On defining a model driven architecture for an enterprise e-health system

Blagoj Atanasovski[a], Milošš Bogdanović[b],Goran Velinov[c], Leonid Stoimenov[b], Aleksandar S. Dimovski[e], Bojana Koteska[c], Dragan Janković[b], Irena Skrceska[d], Margita Kon-Popovska[c], Boro Jakimovski[c]

[a]Sorsix International, Skopje, Macedonia, blagoj.atanasovski@sorsix.com;
[b]University of Nis, Faculty of Electronic Engineering, Niš, Serbia,
{milos.bogdanovic,leonid.stoimenov,dragan.jankovic}@elfak.ni.ac.rs;
[c]Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, Skopje, Macedonia,
{goran.velinov,bojana.koteska,margita.kon-popovska,boro.jakimovski}@finki.ukim.mk;
[d]Faculty of Informatics, European University, Skopje, Macedonia,
irena.skrceska@eurm.edu.mk;
[e]Mother Teresa University, Skopje, Macedonia; aleksandar.dimovski@unt.edu.mk

ARTICLE HISTORY
Compiled November 4, 2021

ABSTRACT
The national e-health systems implemented in Serbia as "MojDoktor" (MyDoctor) and in Macedonia as "MojTermin" (MyAppointment) are based on the same integrated health information platform. It represents a central electronic system, where all medical health related information about patients, health workers, healthcare delivery organizations, documents, procedures is stored and processed on-line. Its architecture was designed to allow for process oriented development with agile methodologies. This methodology allowed for fast deployment and adoption, but a change in the architecture to a more formal approach is required to assure its extensibility, soundness, interoperability and standardization.

In this paper, we propose a formalization of the system design and its implementation as a Model Driven Architecture® [1]. We develop a set of formal models on several abstraction levels and explain how different layers of the MDA framework are covered using them. We also propose a model for data transformations that will provide interfaces for interoperability between an external and our system. We conclude the paper with a complete example that utilizes the formal models to define a RESTful service model that is interoperable with our system.

KEYWORDS
Model driven architecture, National e-health system, Integrated information systems, Formal models, and Verification

CONTACT Aleksandar S. Dimovski. Email: aleksandar.dimovski@unt.edu.mk

[1]The OMG group that launched it holds registered trademarks on the term Model Driven Architecture® and its acronym MDA®

1. Introduction

Electronic health record (EHR) represents a record of any health-related event (e.g. hospital admission, general practitioner visit, allergies) experienced by an individual over their lifespan from in utero to death (see Nguyen et al. (2014)). A national e-health system is a central electronic system, which enables collection and management of national EHRs data. It ensures interoperability of the existing autonomous and heterogeneous systems developed at different times, with different objectives, different data semantic and models, platforms and technology. The model of storage of EHRs data can be centralized (patient-centric) or decentralized (distributed or institution-centric), see Lapsia et al. (2012).

National e-health systems in Macedonia and Serbia were implemented in the period of 2011-2016 and 2015-2016, respectively. Based on the analysis of the health systems and the level of implementation of the local e-health applications in both countries, but also taking into account the risk factors and the experiences from previous implementations of e-health systems in the region, an adapted centralized model of data storage is implemented. In such a model some Healthcare Delivery Organizations (HDOs) have local copies of the data (they have e-health applications), while others do not have copies of the data and they are about to start using the new central system. In order to integrate the e-health applications that are used by HDOs with the national e-health system, the unambiguous semantics of data exchange between them is defined and appropriate Application Programming Interface (API) for integration and interoperability is developed.

The integration of these local e-health applications with the central e-health system enables integrated execution of business processes taking place partly in the e-health applications, and partly in the central system. In addition, the integration provides uni- or bi-directional data exchange: e-health applications provide the necessary data to the central e-health system and/or receive the necessary data for storage from the central e-health system. All systems used in the state institutions: Republic Health Insurance Fund (RHIF), Institutes for Public Health (IPH), Medicines and Medical Devices Agency (MMDA or Drug Agency), and so on, are also integrated, e.g. the data retrieved from the RHIF is insurance status of patients.

For the HDOs and the state institutions that do not have an e-health application, the national e-health system provides authorized access to some specific modules or functionalities through a web user interface (e.g. the users form IPH have access to all functionalities of the BI module). The user interface of the national e-health system is designed and developed according to the user-centred design principles for e-health systems described by Johnson et al. (2005); Chan et al. (2011).

From a data collection point of view, the system continuously creates new and updates existing records. In Serbia, the system creates records for over 240,000 prescriptions, 80,000 referrals and 170,000 examinations, each working day, while handling over 4,800,000 requests through the web APIs. In Macedonia, the system creates records for over 70,000 prescriptions and 30,000 referrals, each working day, while handling over 1,000,000 requests through the web APIs. Although the national e-health systems in both countries are being effectively used, still there are challenges to overcome in order the system to be more productively used and the positive effects of its usage to be increased.

Velinov et al. (2015) recognize that long term strategy goals of a national e-health system should be standardization on a national level, and this should be enhanced to a full interoperability on a European level. The current implementations of the systems

use a number of international codes and nomenclatures with unified significance, like ICD10 and ATC, but the semantic interoperability is defined by internal specification adopted on the national level. For international system interoperability, internationally recognized standards for medical data exchange, such as HL7, OpenEHR, etc. should be implemented.

From the initial data analysis many conclusions can be provided and numerous improvements of the health system in both countries can be planned. Unfortunately, there are cases where the quality of the entered data is not very high. Such examples are incomplete data, incorrect data, etc. Therefore, strict rules for data quality control should be introduced and appropriate changes in the system architecture design should be implemented.

We need to change the software system architecture to a more formal approach in order to modernize and transform the existing e-health systems to be interoperable and standardized, and we need to do such transformation in a secure and sound way. For this aim to be achieved, we have developed a formal model which formalizes all steps taken while implementing an enterprise e-health system, including the first one which is usually left out of the formal model scope - the formalization of work-flow diagrams that describe health-care processes. Our formal model is inspired by Riccobene and Scandurra (2009) and Tao et al. (2017). It incorporates benefits from both approaches and adds additional benefits in terms of simplicity, usability, system design and interoperability. We believe that the developed formal model is simpler and easier to implement, especially in cases when information systems are already in place. In contrast to the above cited works, our model does not focus exclusively to capture behaviour formalism at the PIM level, but follows the complete design or redesign process starting from work-flow diagrams introduced within Computation Independent Model (CIM) of Model Driven Architecture (MDA). The formal model we present in this paper includes data transformation meta-model which can be used for exchanging data between the system implementing our model and any other system implementing electronic health information exchange standard. Thus, we consider our proposal to significantly improve interoperability level of any e-health system that would use it.

2. Motivation and state-of-the-art

In order to design a viable Integrated Health Information System (IHIS), software engineers are facing a variety of problems originating from the high complexity of the health care processes they are attempting to conceptualize. Although a certain level of standardization has been achieved in the past decade, e.g. CEN (2008); HL7 (2011); ISO/TC215 (2009), there still seems to be missing an overall standard that makes it possible for electronic health care data to be easily interchanged between health enterprises. Without the ability to easily interchange data, interoperability remains an unsolved issue for majority of health information system and brings both research and development communities back to a set of questions regarding the architecture of these systems: What methodologies should be used to conceptualize the health care processes? Should health care information system always rely on platform-independent models? What architecture should be used to insure successful deployment of information system in heterogeneous and distributed environments? Which interoperability standards should be adopted to enhance health information exchange? What is the impact of all these issues upon software development process?

In the past decade, Model Driven Architecture (MDA) has been widely adopted as an framework for development of integrated health information systems. This architecture has been foreseen as a possible solution for a wide range of problems addressed by the designers of health information systems. MDA characteristics offer a possibility to develop health information system as a loosely coupled group of applications functioning in a heterogeneous and distributed environment. If MDA is adopted in the early design stage, it offers system architects an opportunity to design a truly lawful, portable and interoperable system by supporting the following architectural characteristics as recognized by Lopez and Blobel (2009):

- system flexibility, scalability and re-usability is ensured through component-orientation;
- architecture is model-driven and separated from the software development process;
- platform-independent and platform-specific modelling processes are separated (imposed through MDA characteristics);
- reference models and domain models can be specified at meta-level which in turn can be used as foundation for the development of shared terminology and/or ontology.

In the context of long-term usability and interoperability of an IHIS, there is a significant number of examples adopting MDA architecture. As stated in Blobel and Pharow (2006), Germany has started a national programme for establishing a health telematics platform which combines card enabled communication with network based interoperability. In order to achieve semantic interoperability, this platform successfully combines model driven architecture with variety of different generic components and development approaches such as ISO reference model for open distributed processing (RM-ODP) IOS (2009), a unified process based on the Rational Unified Process RUP (2017), the HL7 development framework HL7 (2006), service oriented architecture (SOA) (2007) , CORBA services CORBA (2012), advanced EHR architecture approaches CEN (2008), etc.

Another example of development framework for semantically interoperable IHIS has been provided by Lopez and Blobel (2009). The main objective of this research was to enhance existing IHIS architectures to enable semantic interoperability. The authors present an analysis of existing IHIS architectures and provide a harmonization framework based on the usage of The Generic Component Model (GCM). The methodology they propose takes advantage of Service-Oriented Architecture (SOA), MDA, ISO 10746 and HL7 Development Framework (HDF). Further, the modelling process is based on the Rational Unified Process (RUP) to ensure the flexibility of the development while harmonizing different architectural approaches. In Rayhupathi and Umar (2008), authors explore the potential of the model driven architecture in health care information systems development. MDA was used as a mean to develop a system capable of tracking patient information. They present the underlying MDA structure in the form of UML diagrams accompanied by PIM to PSM transformation rules. These rules have been used to generate the prototype application from the model they also present. This development provided additional insights regarding the development of transformation rules. As output, this research provided design guidelines for using MDA in IHIS development and confirmed MDA is usable for generating general models and has the potential to overcome lack of open standards, interoperability, portability, scalability, and the high cost of implementation issues.

Another possible application of MDA is within the information systems focused on achieving semantic interoperability of electronic health records. As an example, the approach described in Garde et al. (2009) focuses on methods which will ensure clinical content is internationally accessible as well as technically and clinically correct. Although this approach can represent the foundation for safely sharing the information clinicians need, decision support and legacy data migration, in order to make a system semantically interoperable all data should conform to shared models, terminologies and/or ontologies which can be introduced through means of MDA. In the same context, Schlieter and et al. (2015) attempt to create a specific MDA for the health-care domain. They needed a mechanism that fosters a sustainable tele-health platform in terms of a methodology that provides an efficient way to deploy new artefacts on the platform and ensure these artefacts are compliant to the platform properties. They show how the MDA approach can be used in order to build a methodological fundament for a systematic creation of an application system. Their use-case is an application for IT based work-flow support for an interdisciplinary stroke care project.

The work by Curcin et al. (2014) uses MDA to develop tools for data collection which allow non-informatics experts to model requirements in their local domain. They recognized a problem in the process of implementing local improvement initiatives in healthcare systems, so they developed a Model-Driven framework and implementation that allows local teams in medical organizations to specify the metrics to track their performance during an intervention, together with data points to calculate these metrics. Based on the metric specification the software generates appropriate data collection pages. In Jones et al. (2005) MDA augmented with formal validation was used to develop m-health components promising portability, interoperability, platform independence and domain specificity. The authors are developing systems based on inter-communicating devices worn on the body (Body Area Networks) that provide an integrated set of personalised health-related services to the user. This mobile healthcare application feeds captured data into a healthcare provider's enterprise information system. In their previous work the same authors propose an extension of the model-driven approach where formal methods are used to support the process of modelling and model transformation. The semi-formal modelling with UML is verified by using tools such as SPIN by Holzmann (2003) for model checking and TORX by Tretmans and Belinfante (1999) for model testing. They refer to Bolognesi et al. (1995) , Jones (1995) and Jones (1997) as possible formal approaches to the transformation.

In this paper we present a set of models that formalize the implementation of e-health system using MDA as a framework. Once the formal approach becomes fully implemented, we believe that both developers and system users will have the following benefits:

(1) The complexity of system will be abstracted through means of MDA models. This way, decisions regarding operational design will be delegated to lower modelling levels. The formal model we introduce in this paper follows the complete system design process - from conceptual design to platform specific implementation. This way, decisions regarding operational design will be delegated to lower modelling levels while retaining the ability to formally validate any of the previous design steps.

(2) The system will become capable of exchanging information with other systems implementing different e-health standards. It will be capable of supporting different e-health standards by aligning (mapping) its Platform Independent Models (PIM) with the structure of the components defined by different standards. This

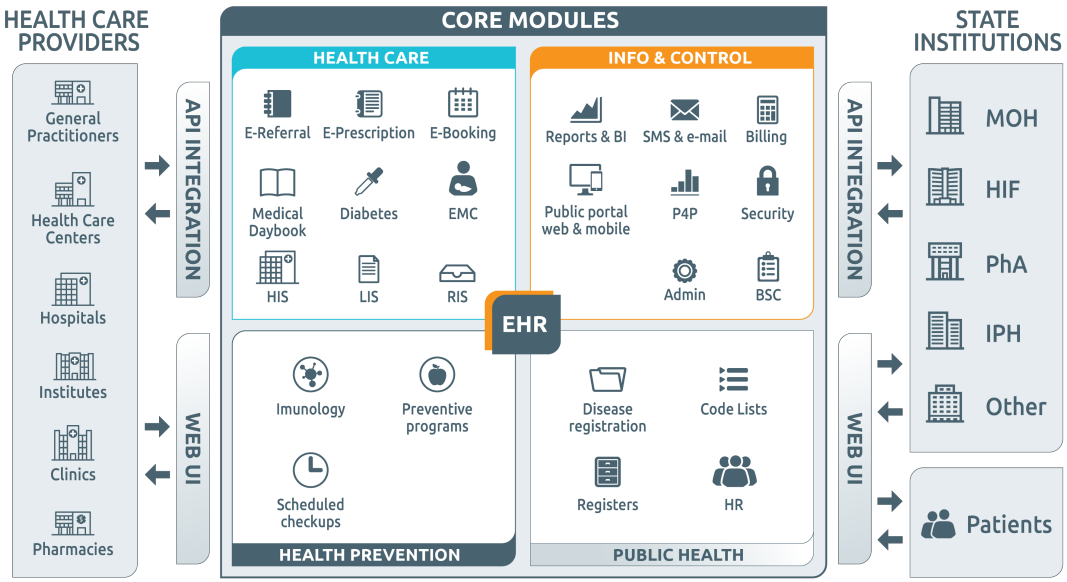

Figure 1. Architecture of the national e-health system

ability is introduced through data transformation meta-model defined within our formal model. As an example, we present data transformation model for HL7 standard messages.

(3) Development costs regarding the future improvements of the system are expected to be decreased due to re-usability of the developed models and alignments (mappings). And the verification capabilities of the models will prevent errors from reaching production.

3. The e-health system overview

3.1. *Functional ecosystem*

The implemented national e-health systems in both countries build functional ecosystems involving all HDOs and state institutions of public healthcare - as providers and users of information in this ecosystem. The systems play an important role in improving the operational management and the strategic planning of the reforms in public health in both countries. The top level architecture of the system is shown in Figure 1. The system consists of functional modules, grouped into four packages as follows:

- Healthcare: e-Referral, e-Prescription, e-Booking, Medical Daybook, Diabetes, E-Mother Card, Hospital Information System (HIS), Laboratory Information System (LIS), Radiology Information System (RIS);
- Information, monitoring and control: Reports and BI, SMS and E-Mail, Billing, Web & Mobile Public Portal, Payment for Performance (P4P), Security, Admin, Balanced ScoreCard (BSC);
- Health prevention: Immunology, Prevention Programs, Scheduled Checkups;
- Public health: Disease Registration, Code Lists, Registers, Human Resources (HR);

The central component of the system is the EHR module. All modules provide web

user interface. The following modules enable integration with other systems via API: E-referral, E-prescription, E-booking, Medical Day-book, HIS, LIS, RIS, while others are used only through a graphical user interface.

The system architecture provides several advantages:

- Existing e-health applications continue to be used in the HDOs and state institutions, where they gain importance because the national e-health system takes data from them. The need for their expansion, upgrading and improvement is created. So local e-health applications are not merely providers of information - with the integration they also receive data from the national e-health system.
- A model of integration is chosen which provides on-line operation of the system - all data are actualized at the moment when they are entered. Accurately defined API for integration allows quick technical integration, minimal technical errors, there is no wait for data synchronization, etc.
- For the users of existing e-health applications there is no change of the way they work or of the application software, thus avoiding resistance to change or the need for mastering new systems.
- For the new users that did not have e-health applications, intuitive user interface with simple and clearly defined operating procedures is developed.
- Due to privacy and confidentiality, a special focus is put on data security.
- A system that is most suitable for users - agile (incremental and iterative) approach to the definition, development and implementation of new functionalities is chosen. In the cases of functionalities that do not meet the real needs, after feedback from the users, a fast repairs and improvements are done.

3.2. *Interoperability and EHR components*

Collecting of EHRs data on the national level, as a central module of the system, has been enabled through the above mentioned modules. A context level Data Flow Diagram (DFD) of the system is shown in Figure 2.

A model of interoperability has been selected by which the local e-health applications are integrated with the central national e-health system. All the existing different e-health applications (74 in Macedonia and 57 in Serbia) were integrated into a national e-health system. Accurately defined API for integration brings the following advantages: there is no wait for data synchronization, there is quick technical integration, technical errors are minimal, etc. The integration is based on interfaces that allow data exchange according to an internal XML based standard. Within the integration, accepted international codification systems are used, such as ICD-10 for diagnosis coding, ATC for medications coding and AR-DRG classification system for health services.

The EHRs combine time-oriented, problem-oriented and source-oriented components (by Häyrinen et al. (2008)): Patient personal data; Health risk factors; Allergies/ intolerances; History of diseases registered in the national registers (cancer, diabetes, cardiovascular diseases, etc.); History of: diagnostic treatments, specialist treatments, inpatient treatment with surgical interventions, medical examinations, radiological treatments, laboratory findings; used medications (chronic and acute therapies, narcotics and medications approved by a consilient opinion); Dental card. This means that the EHR module on the national level contains all data components that are relevant to the long-term health status of the patient, MITRE-corporation (2006). The above mentioned segmentation by components of the EHRs is made according to the

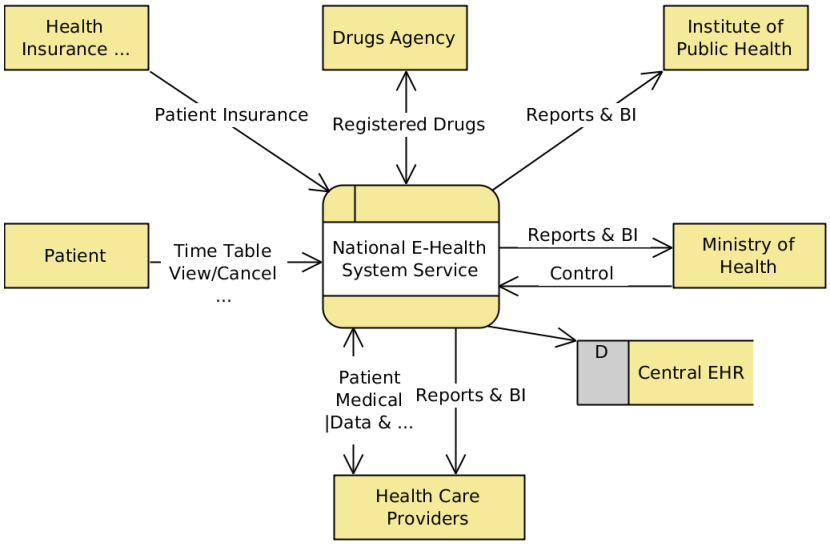

Figure 2. The system context level DFD

requirements and the needs of the Ministry of Health (MoH) of Macedonia, but it is adaptable to any segmentation and display, US-HHS (2010). The manner of defining and monitoring the access to data is provided below.

Due to the sensitivity of the data that are accessed through EHR module, US-HHS (2010); Ricciardi (2010); US-HHS (2015); Xu et al. (2017), there are strict rules of defining this access. The e-health system has the functionality for defining the rules of access to the EHR. This functionality is intended for administrators from the MoH. A very fine granularity for dividing the content of the EHR is provided. The content can be divided into different dimensions (criteria), to the level of each combination of values. The dimensions by which the rules of access could be defined are: EHR components (described above), Specialties, Diagnoses, Episodes of care, Prescribed/used medications, HDOs level of healthcare, etc. There is a flexible tool for creating and saving the filters in the form of predicate expressions. It allows easy selection of different criteria for filtering and building complex expressions by including or excluding one value or a set of values from any dimension (IN or NOT IN), easy embedding or combining (AND or OR).

3.3. *Security*

The system covers the following security aspects: role-based access, network security mechanisms, data encryption, digital signature and access monitoring. It uses a security access for allocation of roles and privileges that are defined in detail at the level of each action. The authentication mechanism allows each user to access the system within the privileges that are permitted to him. Each user can access it by logging the web application or through a web service from an e-health application used in the HDO.

According to the needs for secure access to sensitive data, two instances are used for web-based access to the system: external, open – designed for public access, accessible via HTTP protocol and internal, private – designed for users that log in to the system through the HTTPS protocol for which a security certificate is provided. The internal

communication within the framework of the system is carried out through an encrypted HTTPS protocol that provides security and crypto protection against the interception of data.

In the cases of access through a web service, there is a dual security mechanism. The authentication is carried out on the level of the provider (software vendor) that has a registered e-health application for use in HDOs integrated with the national e-health system through an application token, as well as at the level of the user (doctor, pharmacist, etc.) who uses web services of the national e-health system through a session token.

The system allows a two-phase authentication - using a user name and a password (the passwords are stored in an encrypted form) for login and verification of the login by sending a message to the logged-in user on his or her official e-mail account or a PIN code on his or her official mobile phone number. The physical security of the data and a system for monitoring of the hardware infrastructure, mechanisms for protection against network attacks have been provided. Each activity of the users is recorded in special log files, that is, records are stored for each change or reading of the data from the system.

The system records (logs) all attempts of access to the EHR of the patients. Thus, the access procedure is as follows: the user of the system (doctor) requests access to the EHR of a particular patient, to which he receives a message with a legal lesson on the consequences of accessing personal medical data and a request for confirmation that he or she consciously requires access. Upon confirmation of the request for access to the EHR, the user receives a message on his or her official e-mail with a link that opens the EHR for the given patient. The content displayed depends on the defined privileges for access to certain parts of the EHR, according to the rules of access. Access logs are stored and consist of data by which user has accessed the EHR, which parts of it he could see, from which HDO and the exact time and date.

3.4. *The development methodology*

An adapted agile methodology for the design and implementation of the information system was used. The system was developed and implemented in phases with gradual expansion according to the adapted methodology for iterative and gradual development by Larman and Basili (2003), in the following way:

(1) By consulting domain experts, work-flow diagrams were created that record the processes and rules in the health-care sector.
(2) With these work-flow diagrams and additional discussion, a relational model was made of:
   (a) all the entities required to describe the data;
   (b) all the relations and constraints between entities.
(3) A central database which is instatiated with the relational model developed in Step (2) was created.
(4) A service layer to govern the database operations was created, this layer implements the semantic and security constraints that can not be modelled with the relational model.
(5) A web application was created, which represents an interface between users and the service layer.
(6) A web API was created to allow third party applications to communicate with the service layer.

When all functionalities were not well defined, i.e. there was no precise and detailed description of the functional requirements, a sequential process of development was organized, with elements of Dynamic System Development Method (DSDM) as in Meso and Jain (2006), Voigt (2004). It should be emphasized here that a significant flexibility was expected from the implementer - to accept frequent requests that changed the functionality, after the stages of acceptance and deployment in production were finished, without considering them as change requests. Similar approach was taken for the development of functionalities that were not required in the original version of the official technical documentation. Since interesting cases appeared when the management of the two project teams assessed that some functionality is crucial for the successful use of the system, so that the given functionality was accepted and developed, regardless whether it was formally required in the functional requirements or not.

4. Transforming informal work-flow diagrams to MDA

As mentioned in the previous sections, the national e-health systems 'Moj Doktor' and 'Moj Termin' are in production. With this paper we aim to create a set of formal models that can be used to formally verify the already deployed code, and allow for stable future updates.

This section is organized as follows. A description of the process for creating electronic referrals is presented in Subsection 4.1. The described process will be used as a running example throughout the rest of the section. Using this example, in Subsection 4.2 we start the formalisation by describing the informal work-flow diagrams used with developing the current implementation of the system. In Subsection 4.3, a class diagram of the existing web services is presented as the first model. By abstracting the first model, a formal model for describing any service is defined in Subsection 4.4 and then in Subsection 4.5 we give a formal model for describing complete processes. The section concludes with part of the example process expressed with this model in Subsection 4.6, how models are organized in layers based on MDA framework in Subsection 4.7, and the definition of a valid process in Subsection 4.8.

4.1. *The process of creating electronic referrals*

The *Healthcare* package of modules (see Fig. 1) creates the possibility to define and approve appointment timeslots for doctors, the creation and update of electronic referrals and examinations, a way to make appointments for surgeries, and to follow the activities of doctors in clinics. The process of referral creation is executed over 30,000 and 100,000 times per day in Macedonia and Serbia, respectively. It represents a business process which is taken as a running example for the remaining of this paper.

In the *E-Referral* module, there are multiple types of referrals. Certain types of referrals do not require an appointment to be made, they are created referring to a clinic or laboratory, and the first available doctor receives the patient as soon as they arrive. Regular referrals require an available appointment timeslot, and each timeslot can be assigned to only one referral.

The process of creating an electronic referral depends on the type of referral, the patient's state, and the resources available. A resource can be a doctor or a machine. Referrals that represent a request for a laboratory to perform tests on a patient are a special category. Laboratories serve patients that have been referred on a first-come-

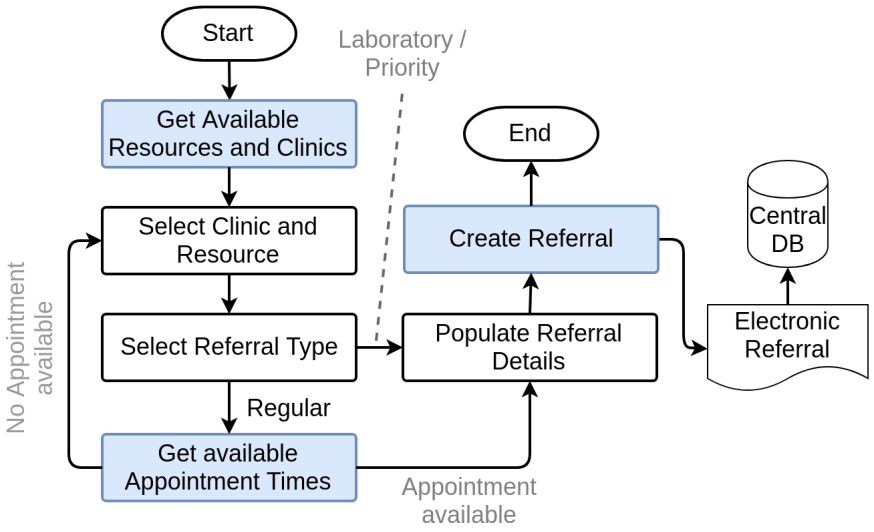

Figure 3. The work-flow of the referral creation process

first serve basis, there are no appointment times, and the required input is the patient, the laboratory and the details about what kind of tests are required. If the patient is referred to a resource then the patient's state determines whether the patient needs an emergency, priority or a regular referral. Emergency referrals are issued for patients in need of urgent care. Emergency referrals require only the patient and accepting clinic to be specified. Priority referrals are for patients with worsening states, they are referred to a clinic and need to be examined in the following few days. Regular referrals need an appointment time to be available, and for regular referrals a specific resource needs to be selected, not just a referred clinic.

A referral is always created by a doctor. A referral can be from a general practitioner to a specialist or a machine/equipment (e.g. MRI Scan), from one specalist to another, or a control referral from one specialist to themselves in the future.

4.2. *An informal work-flow diagram*

The first step taken in implementing the system was creating work-flow diagrams. They represent an informal representation of processes that the system automates. We consider them informal because complex workflows are hard to model, constraints are not presented and interpretation can vary between readers. Because of this, they are not ideal for documenting health-care processes, and significant amount of non-structured text was required to capture all the rules and constraints.

Figure 3 represents the process of referral creation in the "MojDoktor" web application. The process descibed can be started by any authenticated doctor. First the doctor will be presented with a list of available clinics and the resources in each of those clinics in the system. The doctor must choose the clinic and/or resource. Then the doctor selects the referral type. For laboratory referrals and priority referrals, the doctor directly fills in the referral details (patient info, referring diagnosis etc.) and creates the referral. For regular referrals, an available appointment time must be selected or the doctor must go back to the clinic selection. If an available appointment time was found, the referral can be created. The electronic referral is stored in the central database.

The work-flow diagram contains user choices (type of referral, clinic selection), calls to web services (available resources, available appointment times, create referral), and documents are created (and stored in the database). In the diagram in Figure 3 we use the shaded fields to emphasize which steps of the process are actions done by the information system, while the others are user input. After the workflow diagram was created, the implementation follows the remaining steps outlined in Section 3.4.

For each of the shaded steps an appropriate web service exists. The *Get Available Resources and Clinics* web service returns a collection of clinics, and for each clinic a list of available resources that can be referred to. The logic behind which resources and clinics are returned depends on the authenticated doctor and is described by a separate work-flow diagram. For example, only a gynocologyst can refer a patient to a mammogram. The *Get available Appointment Times* web service expects a resource id as input, and it returns a collection of available Appointment Times or an error if no Appointment Times are available. The *Create Referral* web service expects as input a referral type, clinic, resource, appointment time, and referral details. Based on the input, it creates a specific referral document, stores it in the central database, updates the selected timeslot as not available and concludes the process.

4.3. *Creating models for the implemented web services*

For all the steps in the work-flow diagrams that are actions done by the system, there is an implementation in code. The first step in the formalization of the system is to create UML class diagrams that document each of the web services. For the web services in Figure 3, the Class diagrams in Figure 4 are given as an example. The implementation follows the standard layered architecture for web applications. There are models (or entities) that represent instances of the data in the system. In this example the *Clinic*, *Resource*, *Appointment Time* and *Referral* entities are shown with a part of their properties. Each of the entities is accessed and saved through a specific repository for it. Repositories encapsulate all database related code. On top of the repository layer is the service layer. The business logic of each web service is recorded here. The *getAvailableResourcesAndClinics* method of the *ResourcesService* is the implementation of the step of the same name in Figure 3. In the *ReferralsService*, three methods are shown for creating referrals, and depending of the user choices in the process in Figure 3, the appropriate method will be called.

Class diagrams descibe the properties and methods a class should have, and the relationships between classes. But most of the business logic of the complete process is inside the methods of the classes of the Service layer and the logical ordering of the operations required to complete the process. The class diagrams of the implementation fail to capture this logic.

4.4. *A formal model for describing services*

To address the issue that simple class diagrams of the implemented services do not capture the logical ordering required to complete a business process, a formal model for the representation of a Service and a Process is needed. In this section we present a formal model for describing Services. The model utilizes a class diagram. The idea is that "instances" of this model would create the specific services (Create Referral, Get Available Appointment Times, etc).

Figure 5 shows the classes of the Service Model. A *Service* has a unique name, which

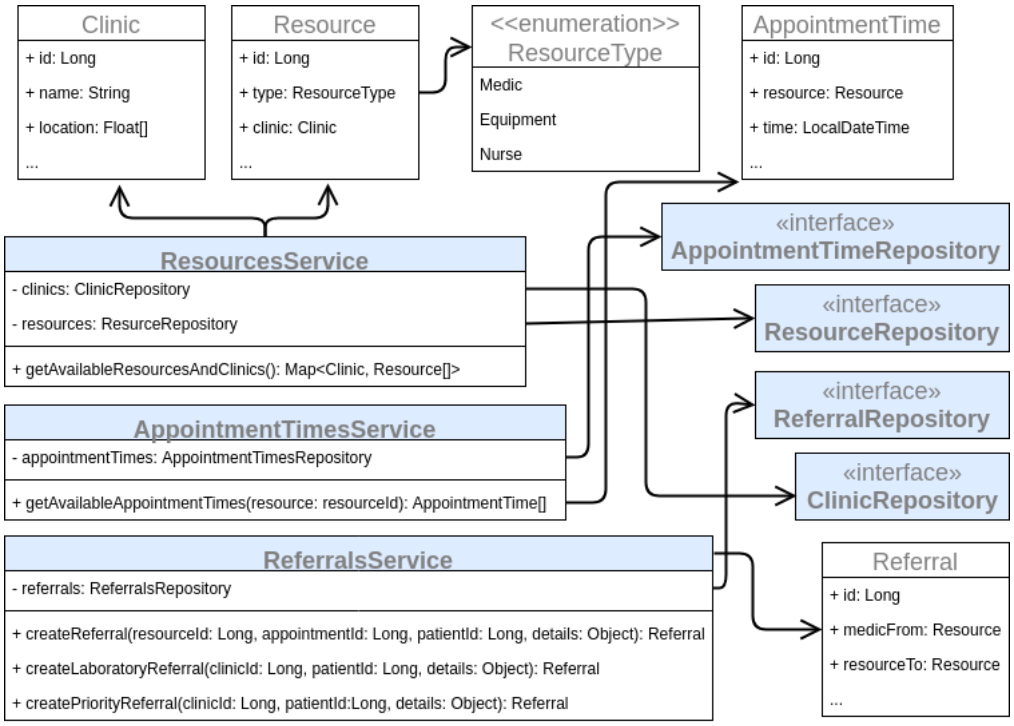

Figure 4. Class diagrams for the web services in the *Create referral* process

represents the method that contains the logic in the implementation. A *Service* also has input arguments, in the model they are given as a dictionary where the keys are the argument names of the method, and the values are the argument types. The output of the service is not named, only the type is given. When this model is translated to the PSM layer it becomes a class definition with a single method.

The *Type* class given in Figure 5 is just a description of a data structure with a set of uniquely named properties, and each property is of a certain *Type*. Some of the Types in the system are Entities that are connected to the tables in the database, others are projections of those Entities, and the rest are just used for state transfer between the Web Application and the APIs, between different services, etc. which we call Data Transfer Objects (DTOs).

The service model also requires that the *SideEffects* be modeled that would result from calling that service. Possible side effects are to create, update or delete one or multiple entities stored in the database. Each modified entity is referred by it's type, and a collection of *Constraints* that must be satisfied from the input or output arguments. A constraint has a type, which gives the type of comparison (Equal, Not Equal, Less than). The property field specifies which property of the modified entity the constraint referrs to. The valueRef property specifies which of the input or output arguments should be compared to the property of the modified entity.

4.5. *A formal model for describing processes*

In the previous section, the requirement for a formal model for describing Processes was given. (Riccobene and Scandurra 2009) propose a formalism based on Abstract State Machines (ASMs) to capture behaviour formalism at the PIM level, and (Tao et al.

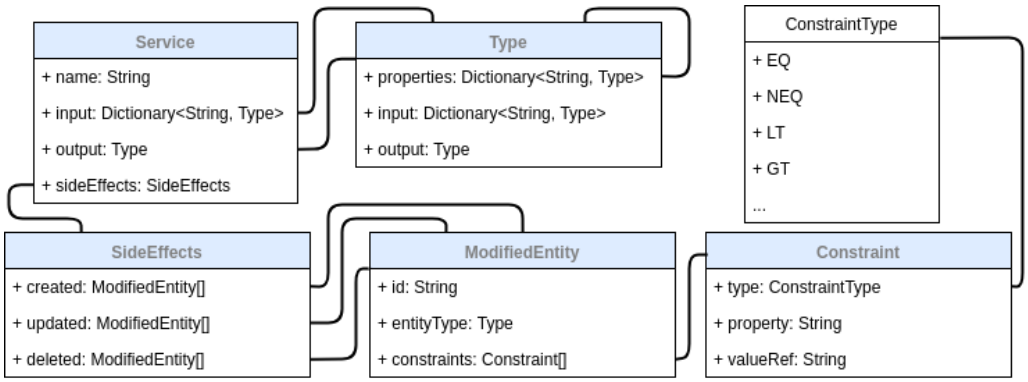

Figure 5. Class diagram representing an abstract model for a *Service*

2017) have proposed a framework for describing software architecture that is based on creating meta-models and constraints between the meta-models. We propose here a simpler model, which is inspired from ASMs and meta-models. With this model, we formalize the work-flow diagrams introduced in Section 4.2. The class diagram shown in Figure 6 is used to describe this model. A *Process* has a set of *Implementation* steps, such that each step is unique and identified with a name. The process knows the starting and terminating steps, identified by their names. The allowed *transitions* between steps are also part of the process.

A step is an instance of the *Implementation* class, which is an abstract class. It can be either a *Service*, *User Input*, or another *Process*. *Services* represent the Web Services in the Information System, *User Inputs* are descriptions of user choices on the front-end applications, and *Processes* can be composed with sub-processes. *Implementations* have names, inputs and outputs. For a *Service*, a name is the method's name. The input represents the input arguments of the method, given as a dictionary from the argument name to the type of each argument. The output represents the return type of the method.

As a process is executed, the output of all the steps is stored in the *Process State*. The process state keeps the output in a dictionary where the key is the step that produced that output. One method exists in this class that returns the Output object for a given step name. The *Process State* is not persisted between executions of the *Process*, data is lost once a process completes.

The transitions between the steps in a *Process* can be conditional. For each step, except the ending step, there should be an *OutgoingTransition[]* entry in the *transitions* dictionary in the *Process*. For each outgoing edge there should be an OutgoingTransition that contains a predicate. The predicate signifies (based on the process state) if a transition between two steps activates). The last OutgoingTransition should always be the default choice, and must always activate in order to guarantee that a process will complete. Since only one transition can be active at one time, we limit our models to not support concurrency.

An *OutputReference* describes a projection of the output from an already executed step in the process. It contains a stepId and a propertyName. The stepId identifies one of the steps, and represents an *Implementation*. The propertyName is one of the properties of the output of the referenced *Implementation*. The propertyName is nullable, and if it is set to null then the complete output Type is referenced (similar to * in SQL). Let us say that there are two steps in a Process with id's *Step 1* and *Step 2*. The output of *Step 1* is an Object with two properties a and b. *Step 1* precedees

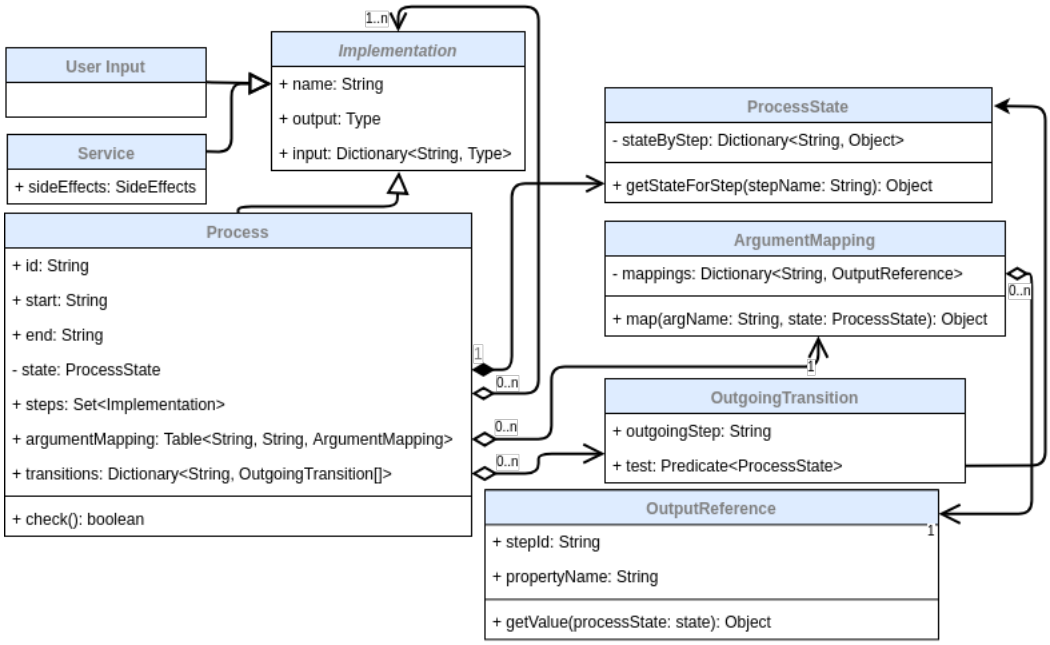

Figure 6. Class diagram representing an abstract model for a *Process*

Step 2 and the Process can transition from Step 1 to Step 2 only if the value stored in the property b of Step 1 is greater than 3. Then the transitions map should contain an instance of an *OutgoingTransition* class where $outgoingStep =$ "$Step2$" and the implementation of test is $test(state) => state[Step1].b > 3$.

To complete the model of a *Process* we need to provide a way to transform the output from one step, to the required format of another step. For that, the *Process* class contains another table, called argumentMapping. For each transition between steps *Step X* and *Step Y*, an *ArgumentMapping* specifies how to generate the required input parameters of *Step Y* from the process state. Each input argument can only be generated from a single *OutputReference*.

A Process model defined like this allows for the system designers to record the business logic in a structured way. Implementation can now follow the constraints put in a Process model clearly, or if a step is ambiguously defined then a feedback can be quickly given to the designers. Most importantly adding new features to the system, or changing existing ones must be compliant with the model.

4.6. *A formal model for describing the process of creating referrals*

We give an example only on the part of the formal description of the "Create Referral" process due to the verbosity required to fully describe the entire process. In the original workflow-diagram in Figure 3 after the "Select Referral Type" step, based on the user selection, the process continues either to the "Get available Appointment Times" step (if the user selected a Regular Referal type), or to the "Populate Referral Details" step. This is modeled in the object diagram in Figure 7 with the *transitions* dictionary instance and the *argumentMapping* Table instance. The transitions contain an entry for the SelectReferralType key, an array of two *OutgoingTransitions*.

The first *OutgoingTransition* object shows that a transition from "Select Referral

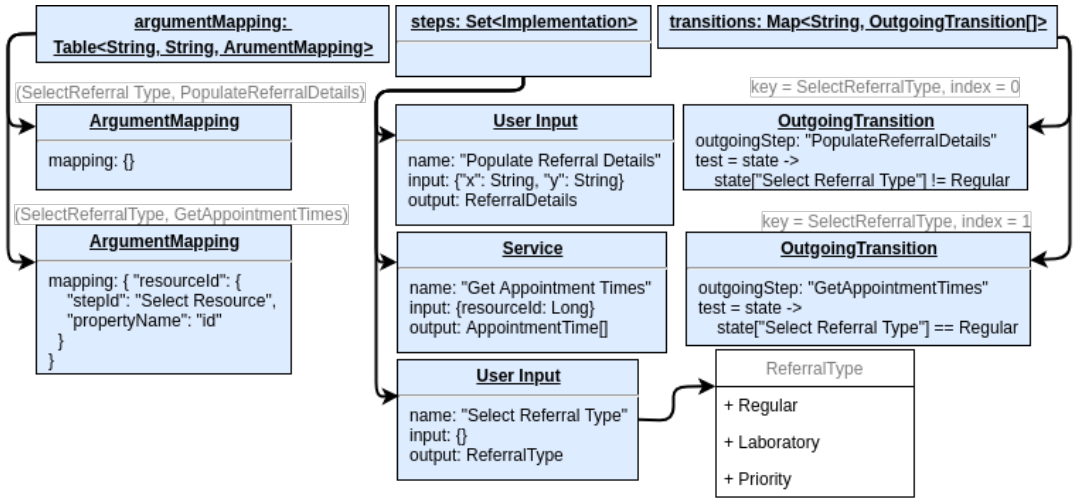

Figure 7. Object diagram of the *Conditional transitions* in the *Create referral* process

Type" to "Populate Details" can happen only if the result of the "Select Referral Type" step saved in the *Process State* is not equal to a *Regular* referral type. The second *OutgoingTransition* object gives the opposite requirement, but for a transition from the "Select Referral Type" step to "Get Appointment Times".

The *argumentMapping* table contains two objects. The first object of type *ArgumentMapping* gives the required transformations for the input arguments of the "Populate Referral Details" step, when transitioning into it from the "Select Referral Type" step. Since the "Populate Referral Type" requires no input, this object is empty. The second object in the *argumentMapping* table shows that the "resourceId" input argument of the "Get Appointment Times" *Service* should be the same as the value stored in the "id" property of the object outputed by the "Select Resource" step that is stored in the process state. The output of the "Select Resource" step is an object of type "Resource", which describes the resource that the patient is being referred to.

4.7. *Organizing the models with MDA*

The different models presented in this section are just abstractions done starting from the current implementation with the goal to capture the business logic in the initial requirements. Figure 8 shows a topological view of how the models can be organized.

Horizontally the models can be grouped based on which part of the system they describe. Three areas are identified: Processes, Services and Documents. Processes describe the different paths a patient can take in health-care, Services are the logic behind a step of the Process, and Documents are the records created in each step. Vertically the models are grouped based on the MDA layers. At the top layer are the work-flow diagrams, free-form text that describes the business logic, and the document descriptions, which all comprise the *Computation Independent Model* (CIM). This layer of the architecture was already present and organized this way when the system was implemented. At the bottom of Figure 8 is the code, that is the classes that implement the Web Services, the definitions of each of the Entities that are saved in the database and all the requests and responses that the API receives and sends out. This layer represents the *Platform Specific Model* (PSM). For the implemented classes, class diagrams can be extracted that will be specific to the Java programming language

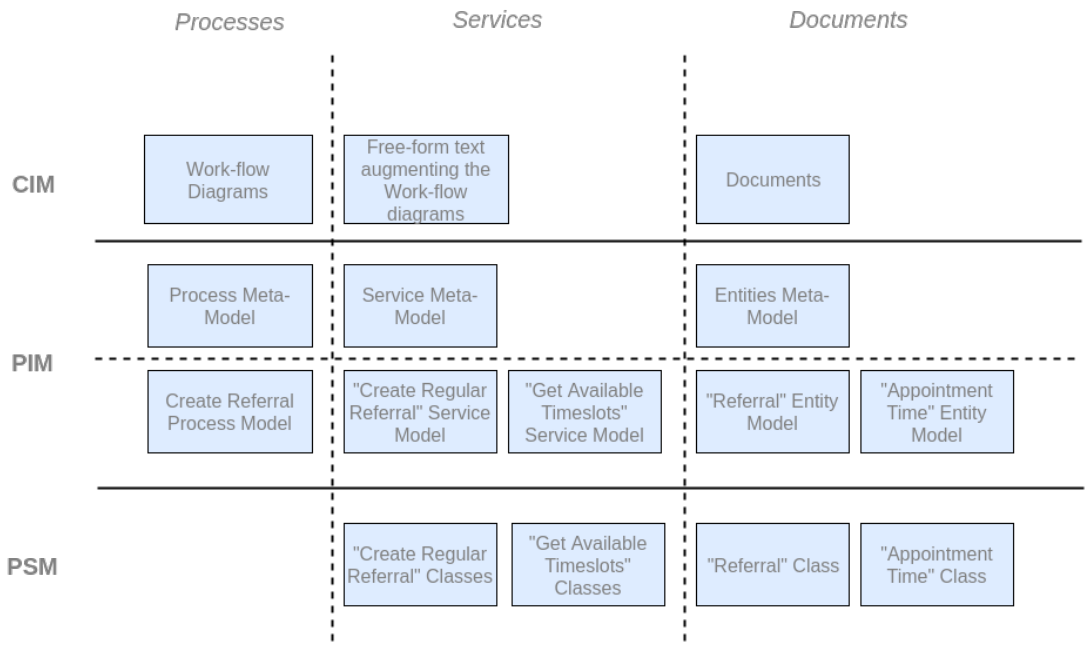

Figure 8. Organization of the models with MDA

in which the system is implemented. The Processes are not implemented in the PSM layer. Section 4.3 describes the models of this layer.

We divide the PIM layer in two sections: models and meta-models. The models sub-layer contains models that describe specific services (Create Regular Referral, Get Available Timeslots) or specific entities (Referral, Appointment Time), for each of them a platform-specific implementation exists. In this sub-layer models for specific processes are introduced (see Section 4.6). The implementations must conform to the models, and the models must conform to the meta-models. The meta-model sublayer defines generic models for Processes, Services and Entities. These meta-models (defined in Section 4.5) give the rules by which the models can be built.

4.8. *Validity of processes*

With the model given in Section 4.5, the validity of a process can be defined. A process is said to be a *valid* if:

- Each step is reachable
- For each step the input requirements can be met
- There is no transitive dependency between processes
- A combination of Service Results and User Input exists that will lead the process to terminate

The process model establishes by design the compatibility of the required input types for a step and the results of previously executed steps. The model allows for the verification of entities also, so that all required data is present in the database. The collection of models for all the processes in the system produces the set of required entities to be saved in the database, and the input and output of each step defines the required projections of each of the entities.

The validity of a process or locating existing issues in the models can be automati-

cally confirmed. Instances of the Abstract Process Model were created by representing the objects using JSON notation. Each object is saved in a separate file. Then a tool was built that reads the object model stored in these files, and parses the files. While parsing of the process model, validation checks whether:

- models are provided for all the steps of the process;
- models are provided for all the input and output types for each of the steps;
- the input arguments for each step are generated by previous steps on the execution path.

Recursively if one of the process steps is a sub-process the model for that process is also parsed. When parsing the models for services, entities and DTOs valid code is genrated. The current implementation of the parsing tools generates Java code, but the process and service models are independent from the generated code, and different backends can quickly be implemented. Having an independent implementation reduces the risk of the system becoming obsolete. The additional information supplied in the Service models about the expected side-effects and the constraints under which the service must operate increase the security of the system by automaticly generating unit tests for each service when parsing the models.

The JSON notation used to store the process models has the advantage that it's easily parsable yet still human readable. The models tend to get verbose when representing larger processes, but the format allows for building simple additional tooling that would provide a visual user interface that would generate the JSON representation for the process models. In addition to the generated code for the Services, with a small effort the tool can be updated to support automatic generation of unit tests based on the supplied side-effects and the conditions given in the models.

The tool was implemented using Java® and is hosted on BitBucket® in a public online repository(Atanasovski 2018). The repository contains example input files for several processes and each of their components. With artifacts built from the source code the input files can be parsed, validated and code is generated for the services and data objects. The only dependency that the system requires upfront to build the artifacts is the Java® SE 8 JDK.

5. Semantic interoperability

In the introduction we stressed the need to improve the system to be more interoperable. In this section we will build upon the ideas presented in Section 4 and (Tu et al. 2014) to create a formal model for data transformations based on creating interfaces between systems instead of modifying the existing systems. First a definition of a meta-model in the PIM layer is given, then example instances of this meta-model are discussed. With the Data Transformation models the requirement for an external system to be integrated with the central system is reduced to implementing the transformation rules between the data types of the external system, and the types used in our system.

5.1. *Data transformation meta-model*

We present the meta-model as a class diagram in Figure 9. The collection of data types in a system is a finite Set that we will call the *Data Model*. The Data Transformation Meta-Model needs to be aware of two *Data Models*, the *from* data model is the source

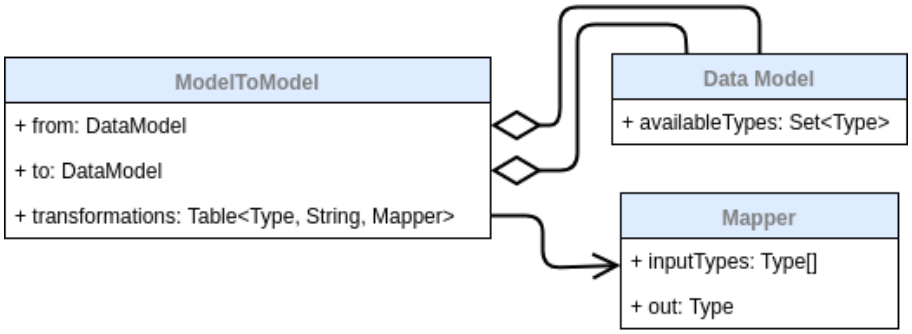

Figure 9. A Meta-Model for describing transformations between two Sets of Types

data model whose *Types* are tranlated into the *Types* of the *to* data model. For one external system to be integrated with our platform two implementations of the *ModelToModel* class must be given. The first maps the types *from* the data model of the external system *to* the types of our platform. The second does the opposite. The first is required to send requests to our platform, the second to recive responses.

The *ModelToModel* instance will keep the transformation logic in the *transformations* property. It is a table where the keys are a *Type* and an *Argument Name*. The *Type* is one of the availableTypes in the target*Data Model*. For each target *Type* and each of the properties of that type there should be an instance of a *Mapper*.

The *Mapper* is a class that specifies a list of input types, and an output type. The *inputTpes* belong to the source *DataModel* specified in the *from* property of the *ModelToModel* class, the *out* Type belongs to the target *DataModel* specified in the *to* property.

5.2. *Example data transformation models*

To demonstrate the transformation meta-model we create a model for a subset of the types present in the HL7 standard to be generated from data types in our system. In HL7 each referral contains information stored in a *ReferralInformationSegment*. This type contains the information about the referral's priority, status, category etc. On the left side of Figure 10 is the diagram of *ReferralInformationSegment* class. For each of the properties a *Mapper* model needs to be defined. In our system, the required information to generate this message can be found in the *Referral* entity and it's properties.

An example transformation model is presented in Figure 11 where *Mappers* are created for the *referralPriority* and *referralType* properties from the *ReferralInformationSegment* external class. The referral priority in the external model can be obtained from the *ReferralType* enumaration class in the internal model. The referal priority in the external model can be obtained from the information in the *Referral* class in the internal model. Finally the last step required before implementing the logic of the transformations in code, is to create interfaces that would be compliant with the object diagram in Figure 11. The implementation logic and the final interfaces of the mappers are given in Figure 12.

Using the presented meta-model, models and implementations can be devised for all the web services in the system. We use the HL7 messages as an example, but

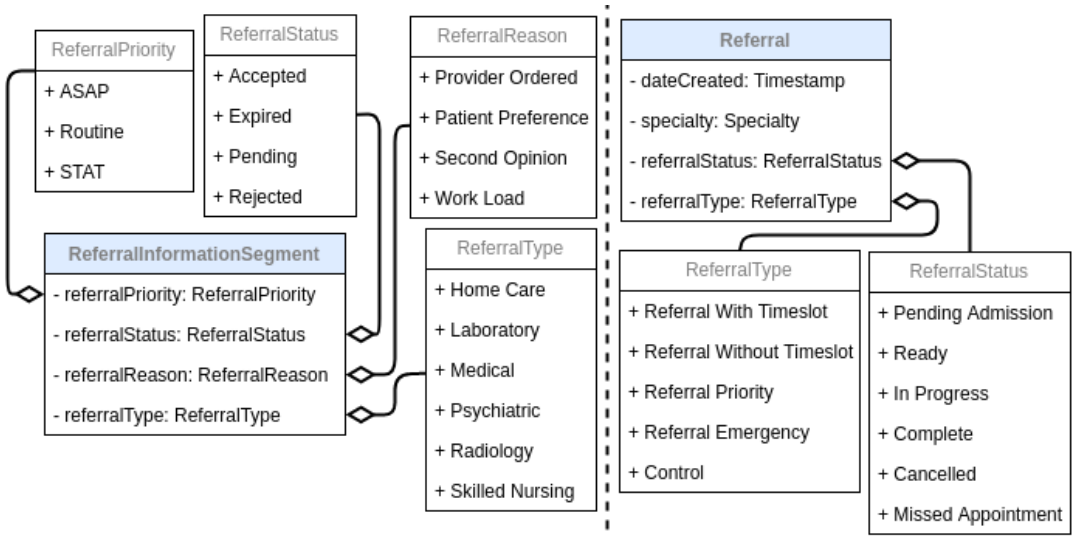

Figure 10. On the left, a part of the ReferralInformationSegment message in the HL7 standard. On the right, a part of the Referral entity in the Moj Doktor system

there is a posibility to create an Interoperability Layer that will translate messages between our system and any external system. An inability to create a *Mapper* for some required property is a direct indication that the systems are not interoperable without modification.

5.3. *Designing a RESTful web service interoperable with the system*

The applicability of UML and model driven engineering can be used for generating RESTful APIs (Schreier 2011). Many of the web services today are using the architectural style called REpresentational State Transfer (REST) which meets the modern software development requirments for web and mobile applications (Rodríguez et al. 2016).

We will conclude the section by presenting a specification for a RESTful web service that can fullfill one of the steps of the "Create Referral" process defined in Section 4.1. We focus on the "Get available Appointment Times" step. The interface of the web service designed here can be the used in the implementation of a new front-end portal. We will utilize the data meta-model defined in this section to show the transformation of the data internal to the system to a representation that gives only the required information outside of the system. Additionally we will show that the model of the RESTful web service can be derived form the meta-model in 4.5.

In the work-flow diagram in Figure 3 the "Get available Appointment Times" step executes after the doctor has selected the referral type, and the resource to which the patient is being referred to. The goal of the step if to retreive from the system a list of Appointment Times that are available for booking. The first step is to create an instance of the *Service* class from the *Process Model* from Figure 6.

The *Get Available Appointment Times* service requires only one argument, the id of the selected resource to which the referral is going to refer to. The output should contain a list of only the available appointment times that belong to the requested resource. Figure 13 shows the instance of the *Service* class. The output of the service is specified as a *Available Appointments Response* type. This type contains a list of *Ap-*

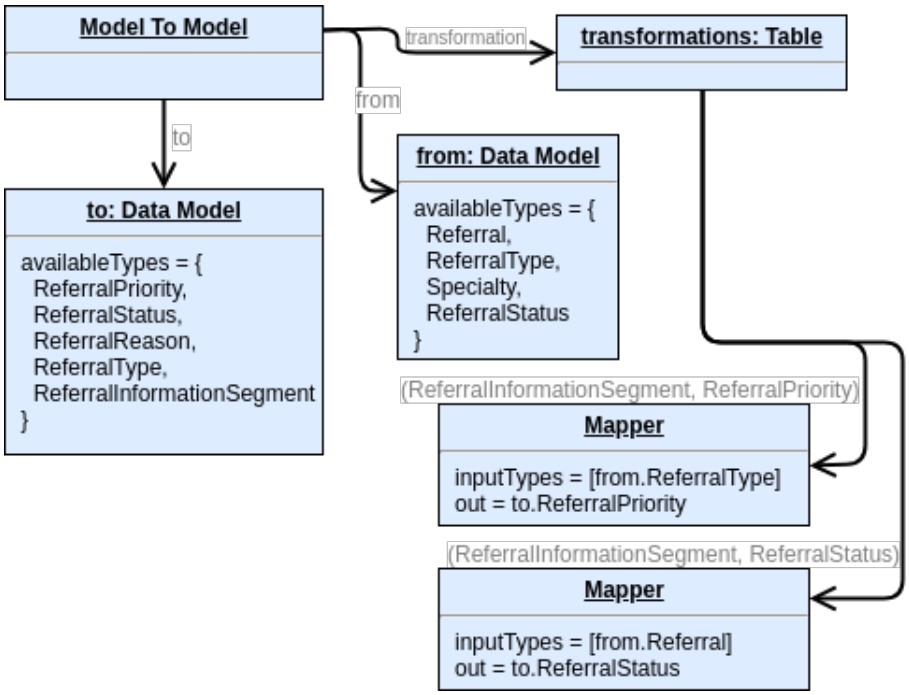

Figure 11. An object diagram depicting the *Mappers* for two of the properties of the *ReferralInformationSegment* class

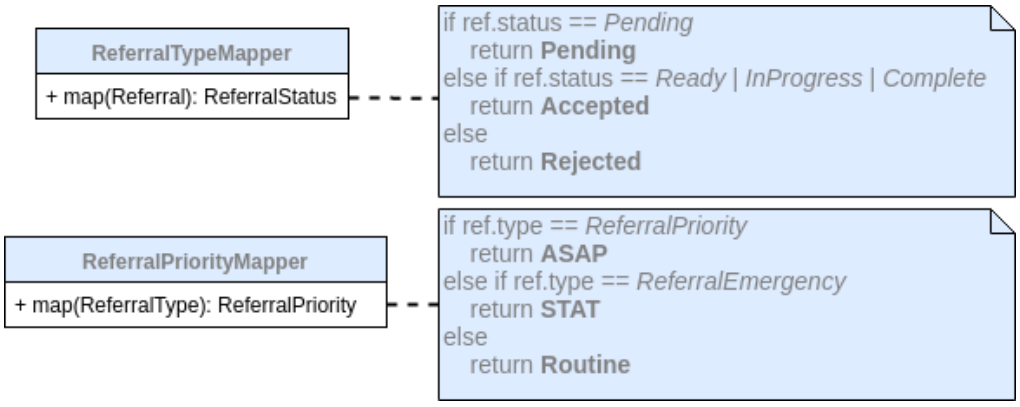

Figure 12. Mappers for HL7 ReferralStatus and ReferralPriority

Figure 13. Object diagram of an instance for the *Get Available Appointment Times* service

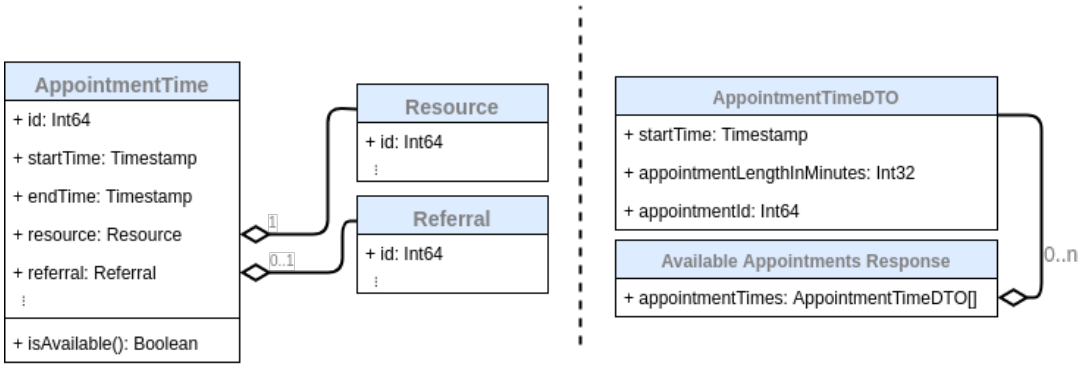

Figure 14. Left - partial class of the *AppointmentTime* entity in the system. Right - class of the required output from the web service

pointmentTimeDTO which in turn only contain only the information needed required for the user to make a choice should be returned.

On the left side of Figure 14 a partial class of the *Appointment Time* entity is shown along with it's relationships to the *Resource* and *Referral* entities in the system. These entities contain a lot more information that what is required for the *Get Available Appointment Times* step.

The required *Mapper* instance that will create an *AppointmentTimeDTO* object from an *AppointmentTime* is straight forward, it is presented in Figure 15. With a Mapper instance like this, and the Service instance from Figure 13 all the requirements are satisfied to define a *Available Appointment Times* RESTfull service.

In Figure 16 a class diagram of the RESTful resource is given. The REST resource is an object with a URI, the HTTP request method, associated parameters and the re-

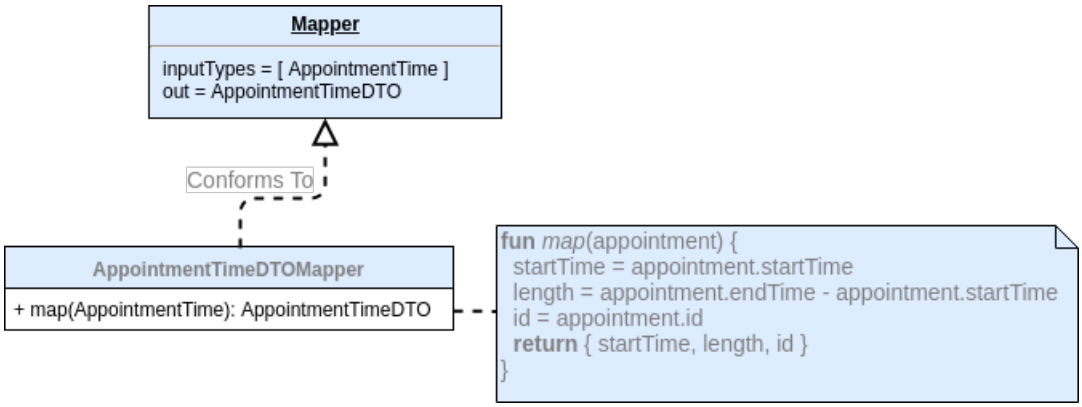

Figure 15. *Appointment Time* to *Appointment Time DTO* mapper

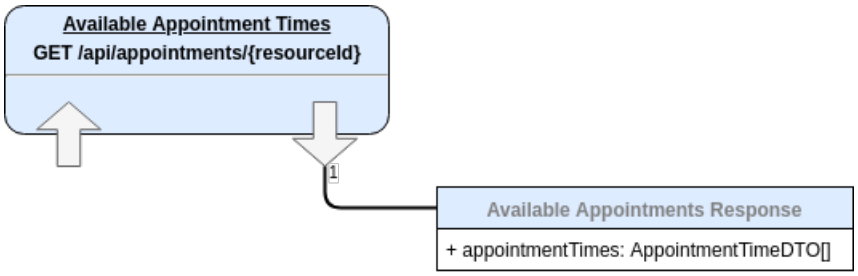

Figure 16. Class diagram of the *Available Appointment Times* RESTful service

quest/response body. It represents a specific service available on the path specified by its URI property, which in our case, is the "Available Appointments Service". This service will respond only to *GET* HTTP requests. Since a GET request does not contain a body, the resourceId input argument is given as a path variable of the HTTP URL. The path that will trigger/call the service is '/api/appointments/resourceId', where resourceId is a placeholder. The service produces an *AvailableAppointmentsResponse*.

Using tools like "Visual Paradigm for UML" (VP-UML 2013) the generation of a REST API and API documentation at this step is a simple process since it is generated automatically. Source code of the communication model and sample source code of client and servlet are generated. API documentation contains HTML files that shows how to use the selected REST Resource.

6. Conclusion

In this paper, we have described the implementation of an enterprise e-health system, where all medical and health related data are stored in a centralized database and processed on-line. The system allows integration with various health organizations, which can communicate with the central database in two directions (read and write).

The goal of this paper was to define the *Platform Independent Model* (PIM). The main reasons were to provide a structured (formal) description of the logic required for each Process, to reduce the possibility of the system becoming obsolete because of the chosen implementation technology, and most importantly to provide a safe and sound platform for introducing changes in the system.

We fit the models defined in the paper in the MDA framework and explain how the different layers are covered with our models. The main contribution is the definition of a *Process Model* that can represent not just the relationships but also execution semantics, and a definition of a valid process. Building upon the *Process Model* we present a model for formalizing data transformation to improve the interoperability of the system and give a working example for transforming data from our system in HL7 messages. We also give a complete example that utilises all of the models defined in the previous sections to create a definition of a RESTful service that can be validated and is interoperable with our system.

We also developed a tool that runs static verification of the processes, whether all input requirements are satisfied for all steps of the processes, whether all required types are defined, if there are duplicates, etc. The next iteration of the tool will include automatic generation of unit tests for web services based on the side effects specified in the models. A second type of tools will do dynamic verification. These tools will do verification by executing the steps on supplied data. They will be based on technolo-

gies for Unit and Integration Testing that support mocking of components, coverage reporting and dynamic code generation and injection.

We also plan on making the models presented here more change-proof by utilizing the findings in (Dam et al. 2016) to evolve the models and prevent the system from becoming obsolete. We can also use the experiences from developing a similar e-health system in Slovenia, see Stanimirovic (2015), in order to strengthen the value of the model suggested here by taking into account other operative, development and implementation aspects.

References

Atanasovski, B. (2018, Jul). Process model parser - online code repository. url-https://bitbucket.org/atanasovskib/pmp/src/master/. Accessed: 2018-07-02.

Blobel, B. and P. Pharow (2006). A model driven approach for the german health telematics architectural framework and security infrastructure. *International Journal of Medical Informatics 76*, 169—-175.

Bolognesi, T., D. De Frutos, R. Langerak, and D. Latella (1995). Correctness preserving transformations for the early phases of software development. In *LOTOSphere: Software Development with LOTOS*, pp. 161—-180. Springer, Boston, MA.

CEN (2008). Electronic health record communication (en 13606).

Chan, J., K. G. Shojania, A. C. Easty, and E. E. Etchells (2011). Does user-centred design affect the efficiency, usability and safety of cpoe order sets? *Journal of the American Medical Informatics 18*(3), 276–281.

CORBA (2012). Common object request broker architecture (corba) 3.3.

Curcin, V., T. Woodcock, A. J. Poots, A. Majeed, and D. Bell (2014). Model-driven approach to data collection and reporting for quality improvement. *Journal of Biomedical Informatics 52*(C), 151—-162.

Dam, H. K., L.-S. Le, and A. Ghose (2016). Managing changes in the enterprise architecture modelling context. *Enterprise Information Systems 10*(6), 666–696.

Erl, T. (2007). *Principles of Service Design*. Prentice Hall.

Garde, S., R. Chen, H. Leslie, T. Beale, I. McNicoll, and S. Heard (2009). Archetype-based knowledge management for semantic interoperability of electronic health records. In *International Congress of the European Federation for Medical Informatics (MIE-09)*, pp. 1007–1011.

Häyrinen, K., K. Saranto, and P. Nykänen (2008). Definition, structure, content, use and impacts of electronic health records: A review of the research literature. *International Journal of Medical Informatics 77*(5), 291–304.

HL7 (2006). Hl7 development framework methodology specification (hdf). Technical report, International Organization for Standardization, Health Level Seven Inc.

HL7 (2011). Health level-7. Technical report, Health Level Seven International (HL7).

Holzmann, G. (2003). *MDA distilled: principles of model-driven architecturThe Spin Model Checker: Primer and Reference Manuale*. Addison-Wesley Professional.

IOS (2009). Reference model of open distributed processing (rm-odp, iso/iec 10746). Technical report, International Organization for Standardization.

ISO/TC215 (2009). Iso/hl7 27931, data exchange standards health level seven version 2.5.

Johnson, C. M., T. R. Johnson, and J. Zhangb (2005). A user-centered framework for redesigning health care interfaces. *Journal of Biomedical Informatics 38*, 75–87.

Jones, V. (1995). Realization of ccr in c. In *LOTOSphere: Software Development with LOTOS*, pp. 348—-368.

Jones, V. (1997). Engineering an implementation of the osi ccr protocol using the information systems engineering techniques of formal specification and program transformation. Technical report, University of Twente, Centre for Telematics and Information Technology

Technical Report series no. 97-19. ISSN 1381-3625.

Jones, V., A. Rensink, and E. Brinksma (2005). Modelling mobile health systems: an application of augmented mda for the extended healthcare enterprise. In *EDOC Enterprise Computing Conference, 2005 Ninth IEEE International*, pp. 58–69. IEEE.

Lapsia, V., K. Lamb, and W. A. Yasnoff (2012). Where should electronic records for patients be stored? *International Journal of Medical Informatics 81*(12), 821–827.

Larman, C. and V. R. Basili (2003). Iterative and incremental development: A brief history. *Journal Computer 36*(6), 47—-56.

Lopez, D. and B. G. Blobel (2009). A development framework for semantically interoperable health information systems. *International Journal of Medical Informatics 78*, 83—-103.

Meso, P. and R. Jain (2006). Agile software development: Adaptive systems principles and best practices. *IS Management 23*(3), 19–30.

MITRE-corporation (2006). Electronic health records overview. Technical report, US NIH National Center for Research Resources (NCRR).

Nguyen, E., E. Bellucci, and L. T. Nguyen (2014). Electronic health records implementation: an evaluation of information system impact and contingency factors. *International Journal of Medical Informatics 83(11)*, 779–796.

Rayhupathi, W. and A. Umar (2008). Exploring a model-driven architecture (mda) approach to health care information systems development. *International Journal of Medical Informatics 77*, 305—-314.

Ricciardi, L. (2010). Protecting Sensitive Health Information in the Context of Health Information Technology. Technical report, Consumer Health, Clinovations LLC.

Riccobene, E. and P. Scandurra (2009). Weaving executability into uml class models at pim level. In *Proceedings of the 1st Workshop on Behaviour Modelling in Model-Driven Architecture*, pp. 1–10. ACM.

Rodríguez, C., M. Baez, F. Daniel, F. Casati, J. C. Trabucco, L. Canali, and G. Percannella (2016). Rest apis: a large-scale analysis of compliance with principles and best practices. In *International Conference on Web Engineering*, pp. 21–39. Springer.

RUP (2017). Best practices for software development teams, rational software white paper, tp026b, rev 11/01.

Schlieter, H. and et al. (2015). Towards model driven architecture in health care information system development. In *12th International Conference on Wirtschaftsinformatik, WI 2015*, pp. 497–511.

Schreier, S. (2011). Modeling restful applications. In *Proceedings of the Second International Workshop on RESTful Design*, WS-REST '11, New York, NY, USA, pp. 15–21. ACM.

Stanimirovic, D. (2015). Modelling the health information system in slovenia — operative, construction and implementation aspects. *International Journal of Engineering Business Management 7*, 13.

Tao, Z., Y. Luo, C. Chen, M. Wang, and F. Ni (2017). Enterprise application architecture development based on dodaf and TOGAF. *Enterprise Information Systems 11*(5), 627–651.

Tretmans, J. and A. Belinfante (1999). Automatic testing with formal methods. In *7th European Int. Conference on Software Testing, Analysis & Review, Barcelona, Spain.*

Tu, Z., G. Zacharewicz, and D. Chen (2014). Building a high-level architecture federated interoperable framework from legacy information systems. *International Journal of Computer Integrated Manufacturing 27*(4), 313–332.

US-HHS (2010). Data segmentation in electronic health information exchange: Policy considerations and analysis. Technical report, ONC, US Department of Health and Human Services (HHS).

US-HHS (2015). Guide to Privacy and Security of Electronic Health Information. Technical report, ONC, US Department of Health and Human Services (HHS).

Velinov, G., B. Jakimovski, D. Lesovski, D. Ivanova Panova, D. Frtunik, and M. Kon-Popovska (2015). Ehr system mojtermin: Implementation and initial data analysis. *Studies in Health Technology and Informatics 210*, 872–876.

Voigt, B. J. J. (2004). *Dynamic System Development Method.* Ph. D. thesis, Department of

Information Technology, University of Zurich.

VP-UML (2013). Visual paradigm for uml. *Visual Paradigm for UML-UML tool for software application development*, 1–72.

Xu, B., L. Xu, H. Cai, L. Jiang, Y. Luo, and Y. Gu (2017). The design of an m-health monitoring system based on a cloud computing platform. *Enterprise Information Systems 11*(1), 17–36.