance Target:	National Health Organization can register new Pharmacy at any time
Issues:	--

Use Case:	**Register New Doctor**
Use Case ID:	06
Actor(s):	National Health Organization
Brief Description:	The National Health Organization registers (or respectively accredits) a new Doctor in the Telemedicine System
Pre-Conditions:	National Health Organization must be logged into the system
Post-Conditions:	Registration process for the Doctor as a new user is completed
Main Success Scenario:	1. National Health Organization clicks on "Register New Doctor" 2. National Health Organization enters the registration data National Health Organization confirms accreditation of new Doctor to the Telemedicine System
Extensions:	--
Priority:	Medium
Performance Target:	National Health Organization can register new Doctor at any time
Issues:	--

Use Case:	**View 'Generate & Export Document' Area**
Use Case ID:	07
Actor(s):	Patient, Doctor, Pharmacy, National Health Organization
Brief Description:	This area shows the option available tailored to the respective user type (depending on their access rights)
Pre-Conditions:	User is logged into the system
Post-Conditions:	Document generation & export is chosen or not
Main Success Scenario:	1. Go to 'Generate & Export Document' Area 2. View available options (depending on the user type's access rights)

Extensions:	- Generate & Export Medical Record Item(s) (Use Case 07a) - Generate & Export Invoice (Use Case 07b)
Priority:	High
Performance Target:	Documents can be generated and exported at any time
Issues:	--

Use Case:	**Generate & Export Medical Record Item(s)**
Use Case ID:	07a
Actor(s):	Patient, Doctor, Pharmacy, National Health Organization
Brief Description:	Each user type can generate and export the document, which he is allowed to access, in the chosen document format
Pre-Conditions:	User is logged into the system and is in the "View 'Generate & Export Documents" area
Post-Conditions:	Medical Record Items information is generated and exported
Main Success Scenario:	1. Click on "Generate & Export Medical Record Items" 2. Choose document export format
Extensions:	--
Priority:	High
Performance Target:	Documents can be generated and exported at any time
Issues:	--

Use Case:	**Generate & Export Invoice**
Use Case ID:	07b
Actor(s):	Patient, National Health Organization
Brief Description:	Patient/National Health Organization can generate and export the Invoice, which he is allowed to access, in the chosen document

	format
Pre-Conditions:	Patient/National Health Organization is logged into the system and has the according rights
Post-Conditions:	Invoice is generated and export
Main Success Scenario:	1. Click on "Generate & Export Invoice Document" 2. Choose the document export format
Extensions:	--
Priority:	High
Performance Target:	Invoices can be generated and exported at any time
Issues:	--

Use Case:	**View Transaction Account**
Use Case ID:	08
Actor(s):	Doctor, Pharmacy, National Health Organization
Brief Description:	The named actors can view their own transaction account
Pre-Conditions:	Doctor/Pharmacy/National Health Organization is logged into the system
Post-Conditions:	--
Main Success Scenario:	1. Click on "View Transaction Account" 2. Read the information on transactions (i.e. paid Invoices)
Extensions:	--
Priority:	Medium
Performance Target:	Transaction account can be viewed at any time
Issues:	--

Use Case:	**View Medical Record Item(s)**

Use Case ID:	09
Actor(s):	Doctor
Brief Description:	The Medical Record Item(s) of a specific Patient are viewed
Pre-Conditions:	Doctor is logged into the system
Post-Conditions:	--
Main Success Scenario:	1. Click on “View Medical Record Item(s) 2. Select/Search for Patient
Extensions:	Edit Medical Record Item(s) (Use Case 09a) Add Device (User Case 09b)
Priority:	High
Performance Target:	Medical Record Item(s) can always be viewed
Issues:	--

Use Case:	**Edit Medical Record Item**
Use Case ID:	09a
Actor(s):	Doctor
Brief Description:	The Doctor can change the content of a Medical Record Item of a specific Patient
Pre-Conditions:	- Doctor is logged into the system - Doctor is in the “View Medical Records Area”
Post-Conditions:	Medical Record Item is modified and saved
Main Success Scenario:	1. Click on “Edit Medical Record Item” 2. Modify the item 3. Save the item
Extensions:	--
Priority:	Medium

Performance Target:	Medical Record Item(s) can be edited by a Doctor at any time
Issues:	--

Use Case:	**Add Device**
Use Case ID:	09b
Actor(s):	Doctor
Brief Description:	The Doctor can add a Device for PatientMonitoring to a specific Patient (i.e. linked to the Patient ID)
Pre-Conditions:	- Doctor is logged into the system - Doctor is in the "View Medical Records Area" and clicked on "View Medical Details" before
Post-Conditions:	Device is added and saved
Main Success Scenario:	1. Click on "Add Device" 2. Device is created by the system 3. Information is serialized by the system
Extensions:	--
Priority:	Medium
Performance Target:	Devices can be edited by a Doctor at any time
Issues:	--

Use Case:	**View Patient Monitoring Item(s)**
Use Case ID:	10
Actor(s):	Doctor
Brief Description:	The Doctor can view the content of a Patient Monitoring Item of a specific Patient
Pre-Conditions:	- Doctor is logged into the system

Post-Conditions:	--
Main Success Scenario:	4. Click on "View Patient Monitoring Item" 5. View the item
Extensions:	--
Priority:	Medium
Performance Target:	Patient Monitoring Item(s) can be viewed by a Doctor at any time
Issues:	--

Use Case:	**Create Medical Record Item**
Use Case ID:	11
Actor(s):	Doctor
Brief Description:	If the Patient visits a Doctor and the Doctor carries out an examination, the doctor can create a new Medical Record Item to capture the result of the examination. After the visit at the Doctor's place, an Invoice (containing the examination/treatment costs) is automatically sent to the Patient. In the Medical Record Item, the Doctor might also include information on the prescription of a medication, which the Patient can order from the Pharmacy.
Pre-Conditions:	The Doctor must be registered in and logged into the Telemedicine System.
Post-Conditions:	The new Medical Record Item will be saved.
Main Success Scenario:	1. Doctor clicks on "Create Medical Record Item" 2. Doctor fills in information required for the Medical Record Item (in the respective form) 3. Doctor saves the Medical Record Item
Extensions:	--
Priority:	High

Performance Target:	System can handle several hundreds of Medical Record Item creation requests on a daily basis.
Issues:	Can several prescriptions be added to one Medical Record Item by a Doctor?

Use Case:	**Claim Insurance Contribution**
Use Case ID:	12
Actor(s):	Insurance
Brief Description:	The Insurance can demand from the Patients that they pay their insurance contribution. For this, the Insurance adds an Invoice that all Patients name will receive.
Pre-Conditions:	The Insurance (employee) must be registered in and logged into the Telemedicine System.
Post-Conditions:	The new Invoice Item will be saved and the Patients will be notified.
Main Success Scenario:	1. Insurance (employee) clicks on "Claim Insurance Contribution" 2. Claim process is performed by the system (i.e. Invoice is added and sent to the Patients)
Extensions:	--
Priority:	High
Performance Target:	System can handle several hundreds of Invoice Item creation requests on a daily basis.
Issues:	-

3.2. Logical View

Class & Package Diagram:

Notes:

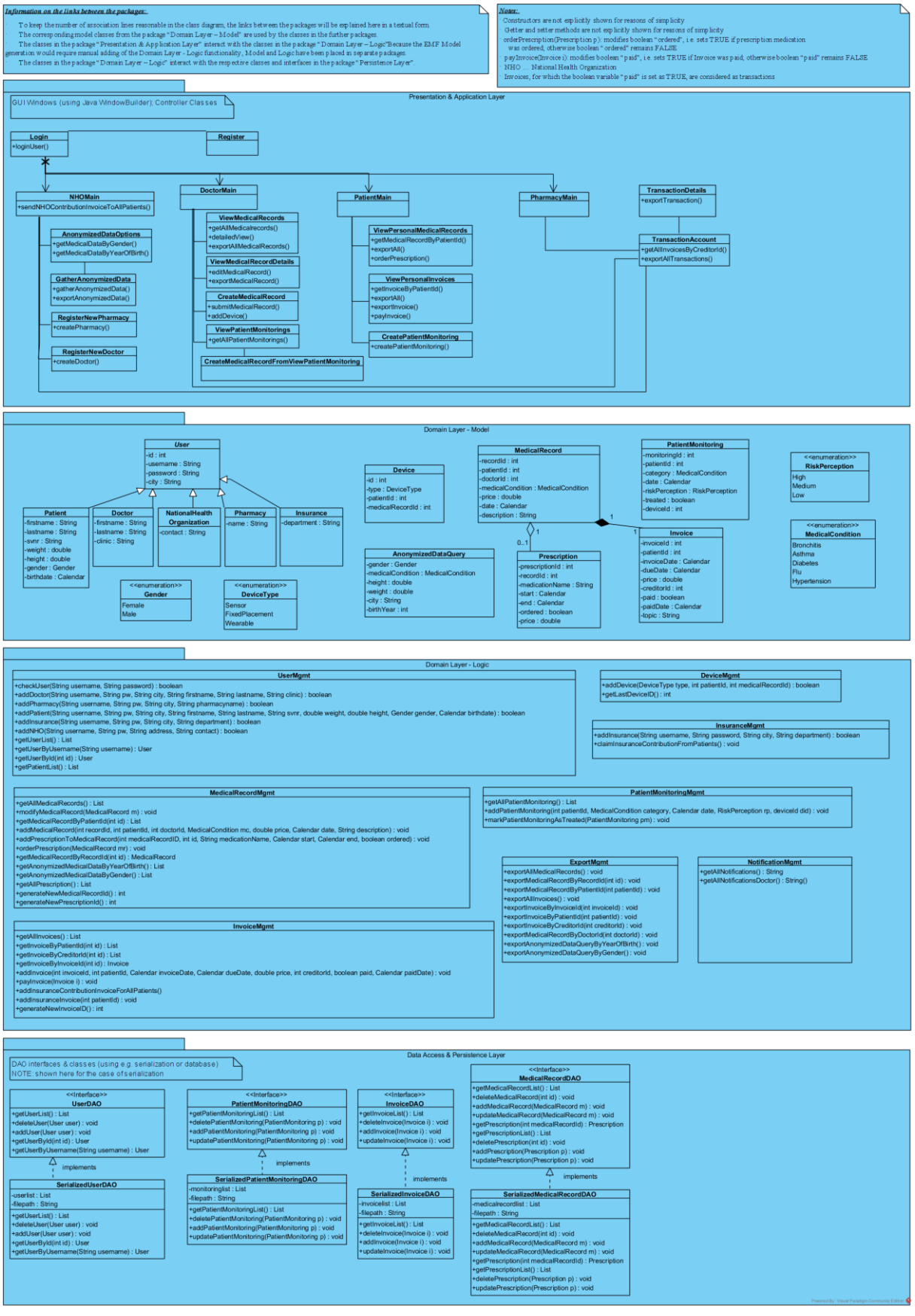
- Constructors are not explicitly shown for reasons of simplicity
- Getter and setter methods are not explicitly shown for reasons of simplicity
- orderPrescription(Prescription p): modifies boolean "ordered", i.e. sets TRUE if prescription medication was ordered; otherwise boolean "ordered" remains FALSE
- payInvoice(Invoice i): modifies boolean "paid", i.e. sets TRUE if Invoice was paid; otherwise boolean "paid" remains FALSE
- NHO … National Health Organization
- Invoices, for which the boolean variable "paid" is set as TRUE, are considered as transactions in the telemedicine system

Information on the links between the packages:

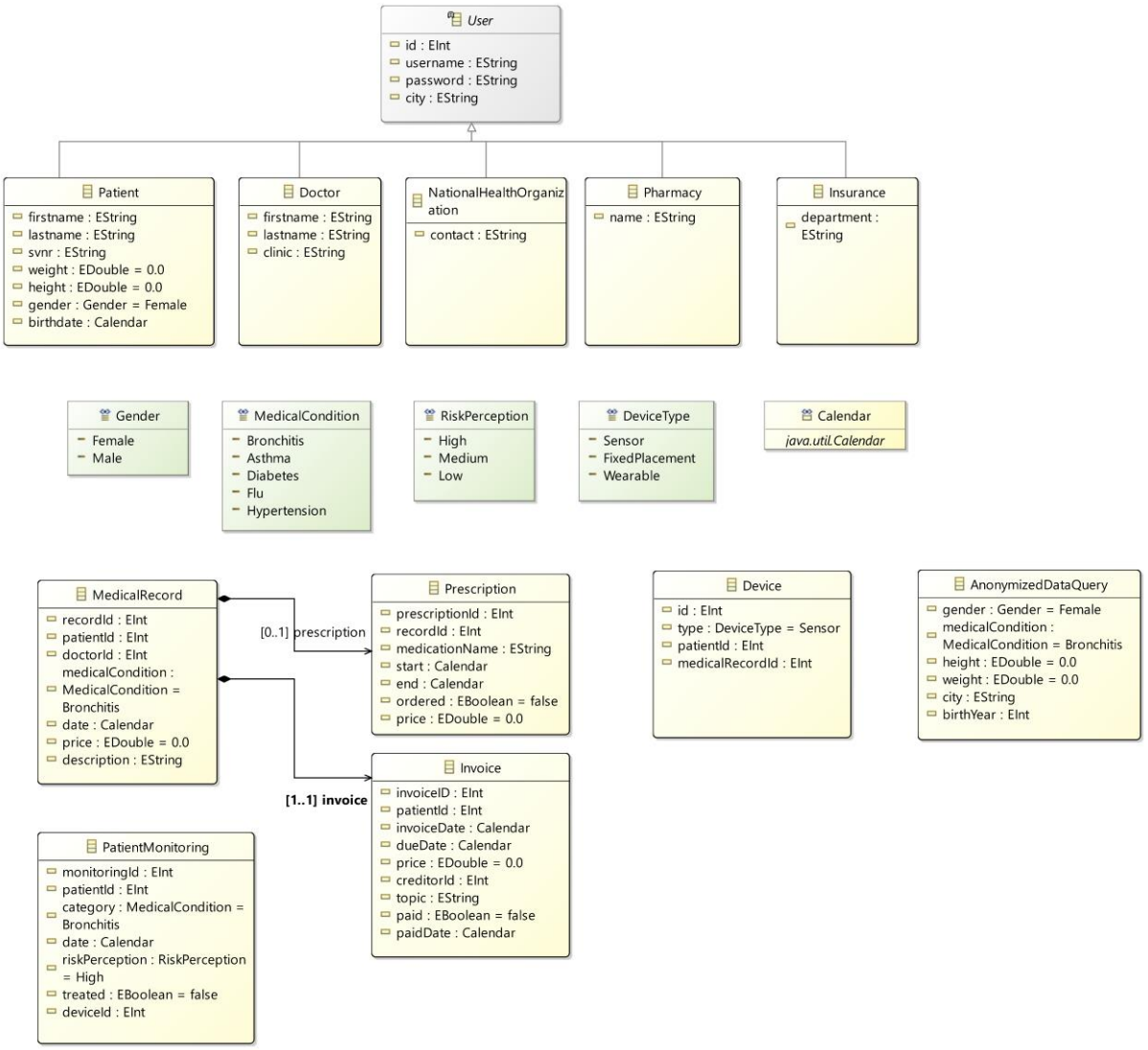
- To keep the number of association lines reasonable in the class diagram, the links between the packages will be explained here in a textual form.
- The corresponding model classes from the package "Domain Layer – Model" are used by the classes in the further packages.
- The classes in the package "Presentation & Application Layer" interact with the classes in the package "Domain Layer – Logic".
- The classes in the package "Domain Layer – Logic" interact with the respective classes and interfaces in the package "Persistence Layer".

Clarification regarding the comments from the SUPD feedback:

- The term "***Data Access & Persistence Laye***r" was chosen in line with the slides of Prof. Zdun according to the Layered Architecture Pattern. According to Prof. Zdun, there is not one single interpretation of a specific "layer" and the meaning of the specific chosen layer depends on the project. From our point of view, we chose to name the layer, which is handling the serialization and writes/reads data from the file, "Data Access & Persistence Layer". This term has been chosen because through this layer, the data is accessed, and the data is made persistent and/or read from a persistent storage (i.e. the corresponding serialization file).
- The ***remote PatientMonitoring items*** can either be produced ***by remote devices or by the Patient himself***. We decided to implement the data collecting functionality in the way showed in the PatientMonitoring model and management (logic) classes. The "device" has also been mentioned as an actor in the use cases.
- The ***Domain Model and Domain Logic*** have been separated within the Domain Layer for the purpose of keeping the project in order and separating concerns for the reason that the EMF Model Generating mechanism would override the implemented method body for the logic, each time, a model change is executed. Therefore, the model classes contain the instance variables, constructor, and getters/setters. The logic functionalities are the "management classes" in the "Domain Layer – Logic".


Information on the links between the packages:
To keep the number of association lines reasonable in the class diagram, the links between the packages will be explained here in a textual form.
The corresponding model classes from the package "Domain Layer – Model" are used by the classes in the further packages.
The classes in the package "Presentation & Application Layer" interact with the classes in the package "Domain Layer – Logic"Because the EMF Model generation would require manual adding of the Domain Layer - Logic functionality, Model and Logic have been placed in separate packages.
The classes in the package "Domain Layer – Logic" interact with the respective classes and interfaces in the package "Persistence Layer".
Notes:
Constructors are not explicitly shown for reasons of simplicity
Getter and setter methods are not explicitly shown for reasons of simplicity
orderPrescription(Prescription p): modifies boolean "ordered", i.e. sets TRUE if prescription medication was ordered, otherwise boolean "ordered" remains FALSE
payInvoice(Invoice i): modifies boolean "paid", i.e. sets TRUE if Invoice was paid, otherwise boolean "paid" remains FALSE
NHO ... National Health Organization
Invoices, for which the boolean variable "paid" is set as TRUE, are considered as transactions
Presentation & Application Layer
GUI Windows (using Java WindowBuilder); Controller Classes
Login
+loginUser()
Register
NHOMain
+sendNHOContributionInvoiceToAllPatients()
DoctorMain
PatientMain
PharmacyMain
TransactionDetails
+exportTransaction()
AnonymizedDataOptions
+getMedicalDataByGender()
+getMedicalDataByYearOfBirth()
GatherAnonymizedData
+gatherAnonymizedData()
+exportAnonymizedData()
RegisterNewPharmacy
+createPharmacy()
RegisterNewDoctor
+createDoctor()
ViewMedicalRecords
+getAllMedicalrecords()
+detailedView()
+exportAllMedicalRecords()
ViewMedicalRecordDetails
+addMedicalRecord()
+exportMedicalRecord()
CreateMedicalRecord
+submitMedicalRecord()
+addDevice()
ViewPatientMonitorings
+getAllPatientMonitorings()
CreateMedicalRecordFromViewPatientMonitoring
ViewPersonalMedicalRecords
+getMedicalRecordByPatientId()
+exportAll()
+orderPrescription()
ViewPersonalInvoices
+getInvoiceByPatientId()
+exportAll()
+exportInvoice()
+payInvoice()
CreatePatientMonitoring
+createPatientMonitoring()
TransactionAccount
+getAllInvoicesByCreditorId()
+exportAllTransactions()
Domain Layer - Model
User
-id : int
-username : String
-password : String
-city : String
Patient
-firstname : String
-lastname : String
-svnr : String
-weight : double
-height : double
-gender : Gender
-birthdate : Calendar
Doctor
-firstname : String
-lastname : String
-clinic : String
NationalHealth Organization
-contact : String
Pharmacy
-name : String
Insurance
-department : String
<<enumeration>>
Gender
Female
Male
<<enumeration>>
DeviceType
Sensor
FixedPlacement
Wearable
Device
-id : int
-type : DeviceType
-patientId : int
-medicalRecordId : int
AnonymizedDataQuery
-gender : Gender
-medicalCondition : MedicalCondition
-height : double
-weight : double
-city : String
-birthYear : int
MedicalRecord
-recordId : int
-patientId : int
-doctorId : int
-medicalCondition : MedicalCondition
-price : double
-date : Calendar
-description : String
Prescription
-prescriptionId : int
-recordId : int
-medicationName : String
-start : Calendar
-end : Calendar
-ordered : boolean
-price : double
PatientMonitoring
-monitoringId : int
-patientId : int
-category : MedicalCondition
-date : Calendar
-riskPerception : RiskPerception
-treated : boolean
-deviceId : int
Invoice
-invoiceId : int
-patientId : int
-invoiceDate : Calendar
-dueDate : Calendar
-price : double
-creditorId : int
-paid : boolean
-paidDate : Calendar
-topic : String
<<enumeration>>
RiskPerception
High
Medium
Low
<<enumeration>>
MedicalCondition
Bronchitis
Asthma
Diabetes
Flu
Hypertension
Domain Layer - Logic
UserMgmt
+checkUser(String username, String password) : boolean
+addDoctor(String username, String pw, String city, String firstname, String lastname, String clinic) : boolean
+addPharmacy(String username, String pw, String city, String pharmacyname) : boolean
+addPatient(String username, String pw, String city, String firstname, String lastname, String svnr, double weight, double height, Gender gender, Calendar birthdate) : boolean
+addInsurance(String username, String pw, String city, String department) : boolean
+addNHO(String username, String pw, String address, String contact) : boolean
+getUserList() : List
+getUserByUsername(String username) : User
+getUserById(int id) : User
+getPatientList() : List
DeviceMgmt
+addDevice(DeviceType type, int patientId, int medicalRecordId) : boolean
+getLastDeviceID() : int
InsuranceMgmt
+addInsurance(String username, String password, String city, String department) : boolean
+claimInsuranceContributionFromPatients() : void
MedicalRecordMgmt
+getAllMedicalRecords() : List
+modifyMedicalRecord(MedicalRecord m) : void
+getMedicalRecordByPatientId(int id) : List
+addMedicalRecord(int recordId, int patientId, int doctorId, MedicalCondition mc, double price, Calendar date, String description) : void
+addPrescriptionToMedicalRecord(int medicalRecordID, int id, String medicationName, Calendar start, Calendar end, boolean ordered) : void
+orderPrescription(MedicalRecord mr) : void
+getMedicalRecordByRecordId(int id) : MedicalRecord
+getAnonymizedMedicalDataByYearOfBirth() : List
+getAnonymizedMedicalDataByGender() : List
+getAllPrescription() : List
+generateNewMedicalRecordId() : int
+generateNewPrescriptionId() : int
PatientMonitoringMgmt
+getAllPatientMonitoring() : List
+addPatientMonitoring(int patientId, MedicalCondition category, Calendar date, RiskPerception rp, deviceId did) : void
+markPatientMonitoringAsTreated(PatientMonitoring pm) : void
ExportMgmt
+exportAllMedicalRecords() : void
+exportMedicalRecordByRecordId(int id) : void
+exportMedicalRecordByPatientId(int patientId) : void
+exportAllInvoices() : void
+exportInvoiceByInvoiceId(int invoiceId) : void
+exportInvoiceByPatientId(int patientId) : void
+exportInvoiceByCreditorId(int creditorId) : void
+exportMedicalRecordByDoctorId(int doctorId) : void
+exportAnonymizedDataQueryByYearOfBirth() : void
+exportAnonymizedDataQueryByGender() : void
NotificationMgmt
+getAllNotifications() : String
+getAllNotificationsDoctor() : String()
InvoiceMgmt
+getAllInvoices() : List
+getInvoiceByPatientId(int id) : List
+getInvoiceByCreditorId(int id) : List
+getInvoiceByInvoiceId(int id) : Invoice
+addInvoice(int invoiceId, int patientId, Calendar invoiceDate, Calendar dueDate, double price, int creditorId, boolean paid, Calendar paidDate) : void
+payInvoice(Invoice i) : void
+addInsuranceContributionInvoiceForAllPatients()
+addInsuranceInvoice(int patientId) : void
+generateNewInvoiceID() : int
Data Access & Persistence Layer
DAO interfaces & classes (using e.g. serialization or database)
NOTE: shown here for the case of serialization
<<Interface>>
UserDAO
+getUserList() : List
+deleteUser(User user) : void
+addUser(User user) : void
+getUserById(int id) : User
+getUserByUsername(String username) : User
implements
SerializedUserDAO
-userlist : List
-filepath : String
+getUserList() : List
+deleteUser(User user) : void
+addUser(User user) : void
+getUserById(int id) : User
+getUserByUsername(String username) : User
<<Interface>>
PatientMonitoringDAO
+getPatientMonitoringList() : List
+deletePatientMonitoring(PatientMonitoring p) : void
+addPatientMonitoring(PatientMonitoring p) : void
+updatePatientMonitoring(PatientMonitoring p) : void
implements
SerializedPatientMonitoringDAO
-monitoringlist : List
-filepath : String
+getPatientMonitoringList() : List
+deletePatientMonitoring(PatientMonitoring p) : void
+addPatientMonitoring(PatientMonitoring p) : void
+updatePatientMonitoring(PatientMonitoring p) : void
<<Interface>>
InvoiceDAO
+getInvoiceList() : List
+deleteInvoice(Invoice i) : void
+addInvoice(Invoice i) : void
+updateInvoice(Invoice i) : void
implements
SerializedInvoiceDAO
-invoicelist : List
-filepath : String
+getInvoiceList() : List
+deleteInvoice(Invoice i) : void
+addInvoice(Invoice i) : void
+updateInvoice(Invoice i) : void
<<Interface>>
MedicalRecordDAO
+getMedicalRecordList() : List
+deleteMedicalRecord(int id) : void
+addMedicalRecord(MedicalRecord m) : void
+updateMedicalRecord(MedicalRecord m) : void
+getPrescription(int medicalRecordId) : Prescription
+getPrescriptionList() : List
+deletePrescription(int id) : void
+addPrescription(Prescription p) : void
+updatePrescription(Prescription p) : void
implements
SerializedMedicalRecordDAO
-medicalrecordlist : List
-filepath : String
+getMedicalRecordList() : List
+deleteMedicalRecord(int id) : void
+addMedicalRecord(MedicalRecord m) : void
+updateMedicalRecord(MedicalRecord m) : void
+getPrescription(int medicalRecordId) : Prescription
+getPrescriptionList() : List
+deletePrescription(Prescription p) : void
+updatePrescription(Prescription p) : void

Ecore/EMF Diagram:



Explanatory Comments on Class & Package Diagram:

Design pattern: Layered Architecture Pattern.

***Layers*:** Persistence Layer, Model, Logic, Presentation and Application Layer.

Persistence Layer:

Persistence Layer is responsible for storing and retrieving of data e.g. from a file (which is used in line with serialization) and is used by Domain layer. The DAO interfaces can, however, be implemented by further persistence mechanisms such as DatabaseDAO classes, which enable storing and retrieving the concerned data in databases.

The Persistence Layer consists of interfaces UserDAO, PatientMonitoringDAO, InvoiceDAO, MedicalRecordDAO and classes, which implement them. It is possible to add, retrieve or delete any data. It is also possible to control prescription items through the MedicaRecordDAO interface.

Domain Model:

The Domain Model is responsible for managing data and modeling the relevant parts of the domain to form a viable representation of the domain.

It consists of User, extended by several User Types: Patient, Doctor, Pharmacy, Insurance and NHO (National Health Organization). Each User has id, username, password and city. There are also some specific characteristics for every user type, such as Sozialversicherungsnummer (SVNR; social insurance number) for patient, or clinic for doctor.

Medical Record Object has a unique identification number, but also patient and doctor ids, medical condition, price, description and date. Each medical record must include only one invoice. Invoice has also unique identification number, patient id, invoice date, due date, price, creditor id, date of payment and Boolean variable paid, which should define if patient has already paid. Medical Record can have one prescription. Prescription has also unique id, id of medical record it counted among, price, medical name, start and end date, Boolean variable ordered, which should detect if patient has already ordered the medication.

Patient monitoring object should help to detect the exact problem of the patient. Among characteristics we can find monitoring id, patient id, disease category of type MedicalCondition, risk perception of type RiskPerception, creation date, device id and Boolean variable treated, that indicated if patient has already get treatment from one of the

doctors. RiskPerception and MedicalCondition implement Enumeration and are user to obtain risk perception (high, medium, low) and medical condition(bronchitis, asthma etc.).

Device object is equipped with id, type of type DeviceType, patient id and medical record id. AnonymizedDataQuery object has variable gender of type Gender, medical condition, height, weight, city and birth date of patient defined. DeviceType and AnonymizedDataQuery implement Enumeration interface. They are used to enumerate gender and device type.

Domain Logic:

Domain Logic handles user requests with regard to the business logic of the domain and is responsible for the interaction between the model and user interface.

In our case it is be defined through the multiple management classes. User management is in charge of creation and check of users. Patient monitoring management is responsible for all the patient monitoring items. Invoice management has the whole invoice logic: payment, creation, search for the defined invoice, creation of insurance contribution for all patients. Medical record management enables controlling of medical record data and prescriptions that belong to medical records: creation, modification, search. Export management enables to export different data items: medical records, invoices. Device management is in charge of creation of devices and insurance management - of creation of insurance contribution and adding of new insurances.

Presentation & Application Layer:

Presentation and Application layer is in charge of the user interface. Using information provided, it produces the application interface needed and takes in the users' inputs.

We have defined the following first UI pages: login & register page. After the user was logged in, he is able to use the rest of the application functionalities, which depend on user type. We have provided pages, that enable the user to interact with application.

NHO: GatherAnonymizedData (gather & export data), RegisterNewPharmacy, RegisterNewDoctor, AnonymizedDataOptions, TransactionAccount (retrieving and export of transaction information), TransactionDetails (export of specific transaction).

Doctor: ViewMedicalRecords (retrieve all medical records, show details. Export), ViewMedicalRecordDetails (edit and export of medical record), CreateMedicalRecord, ViewPatientMonitorings (retrieving and creation of monitoring items), ViewPatientMonitorings (retrieving of all patient monitoring items), TransactionAccount

(retrieving and export of transaction information), TransactionDetails (export of specific transaction). From ViewPatientMonitorings page it is possible to get to CreateMedicalRecordFromViewPatientMonitoring to create a medical record.

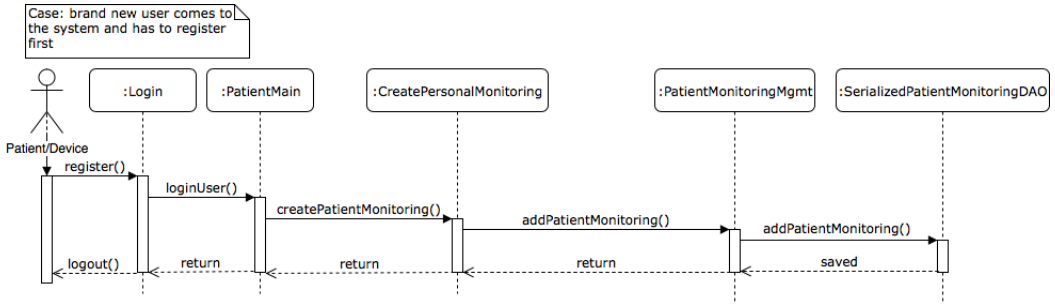
Patient: ViewPersonalMedicalRecords (retrieving of all the personal medical records, export, ordering of prescriptions), ViewPersonalInvoices (retrieving and export of all invoices, export of specific one, payment), CreatePatientMonitoring, TransactionAccount (retrieving and export of transaction information), TransactionDetails (export of specific transaction).

Pharmacy: TransactionAccount (retrieving and export of transaction information), TransactionDetails (export of specific transaction).

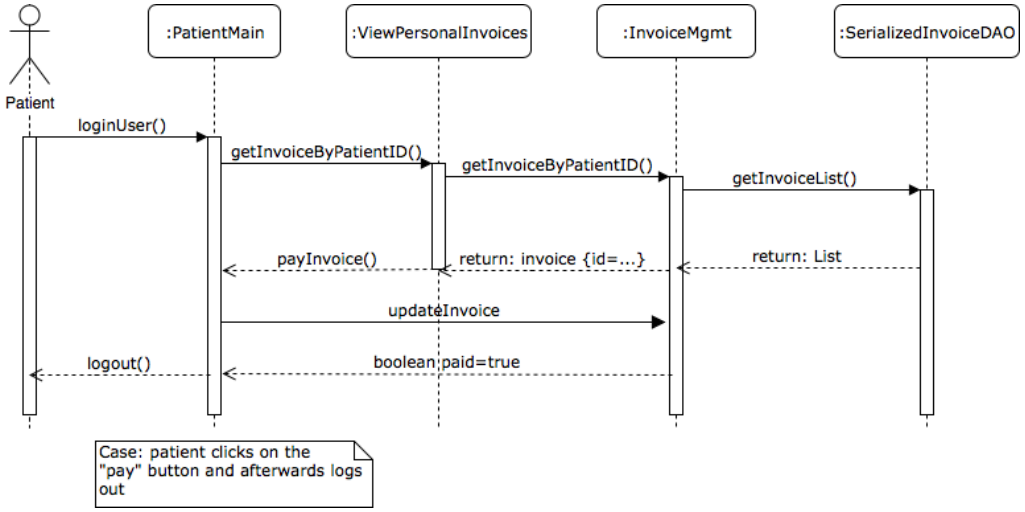
3.3. Process View

Sequence Diagrams:

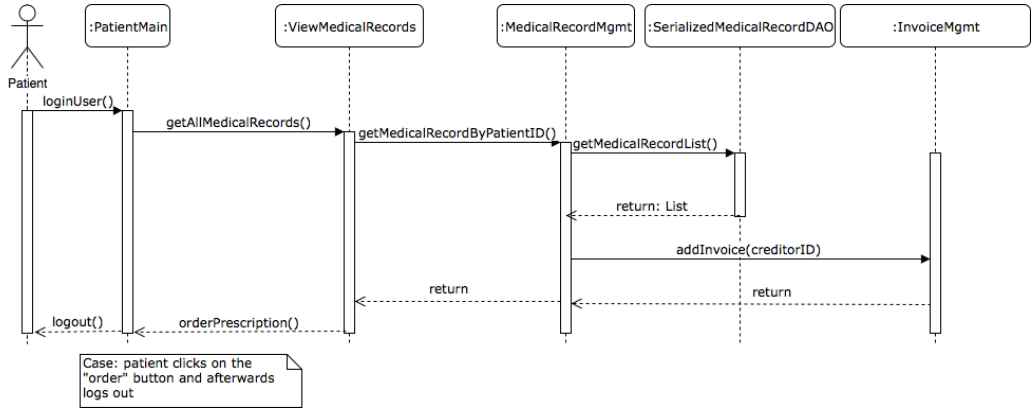
Use Case 01



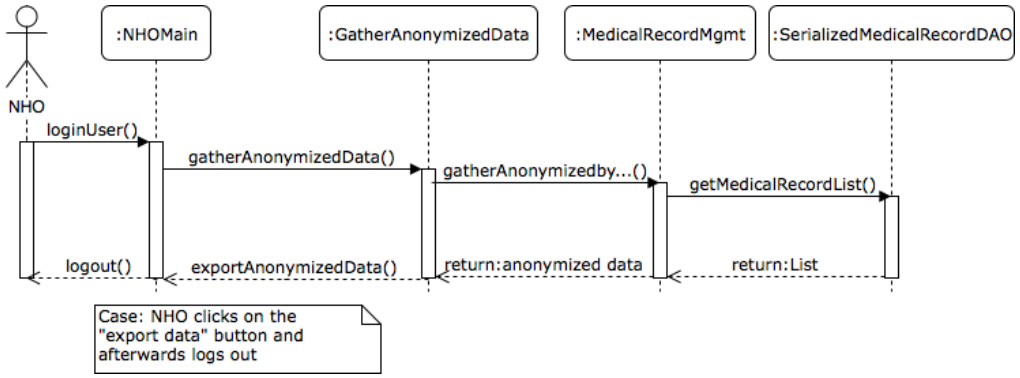
Use Case 02



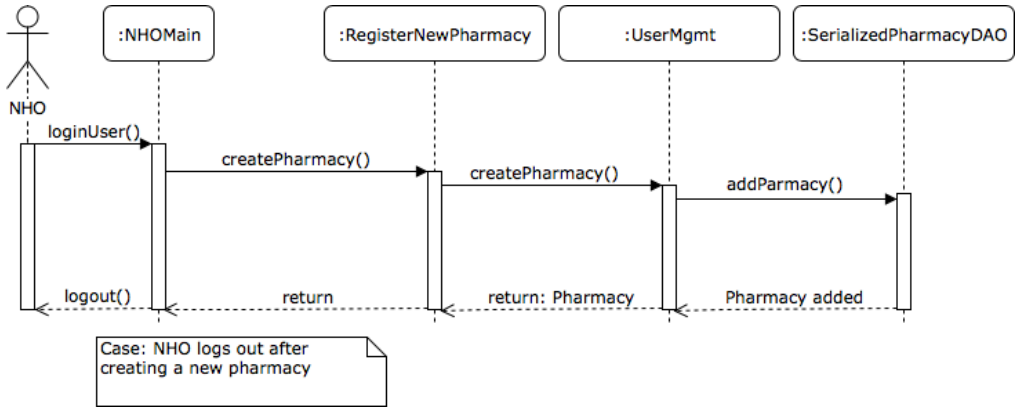
Use Case 03



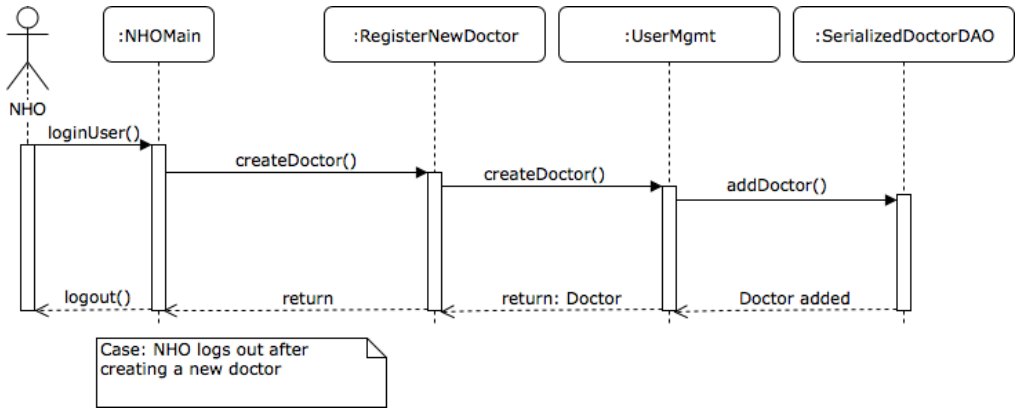
Use Case 04



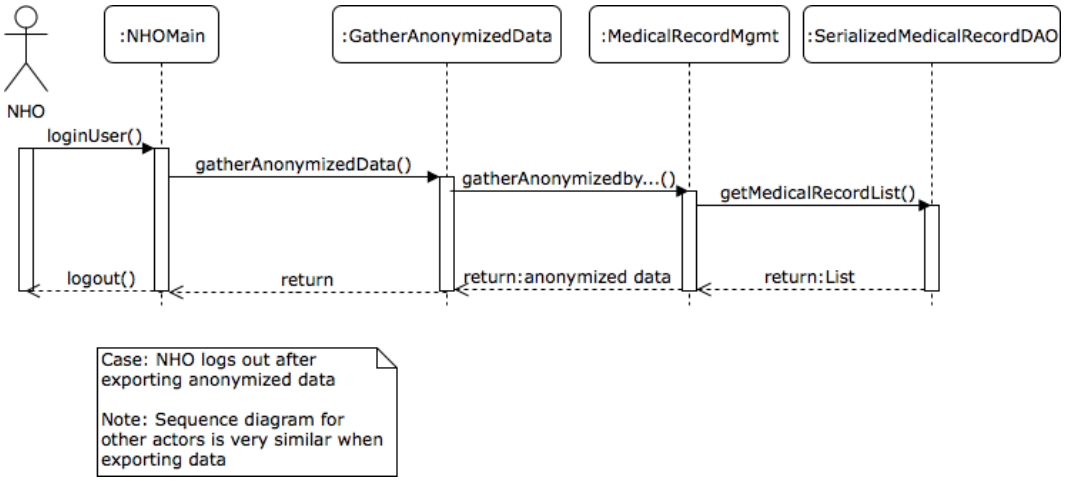
Use Case 05



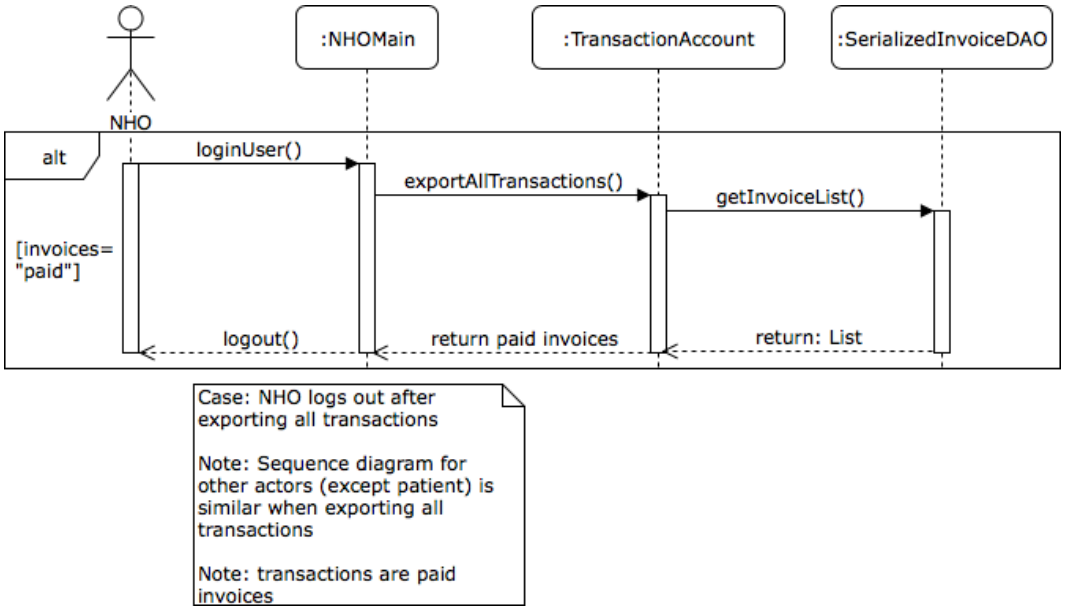
Use Case 06



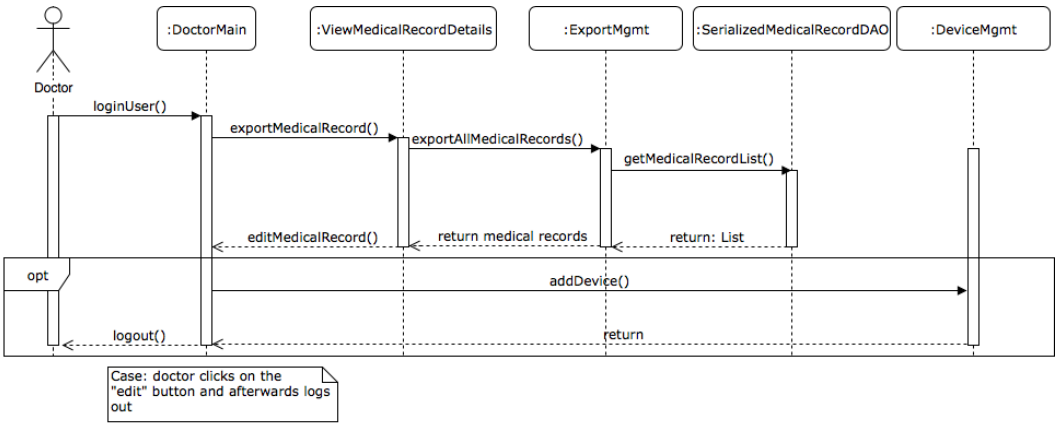
Use Case 07



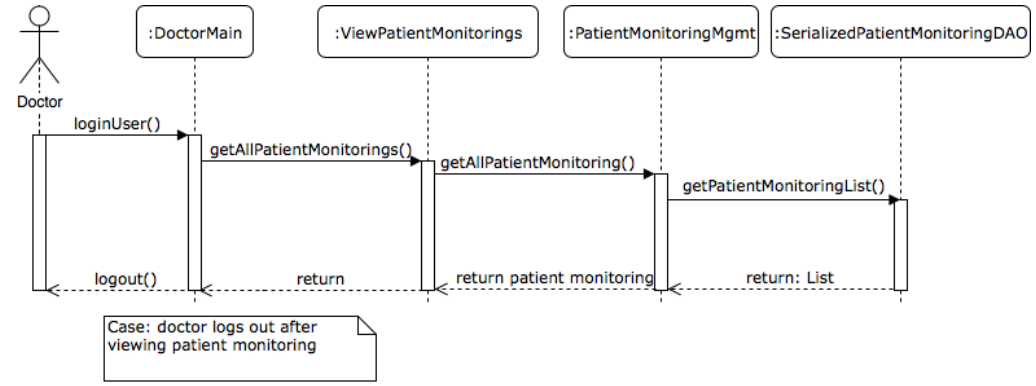
Use Case 08



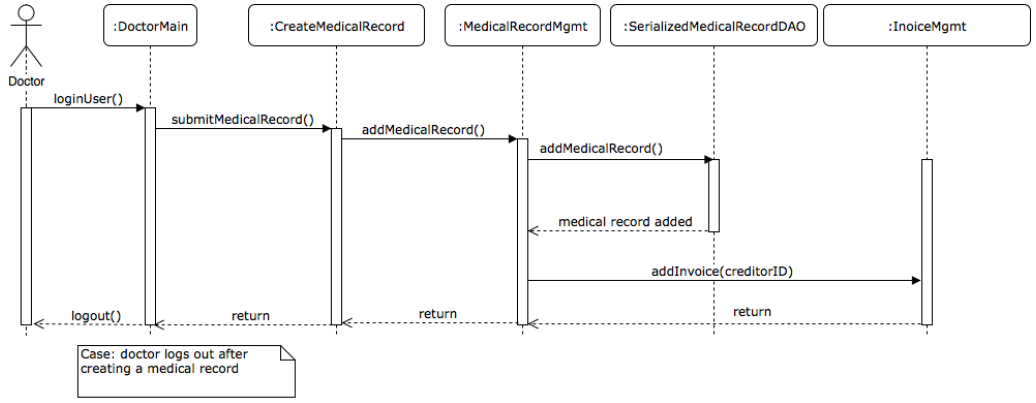
Use Case 09



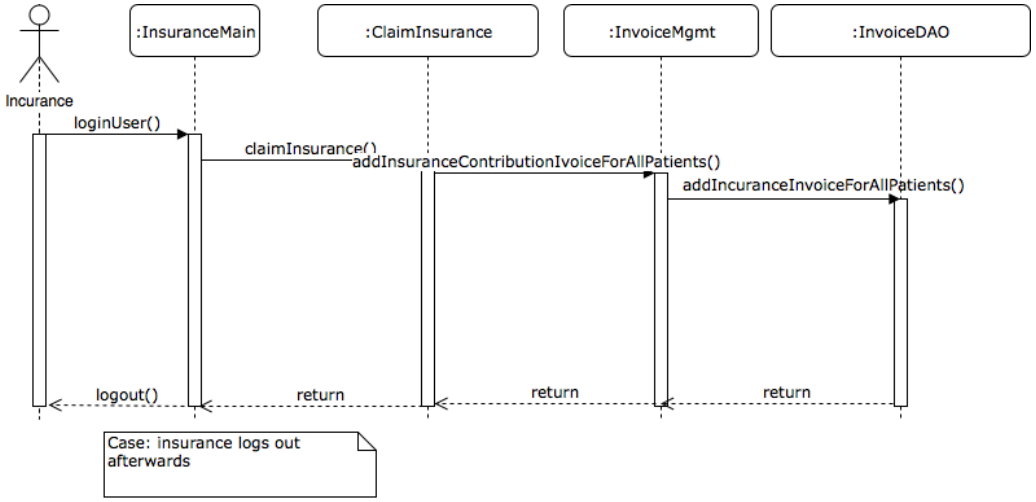
Use Case 10



Use Case 11

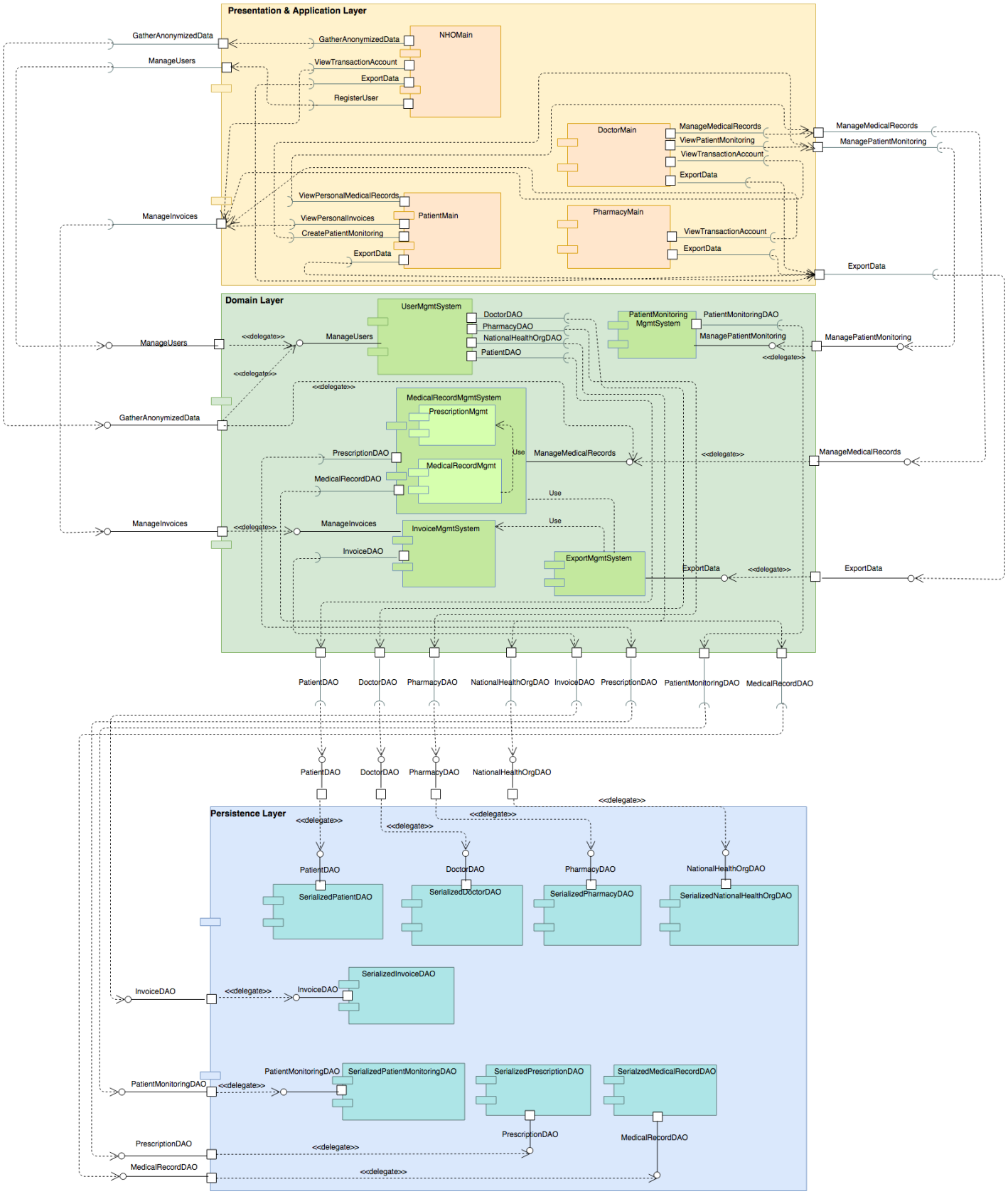


Use Case 12



3.4. Development View

Component Diagram:



Changes made with regard to SUPD feedback:

- Create/View/Manage summed up to a single interface for each instance type (MedicalRecord, PatientMonitoring, Invoice, User)

  Resulting interfaces: ManageUsers, ManagePatientMonitoring, ManageInvoices, ManageMedicalRecords

- Remark: Classes are not components, but a component may consist of a single class. No reasonable function blocks, composed from from more than one class, recognized **except for** MedicalRecordMgmtSystem (MedicalRecordMgmt + PrescriptionMgmt).

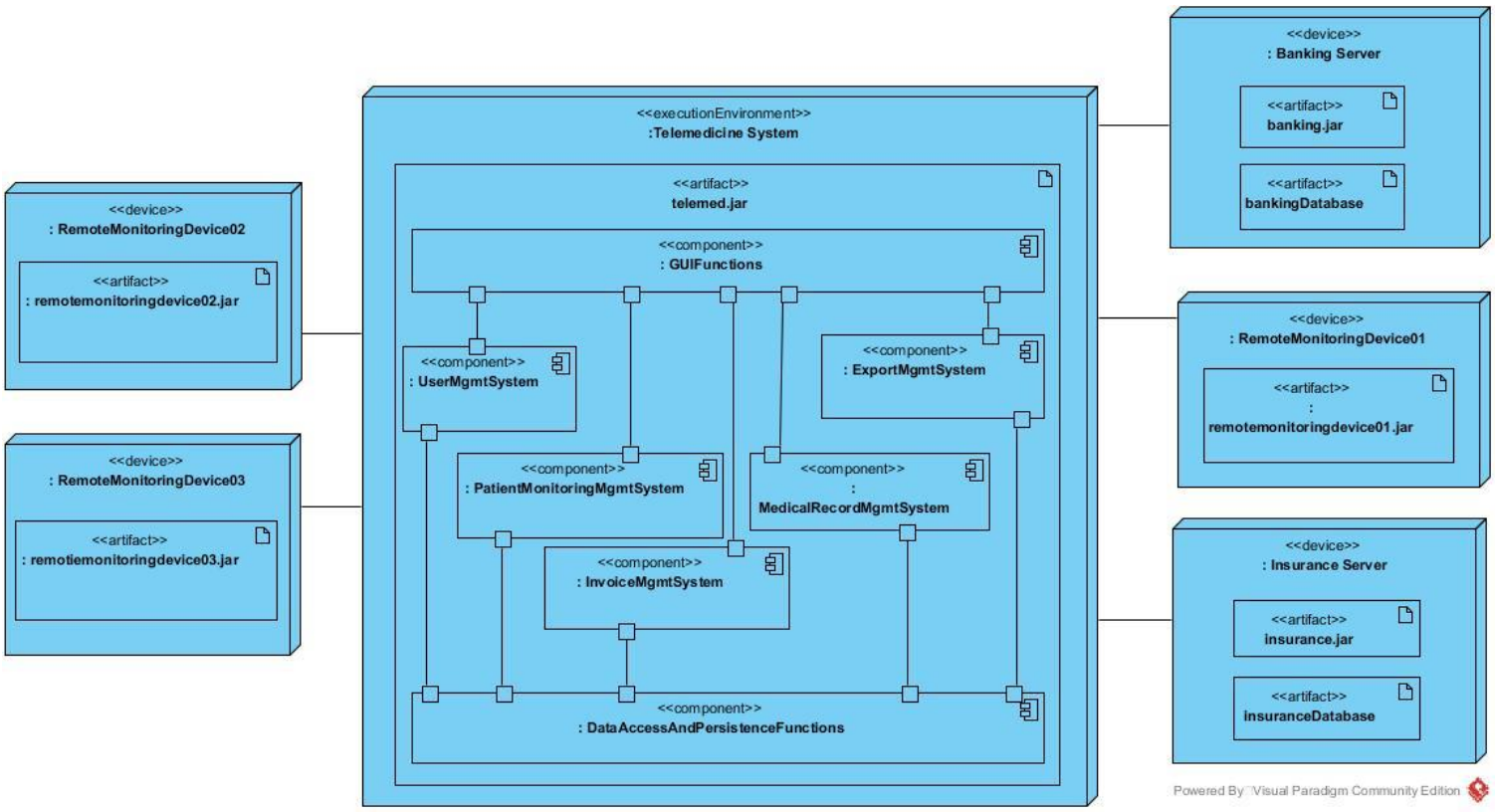
  Resulting components: UserMgmtSystem, ExportMgmtSystem, PatientMonitoringSystem, MedicalRecordMgmtSystem, InvoiceMgmtSystem.

3.5. Physical View

Deployment Diagram:

Within the Telemedicine System, the telemed.jar contains the implementation of the functionalities laid out in the use cases. The telemed.jar refers to the program, with which the user can interact via a GUI based on Java WindowBuilder/SWT/Swing Designer. To provide a more precise picture of the telemd.jar, the main components of the program as well as the links inbetween have been illustrated.

The system is also open to external devices, of which three exemplary Remote Monitoring Devices are modelled in the deployment diagram below. The remote devices interact with the Telemedicine System. In addition to this, for instance the Banking Server and the Insurance Server, which are also external, interact with the system and perform the corresponding banking and insurance operations.[2]



[2] ***NOTE:*** The external devices have been added as it was suggested in the SUPD feedback. As it was stated in the SUPD feedback, the external devices should be showed in this deployment diagram to create an environment of "distributed systems", although this project comprises the implementation of the Telemedicine System as such.