CONTRIBUTIONS ON THE MULTI-TIER ARCHITECTURE OF ELECTRONIC HEALTH RECORD SYSTEMS

Teză destinată obţinerii
titlului ştiinţific de doctor inginer
la
Universitatea Politehnica Timişoara
în domeniul
CALCULATOARE ŞI TEHNOLOGIA INFORMAŢIEI
de către

Ing. ec. inf. Daniel-Alexandru JURCĂU

Conducător ştiinţific:	prof.univ.dr.ing. Vasile STOICU-TIVADAR
Referenţi ştiinţifici:	prof.univ.dr.ing. Rodica POTOLEA
	prof.univ.dr.fiz. Gheorghe-Ioan MIHALAȘ
	prof.univ.dr.ing. Horia CIOCÂRLIE

Ziua susţinerii tezei: 06.10.2016

România, 300159 Timişoara, Bd. Republicii 9,
tel. 0256 403823, fax. 0256 403221
e-mail: editura@edipol.upt.ro

Cuvânt înainte

Teza de doctorat a fost elaborată pe parcursul activităţii mele în cadrul Departamentului de Automatică şi Informatică Aplicată al Universităţii Politehnica Timişoara.

Lucrarea se adresează tuturor celor interesați de dezvoltarea sistemelor informatice de gestiune a înregistrărilor electronice de sănătate, aducând contribuții în ceea ce privește găsirea metodelor de a proiecta sisteme software ce gestionează înregistrări electronice de sănătate.

Lucrarea investighează un mod de a structura datele pe baza tipurilor de înregistrări identificate. Ulterior caută să îmbunătățească flexiblitatea prin găsirea unor moduri de a integra specificații, standarde și terminologii existente în solutie.

Întrucât personalul medical interacționează zilnic cu astfel de sisteme, lucrarea investighează și evaluează de asemenea moduri de creare a interfețelor moderne de interacțiune cu utilizatorul în ceea ce privesc înregistrările electronice de sănătate.

Consider că lucrarea este un suport ştiinţific de luat în seamă pentru cercetările viitoare asupra modalităților de dezvoltare și integrare a aplicațiilor ce gestionează înregistrări electronice de sănătate.

Mulţumiri deosebite se cuvin conducătorului de doctorat prof.dr.ing. Vasile STOICU-TIVADAR, alături de membrii comisiei de îndrumare: prof.univ.dr.ing. Ionel JIAN, prof.univ.dr.ing. Diana LUNGEANU respectiv șl.dr.ing. Dorin BERIAN.

Timişoara, septembrie 2016 Daniel-Alexandru Jurcău

Jurcău, Daniel-Alexandru

Contributions on the Multi-Tier Architecture of Electronic Health Record Systems

Teze de doctorat ale UPT, Seria 14, Nr. 31, Editura Politehnica, 2016, 156 pagini, 70 figuri, 19 tabele.

ISSN: 2069-8216

ISSN-L: 2069-8216

ISBN: 978-606-35-0090-9

Cuvinte cheie: electronic health records, HL7 CDA, openEHR, LOINC, ICD10, architecture, usability,

Rezumat,

This thesis brings contributions to finding methods for designing software systems that manage electronic health records. It investigates a way to structure the data based on types of inputs identified and then goes on to improve the flexibility by finding ways to integrate existing specifications, standards and terminologies into the solution. The thesis handles multiple areas of interest concerning the development of electronic-heath record applications: storage, processing and the presentation of data; all in a structured manner.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
AOP	Aspect-Oriented Programming
CDA	Clinical Document Architecture
CSS	Cascading Style Sheets
DBMS	Database Management System
DI	Dependency Injection
DOM	Document Object Model
EHR	Electronic Health Record
GOMS	Goals, Objects, Methods and Selection rules
HL7	Health Level Seven
HTTP	HyperText Transfer Protocol
HTML	HyperText Markup Language
ICD	International Classification for Diseases
JSON	JavaScript Object Notation
LOINC	Logical Observation Identifiers Names and Codes
MVVM	Model View ViewModel
NoSQL	Non Structured Query Language
OLTP	Online Transactional Processing
OOP	Object-Oriented Programming
ORM	Object-Relational Mapping
RDBMS	Relational Database Management System
REST	Representational State Transfer
RIM	Reference Information Model
SDK	Software Development Kit
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SPA	Single-Page Application
SQL	Structured Query Language
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language

1. INTRODUCTION

Healthcare is a domain which generates a lot of data every day, however, traditionally, in paper form. Automating healthcare processes by using computer software systems opens the way to a lot of improvements, among which: [1]

- Recording information with less room for error
- Allowing for much easier/faster information retrieval
- Providing a higher degree of understanding information recorded by someone else
- Providing easier ways of sharing information between systems
- Allowing complex queries to be performed on pools of data for analytic/statistical purposes.

The first obstacle encountered when setting of to build a software system for dealing with healthcare data is that the domain in question is both complex and huge. Depending on variables such as the medical specialty, the clinical setting or the country in which one operates, each medic might encounter a completely different slice of the healthcare information domain on a day to day basis. Considering that some type of information is useful from a clinical perspective, while a different piece is only used for administrative purposes like health insurance further complicates matters.

Because of this diversity, creating simple Create-Read-Update-Delete (CRUD) applications that merely store a few bits and pieces of information in a database is not an option as it provides no more utility than simply using a word processor for recording information instead of pen and paper. The challenges involved in creating a flexible system have inspired many research projects.

This thesis aims to bring contributions to finding methods of designing software systems that manage electronic health records. It investigates a way to structure the data based on types of inputs identified and then goes on to improve the flexibility by finding ways to integrate existing specifications, standards and terminologies into the solution.

Because medical professionals need to interact with such systems on a daily basis, the thesis also investigates and evaluates ways of creating a modern user interface for storing electronic health records.

1.1. Thesis Goals

The aim of this thesis is to bring valid knowledge which is both useful and usable in practice for the creation and improvement of software solutions dealing with electronic health records, in all the areas of interest concerning their development:

storage, processing and presentation of data; all in a structured manner. As such, the author brings contributions with the purpose of obtaining medical applications that are both efficient and easy to use by medics.

To this end, the goals taken into consideration are the following:

- Identifying the type of data stored in electronic health records
- Researching into a way of structuring the required data in a manner that is both easy to implement and flexible enough to change
- Identifying a way of implementing domain and service logic for dealing with the structured electronic health record data
- Finding ways to include standard medical terminologies in an existing solution
- Researching ways of building an appropriate user-interface and how to evaluate the usability.

1.2. Thesis Structure

This thesis is divided into four major chapters, accompanied by an introduction and conclusions. The structure of each chapter and the interrelations between them are summarized in figure 1.1.

Chapter 2, *Medical Standards and Terminologies* starts by providing information about existing medical approaches and standards for building applications. It presents the Health Level 7 (HL7) Clinical Document Architecture (CDA) standard which provides a way to format (usually as XML) documents which exchange medical information.

An as alternative to HL7 standards, the chapter also presents the openEHR specification. Medical coding standards like the Logical Observation Identifiers Names and Codes (LOINC) and the International Classification for Diseases (ICD) are also introduced.

The information presented in this chapter is later used in chapter 4 which analyses ways of integrating such standards into an existing application.

Chapter 3, *Designing a Solution for the Structured Collection of Medical Data* analyses the requirements for building an application that manages electronic health records. It then proceeds with suggesting an architecture and building a prototype of such a system that uses a flexible approach of storing metadata on the actual inputs that compose each form, instead of simply storing unrelated columns in a database.

This approach is designed to easily provide a solution to current requirements and does not take specifications such as openEHR into account. The prototype created is used in the next chapter as a starting point for linking openEHR artifacts and also finding ways of using Health Level 7 (HL7) standardized documents for exchanging medical data.

Chapter 4, *Integrating Medical Standards*, presents the implications of integrating standard medical terminologies into an existing electronic health record application and also deals with ways of inter-operating with openEHR methodologies in order to more easily deal with terminologies.

An analysis is performed on the structure of Logical Observation Identifiers Names and Codes (LOINC) codes and ways to more easily translate them into languages in which they are currently not available.

International Classification for Diseases (ICD) codes are also discussed in an effort to evaluate open-source full-text search engines for providing better and more precise ways of searching and matching such codes. Lastly, the chapter proposes a solution for combining the elements above in an architecture which provides interoperability by generating specific connectors which consume or produce health data in the form of Clinical Document Architecture (CDA) documents.

Chapter 5, *Developing a Web Front-End for Electronic Health Records* deals with the aspects of building a front-end for an application that deals with electronic health records. The chapter presents the links between the front-end and the back-end architectures and offers a flexibly way of using metadata in combination with a rich client-side application in order to generate UI elements on the fly. As the front-end is the gateway to the application, seen from the eyes of the users, its usability is very important. Ways of evaluating the usability are taken into account and applied on a prototype application.

While working on this thesis, various parts have been published in the following articles:

- *Modern Technologies for Improving Interoperability in Health Information Systems*. **Daniel-Alexandru Jurcău** and Vasile Stoicu-Tivadar. Buletinul Științific al Universității Politehnica Timișoara, 2014, pp. 53–58
- *Using Modern Technologies to Facilitate Translating Logical Observation Identifiers Names and Codes*. **Daniel-Alexandru Jurcău**, Vasile Stoicu-Tivadar and Alexandru Șerban. Proceedings of the 6th International Workshop Soft Computing Applications (SOFA 2014), pp. 219–229
- *Incidence Rate of Canonical vs. Derived Medical Terminology in Natural Language*. Vasile Topac, **Daniel-Alexandru Jurcau** and Vasile Stoicu-Tivadar. Digital Healthcare Empowering Europeans. Proceedings of MIE 2015, pp. 5–9
- *Evaluating Open-Source Full-Text Search Engines for Matching ICD-10 Codes*. **Daniel-Alexandru Jurcău** and Vasile Stoicu-Tivadar. 14th International Conference on Informatics, Management and Technology in Healthcare, 2016
- *Evaluating the User Experience of a Web Application for Managing Electronic Health Records*. **Daniel-Alexandru Jurcău** and Vasile Stoicu-Tivadar. Proceedings of the 7th International Workshop Soft Computing Applications (SOFA 2016)

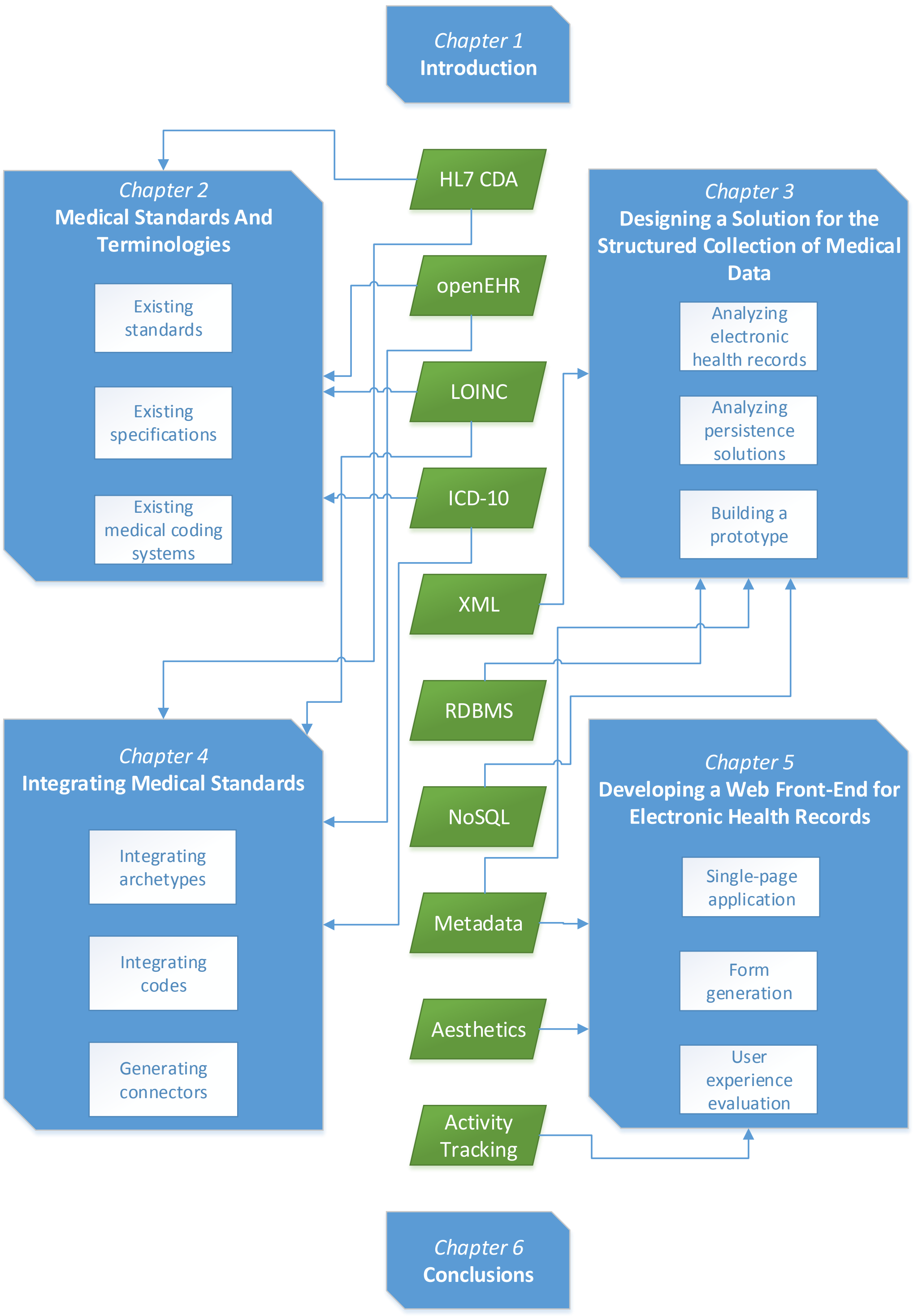

Chapter 1
Introduction
Chapter 2
Medical Standards And Terminologies
Existing standards
Existing specifications
Existing medical coding systems
HL7 CDA
openEHR
LOINC
ICD-10
XML
RDBMS
NoSQL
Metadata
Aesthetics
Activity Tracking
Chapter 3
Designing a Solution for the Structured Collection of Medical Data
Analyzing electronic health records
Analyzing persistence solutions
Building a prototype
Chapter 4
Integrating Medical Standards
Integrating archetypes
Integrating codes
Generating connectors
Chapter 5
Developing a Web Front-End for Electronic Health Records
Single-page application
Form generation
User experience evaluation
Chapter 6
Conclusions

Fig. 1.1: Diagram detailing the subjects and technologies encountered in the thesis chapters

2. MEDICAL STANDARDS AND TERMINOLOGIES

The healthcare domain is one of the most complex in the world and, as such, poses many challenges when it comes to being automated with the help of computer software.

Among the reasons for automating health care data is the possibility to document actions taken to treat patients: [2]

- The request of a test
- The report of a test result
- The creation of a diagnosis
- The prescription of treatment.

The goal of software systems that deal with electronic health records is to allow users to digitally record medical information that has been traditionally stored on paper. As the diversity of this type of information is large, so are the amount of ways in which such software can be written. Simple approaches might find it easy to fulfill a very limited subset of requirements but evolving the system with new goals will be ever more difficult.

The most important challenge when building software that automates working with health records is in finding efficient ways of dealing with medical data. As documented in literature, physicians are usually passive users that show no concern on what the data type is and how it is organized [3]. This creates a need for building efficient user interfaces that manage to extract as much information as possible from the physicians with as little effort as possible.

This chapter introduces medical standards and specifications which have been developed to help software vendors provide solutions that operate in a similar way and are able to export data from one system so that other systems will automatically understand it. Adopting such standards is not a bullet-proof solution, each of them presenting various strengths and weaknesses which have given rise to new suggestions and alternative approaches.

The chapter also deals with common medical coding systems and the issues of using them. Such systems provide a good introduction into the healthcare domain and can be readily used by developers as a source of domain entities.

Using standardization leads to easier interoperability which is defined as "the ability of two or more systems or components to exchange information and to use the information that has been exchanged" [4].

Why is standardization needed? In order to treat a medical problem, patients often need to consult multiple physicians [3]. Without interoperable systems, physician #2 has no way of knowing what medical information physician #1 has about the patient's condition. Typical solutions which involve the first physician issuing a hand-written letter describing facts about the patient, or, even worse, relying only on information provided verbally by the patient himself are seldom appropriate.

In Romania, integration between medical systems is mostly limited to reporting administrative information to the *Unique Integrated Information System of Social Health Insurances* (SIUI), a complex national system with the goal of managing the acquisition, storage and processing of data in regards to the national health insurance system [5, 6].

Established at the University of Pennsylvania in 1987, Health Level Seven® (HL7) is one of several ANSI-accredited standards developing organizations (SDOs) operating in the healthcare domain [7]. HL7 provides "a standard for the exchange, management and integration of data that support clinical patient care, management, delivery and evaluation of healthcare services" [7].

HL7 standards provide rules on how data contents should be exchanged between healthcare applications [7]. Among the functional domains covered by HL7 standards are financial transactions, observation reporting, order entry and patient administration [8].

The essence of version 3 of the HL7 standards is to "apply the 'best practices' of software development to developing standards – a model-based methodology" [2]. In contrast with version 2 of the standards which are "primarily concerned with organizational issues and rarely provide means for terminological control", HL7 v3 intends to support the interchange of medical contents at context level [3]. Among HL7's v3 standards, CDA is the most widely adopted [9].

This chapter is divided into three sections. Section *Health Level Seven Clinical Document Architecture* introduces HL7's Clinical Document Architecture (CDA), a document standard for exchanging clinical information.

Section *Medical Code Systems* introduces the world of coding systems and terminologies. Instead of using long terms when communicating medical information, a practice that can easily lead to confusion or misinterpretation, various coding systems have been developed which provide simple codes, that, when transmitted, allow the recipient to easily understand what the sender is talking about. This section presents the Logical Observation Identifiers Names and Codes (LOINC) which are mainly used for coding laboratory data and International Classification for Diseases (ICD) which encode medical diagnostics.

In order to also look at alternatives, *The openEHR Specifications* section describes an approach to building electronic health record systems known as openEHR. This specification is built on a multi-model approach and aims to improve in certain areas where HL7 standards are lacking.

2.1. Health Level Seven Clinical Document Architecture

The HL7 Clinical Document Architecture (CDA®) is a standard which provides a common architecture, coding, semantic framework, and markup language for the creation of electronic clinical documents. Among its goals are: [10]

- Allowing cost effective implementations across a wide variety of systems
- Allowing the exchange of human-readable documents

- Compatibility with a wide range of applications used to create documents
- Promoting document exchange in a manner independent of the actual storage of transfer mechanism used
- Promoting longevity of all the encoded information.

The HL7 CDA standard can be used to transmit messages in a simple and structured manner, thus allowing access to unstructured databases [11]. An important challenge lies in the local adaptation of computer-based guidelines and protocols [11].

Data stored in CDA documents can be quite diverse: clinical summaries, diagnostic reports, discharge summaries, history/physical examinations, prescriptions, etc [11].

CDA documents are coded using the Extensible Markup Language (XML) and are created with a focus on document exchange [10]. Release 2 of CDA was launched in 2005 and the model is characterized by context, human readability, persistence and the ability to sign documents [12].

The characteristics and goals of clinical documents differ from those of messages, implemented in the form of the HL7 v3 messaging standard: [13]

- *Documents* promote human readability, persistence and self containment
- *Messages* promote machine processability, are based on the status change of one or more business-objects and are capable of providing real-time information.

The CDA standard is derived from the HL7 Reference Information Model (RIM), shown in figure 2.1. The model contains four basic classes: [3]

- *Entity* – a physical thing or group thereof: patients, clinicians, nurses, rooms, wards, etc.
- *Act* – a record of something that "is being done, has been done, can be done, or is intended or requested to be done" [3]: admissions, treatments, transfers, etc.
- *Participation* – "an association between an *Act* and a *Role* with an *Entity* playing that *Role*" [3]
- *Role* – "a competency of the *Entity* playing the *Role* as identified, defined, guaranteed, or acknowledged by the *Entity* that scopes the *Role*" [3].

As such, composing CDA entries presents challenges such as requiring RIM modeling expertise in order to express any particular piece of clinical information, as the representations are not obvious out of the box [9]. Moreover is does also lead to common clinical concepts being modeled differently under different circumstances [9].

The HL7 v3 RIM has also been specialized to the medical device domain resulting in a Refined Message Information Model (RMIM) [15]. This allowed the authors to achieve interoperability as they used concepts derived from a common RIM, allowing the building blocks of the interfaces to be similar and traced back to it [15].

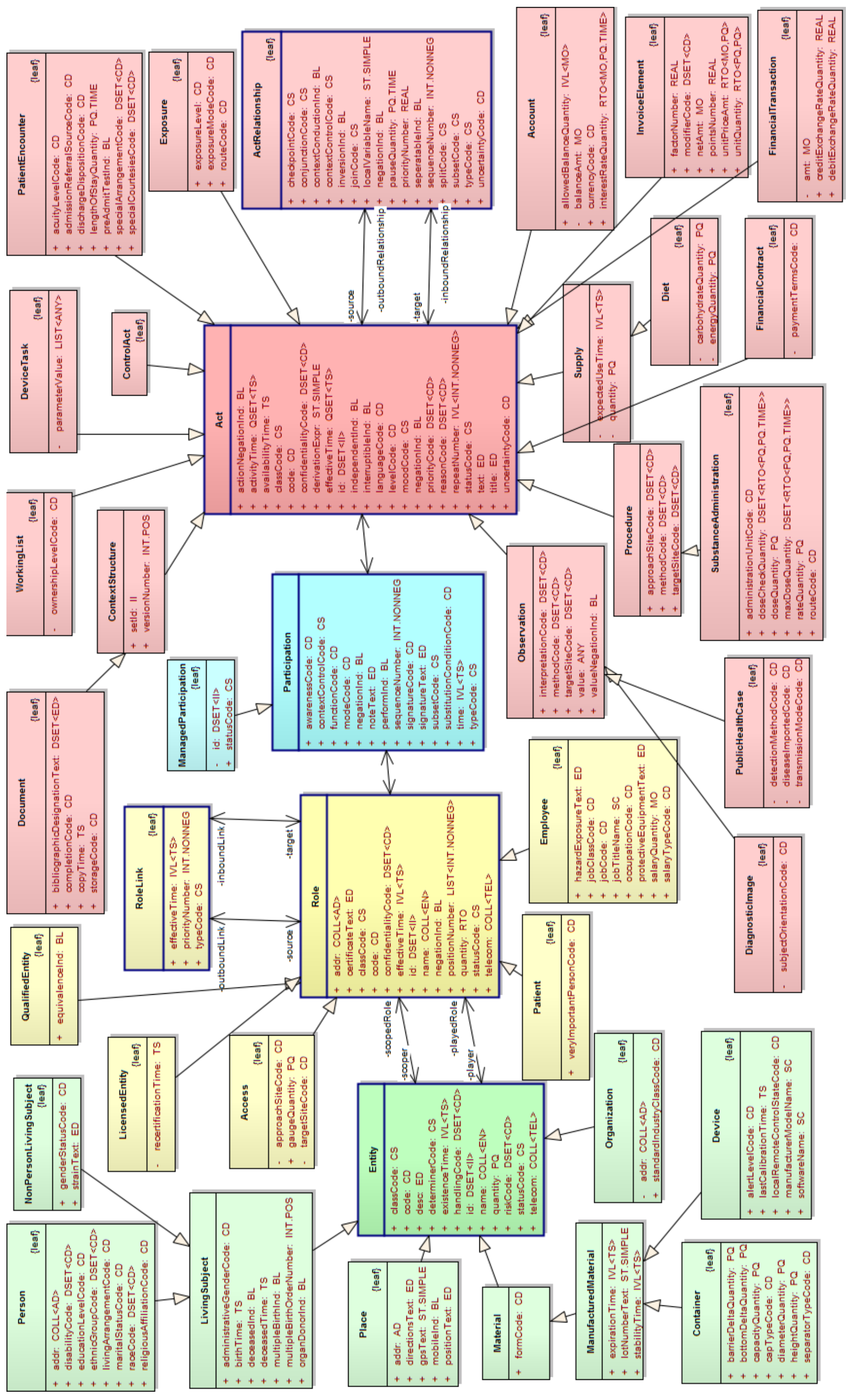


Fig. 2.1: The HL7 Reference Information Model [14]. The four main classes and their hierarchies are highlighted in different colors: Entity (green), Role (yellow), Participation (cyan) and Act (red).

BUPT

CDA documents rely heavily on code sets for document types, document sections, clinical procedures, and clinical findings. Codes include Current Procedural Terminology (CPT®), International Statistical Classification of Diseases and Related Health Problems (ICD), Logical Observation Identifiers Names and Codes (LOINC®), MEDCIN®, Systematized Nomenclature of Medicine (SNOMED®) or any code set used in the RIM or internal RIM vocabularies [10]. As can be seen in the listing below, using a code involves specifying:

- The code system's unique object identifier (OID): *2.16.840.1.113883.6.1*
- The code system's human readable name: *LOINC*
- The code identifier: *11488-4*
- The code identifier's human readable name: *Consultation note*

```
<code code="11488-4" codeSystem="2.16.840.1.113883.6.1"
   codeSystemName="LOINC" displayName="Consultation note" />
```

The structure of a CDA document consists of two main parts: [16]

1. The *Header* provides information on authentication, the document's identity, the document's classification, the encounter, the patient and the providers involved
2. The *Body* contains the clinical report and can be structured or unstructured.

The listing below exemplifies the metadata present in a CDA document standardized header, showing information about the document's author: [10]

```
<author>
  <time value="2000040714"/>
  <assignedAuthor>
    <id extension="KP00017" root="2.16.840.1.113883.19.5"/>
    <assignedPerson>
      <name>
        <given>Robert</given>
        <family>Dolin</family>
        <suffix>MD</suffix>
      </name>
    </assignedPerson>
    <representedOrganization>
      <id root="2.16.840.1.113883.19.5"/>
      <name>Organization Name</name>
    </representedOrganization>
  </assignedAuthor>
</author>
```

The hierarchy of a CDA document extends on three levels: [16]

- *Level I* – largely narrative text, no structured data (non XML)
- *Level II* – provides a structured narrative, broken into sections (XML body, narrative block)

- *Level III* – includes additional formal expressions of clinical content by means of coding and explicit data representations (XML body, clinical statement). Level III markup cannot contain more information than the narrative, in order to adhere to CDA's human readability principle [17].

CDA documents provide a *narrative block*, a mechanism to dress up information in the user's browser. This block provides representations of tables, item lists and various text formatting [3]. XSL style sheet transformations can be applied on the CDA document resulting in an HTML file ready to display inside the user's browser. Having a higher level CDA document which also contains information for computers to parse is also beneficial when the document is read by a person as it helps clarify the information, should the narrative block contain ambiguities.

The document content, when structured, contains multiple sections, as in the example below provided by [18]:

```
<ClinicalDocument
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:mif="urn:hl7-org:v3/mif" xmlns="urn:hl7-org:v3">
   ...
   <component>
      <structuredBody>
         <component>
            <section>
            ...
            </section>
            ...
         </component>
      </structuredBody>
   </component>
</ClinicalDocument>
```

Each section can contain any number of CDA entries, but only a single narrative block [3]:

```
<section>
   <templateId root="2.16.840.1.113883.10.20.5.5.15"/>
   <code codeSystem="2.16.840.1.113883.6.1"
         codeSystemName="LOINC" code="18769-0"
         displayName="Findings Section"/>
   <title>Bug-Drug Tests</title>
   <text>...</text>
   <entry>...</entry>
</section>
```

The content a CDA entry is expressed using a "complex and extremely abstract model based on HL7's *Clinical Statement* project" [9] with the purpose of allowing any degree of rigor and granularity. The clinical statement consists of 9 *Acts* classes: Act, Encounter, Observation, ObservationMedia, Organizer, Procedure, RegionOfInterest, SubstanceAdministration and Supply [3]. The listing below shows an example entry which codes the value of the *body temperature* as an observation [11]:

```
<entry>
   <observation classCode="OBS" moodCode="EVN">
      <code code="386725007" codeSystem="2.16.840.1.113883.6.69"
         codeSystemName="SNOMED CT" displayName="Body temperature" />
```

```
    <statusCode code="completed" />
    <effectiveTime value="200004071430" />
    <value xsi:type="PQ" value="36.9" unit="Cel" />
  </observation>
</entry>
```

An issue still present in today's healthcare facilities is that, while they introduce electronic medical record systems, some facilities still use paper when exchanging clinical information [19]. When switching medical facilities, patients receive referral form inputs from medics at the former one and manually pass them along to the staff of the current facility, a process that wastes resources and also prevents medics at the accepting facility from reading the documents in advance [19].

N. Mihara et al [19] have made the transition from traditional paper documents to electronic ones by generating a PDF file of the medical document, using a virtual printer, and appending it to an XML file containing the meta-information of the document in the format of a CDA Release 2 header. This a good example of implementing CDA step-by-step, at first in a narrative manner.

The HL7 application methodology is often enhanced by using web services [20]. Release 2 of HL7 CDA is being used inside service oriented architectures to facilitate integration between healthcare facilities, as is, for example, the one developed by Medinfo researchers that connects medical records from the Ligurian HIV Network to the Biobanca database [21].

The emergence of the CDA standard was followed by the development and publishing of several implementation guides that target specific types of CDA documents. Among them is the The Continuity of Care Document (CCD), a joint project by HL7 and ASTM International, which implements the clinical requirements specified in the Continuity of Care Record (CCR) using the CDA architecture [10].

The Windows Communication Foundation (WCF) framework has been used to build service-oriented applications which transfer HL7 CDA documents as XML encapsulated in the body of SOAP messages [5].

HL7's decision to adopt a RIM-based methodology is questioned by some authors which consider that after ten years of effort, "the promised benefits of interoperability which were to have been engendered by its use remain elusive" [22]. The classes which compose the RIM, having associated "a rich stock of attributes derived from the specific domain of US hospital billing practices" [22] are considered counterproductive when applied to new domains as one must constantly delete existing attributes and replace them by others [22], a methodology considered to break the central rules of object-oriented software design [23].

By storing information about *observations* and not information on various medical issues like fractures or infections, the RIM makes it difficult to infer that two different messages actually refer to the same fracture or infection [22].

The requirement to model the clinical content of CDA entries according to the RIM leads to most of CDA's primary shortcomings according to [24]:

- Lack of cohesion between the *text* and the *coded entry*. The coded entries meant for computers to process are also optional and only considered "nice-to-have" [24]. No mechanism is provided for adequately tying the narrative part to coded entries or transforming one into the other.

- Inconsistent representation of clinical concepts
- Lack of specialization semantics
- Lack of tools to assist the design, evaluation and standardization of detailed clinical models by clinicians.

The RIM is also criticized for not providing associated tooling that can facilitate easily designing CDA documents, requiring years until one is "extensively versed in much of the HL7 RIM's architecture, language and undocumented usage practices" before being able to design, understand or communicate CDA or construct messages that can accomplish actions [24, 8].

Although HL7 CDA enables the exchange of clinical information from a data perspective, additional efforts are required to "enable a more flexible interaction with EHR systems" [25]. The functionality of current systems, from the users' point of view, is often limited to performing basic queries on existing documents [25].

M. Schweitzer et al [25] have performed a study using direct observations to record all common activities executed by clinical personnel during routine diabetes consultations. Their observations show that some medics still used printouts and recorded notes on paper while all nurses were observed to document electronically. The most used IT action in the case of medics was information retrieval and, in the case of nurses, documenting data.

In practice, according to [26] "HL7 interfaces always end up being a thin wrapper around the structure of the database of the application which feeds them" . Even though standards are used, there are areas where it is unclear how to structure the data, thus leading to pain [26]. According to [26], "the best any vendor can ever do is provide a stream of messages with fields that map adequately to most of the data from their application".

B. Smith [26] finds using a single data model for serving all purposes wrong and suggests using small, simple models for various problems.

2.2. Medical Code Systems

2.2.1. Logical Observation Identifiers Names and Codes

When it comes to terminology standards, some authors argue that "the existence of the perfectly coded biomedical data set may be more of a theoretical concept to aim for rather than an obtainable goal" [27].

Developed by the Regenstrief Institute, the LOINC project aims to "create universal identifiers (names and codes) used in the context of existing ASTM E1238, HL7, CEN TC251, and DICOM observation report messages employed in the various sub-domains of healthcare informatics such as Clinical Laboratory Information Management Systems and Computer-Based Patient Record Systems" [28].

From the beginning, LOINC has been developed as an open standard and made available worldwide through the LOINC website[1] [29]. LOINC codes are available for

[1] http://loinc.org

commercial use without charge [10]. The current available version is 2.54 released on December 21st 2015.

Initially released in April 1996, LOINC codes have been greeted enthusiastically and managed to gather 27 000 users in 158 different countries: clinical institutions to health systems, government agencies, international e-Health projects, IT vendors, research projects, etc [28]. LOINC has been even adopted as a national standard in countries like: Australia, Brazil, Canada, Cyprus, Estonia, France, Germany, Mexico, Mongolia, the Netherlands, Rwanda, Thailand, Turkey, and the United States; and is also seeing large-scale use in Hong Kong, Italy, the Philippines, Spain, Singapore, and Korea [28].

HL7 messages can include LOINC codes which are identified as the code system LN [28]. In the context of messaging standards, using LOINC codes enables the exchange of clinical laboratory data between heterogeneous digital environments [28]. LOINC codes are also frequently used in CDA documents [10].

The LOINC codes were developed in English and, over time, the received contributions for the following linguistic variants: Chinese (China), Dutch (Netherlands), Estonian (Estonia), French (Belgium, Canada, France, Switzerland), German (Austria, Germany, Switzerland), Greek (Greece), Italian (Italy, Switzerland), Korean (Korea, Republic of), Portuguese (Brazil), Russian (Russian Federation), Spanish (Argentina, Mexico, Spain) and Turkish (Turkey) [30].

The LOINC database comes in the form of a tabular file and contains standard text names and codes. LOINC names are *fully specified*, containing all the information needed to map a local test name to one of the fully specified names when a person wants to map a local test dictionary to LOINC codes [28]. The current version of the database contains 78 959 rows. Table 2.1 shows a section of the database.

The LOINC database's laboratory portion's scope includes all observations reported by clinical laboratories including: blood bank, chemistry, cytology, fertility, hematology, microbiology, serology and surgical pathology [28].

As can be seen in table 2.1, each LOINC entry has a unique, permanent code assigned to it (LOINC_NUM). This code is used to identify test results in electronic reports and is composed of two numeric values, the last one being a modulo 10 check digit [28].

As shown in figure 2.2, the LOINC concept names are composed of six core name parts which provide information necessary to map local test names to LOINC: [31]

1. The component or analyte
2. A property indicating various kinds of quantities
3. The time aspect of the measurement
4. The system or specimen
5. The type of scale
6. The method of performing the test.

Besides the LOINC database, the Regenstrief institute also freely provides a software program called RELMA® (the REgenstrief LOINC Mapping Assistant) which

LOINC_NUM	*COMPONENT*	*P*[a]	*T*[b]	*SYSTEM*	*S*[c]	*LONG_COMMON_NAME*
1000-9	DBG Ab	Pr	Pt	Ser/ Plas^BPU	Ord	DBG Ab [Presence] in Serum or Plasma from Blood product unit
10322-6	Potassium intake	SRat	24H	^Patient	Qn	Potassium intake 24 hour
16415-2	Aspergillus flavus Ab	ACnc	Pt	CSF	Ord	Aspergillus flavus Ab [Presence] in Cerebral spinal fluid
1896-0	Argininosuccinate	MCnc	Pt	Ser/Plas	Qn	Argininosuccinate [Mass/volume] in Serum or Plasma
20645-8	Histidine	SCnc	Pt	Ser/Plas	Qn	Histidine [Moles/volume] in Serum or Plasma
35006-6	Plasma cells	NCnc	Pt	Body fld	Qn	Plasma cells [#/volume] in Body fluid
47939-4	Sulfocysteine	SCnc	Pt	Plas	Qn	Sulfocysteine [Moles/volume] in Plasma
51754-0	Histoplasma capsulatum Ag	MCnc	Pt	CSF	Qn	Histoplasma capsulatum Ag [Mass/volume] in Cerebral spinal fluid by Immunoassay
55557-3	Nicotine	SCnc	Pt	Ser/Plas	Qn	Nicotine [Moles/volume] in Serum or Plasma
57794-0	Newborn screening test results panel	-	Pt	Bld.dot	-	Newborn screening test results panel - Dried blood spot
59574-4	Body mass index	Prctl	Pt	^Patient	Qn	Body mass index (BMI) [Percentile]
66995-2	How many times in the last Y have you been drunk in a public place	NRat	Pt	^Patient	Qn	How many times in the last year have you been drunk in a public place [PhenX]

Table 2.1: Example entries in the LOINC table. Each entry contains detailed information such as the time aspect, system and scale type.

[a] PROPERTY
[b] TIME_ASPCT
[c] SCALE_TYP

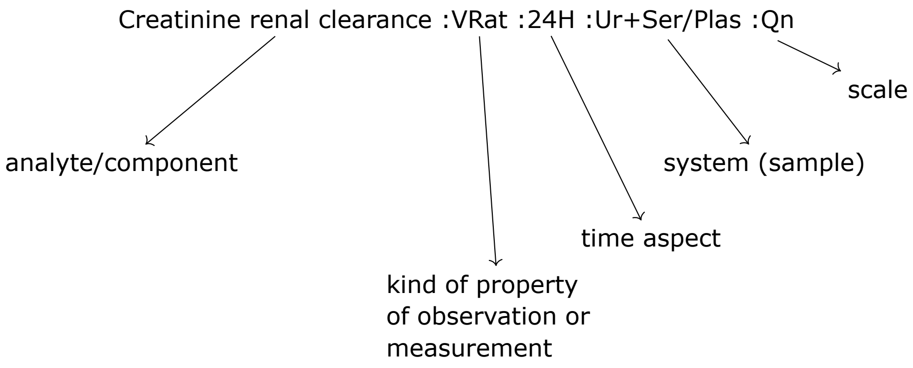
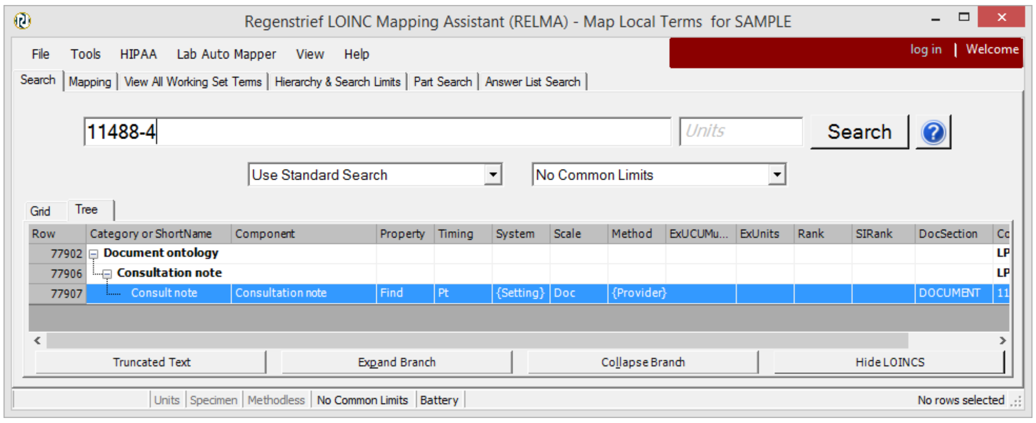

Creatinine renal clearance :VRat :24H :Ur+Ser/Plas :Qn
analyte/component
kind of property of observation or measurement
time aspect
system (sample)
scale

Fig. 2.2: Major parts of a LOINC concept name [28]

facilitates "searching the LOINC database, viewing detailed accessory content about the terms, and mapping local terminology to LOINC terms" [29].



11488-4

Use Standard Search

No Common Limits

Row	Category or ShortName	Component	Property	Timing	System	Scale	Method	ExUCUMu...	ExUnits	Rank	SIRank	DocSection	Cc
77902	Document ontology												LP
77906	Consultation note												LP
77907	Consult note	Consultation note	Find	Pt	{Setting}	Doc	{Provider}					DOCUMENT	11

No Common Limits

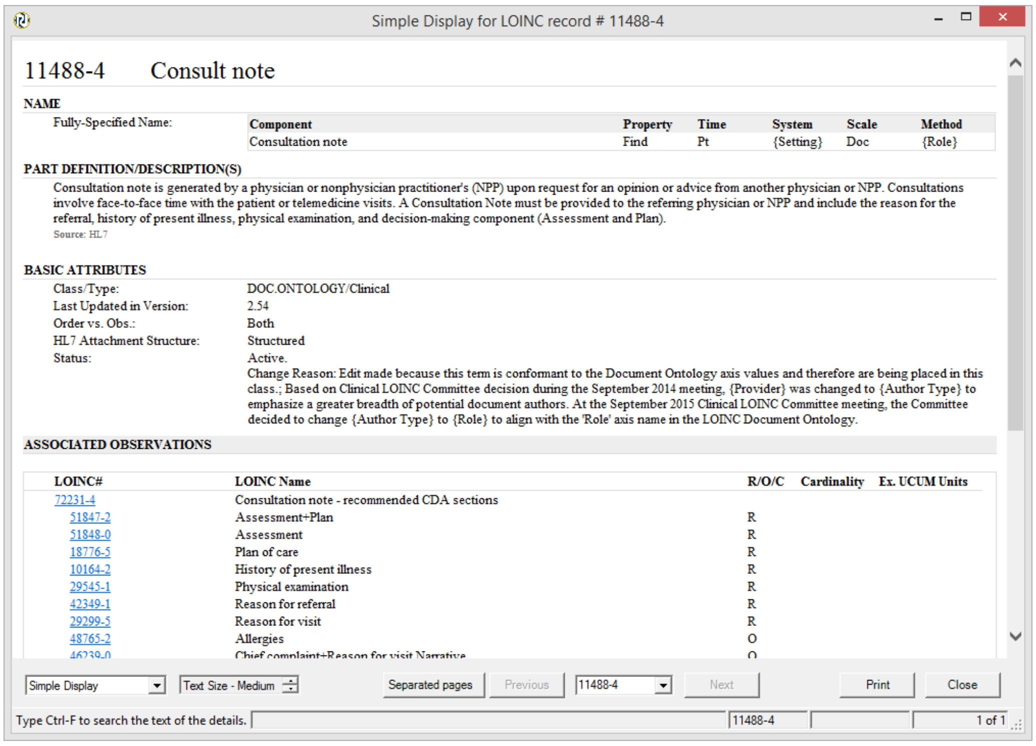
No rows selected

Fig. 2.3: Searching a LOINC code using the Regenstrief LOINC Mapping Assistant (RELMA)

Research around LOINC is focused around mapping local terms. H. Kim et al [31] found mapping using RELMA challenging due to two major reasons: some local name variants were not recognized by RELMA and tests showed incomplete information. A study performed by Ch. Zunner [32] et al shows that they managed to map 1660 interface terms (∼77%) to LOINC out of 2148 which were processed, resulting in 1208 individual LOINC codes. The time necessary was 4 working days with a wide variation between the time needed for simple terms (>200 terms mapped/hour) and the time needed for terms requiring further investigation (<10 terms/hour) [32].

2.2.2. International Classification for Diseases

The International Statistical Classification of Diseases and Related Health Problems (ICD) is an internationally used medical classification list from the World Health Organization. Its current version, 10, has been in use since 1994 [33].



11488-4 Consult note

NAME

Fully-Specified Name:

Component	Property	Time	System	Scale	Method
Consultation note	Find	Pt	{Setting}	Doc	{Role}

PART DEFINITION/DESCRIPTION(S)

Consultation note is generated by a physician or nonphysician practitioner's (NPP) upon request for an opinion or advice from another physician or NPP. Consultations involve face-to-face time with the patient or telemedicine visits. A Consultation Note must be provided to the referring physician or NPP and include the reason for the referral, history of present illness, physical examination, and decision-making component (Assessment and Plan).
Source: HL7

BASIC ATTRIBUTES

Class/Type:	DOC.ONTOLOGY/Clinical
Last Updated in Version:	2.54
Order vs. Obs.:	Both
HL7 Attachment Structure:	Structured
Status:	Active. Change Reason: Edit made because this term is conformant to the Document Ontology axis values and therefore are being placed in this class.; Based on Clinical LOINC Committee decision during the September 2014 meeting, {Provider} was changed to {Author Type} to emphasize a greater breadth of potential document authors. At the September 2015 Clinical LOINC Committee meeting, the Committee decided to change {Author Type} to {Role} to align with the 'Role' axis name in the LOINC Document Ontology.

ASSOCIATED OBSERVATIONS

LOINC#	LOINC Name	R/O/C	Cardinality	Ex. UCUM Units
72231-4	Consultation note - recommended CDA sections			
51847-2	Assessment+Plan	R		
51848-0	Assessment	R		
18776-5	Plan of care	R		
10164-2	History of present illness	R		
29545-1	Physical examination	R		
42349-1	Reason for referral	R		
29299-5	Reason for visit	R		
48765-2	Allergies	O		
46239-0	Chief complaint+Reason for visit Narrative	O		

Fig. 2.4: Regenstrief LOINC Mapping Assistant (RELMA) showing details on item 11488-4

ICD version 10 codes are organized in a tree structure, becoming more and more specific when navigating deeper inside the hierarchy. The actual codes consist of a letter prefix followed by digits. Each code is also accompanied by a description and optional notes.

At the root of the hierarchy are multiple chapters, as shown in the list below, an extract from *ICD-10-CM Tabular List of Diseases and Injuries* [34]:

1. Certain infectious and parasitic diseases (A00-B99)
2. Neoplasms (C00-D49)
3. Diseases of the blood and blood-forming organs and certain disorders involving the immune mechanism (D50-D89)
4. Endocrine, nutritional and metabolic diseases (E00-E89)
5. Mental, Behavioral and Neurodevelopmental disorders (F01-F99)
6. Diseases of the nervous system (G00-G99)
7. Diseases of the eye and adnexa (H00-H59)
8. Diseases of the ear and mastoid process (H60-H95)

9. Diseases of the circulatory system (I00-I99)
10. Diseases of the respiratory system (J00-J99)
11. Diseases of the digestive system (K00-K95)
12. Diseases of the skin and subcutaneous tissue (L00-L99)
13. Diseases of the musculoskeletal system and connective tissue (M00-M99)
14. Diseases of the genitourinary system (N00-N99)
15. Pregnancy, childbirth and the puerperium (O00-O9A)
16. Certain conditions originating in the perinatal period (P00-P96)
17. Congenital malformations, deformations and chromosomal abnormalities (Q00-Q99)
18. Symptoms, signs and abnormal clinical and laboratory findings, not elsewhere classified (R00-R99)
19. Injury, poisoning and certain other consequences of external causes (S00-T88)
20. External causes of morbidity (V00-Y99)
21. Factors influencing health status and contact with health services (Z00-Z99)

For each chapter, the hierarchy branches further, providing more specific codes. The first part of the first chapter, *Intestinal infectious diseases (A00-A09)* starts with the following codes: [34]

- A00 – Cholera
  - A00.0 – Cholera due to Vibrio cholerae 01, biovar cholerae
  - A00.1 – Cholera due to Vibrio cholerae 01, biovar eltor
  - A00.9 – Cholera, unspecified
- A01 – Typhoid and paratyphoid fevers
  - A01.0 – Typhoid fever
    - A01.00 – Typhoid fever, unspecified
    - A01.01 – Typhoid meningitis
    - A01.02 – Typhoid fever with heart involvement
    - ...
  - A01.1 – Paratyphoid fever A
  - A01.2 – Paratyphoid fever B
  - ...
- ...

A very common task for clinicians is to query a code for the diagnostics they have come up with. Due to time constraints, browsing through the codes is not normally an option, so having a readily available search functionality which can provide the correct answer in a quick and easy fashion is important [35].

As ICD codes classify diseases in a comprehensive way, there have been found uses for them as a foundation in building a search system for identifying doctors by their specialty in large health facilities [36].

ICD-10 is also being used in Romania. The Australian version, ICD-10-AM, has been translated into Romanian and is available at https://www.drg.ro/DocDRG/download.php?fi=2.

2.3. The openEHR Specifications

Established in 2000 in the UK, the openEHR Foundation[2] publishes e-health domain models (archetypes), educational material, open source software and specifications around a platform architecture [37]. Although it is not a standards body itself, openEHR is dedicated to work with standards organizations and has influenced the revised European standard CEN 13606 [17].

Specifications provided by openEHR impact: [37]

- Clinical (EHR) and demographic data (the openEHR Information Models)
- Clinical (EHR) and demographic content models, and connection points to terminology (the openEHR archetypes and templates)
- Guidelines
- Portable Queries
- Key Services and APIs.

As can be seen in figure 2.5, at the core of openEHR is providing a layer which houses models built by domain experts. This makes using terminology much easier and has given rise to an international repository of these models, also called *archetypes* [37]. Archetypes and templates are models of semantics, independent of any document or messaging standards [37]. The goal is that, once the modeling is established, other artifacts such as specific UI forms or source code can be generated.

The approach taken by openEHR is to a have a stable reference model which can be implemented in software and to also provide a flexible domain model expressed as *archetypes* and *templates* [38]. Archetypes are there to give semantic meaning to objects persisted via the reference model and are intended to be created and edited by domain experts and not by programmers [38]. Preventing structural or business rules changes in the reference model and directing them to the archetypes ensures that the persistence mechanism doesn't need to change [38].

An openEHR electronic health record is structured based on the following model: [39]

- *EHR* – the root object – a globally unique EHR identifier

[2]http://www.openehr.org/

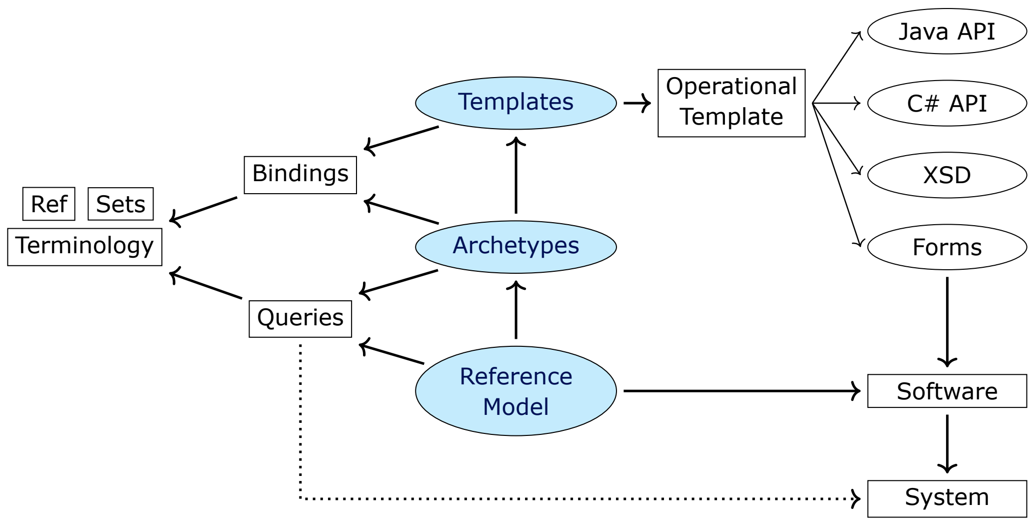


Fig. 2.5: The openEHR paradigm (adapted after [37]). At the center lies the *Reference Model* which is used to form *Archetypes* that are further specialized into *Templates*.

- *EHR_access (versioned)* – an object describing the access control information for the record
- *EHR_status (versioned)* – various control and status information
- *Directory (versioned)* – an optional hierarchy of folders used to organize *Compositions*
- *Compositions* – contains all the administrative and clinical content of the record.

The openEHR specifications "aim at providing a way of implementing more flexible EHRs, by gracefully embracing change" [40]. Figure 2.6 presents a simplified view of the multilevel model used by openEHR. The reference model (RM) contains commonly occurring structures, building blocks which are used to create archetypes that express domain concepts. Combining archetypes leads to templates which are further used to generate messages or UI forms [40].

The official documentation defines an *archetype* as "a computable expression of a domain content model in the form of structured constraint statements, based on a reference (information) model" and a *template* as "a directly locally usable definition which composes archetypes into a larger structures often corresponding to a screen form, document, report or message" [41]. As archetypes are generally broad models, templates play an important role in narrowing the choice of archetypes for specific purposes.

Archetypes are expressed using a formal language called the Archetype Definition Language (ADL) [42]. ADL documents are parsed into a network of objects defined by a formal, abstract object model – the openEHR Archetype Object Model (AOM), which can be further represented in a number of ways, including XML [42].

Am important strength of openEHR archetypes is the ability to store parts of a component. The listing below, taken from an archetype available at

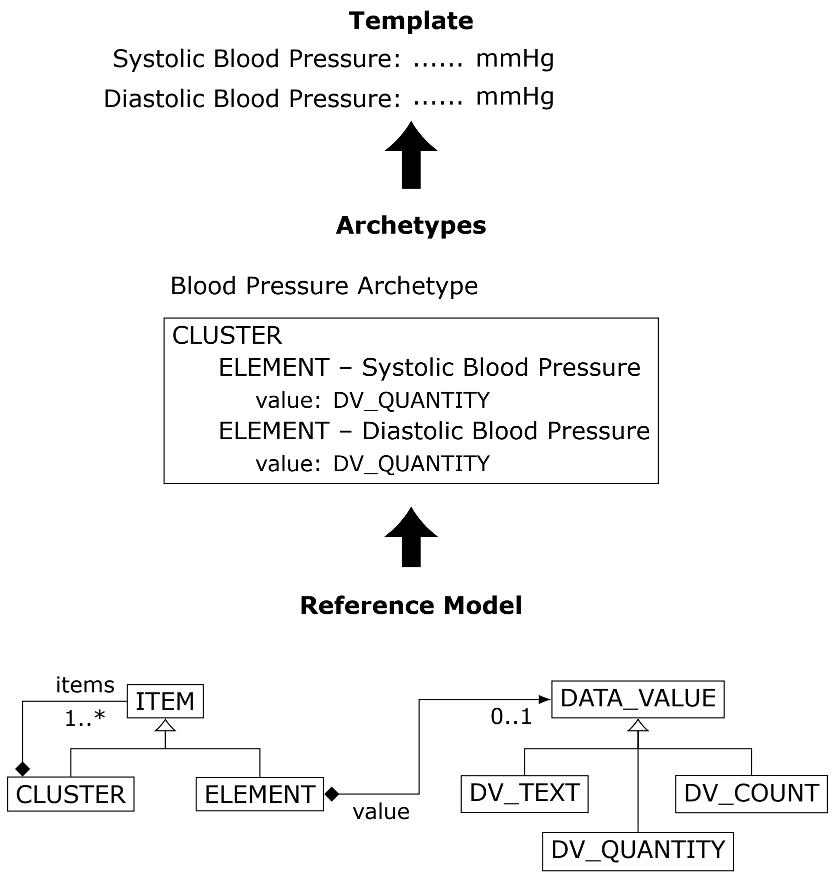


Fig. 2.6: The openEHR multilevel model (adapted from [40]). The figure shows how archetypes are built using primitives from the reference model. Archetypes are further used to create templates which detail which inputs are to be presented to users.

https://github.com/openEHR/adl-archetypes.git, exemplifies how *Blood Pressure* is composed of (among others) a *systolic* and a *diastolic* value.

```
OBSERVATION[at0000] matches { -- Blood Pressure
   data matches {
      HISTORY[at0001] matches { -- history
         events cardinality matches {1..*; unordered} matches {
            EVENT[at0006] occurrences matches {0..*} matches {  -- Any event
               data matches {
                  ITEM_TREE[at0003] matches { -- blood pressure
                     items cardinality matches {0..*; unordered} matches {
                        ELEMENT[at0004] occurrences matches {0..1} matches {
                        -- Systolic
                           value matches {
                              C_DV_QUANTITY <
                                 property = <[openehr::125]>
                                 list = <
                                    ["1"] = <
```

```
                                units = <”mm[Hg]”>
                                magnitude = <|0.0..<1000.0|>
                                precision = <|0|>
                            >
                        >
                    >
                }
            }
            ELEMENT[at0005] occurrences matches {0..1} matches {
            -- Diastolic
                value matches {
                    C_DV_QUANTITY <
                        property = <[openehr::125]>
                        list = <
                            [”1”] = <
                                units = <”mm[Hg]”>
                                magnitude = <|0.0..<1000.0|>
                                precision = <|0|>
                            >
                        >
                    >
                }
            }
```

The listing above presented the systolic blood pressure using a local term, [at0004]. An archetype contains a *term_definitions* section which defines all the local terms in various languages:

```
term_definitions = <
    [”en”] = <
        items = <
            [”at0004”] = <
                text = <”Systolic”>
                description = <”Peak systemic arterial blood pressure -
                    measured in systolic or contraction phase of the heart cycle.”>
            >
            [”at0005”] = <
                text = <”Diastolic”>
                description = <”Minimum systemic arterial blood pressure -
                    measured in the diastolic or relaxation phase of the heart cycle.”>
            >
        >
    >
    [”de”] = <
        items = <
            [”at0004”] = <
                text = <”Systolisch”>
                description = <”Der höchste arterielle Blutdruck eines Zyklus -
                    gemessen in der systolischen oder Kontraktionsphase des Herzens.”>
            >
            [”at0005”] = <
                text = <”Diastolisch”>
                description = <”Der minimale systemische arterielle Blutdruck
                    eines Zyklus - gemessen in der diastolischen oder Entspannungsphase
                    des Herzens.”>
            >
        >
    >
>
```

Terminologies are also linked:

```
term_bindings = <
   ["SNOMED-CT"] = <
      items = <
         ["at0004"] = <[SNOMED-CT(2003)::163030003]>
         ["at0005"] = <[SNOMED-CT(2003)::163031004]>
      >
   >
>
```

In contrast, HL7 does not have a data type for storing a pair of component parts. This leads to the creation of an extra pair of *Observation* classes and *Act-Relationship* objects with the downside that these components gain so much importance that the parent *Observation* is discarded [24]. An example of this is the ASTM/HL7 Continuity of Care Document which contains two separate and uncorrelated *Observations* for systolic and diastolic blood pressure [24].

When it comes to persistence, openEHR does not specify or require a specific persistence method or strategy. As shown by [43], modeling a relational database for persisting EHR records according to openEHR would result in performance issues due to the large number of joins required by the complex tree structure of openEHR archetypes. Figure 2.7 demos such a query.

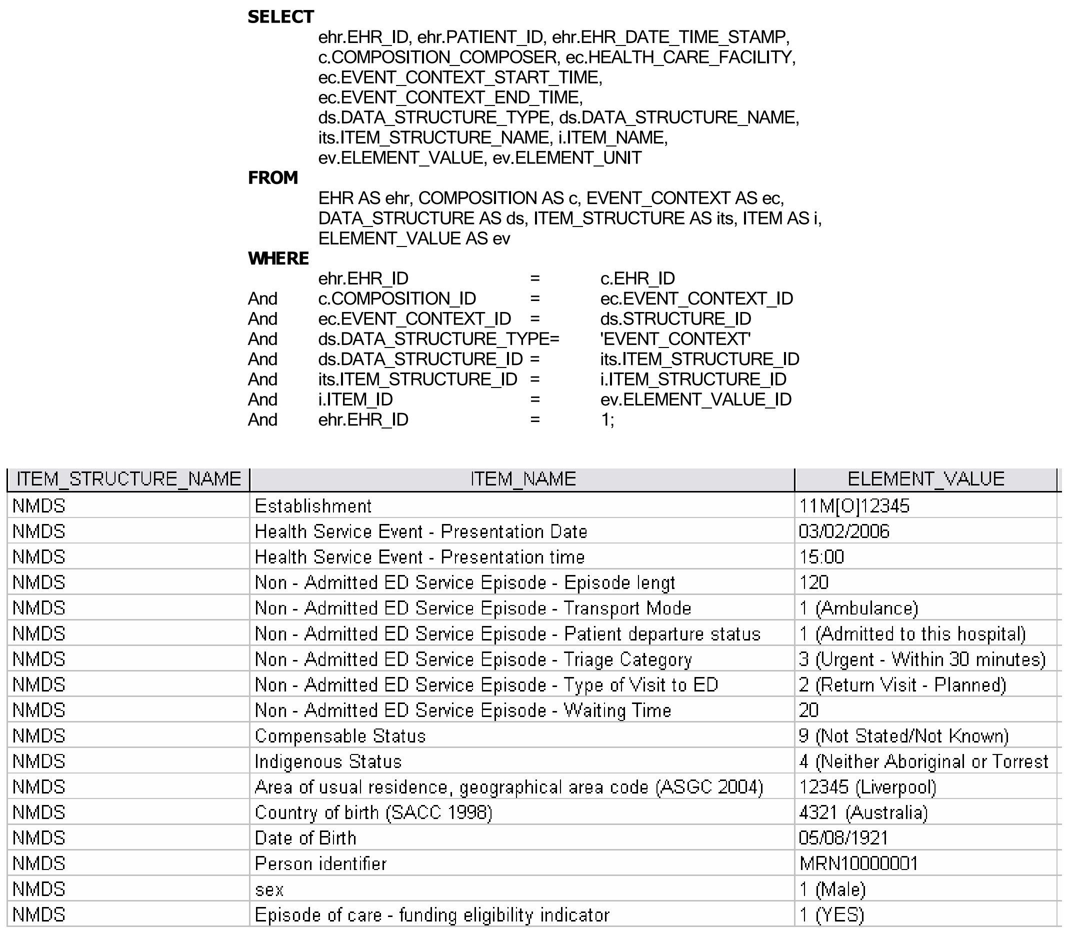
As a practical persistence solution for future-proof electronic health record systems, L. Wang et al propose a relational database schema based on the following mapping rules: [44]

- Each archetype is mapped to a table
- Each basic data item represented by the archetype basic data type is mapped
- The identification data item is constrained as the key column
- The query data item is constrained as an indexed column
- Archetype slots are mapped
- Collection data items are mapped according to the collection data structure
- Query data items are propagated in order to reduce the recursive level

The structure of the reference model allows generated EHR data to be serialized in several popular formats like XML or JSON. When it comes to storing EHR data in a relational database, performing a pure object-relational mapping may not yield an efficient solution because the reference model contains a large set of classes which can easily form relatively deep tree hierarchies [38].

One solution is to use XML databases which are readily available. If using them in production systems, [38] identifies an important requirement in the fact that such databases must present good performance "not only when querying for data about an individual (clinical query) but also for data about a whole population (epidemiological query)" when performing research studies, an important secondary use of electronic health records.

A study has been performed by [38] on the viability of using XML databases for storing EHR data. They benchmark a relational database (MySQL version 5.5.24)



```
SELECT
         ehr.EHR_ID, ehr.PATIENT_ID, ehr.EHR_DATE_TIME_STAMP,
         c.COMPOSITION_COMPOSER, ec.HEALTH_CARE_FACILITY,
         ec.EVENT_CONTEXT_START_TIME,
         ec.EVENT_CONTEXT_END_TIME,
         ds.DATA_STRUCTURE_TYPE, ds.DATA_STRUCTURE_NAME,
         its.ITEM_STRUCTURE_NAME, i.ITEM_NAME,
         ev.ELEMENT_VALUE, ev.ELEMENT_UNIT
FROM
         EHR AS ehr, COMPOSITION AS c, EVENT_CONTEXT AS ec,
         DATA_STRUCTURE AS ds, ITEM_STRUCTURE AS its, ITEM AS i,
         ELEMENT_VALUE AS ev
WHERE
         ehr.EHR_ID                =     c.EHR_ID
And      c.COMPOSITION_ID          =     ec.EVENT_CONTEXT_ID
And      ec.EVENT_CONTEXT_ID   =     ds.STRUCTURE_ID
And      ds.DATA_STRUCTURE_TYPE=     'EVENT_CONTEXT'
And      ds.DATA_STRUCTURE_ID =     its.ITEM_STRUCTURE_ID
And      its.ITEM_STRUCTURE_ID  =     i.ITEM_STRUCTURE_ID
And      i.ITEM_ID                 =     ev.ELEMENT_VALUE_ID
And      ehr.EHR_ID                =     1;
```

ITEM_STRUCTURE_NAME	ITEM_NAME	ELEMENT_VALUE
NMDS	Establishment	11M[O]12345
NMDS	Health Service Event - Presentation Date	03/02/2006
NMDS	Health Service Event - Presentation time	15:00
NMDS	Non - Admitted ED Service Episode - Episode lengt	120
NMDS	Non - Admitted ED Service Episode - Transport Mode	1 (Ambulance)
NMDS	Non - Admitted ED Service Episode - Patient departure status	1 (Admitted to this hospital)
NMDS	Non - Admitted ED Service Episode - Triage Category	3 (Urgent - Within 30 minutes)
NMDS	Non - Admitted ED Service Episode - Type of Visit to ED	2 (Return Visit - Planned)
NMDS	Non - Admitted ED Service Episode - Waiting Time	20
NMDS	Compensable Status	9 (Not Stated/Not Known)
NMDS	Indigenous Status	4 (Neither Aboriginal or Torrest
NMDS	Area of usual residence, geographical area code (ASGC 2004)	12345 (Liverpool)
NMDS	Country of birth (SACC 1998)	4321 (Australia)
NMDS	Date of Birth	05/08/1921
NMDS	Person identifier	MRN10000001
NMDS	sex	1 (Male)
NMDS	Episode of care - funding eligibility indicator	1 (YES)

Fig. 2.7: Complexity of a openEHR query against a SQL store [43]. The data is normalized throughout many tables resulting in a lot of table join operations being required to fetch the requested records.

against the following XML databases: eXist (version 1.4.2), BaseX (version 7.3), Sedna (version 3.5), and Berkeley DB XML (version 11g). When it comes to the necessary storage space, they have observed large discrepancies between the XML databases which require between 38 and 164 times more storage space than the relational database. The response times when performing epidemiological queries on XML databases also leave much to be desired when compared to the relational database [38].

An explanation on the poor performance of XML databases might lie in the way openEHR archetypes are designed. They usually contain many attributes with the same value thus making XML text and attribute indexes point to a huge number of entries in the database, rendering them inefficient [38].

A later study performed by [40] also includes a benchmark of a modern, distributed NoSQL database based on the MapReduce approach, Couchbase is a document database and provides native support for the JSON format. The results of the benchmark show an overall better response time in the case of Couchbase compared to MySQL, however with the note that Couchbase requires indexing for each different query [40]. The indexing performance can however be increased proportionally by introducing more nodes in the cluster.

E. Sundvall et al [45, 46] have investigated ways of creating REST services as interfaces to openEHR implementations. They use XML for passing messages and an XML databases with support for XQuery for storage. The REST architecture benefits from the features offered by HTTP, making it easy to switch between XML or JSON content by just changing the mime type of the request.

The system presented by [45] supports queries in two ways:

- By constructing them with the help of URI paths, a convenient method thanks to the documents' hierarchical structure. This method is detailed in figure 2.8.
- By providing AQL queries which the system then translates to XQuery. As example AQL query is shown in figure 2.9.

ehrID	ehr://1234567/
Compositions (versioned)	87284370-2D4B-4e3d-A3F3-F303D2F4F34B @latest_trunk_version/
Sections	content[openEHR-EHR-SECTION.vital_signs.v1]/
Entries	items[openEHR-EHR-OBSERVATION.heart_rate-pulse.v1]/
Data structures	data/events[at0006]/data/items[at0004]/value/
Values	magnitude

Fig. 2.8: Using URI paths to construct an openEHR query [45]. Each part is explained on the left.

T. Beale [47] suggests the following steps for a template-based integration of non-EHR systems into an openEHR backend, as shown in figure 2.10. Templates can also be used to other way round, to produce documents based on other medical standards.

- Building openEHR templates for each message

```
SELECT v/uid/value as composition_id,
 obs/data/origin/value as measurement_time,
 obs/data/events/data/items[at0004]/value/magnitude as systolic,
 obs/data/events/data/items[at0005]/value/magnitude as diastolic,
 c/context/setting/value as composition_setting,
 c/name/value as composition_title,
 c/context/start_time/value as composition_start_time
FROM Ehr [uid=$current_ehr_uid]
CONTAINS VERSION v
CONTAINS COMPOSITION c[openEHR-EHR-COMPOSITION.encounter.v1]
CONTAINS OBSERVATION obs[openEHR-EHR-
OBSERVATION.blood_pressure.v1]
WHERE obs/data/events/data/items[at0004]/value/magnitude > 185
ORDER BY c/context/start_time/value
```

Fig. 2.9: Example of a Archetype Query Language query [46]. The language is quite similar to SQL, with the added complexity if being able to reference elements of an archetype by means of a path and being able to check for the existence of archetypes inside records.

- Generating XML schemas from the template definition
- Populating XML documents from source data
- Transforming to standard openEHR XML
- Using the integration engine to do XML processing.

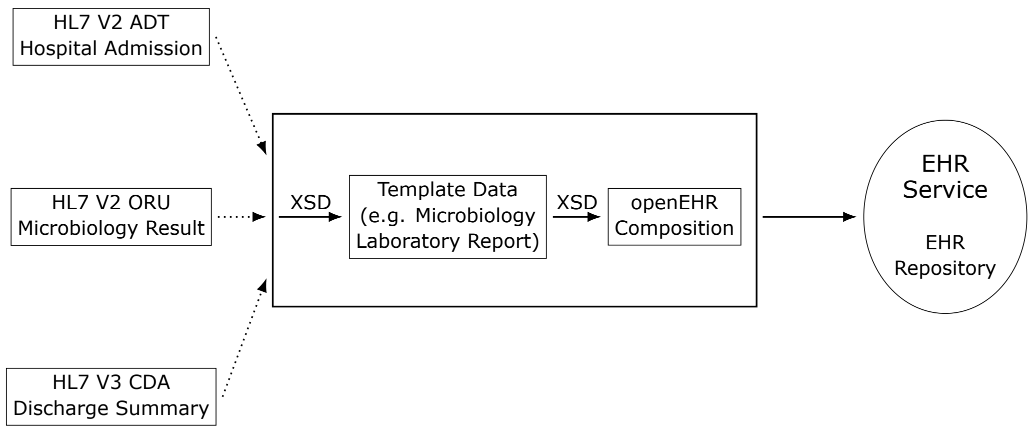


Fig. 2.10: openEHR Template-based Integration (adapted from [47]). This approach from literature makes use of XML technologies like schemas and transformations to integrate HL7 documents into an openEHR repository.

Similar to openEHR's dual model approach, HL7 has developed a specification for the electronic exchange of healthcare information called Fast Healthcare Interoperability Resources (FHIR) [48]. The main difference in the approach compared to openEHR archetypes is that archetypes are maximal datasets, expected to represent all clinical content, while the FHIR resources are meant to only contain the most commonly used clinical information, providing ways to be extended for specific use cases [48].

Conclusions

This chapter focused on providing a literature study concerning coding systems, methodologies, standards and terminologies from the field of medical informatics.

The chapter presented details about the Health Level Seven Clinical Document Architecture, a standard which defines the structure of electronic documents that store health related information, mostly for the purpose of document exchange. CDA documents use XML as a markup language for structuring documents and place a high emphasis on also storing the data in a human readable format. The study on CDA also reveals limitations such as the possibility that documents may not provide information in a structured manner targeted at being processed by a computer. Authors also question the reference information model on which CDA is based, considering it counter-productive and leading to inconsistent representations of clinical concepts.

When it comes to building electronic health record applications, the openEHR Foundation provides methodologies and specifications for aiding software development. Similar to CDA, openEHR is also based on a reference information model, which is then used to construct one of openEHR's distinguishing features, models, called *archetypes*, built by domain experts. Repositories of such models have been created, containing important details, including links to medical terminology, making them attractive for use in electronic health record applications. One of the limitations of the openEHR specifications is that they do not provide any solution for persisting the data, a requirement that each EHR implementer needs to deal with.

In regards to medical coding systems, this chapter presents information on the Logical Observation Identifiers Names and Codes and the International Classification for Diseases. LOINC codes are mostly used for clinical laboratory data and provide challenges when requiring new mappings, especially if they are to be used in a language for which no official translation is currently available. ICD codes, organized in a tree hierarchy, benefit from translations in many more languages; however they still pose challenges when it comes to accurately choosing the correct code.

3. DESIGNING A SOLUTION FOR THE STRUCTURED COLLECTION OF MEDICAL DATA

This chapter sets out to analyze the implications of building an electronic health record application. Building the application follows the principle of creating only what is strictly necessary for achieving the current goals. A prototype application that is built in this chapter serves as a base for the next chapters that deal with integrating medical standards and interoperability.

The first section, *Identifying the Business Domain* presents an analysis of the targeted business domain. The goal is to investigate the requirements needed when creating an electronic health record application and what influences these requirements. For the purpose of building a prototype application, the scope of the medical documents is restricted to medical forms that register information regarding patients with chronic illness.

Section *Rapid Prototyping Using XML* takes the requirements previously identified and uses hand crafted XML documents to rapidly prototype the structure of domain classes capable of dealing with a handful of input forms regarding chronically ill patients. In contrast to some legacy applications which simply create unrelated create-read-update-delete (CRUD) forms for whatever the daily requirement is, the goal of the approach is to store the information regarding what inputs are present in what forms in a reusable manner. This is built around the observation that medical forms have a tendency to change or be later added and thus provides flexibility by allowing a simpler manipulation of forms by users that are not developers.

With this approach in place, *Designing Business Domain Classes* goes into the details of building domain classes. While being kept flexible for future expansions, these classes are still mostly focused on the current requirements. In contrast with approaches such as building openEHR systems based on archetypes and templates, this approach is kept simple for developers and does not require them to have expert knowledge on complicated systems such as openEHR. As will be shown in a later chapter, this architecture is flexible enough to be integrated with openEHR artifacts at a latter stage.

When it comes to persisting data, this chapter discusses the implications of using a traditional relational database management system but also take the possibility of using a modern NoSQL document database into account.

3.1. Identifying the Business Domain

Medical information pertaining to a patient is organized in the so called patient records which account for the health and disease after the patient has sought medical help [1].

Traditionally recorded on paper, early patient records had numerous disadvantages [1]:

- Existing in only one copy, the records are only available at one place at a time
- Their free-text content may be illegible, incomplete and/or ambiguous leading to potential errors when interpreting or transcribing for scientific analysis purposes
- Active advice, reminders or warnings cannot be implemented while the various notes are only stored on paper.

All of the above argument the need of finding ways of using computer systems for storing patient records, a topic under research for the past decades [1]. It is important to distinguish between health information only used for administrative purposes, such as health insurance and clinically meaningful applications [49].

Much of the new data that is being introduced into the system comes from medical observations of the patient's condition along with information measured using various medical equipment. One of the challenges faced by general practitioners as well as specialized medics, in Romania, is how to collect all this information into a digital system as easy and as efficiently as possible. Improvements in this area can lead to advances on the use of information and communication technology by citizens on communication with their general practitioners as implemented in some European countries [50].

Studies have identified two major strategies for collecting patient data: [1]

- *Natural Language Processing* – The computer system inputs free text and processes it to extract meaningful information,
- *Direct Entry* – The system instructs the medic how to enter the data in a structured fashion.

This chapter investigates a way of designing a highly flexible software system aimed at collecting medical data in a structured way. The medics interact with the system via a web interface which allows them to easily manage information regarding the medical condition of their patients (at any time and place [51]). For demonstration purposes, the scope of the medical information will be limited to the monitoring of patients with chronic illnesses.

Information collected while investigating and monitoring patients with a chronic condition is often grouped into various forms, organized by specific conditions. Medics investigating such patients need to fill in a lot of information present in such forms. To make matters even more complicated, the information requested by each form differs based on whether it is the first time the form is filled for a specific patient, or whether it is a recurring entry.

From the software point of view, presenting a user with a web page that requests the input of data and storing that data in a database is a use case common to virtually all web applications. What makes this case different however is the high diversity of the medical data. Hard-coding an HTML input form might be a solution for small inputs such as a log-in form or even a contact form, but requiring the intervention of a software developer to update each form quickly becomes inflexible when it comes to medical data inputs which need constant updates.

These high frequency changes caused by the need to support more and more types of inputs (e.g. support the addition of information concerning patients with a different chronic condition than patients already in the system) or by changes in regulations that require medics to obtain and store additional data when it comes to specific conditions, put additional strain of the way the data is stored in databases and on the way the inputs are organized.

Because medical information about a patient (or even between various patients suffering from the same condition) is interconnected in a way that can provide very useful statistical information, is it important to store the information in a way that facilitates automatic processing and analytics: e.g. numerical inputs, stored appropriately in a database, alongside their measurement units can be easily processed automatically to extract information as opposed to storing a scanned image of a handwritten note which can only be manually interpreted when viewed by a medic. This goal aligns with the overall one of reducing medical errors and the rising costs of delivering health care [52].

As such, an application that can handle these requirements basically becomes a content management system, but one involving two layers:

1. It stores medical data concerning various patients, collected over time
2. It stores meta-data that describes the structure of the medical data, and, more importantly, the actual inputs that each medic must provide when filling in a specific form related to a specific patient.

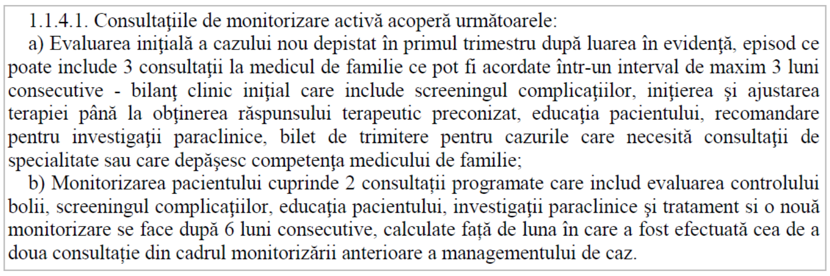
This two layer approach provides the system with the required flexibility, allowing the continuous update of the system to be handled by administrators with medical background (who better understand new requirements as they appear) instead of an IT background. The complexity of the system also increases, bringing along technical challenges.

A first step in designing a web solution for medical structured data collection is to correctly identify the business domain. This allows building a solution that can also semantically interpret the data it is processing instead of simply storing opaque input.

3.1.1. Analyzing Medical Input Forms

In the case of monitoring and consulting patients with chronic diseases, the healthcare legislation governs which steps need to be followed by medics in Romania [53]. Among them are:

- Which medical conditions are to be monitored
- How often the patient needs to be consulted
- What information needs to be gathered on each initial/recurring consultation
- Classification of risks
- Details concerning medical insurance

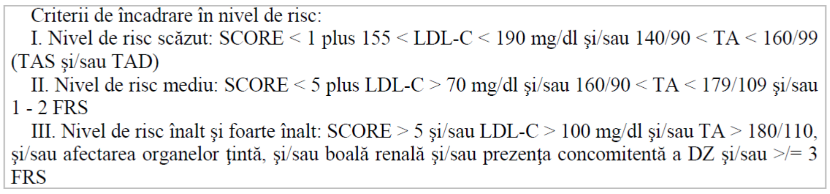


1.1.4.1. Consultaţiile de monitorizare activă acoperă următoarele:

a) Evaluarea iniţială a cazului nou depistat în primul trimestru după luarea în evidenţă, episod ce poate include 3 consultaţii la medicul de familie ce pot fi acordate într-un interval de maxim 3 luni consecutive - bilanţ clinic iniţial care include screeningul complicaţiilor, iniţierea şi ajustarea terapiei până la obţinerea răspunsului terapeutic preconizat, educaţia pacientului, recomandare pentru investigaţii paraclinice, bilet de trimitere pentru cazurile care necesită consultaţii de specialitate sau care depăşesc competenţa medicului de familie;

b) Monitorizarea pacientului cuprinde 2 consultaţii programate care includ evaluarea controlului bolii, screeningul complicaţiilor, educaţia pacientului, investigaţii paraclinice şi tratament si o nouă monitorizare se face după 6 luni consecutive, calculate faţă de luna în care a fost efectuată cea de a doua consultaţie din cadrul monitorizării anterioare a managementului de caz.

Fig. 3.1: Legislation detailing organizational aspects of patient monitoring [53]



Criterii de încadrare în nivel de risc:

I. Nivel de risc scăzut: SCORE < 1 plus 155 < LDL-C < 190 mg/dl şi/sau 140/90 < TA < 160/99 (TAS şi/sau TAD)

II. Nivel de risc mediu: SCORE < 5 plus LDL-C > 70 mg/dl şi/sau 160/90 < TA < 179/109 şi/sau 1 - 2 FRS

III. Nivel de risc înalt şi foarte înalt: SCORE > 5 şi/sau LDL-C > 100 mg/dl şi/sau TA > 180/110, şi/sau afectarea organelor ţintă, şi/sau boală renală şi/sau prezenţa concomitentă a DZ şi/sau >/= 3 FRS

Fig. 3.2: Legislation detailing the classification of medical risks [53]

Figures 3.1 and 3.2 exemplify such pieces of legislation detailing the coverage of the initial and active monitoring procedures and also the exact numeric values for classifying risk.

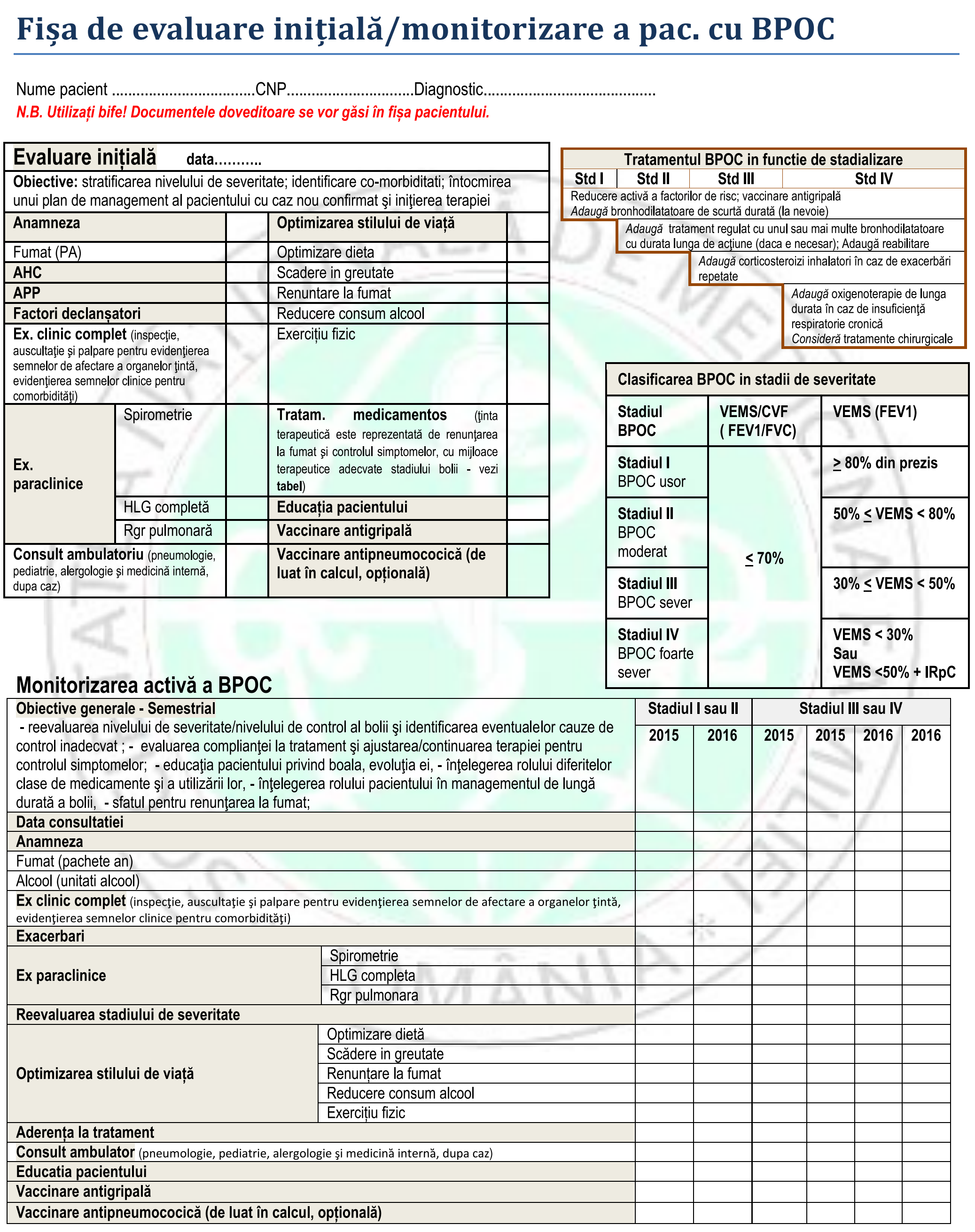
Web sites targeting healthcare professionals present detailed medical input forms for chronic diseases, as shown in figure 3.3 which presents a form for the initial and active monitoring of patients with chronic obstructive pulmonary disease.

3.1.2. Identifying Types of Input

At first glance, all of the medical input forms request textual input. However, a deeper analysis reveals that some fields only accept numeric values, boolean (*yes*/*no*) values or can even be computed automatically based on other inputs. Separating these types of input from the usual textual ones is important as it can lead to better data storage, better semantics and gives way to implement input field relations and analytics.

The analysis of various sources of medical input forms shows a high occurrence of free text input fields. These fields range from asking for the input of small pieces of information (e.g. *the state of the patient's pupils*) to complete sentences or even phrases (e.g. *the patient's case history*) and might even be accompanied by detailed descriptions on exactly what details to provide. Textual inputs that ask for small pieces of information often present the following extra characteristics:

- The variety of submitted values is low. This allows for the creation of a set of predefined values which can accompany the input field as suggestions.
- A specific value is provided much more often than others. This allows for treating that value as a *default value* of the field.



Fișa de evaluare inițială/monitorizare a pac. cu BPOC

Nume pacient ....CNP..............................Diagnostic...

N.B. Utilizați bife! Documentele doveditoare se vor găsi în fișa pacientului.

Evaluare inițială **data...........**

Obiective: stratificarea nivelului de severitate; identificare co-morbiditati; întocmirea unui plan de management al pacientului cu caz nou confirmat şi iniţierea terapiei

Anamneza			**Optimizarea stilului de viață**	
Fumat (PA)			Optimizare dieta	
AHC			Scadere in greutate	
APP			Renuntare la fumat	
Factori declanșatori			Reducere consum alcool	
Ex. clinic complet (inspecţie, auscultaţie şi palpare pentru evidenţierea semnelor de afectare a organelor ţintă, evidenţierea semnelor clinice pentru comorbidităţi)			Exercițiu fizic	
Ex. paraclinice	Spirometrie		**Tratam. medicamentos** (ţinta terapeutică este reprezentată de renunţarea la fumat şi controlul simptomelor, cu mijloace terapeutice adecvate stadiului bolii - vezi **tabel**)	
	HLG completă		**Educația pacientului**	
	Rgr pulmonară		**Vaccinare antigripală**	
Consult ambulatoriu (pneumologie, pediatrie, alergologie şi medicină internă, dupa caz)			**Vaccinare antipneumococică (de luat în calcul, opțională)**	

Tratamentul BPOC in functie de stadializare			
Std I	**Std II**	**Std III**	**Std IV**
Reducere activă a factorilor de risc; vaccinare antigripală *Adaugă* bronhodilatatoare de scurtă durată (la nevoie)			
	Adaugă tratament regulat cu unul sau mai multe bronhodilatatoare cu durata lunga de acţiune (daca e necesar); Adaugă reabilitare		
		Adaugă corticosteroizi inhalatori în caz de exacerbări repetate	
			Adaugă oxigenoterapie de lunga durata în caz de insuficienţă respiratorie cronică *Consideră* tratamente chirurgicale

Clasificarea BPOC in stadii de severitate		
Stadiul BPOC	**VEMS/CVF (FEV1/FVC)**	**VEMS (FEV1)**
Stadiul I BPOC usor	**$\leq$ 70%**	**$\geq$ 80% din prezis**
Stadiul II BPOC moderat		**50% $\leq$ VEMS < 80%**
Stadiul III BPOC sever		**30% $\leq$ VEMS < 50%**
Stadiul IV BPOC foarte sever		**VEMS < 30% Sau VEMS <50% + IRpC**

Monitorizarea activă a BPOC

Obiective generale - Semestrial - reevaluarea nivelului de severitate/nivelului de control al bolii şi identificarea eventualelor cauze de control inadecvat ; - evaluarea complianţei la tratament şi ajustarea/continuarea terapiei pentru controlul simptomelor; - educaţia pacientului privind boala, evoluţia ei, - înţelegerea rolului diferitelor clase de medicamente şi a utilizării lor, - înţelegerea rolului pacientului în managementul de lungă durată a bolii, - sfatul pentru renunţarea la fumat;		**Stadiul I sau II**		**Stadiul III sau IV**			
		2015	**2016**	**2015**	**2015**	**2016**	**2016**
Data consultatiei							
Anamneza							
Fumat (pachete an)							
Alcool (unitati alcool)							
Ex clinic complet (inspecţie, auscultaţie şi palpare pentru evidenţierea semnelor de afectare a organelor ţintă, evidenţierea semnelor clinice pentru comorbidităţi)							
Exacerbari							
Ex paraclinice	Spirometrie						
	HLG completa						
	Rgr pulmonara						
Reevaluarea stadiului de severitate							
Optimizarea stilului de viață	Optimizare dietă						
	Scădere in greutate						
	Renunțare la fumat						
	Reducere consum alcool						
	Exercițiu fizic						
Aderența la tratament							
Consult ambulator (pneumologie, pediatrie, alergologie şi medicină internă, dupa caz)							
Educatia pacientului							
Vaccinare antigripală							
Vaccinare antipneumococică (de luat în calcul, opțională)							

Fig. 3.3: Example of a monitoring form [54]. The form contains various types of inputs grouped into categories and also organized into the initial evaluation and the active monitoring.

Type	*Characteristic*
Text	Varying amount of free text May contain an extra description May contain a set of predefined values and/or a default value
Numeric	Accompanied by a unit of measurement May be calculated from the values of other fields May be referenced by a calculated field
Boolean	Simple *yes* or *no* value May be referenced by a calculated field
Composed	A combination of two or more *Text*, *Numeric* or *Boolean* inputs

Table 3.1: Summary of medical input types

In contrast to free text inputs, boolean inputs (e.g. *Does the patient suffer from ...?*) require the least amount of typing for providing the necessary information. They must however be handled with care as to not confuse a *no* value with an unknown/not provided value.

Numeric information is also present when detailing the condition of a patient. Values range from integers to decimals, positive or negative. The fields requesting numeric inputs often stand out by providing a measurement unit – e.g. *Temperature (°C)*. A special type of numeric input is one whose value can be calculated by applying a formula on other numeric inputs from the same form. These special cases provide a good opportunity for automated optimizations that reduce the amount of input required while still providing the same amount of information.

The complexity of medical information leads to the existence of more complex input fields which can be viewed as a combination of two or more "primitive" inputs (as are those described above) that are semantically linked. A classic example of such a composed field is *blood pressure*, which is composed of two numeric fields: *Systolic* and *Diastolic*, both measured in *mmHg* [55].

Table 3.1 lists a summary of the types of input fields and their defining characteristics.

3.2. Rapid Prototyping Using XML

The availability of medical data input forms for use as examples allows for a data-first approach on modelling the domain classes. This involves a first step of gathering some input forms and storing them as structured data before moving on to the second step of extracting the domain entity classes from the structured data (detailed in section 3.3).

For the purpose of easily evaluating the system early on, prototyping has been employed as a development approach [56]. Prototyping, in general, comes with the following advantages: [57]

- It allows filtering and adjusting the requirements early in the development process
- It facilitates early feedback from users
- It shields from dealing low-level details not mandatory in the early phases of development.

Although using a relational database management system is what first comes to mind when thinking about storing the structured data, using a mark-up language like XML for prototyping the system comes with an important advantage: no schema $\Rightarrow$ no restrictions on the structure. As such, there is no need to define the entire structure of the document upfront (like one would do by defining the tables in an RDBMS before populating them), but, instead, the document can grow and adapt iteratively as the data is being added and new properties of the input elements are discovered. Also, the cost of provisioning a database in an RDBMS can be deferred for later stages of development.

By analyzing medical input forms that deal with monitoring patients with chronic illnesses, one can observe two distinct types of entities:

- *Input Elements* – each of the distinct (text/numeric/etc) inputs that the medic must address
- *Input Forms* – collections of select input elements, displayed in a specific order and optionally grouped by various criteria.

3.2.1. Structuring Input Elements

Although the input elements appear to be sub-components of the input forms, the analysis of multiple input forms reveals a lot of common input elements, as multiple forms ask for the same information (e.g. *When did the patient start smoking?*). This allows for the grouping of the input elements (removal of duplicates) and treating them as separate entities.

The new input element entities must now be associated a unique identity so that they can be referenced by the input forms, resulting in a N-N relationship[1]. For simplicity, the unique identities will take the form of integers in ascending order and stored in an XML attribute called *id*.

Each input element will be stored as a node in the XML document, with the name of the node indicating the type of the input element, as shown in table 3.2. Multiple nodes can be grouped together under a logical parent, yielding a tree structure. This can be easily modeled using XML by assigning the parent node the name *group* and placing it's child nodes below, as be observed in the listing below showing a group of input elements related to *family history*: [58]

```
<group id="28" label="AHC" longLabel="Antecedente heredo-colaterale" >
  <boolean id="29" label="HTA" />
  <boolean id="30" label="DZ" />
```

[1]Each input form can reference multiple input elements and each input element be can referenced by multiple forms

BUPT

```
    <boolean id="31" label="Guta" />
    <boolean id="32" label="Dislipidemie" />
</group>
```

	Name	*Characteristic*
primitive	**text**	Short textual input
	largeText	Medium or large free text input
	numeric	Numeric values
	boolean	Yes/No values
	calculated	Numeric values automatically calculated based on others
	group	Groups semantically related primitive inputs of any type
	combined	Complex input element composed of primitive elements

Table 3.2: XML node names of *input elements*

Besides the *id* attribute, a few others are also common to all of the input elements:

- *label* – stores the name of the input element
- *longLabel* – optionally stores the extented name of the input element, should the *label* be an abbreviation
- *description* – stores additional information to be presented to the medic.

Text elements can then optionally contain value suggestions, stored as sub-nodes of the input element node. They can also contain a default value, stored under the attribute *default*.

Numeric elements have an optional attribute for specifying the required precision and also an *identifier*. This is an internal name given to an element so that it's value can be referenced in a formula of a calculated field. What's also special about numeric elements, as opposed to free text ones, is that their value can be automatically interpreted to provide extra information [2]. This information can eliminate the need for the medic to consult various data tables and can also be used to prevent typing mistakes: presenting a value as abnormal can force the medic to double check it and notice a typing mistake.

Because the rules for interpreting numeric values are often static and found in tables, they can be incorporated into the business domain. Structuring them in an XML document involves creating sub-nodes of type *classification*. Each *classification* contains a set of *rules* by which to interpret the numeric value.

The need for multiple classifications comes from the fact that a value can be interpreted in different ways based on other inputs (e.g. a specific value can be considered normal for an adult, but too high for a child). The rules of each *classification* are composed of a *name*, a *condition* (a formula that triggers the rule) and an optional

[2]A simple example of this would be to interpret the provided blood pressure and immediately present whether it's a normal value or not

color code hinting at the normal/abnormal state of the value. Below is an example of such a *classification* for systolic blood pressure [55] in the case of adults. Notice that the *classification* itself also has a *condition* which enforces it's use only for adult patients:

```
<numeric label="Sistolic (mmHg)" precision="0" identifier="tensSistolica">
   <classification name="Pacienti peste 18 ani"
      condition="@patient['age'] >= 18">
      <rule name="Normala" condition="@value &lt; 120" color="success" />
      <rule name="Prehipertensiune"
          condition="120 &lt;= @value &amp;&amp; @value &lt;= 139"
          color="warning" />
      <rule name="Hipertensiune stadiul 1"
          condition="140 &lt;= @value &amp;&amp; @value &lt;= 159"
          color="danger" />
      <rule name="Hipertensiune stadiul 2"
          condition="@value &gt;= 160" color="danger" />
   </classification>
</numeric>
```

Calculated elements are characterized by the presence of a formula which in turn references other numeric/calculated elements by their identifier, and can also use the same *classification* structure as numeric elements. The formula, which is used for internal calculations, can also be accompanied by a textual description which details the formula to the medic. This description can be plain text, or it can contain rich mathematical expressions if written using a mathematical markup language like, for example, TEX and LATEX.

```
<calculated id="152" label="Ani de fumat" identifier="smokingYears">
   <formula>(@smokingEndAge > 0 ? @smokingEndAge : @patient['age']) -
      @smokingStartAge</formula>
   <formulaLabel>$$\textrm{Varsta oprire} =
      \left\{\begin{matrix}
      \textrm{Varsta oprire}, &amp; \textrm{Varsta oprire} > 0 \\
      \textrm{Varsta curenta}, &amp; \textrm{Varsta oprire} = 0
      \end{matrix}\right. $$
   </formulaLabel>
</calculated>
```

The formula description used in the above example would expand to a clear description for the medic, as presented by figure 3.4.

$$\textrm{Vârstă oprire} = \left\{\begin{matrix} \textrm{Vârstă oprire}, & \textrm{Vârstă oprire} > 0 \\ \textrm{Vârstă curentă}, & \textrm{Vârstă oprire} = 0 \end{matrix}\right.$$

Fig. 3.4: Rendering of a formula description expressed using LATEX

3.2.2. Structuring Input Forms

In contrast to the complexity of input elements, the structure of input forms is quite straightforward. Each one contains the following:

- An id (conventionally numeric, as in the case of input elements)
- A unique name
- An expansion of the name, should it be an abbreviation
- An ordered list of input elements, referenced by their id

These input forms allow for two types of data acquisition: the initial input, performed only once for a patient, and the active input, performed periodically. As such, each of the input element reference must also specify whether it involves an initial input or an active one. Multiple forms can reference the same input element [54].

Storing this data in XML yields a document having a structure similar to the following:

```
<form id="1" name="BPOC" longName="Bronhopneumopatie obstructiva cronica">
   <input id="1"   initial="true" active="true" />
   <input id="148" initial="true" active="true" />
   <input id="3"                  active="true" />
   <input id="4"   initial="true"               />
   <input id="5"   initial="true"               />
   <input id="6"   initial="true"               />
   ...
</form>
<form id="2" name="Astm">
   <input id="1"   initial="true" active="true" />
   <input id="148" initial="true" active="true" />
   <input id="3"                  active="true" />
   <input id="4"   initial="true"               />
   <input id="5"   initial="true"               />
   <input id="6"   initial="true" active="true" />
   ...
</form>
```

The references between the input forms and constituent input elements can be observed in figure 3.5.

3.3. Designing Business Domain Classes

Software applications targeting the administration of medical data fall in the category of *Enterprise Applications*, described by specialists as "about the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data" [59].

Enterprise applications benefit from a multilayer architecture with clear separation of concerns. One of these layers is the *Domain Model* which "creates a web of interconnected objects, where each object represents some meaningful individual" [59]. The classes comprising the domain model come in multiple flavors:

- *Entity (Data) Classes* – only store information
- *Behavioral (Business Logic) Classes* – perform business logic

The following sections presents models of the domain classes using a standard UML notation.

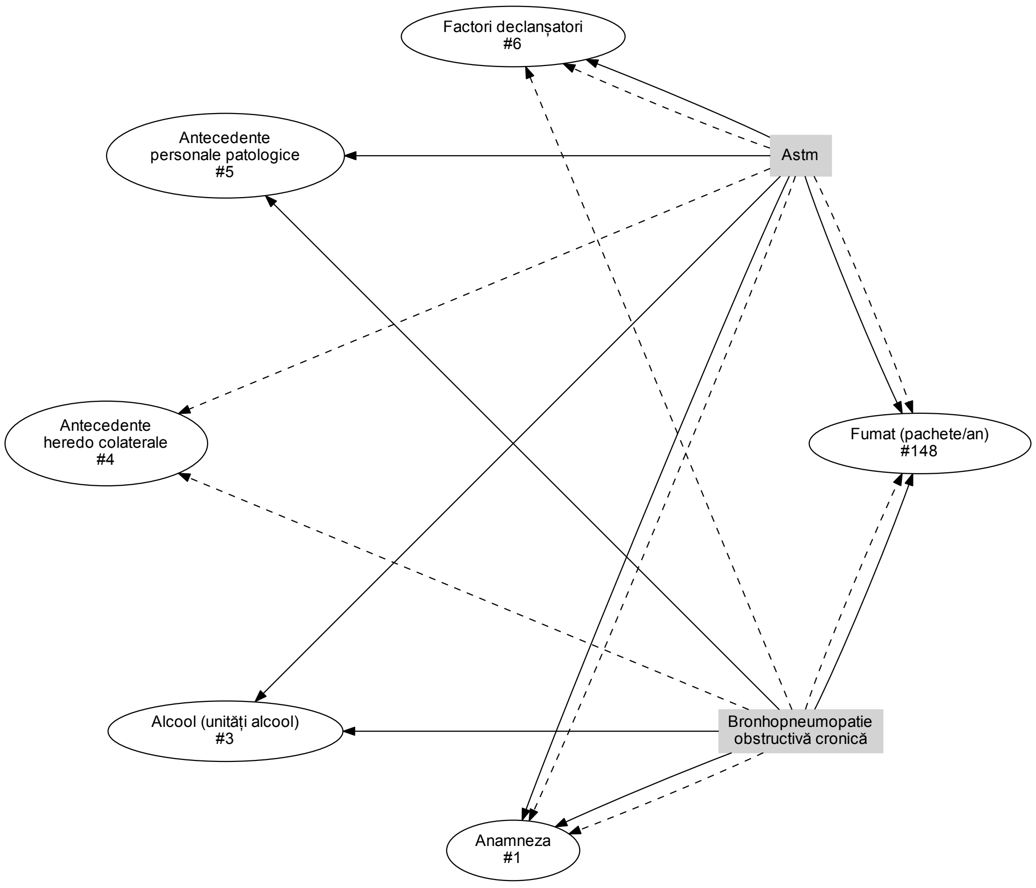


Fig. 3.5: References between *input forms* and *input elements*. Initial elements are referenced with a dashed arrow, active ones with a normal one.

3.3.1. Modeling Entity Classes

The entity classes need to modeled, in an object-oriented manner, in such a way as to be able to store all the information organized in XML, as presented in section 3.2. Because the entity classes are basically used for storing information, their methods are limited to getters and setters for various private fields.

The structure of the entity classes will be closely related to that of the originating XML document. As such, there will be a class called *InputElement*, storing information about a individual inputs, and a class called *InputForm*, containing the details of a form and a collection of input elements.

Modeling Input Elements

Because there are multiple types of input elements, with common as well as individual attributes, they can be modeled using a class hierarchy, with an abstract

InputElement as root.

Figure 3.6 shows the class diagram of the *InputElement* abstract class, containing all the common fields. Most of the subclasses then introduce additional fields, as presented in figure 3.7. Finally, figure 3.8 presents the entire class hierarchy related to the input elements.

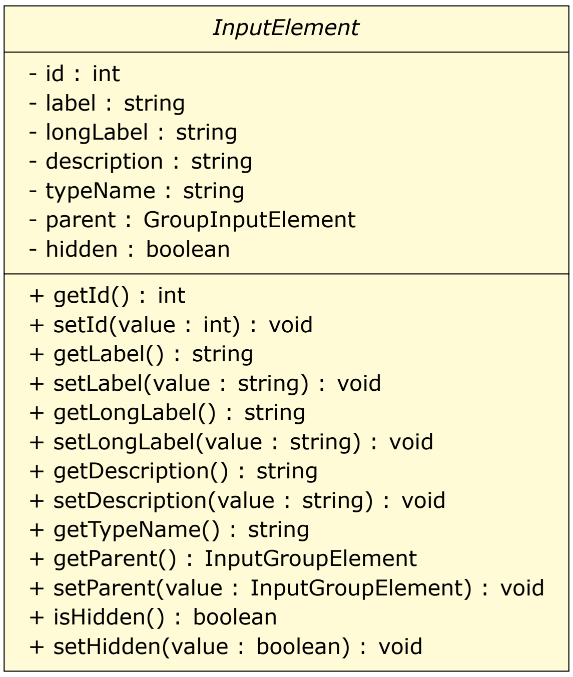

InputElement
- id : int
- label : string
- longLabel : string
- description : string
- typeName : string
- parent : GroupInputElement
- hidden : boolean
+ getId() : int
+ setId(value : int) : void
+ getLabel() : string
+ setLabel(value : string) : void
+ getLongLabel() : string
+ setLongLabel(value : string) : void
+ getDescription() : string
+ setDescription(value : string) : void
+ getTypeName() : string
+ getParent() : InputGroupElement
+ setParent(value : InputGroupElement) : void
+ isHidden() : boolean
+ setHidden(value : boolean) : void

Fig. 3.6: UML class diagram of the *input element* abstract class. This class contains common fields such as id, label or description.

Modeling Input Forms

The input forms are basically ordered collections of input elements grouped into two categories: elements to be displayed when the form is first used for a patient (*initial elements*), and elements to be displayed when filling in the form periodically (*activeElements*). As can be seen in figure 3.9, forms also have a (unique) numeric id, a short and an optionally long version of their name [3].

Modeling Value Classes

The classes and class hierarchies presented so far are related to storing the constructive details of the medical input forms and their comprising elements. Besides

[3]The *long* name is essentially the name of the form if the normal name is just an abbreviation, as is common in medical terms

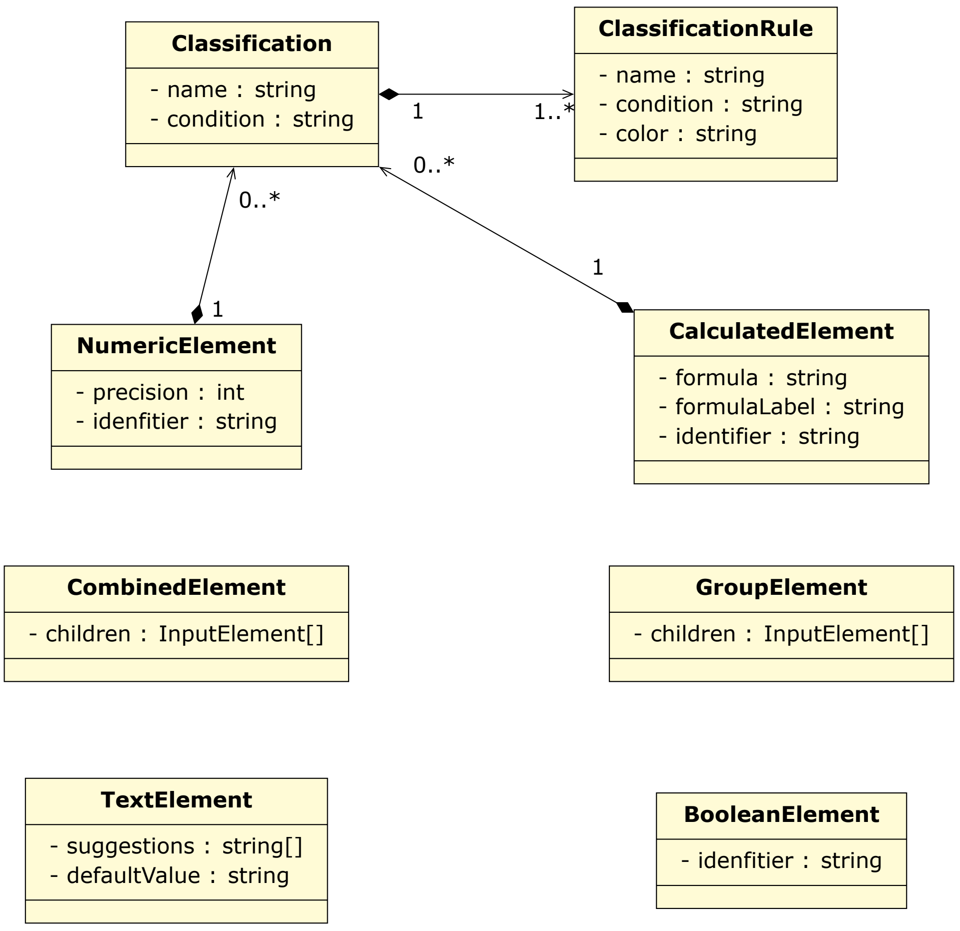


Fig. 3.7: UML Class Diagram of the *input element* subclasses (getters and setters omitted)

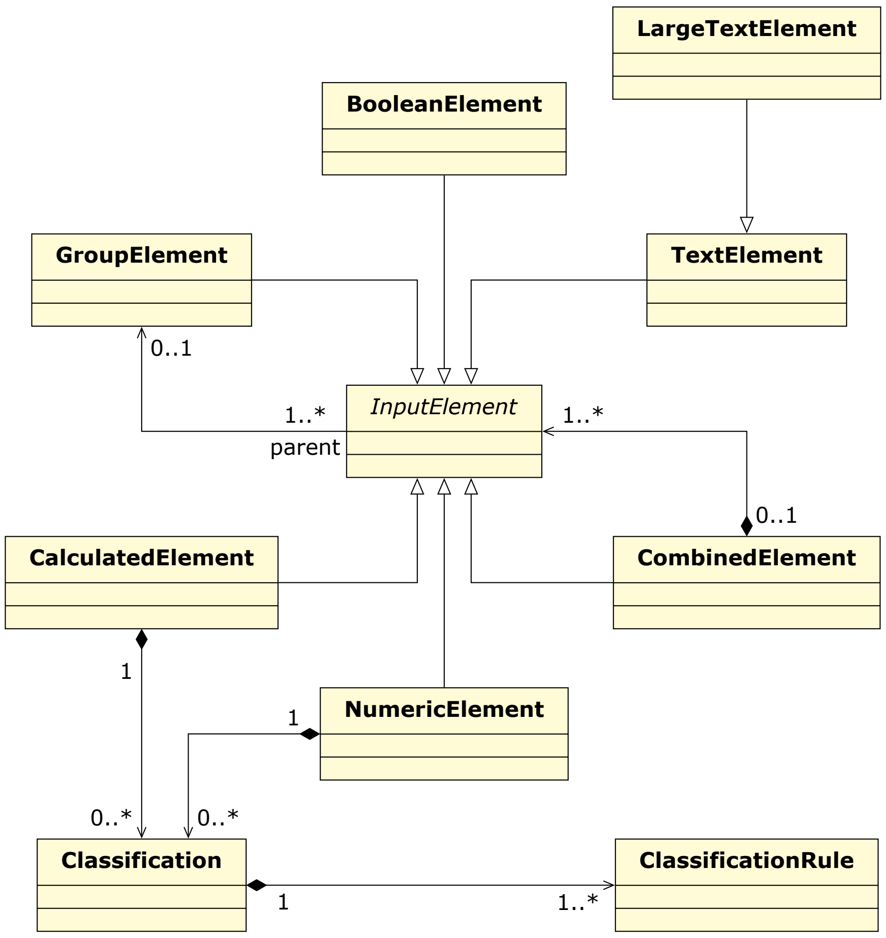


Fig. 3.8: UML class diagram of the *input element* class hierarchy

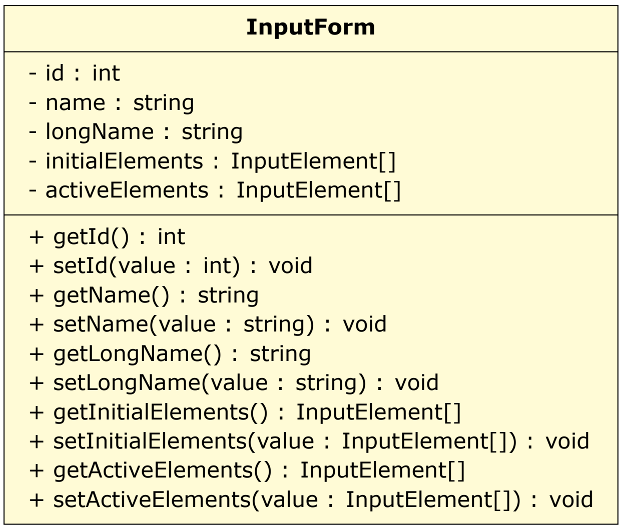

InputForm
- id : int
- name : string
- longName : string
- initialElements : InputElement[]
- activeElements : InputElement[]
+ getId() : int
+ setId(value : int) : void
+ getName() : string
+ setName(value : string) : void
+ getLongName() : string
+ setLongName(value : string) : void
+ getInitialElements() : InputElement[]
+ setInitialElements(value : InputElement[]) : void
+ getActiveElements() : InputElement[]
+ setActiveElements(value : InputElement[]) : void

Fig. 3.9: UML class diagram of the *input form* class. Besides an id and name, the input form aggregates multiple input elements into two groups: initial and active ones.

them, other entity classes are required for storing the actual values that are stored in the system.

One such class is the *Patient* class, presented in figure 3.10 with fields storing general information about a patient. Although instances of this class do not store medical information, they are referenced in formulas used by calculated input elements or classifications, the most common properties accessed being the person's birth date (for calculating the current age) and sex.

A similar entity class, *Employee*, stores information regarding each user that has access to the system. Such a user ca be a medic or a system administrator, the distinction being stored in form of an enumeration field called *role*.

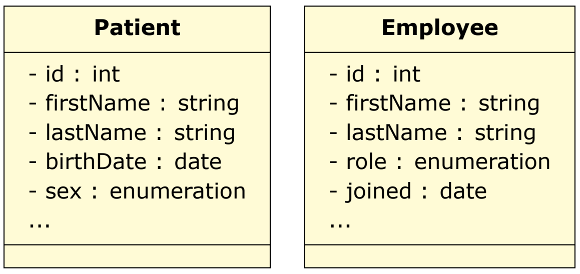


Fig. 3.10: UML class diagram modeling a patient and an employee class (getters and setters omitted)

The actual medical data entered in the system needs to be structured in a similar way as the input elements and forms. As such, an *InputFormValue* class would be similar to an *InputForm* class, storing two sets of *InputElementValue* collections, one for the initial elements of the form and one for the active ones. The class also references the patient involved, the medic that entered the value and stores the date of the recording.

As can be seen in figure 3.11, the *InputElementValue* class stores the actual value, and then references the *InputElement* class for which the value is intended. Programmatically, the actual medical value can be stored as a string, decimal, boolean or date/time value. A secondary field stores the type of the value, using an enumeration. Combined input elements represent a special case as they need to store multiple values, one from each child element. To accommodate for this, *InputElementValue* can also store an array of self-referenced *InputElementValue* elements, only used in the case of combined elements.

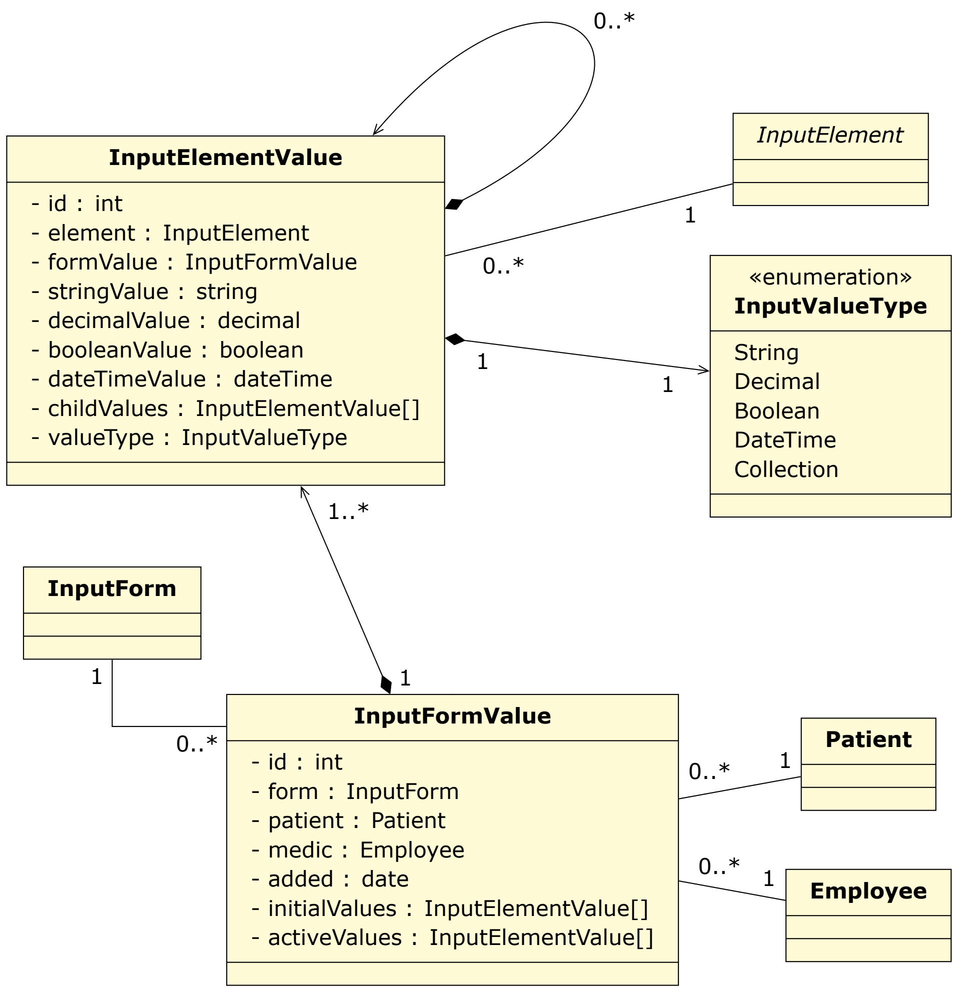


Fig. 3.11: UML class diagram related to the medical data classes (getters and setters omitted)

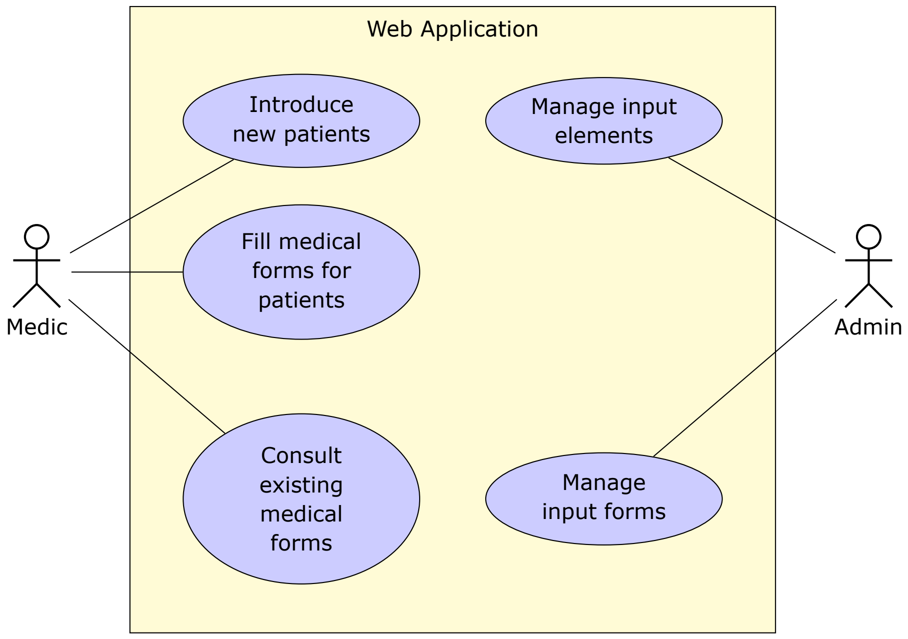


Fig. 3.12: Use cases of the web application. The system administrator is responsible for updating requested input forms and/or their constituent elements while the normal user (usually a medic) is tasked with entering new medical information about patients.

3.3.2. Modeling Behavioral Classes

With the entity classes in place, the domain model now demands behavioral classes, the ones that actually perform the business logic. In order to understand what behavioral classes are actually need, one must study the use cases of the system.

Figure 3.12 presents how the two main actors interact with the web application:

- The system administrator manages the input forms and their constituent input elements, adapting them to new requirements originating from the medics or changes in legislation.
- The medics use the web application to record medical information gathered at each patient consultation and to retrieve medical information previously recorded.

As such, the web application can be viewed as a large data store. Literature on software design presents numerous ways of building data-centric applications, however one design pattern stands out: the *Repository*, presented as "mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects" [59].

By creating a repository, the actual source and destination of the data is abstracted away, achieving loose coupling. The domain objects are presented with a repository instance which resembles a collection, returning the desired entities when queried. Behind the scenes, the repository makes use of a specific strategy to extract the data from a data store (usually a relational database) and map to into entity objects.

Figures 3.13 and 3.14 show the sequence of operations when using the repository to retrieve or store entities (in this case input forms):

- The client of the repository creates a *query* object which stores information on the requested operation
- The client then passes the query to the repository which forwards it to an appropriate method of it's current strategy
- The strategy (one for operating with relational databases in this example) creates an appropriate SQL query and executes it against the database.
- The rows resulting from querying the database are then passed to a mapper which creates appropriate *InputForm* classes and populates their properties based on the information in the database rows.

Having a mapping layer prevents the entity domain classes from being tightly coupled to a specific database structure. This, together with the use of the repository, makes changing the structure, or even the type, of the database much easier.

3.3.3. Persisting Data to a Relational Database

After using XML to prototype the structure of the system's data, as described in section 3.2, the structured data must be stored in a way more adequate for production use.

Traditionally, Electronic Health Record systems persist their data in relational or object-relational databases [40]. Such systems fall into the category of Online Transactional Processing (OLTP) and are generally used for storing and retrieving individual patient records [40].

Choosing a relational database for storing medical forms and inputs is motivated by the features offered for ensuring persistence and consistency – CP in the *Consistency, Availability and Partition tolerance* (CAP) theorem [60] – both important for OLTP systems. Relational databases are also easily connected to input data into data warehouses used for Online Analytical Processing (OLAP) applications. This is important as EHR data can also be used for other purposes such as clinical and epidemiological studies or the determination of a population epidemiological profile [40].

Relational databases operate on normalized data stored in tables (relations) that follow a strict schema. Storing the data contained by the object-oriented models presented earlier requires multiple database tables to be created and linked with each other.

Because the system is designed around various types of input forms and their characteristics, the rigid database schema needs only to accommodate the properties of the input elements, few in number. Had input forms been the focus of attention, without modeling the individual input elements, then the database schema will had

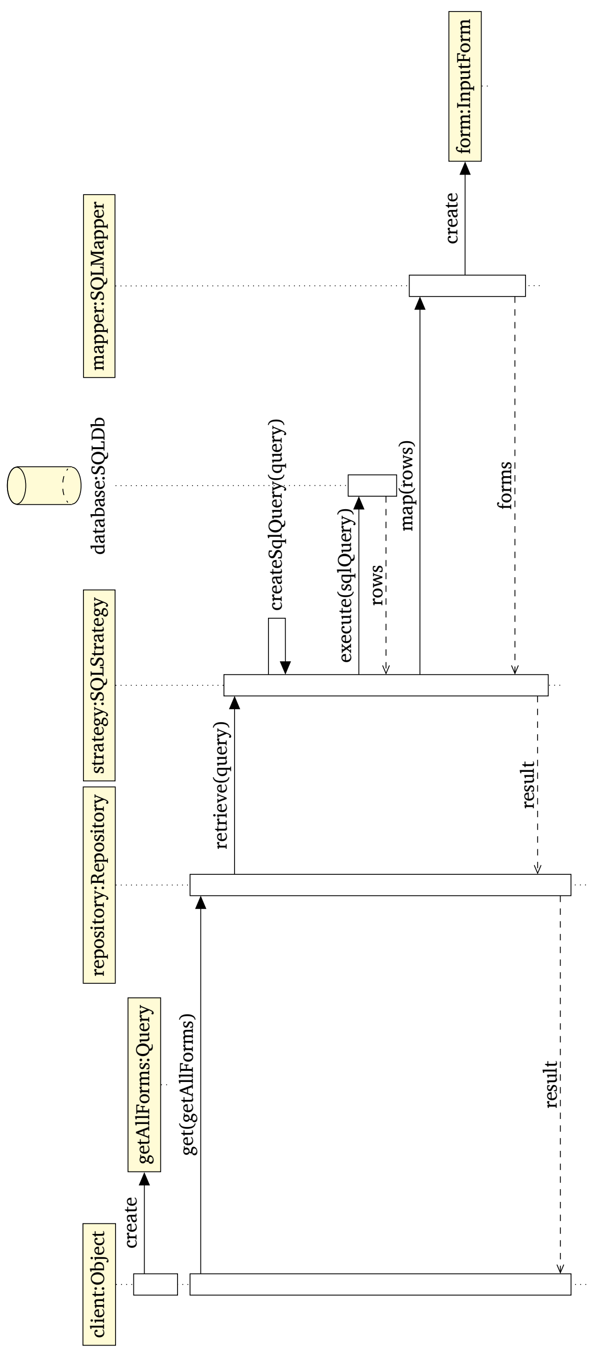


Fig. 3.13: Fetching input forms from a repository. The structure of each input form is persisted via multiple rows in a database.

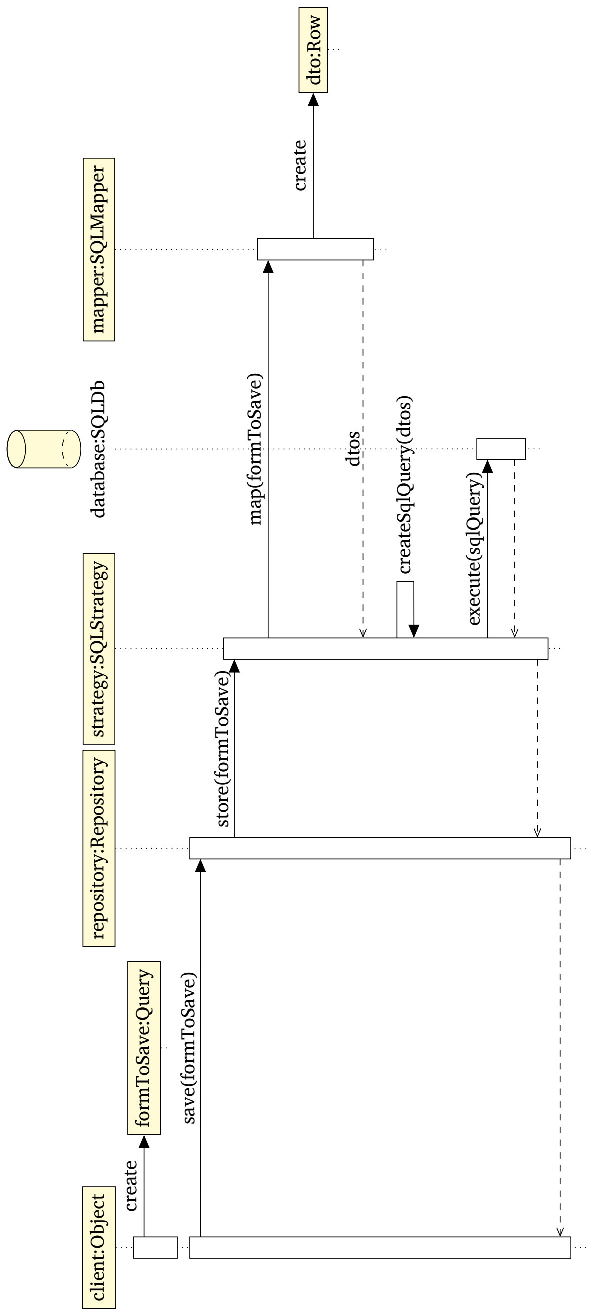


Fig. 3.14: Saving input forms to a repository. The structure of each input form is persisted via multiple rows in a database.

to have been strongly dependent of the structure of each form, thus leading to an inflexible approach that would have required constant change.

Two approaches have been identified for creating a schema:

- Creating a single table for storing the simple properties of all input elements, as shown in figure 3.15. This approach leads to the addition of multiple columns that only apply for some input elements, remaining empty for others. Properties that exhibit a $1 - N$ or $N - N$ relationship are implemented with the help of extra tables.
- Creating a table for all the common properties of input elements and one extra table for each type of input element for storing additional properties, as shown in figure 3.16. This approach results in many more smaller tables which do a better job at isolating data but present overhead when querying.

From the programmatic point of view, the details on how to store and retrieve the data from a relational database are handled by a specialized repository strategy, together with appropriate mappers. The paragraphs that follow will look into how to use a commercially available relational database, Microsoft SQL Server and connect to it from a Microsoft .NET application using Microsoft Entity Framework.

Microsoft's Entity Framework is a framework for object-relational mapping (ORM). It bridges the gap between object oriented programming and the relational structure of databases by allowing users to work with entity classes while automatically *mapping* the data between them and the database tables [61]. Using object-relational mappers has become one of the most used persistence mechanisms in object-oriented systems [40].

Microsoft Entity Framework allows for three major ways to obtain the database entity classes [62]:

- *Database First* – point Microsoft Entity Framework to an existing database schema and let it generate the database entity classes
- *Model First* – create a new model using UI tools and then let Microsoft Entity Framework generate a database from it
- *Code First* – write the database entity classes by hand and let Microsoft Entity Framework generate an appropriate database schema for them.

Having already modeled classes in section 3.3.1, the logical way to proceed was to create database entity classes and then generate a new database from their structure. The structure of the newly created database entity classes is presented in figure 3.17. At the center of the class diagram lies a hierarchy rooted in the *InputElement* class. From the metadata of these classes, the object-relational mapping framework was able to generate a database, the structure of which is presented in figure 3.15.

The classes used with Microsoft Entity Framework have a similar structure to the entity classes that are part of the domain model (presented in section 3.3.1) but they are not identical. Using the same classes, although tempting, would have broken the single-responsibility principle and contaminates the entity classes with properties

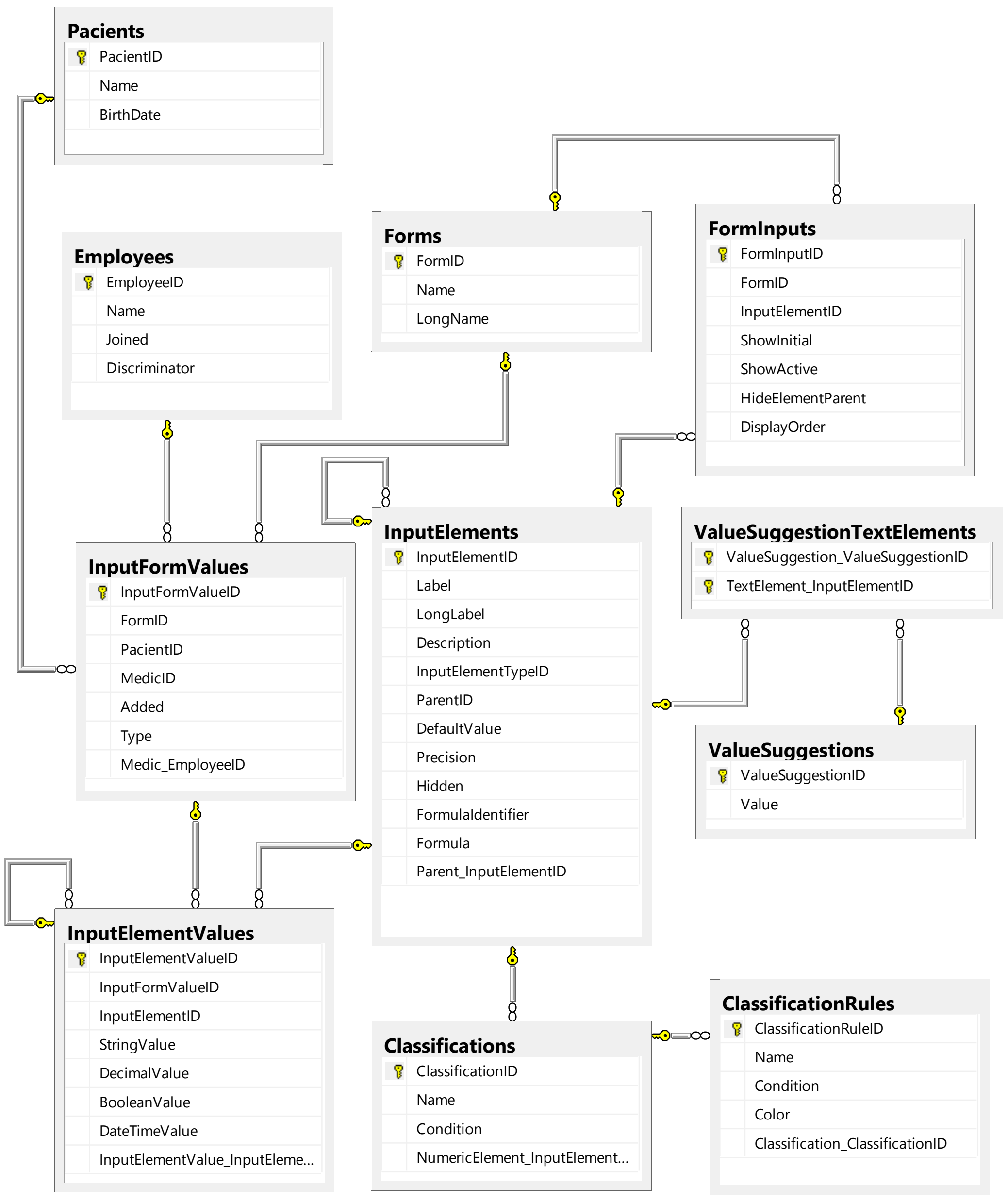


Fig. 3.15: Database diagram with flattened hierarchies

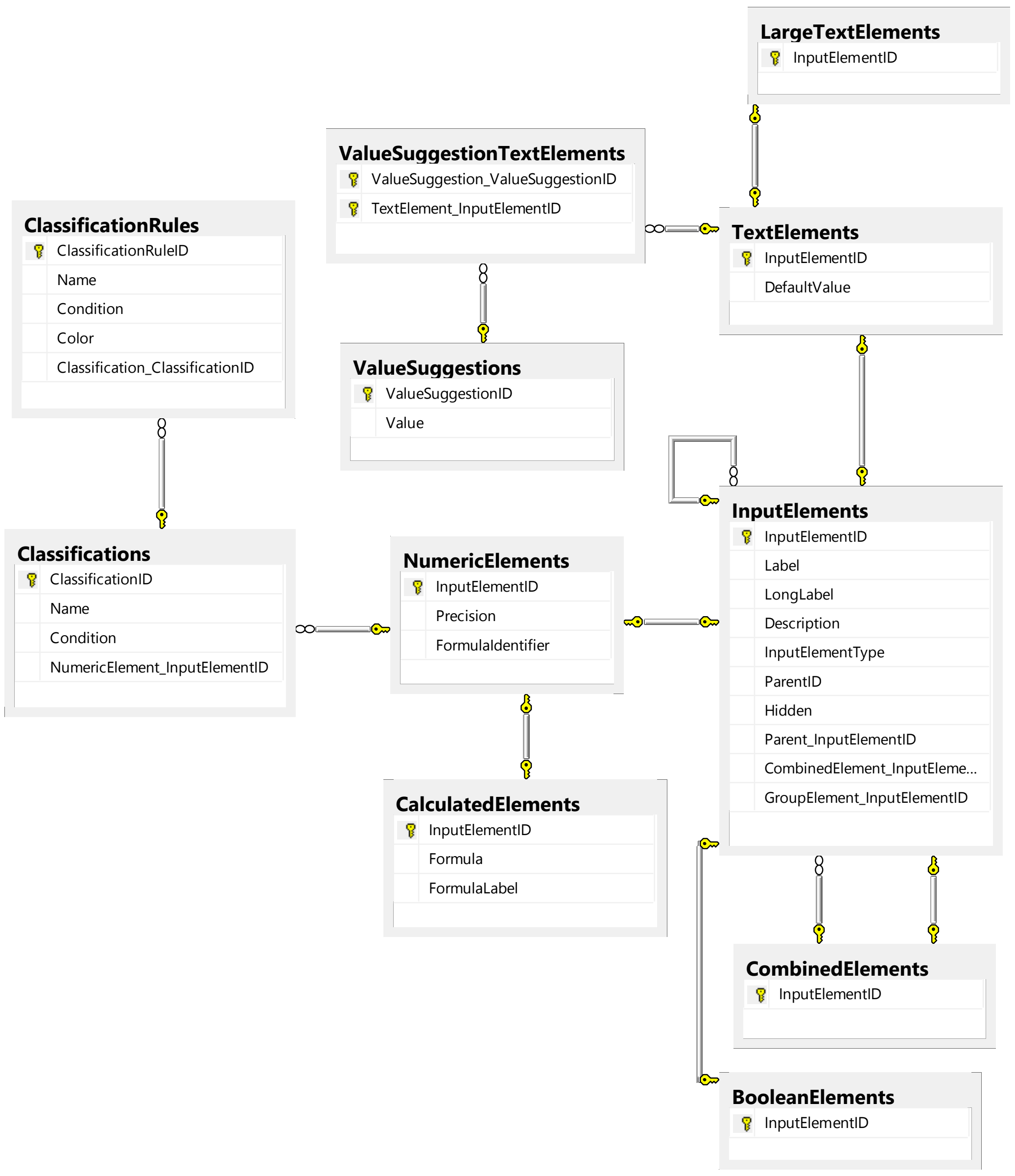


Fig. 3.16: Database diagram preserving input element hierarchies

strictly related to the way they are stored in a database. Maintaining a different set of classes for use with Microsoft Entity Framework, and translating between them and the entity classes with the help of mappers, helps keep the ORM logic isolated from the rest of the domain model, also making it easy to replace if necessary.

When it comes to the structure of relational database tables, it is quite different from the one common in object-oriented data structures. Most notably, parent-child relationships are implemented using foreign keys and can even require additional tables, if there are $N - N$ relationships. Inheritance is also not present in relational databases and must be emulated.

Microsoft Entity Framework makes use of *convention over configuration* providing default behaviors for the way it names tables or identifies primary keys or table links. This allows the code to remain very clean, similar to a plain object:

```
public class InputElement
{
   public int InputElementID { get; set; }
   public string Label { get; set; }
   public string LongLabel { get; set; }
   public string Description { get; set; }
   public InputElementType InputElementType { get; set; }
   public int ParentID { get; set; }
   public bool Hidden { get; set; }

   public virtual InputElement Parent { get; set; }
}
```

By default, Microsoft Entity Framework collapses class hierarchies into a single table, named after the base class. This can be observed in figure 3.15 which lists only a single, flattened *InputElements* table which contains columns specific the all the different input elements. Microsoft Entity Framework also supports using separate tables for each subclass. This leads to a structure shown in figure 3.16 and triggered by simply decorating the database entity class with an appropriate attribute:

```
[Table("TextElements")]
public class TextElement : InputElement
{
   public string DefaultValue { get; set; }
   public virtual ICollection<ValueSuggestion> ValueSuggestions { get; set; }
}
```

The following example query selects all input elements shown in the initial evaluation of the *Asthma* input form:

```
using (var context = new EFPersistence.Context())
{
   var formElements = context.Forms
      .First(f => f.Name.Equals("Astm", StringComparison.OrdinalIgnoreCase))
      .Inputs.Where(i => i.ShowInitial)
      .OrderBy(i => i.DisplayOrder)
      .ToArray();
}
```

Thus, such a request gets converted by the Microsoft Entity Framework into the following two queries, which are forwarded to the relational database management system.

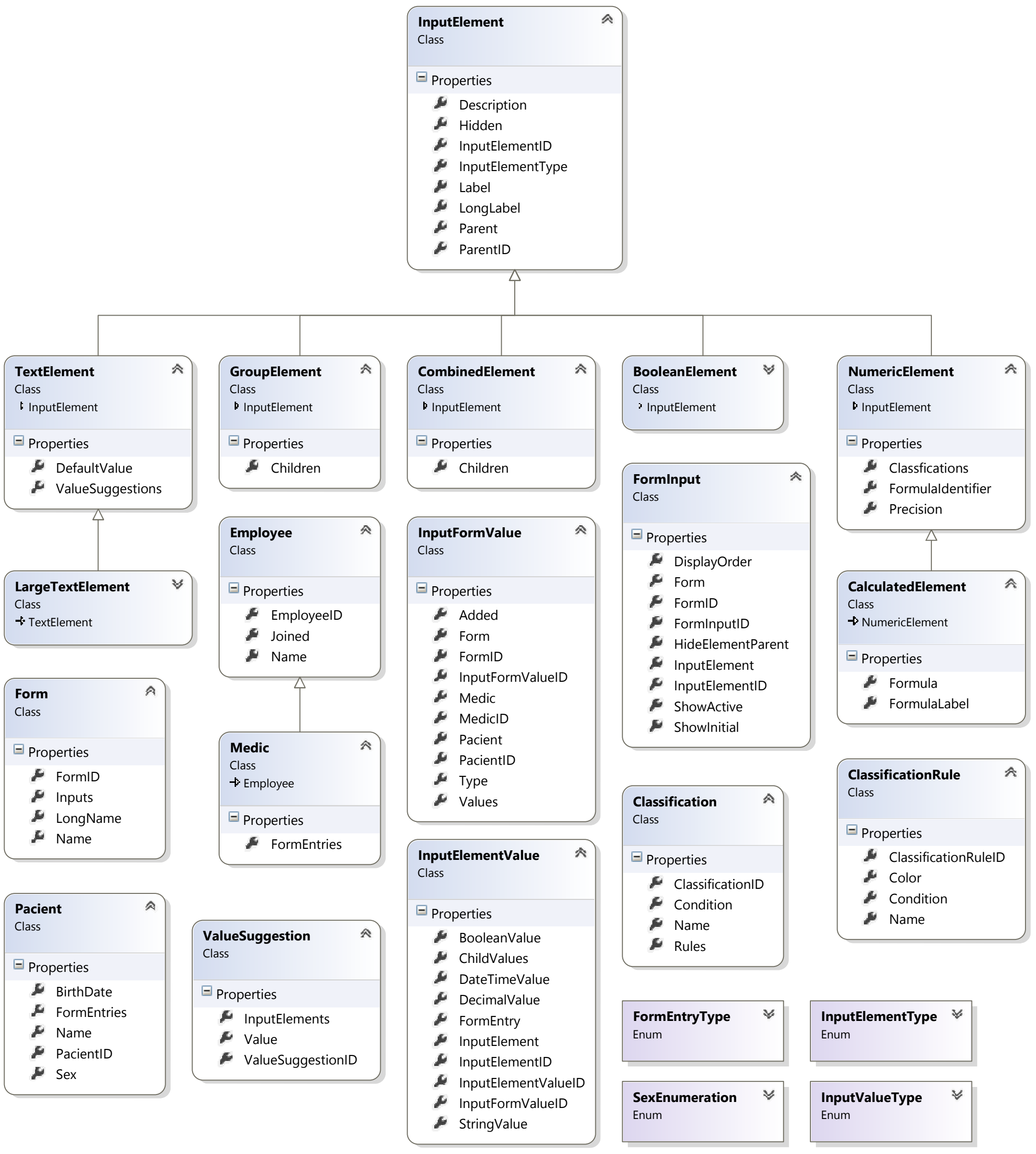


Fig. 3.17: Database entity classes used with Microsoft Entity Framework

```
SELECT TOP (1)
    [Extent1].[FormID] AS [FormID],
    [Extent1].[Name] AS [Name],
    [Extent1].[LongName] AS [LongName]
    FROM [dbo].[Forms] AS [Extent1]
    WHERE N'Astm' = [Extent1].[Name]

SELECT
    [Extent1].[FormInputID] AS [FormInputID],
    [Extent1].[FormID] AS [FormID],
    [Extent1].[InputElementID] AS [InputElementID],
    [Extent1].[ShowInitial] AS [ShowInitial],
    [Extent1].[ShowActive] AS [ShowActive],
    [Extent1].[HideElementParent] AS [HideElementParent],
    [Extent1].[DisplayOrder] AS [DisplayOrder]
    FROM [dbo].[FormInputs] AS [Extent1]
    WHERE [Extent1].[FormID] = @EntityKeyValue1
```

3.3.4. Persisting Data to a NoSQL Document Database

Although relational databases have become very popular in the last decades, companies have also been looking at other ways to store and query data. The last decade has seen the appearance of various database products, which fall under the umbrella term of *NoSQL*.

In contrast with relational databases which share a similar structure and are queried using the powerful SQL language (or dialects thereof), NoSQL products are much more diverse and are focused on solving more specific problems. By their data model, NoSQL databases fall into the following categories: [60]

- *Key-Value Databases* – are the simplest as they only store a map of keys and values. The only allowed operations are putting a value for a key, retrieving that value and deleting the value of a key. From the database's perspective, the actual value is a blob [60].

- *Document Databases* – are similar to Key-Value databases but differ in the fact that the actual value isn't a blob anymore, it's a document stored using popular formats such as BSON, JSON, XML, ... which the database can understand and can manipulate when responding to queries [60].

- *Column-Family Stores* – allow storing data with keys mapped to values and then grouping the values into multiple column families [60]. Similar to relational databases, column-family stores require the creation of a schema, however optimizations allow rows to only present values for few of the available columns.

- *Graph Databases* – allow storing entities and relationships between them, each with various properties [60].

Key-Value databases, due to their simplicity, are not suited for persisting complex electronic health records. They can be useful however for storing volatile data (usually in volatile memory) such as temporary session state when serving a web application from a cluster of servers.

Column-family stores share similarities to the way relational databases require a schema to be designed up-front and also share the same weaknesses when dealing with highly interconnected records. The query languages offered are also usually less flexible and feature rich than traditional SQL. Where column-family databases have made a name for themselves is domains that require the storage and processing of large amounts of data, Big Data.

A popular document database which can be used to store medical forms and inputs is MongoDB[4]. Unlike a relational database which stores normalized data, a MongoDB document can store an entire hierarchy and even allows it's structure to vary from document to document. For example, an entire input form model can be stored in just one document, as shown in the listing below:

```
{
  "_id" : 1,
  "name" : "BPOC",
  "longName" : "Bronhopneumopatie obstructiva cronica",
  "inputs" : [{
      "showInitial" : true,
      "showActive" : true,
      "displayOrder" : 1,
      "inputElement" : {
        "label" : "Anamneza",
        "type" : "LargeText",
      }
    }, {
      "showInitial" : true,
      "showActive" : false,
      "displayOrder" : 4,
      "inputElement" : {
        "label" : "AHC",
        "longLabel" : "Antecedente heredo colaterale",
        "type" : "Text",
      }
    }, {
      "showInitial" : true,
      "showActive" : true,
      "displayOrder" : 9,
      "inputElement" : {
        "label" : "Spirometrie (%)",
        "type" : "Numeric",
      }
    },
  ]
}
```

One of the first questions that appears, thanks to the possibility of storing an entire hierarchy in a document is how to find a balance between a full hierarchy and a partial one. Storing the details of each input element inside each input form introduces data duplication and makes updating an input on all forms difficult. Is it more flexible to store input elements in a separate collection and just reference them in the forms where they appear, similar to a relational model. In order to avoid querying the input element collection every time a form needs to be loaded, the input element collection can be cached in memory.

[4]https://www.mongodb.com/

The hierarchies are much better suited for storing patient records. Thus, all inputs provided for a specific patient on a specific occasion can be grouped inside a single document:

```
{
  ”_id” : 1542,
  ”patient” : {
    ”id” : ”4803”,
    ”firstName” : ”Ion”,
    ”lastName” : ”Popescu”
  },
  ”consult” : ”2016-04-15”,
  ”entries” : [
    ”form” : {
      ”id” : 1,
      ”name” : ”BPOC”,
      ”longName” : ”Bronhopneumopatie obstructiva cronica”,
    },
    ”values” : [{
        ”inputId” : 119,
        ”inputName” : [”Examen clinic - Aparat respirator”, ”Palpare”],
        ”value” : ”Transmiterea vibratiilor vocale”
      }, {
        ”inputId” : 9,
        ”inputName” : [”Examene paraclinice”, ”Spirometrie (%)”],
        ”value” : 70
      }, {
        ”inputId” : 9,
        ”inputName” : [”Examene paraclinice”, ”HLG completa”],
        ”value” : [{
            ”subItem” : ”Hemoglobina (g/dl)”,
            ”value” : 11
          }, {
            ”subItem” : ”Hematii (x10^{6} / mm^{3})”,
            ”value” : 4.7
          }, {
            ”subItem” : ”Leucocite (/ mm^{3})”,
            ”value” : 10000
          }, {
            ”subItem” : ”Trombocite (/ mm^{3})”,
            ”value” : 265000
          }
        ]
      },
    ]
  ]
}
```

By storing the medical inputs regarding a specific patient at a specific time in a single document, the document is immune to any future changes in the structure of input elements or forms. Adding/changing or deleting an input element will have no undesired effect on past entries. Should the database grow to big, documents can be split up between multiple instances using a technique called *sharding*. Having related data about a patient in one place helps avoid the overhead that would have been encountered when querying data from multiple sources.

An area where storing data as documents is less than ideal is when doing analytics, complex analysis of data across multiple patients. A solution in this case is the creation of materialized views [60]. These provide an alternate view of the data, suited for a specific purpose, but, in contrast to traditional database views, they are updated periodically, on demand, not on every query. A popular method for creating materialized views for NoSQL databases is by using the *Map-Reduce* programming model [60].

A study shows that MongoDB can be successfully used in a distributed manner to process large amounts of HL7 messages [63]. By creating a virtual file system they manage to create a scalable solution that overcomes the limitations of traditional file systems while using legacy systems that are designed to use the file system.

Conclusions

This chapter focused on investigating ways of architecting an electronic health record application which is flexible to change while also requiring as less medical domain knowledge as possible from the developers side.

The main original contribution of this chapter consists of a proposed back-end architecture which focuses on modeling the data around medical input types. These are then combined into medical input forms and allow the creation of a configuration interface in which a medical specialist is able to define new input forms from existing types without requiring the intervention of a software developer to write new code or change the database (except in the rare case in which a completely new input type is required).

Types of input forms were identified by analyzing multiple input forms and structured in XML with the help of rapid prototyping. The chapter also provided contributions regarding the data layer, presenting ways to structure the data, based on input types, in relational as well as document databases.

4. INTEGRATING MEDICAL STANDARDS

When writing applications which operate in the healthcare industry, the requirement to interconnect with other systems, at one point, is almost unavoidable. This chapter presents an analysis of issues identified when adapting an existing application that deals with electronic health records to common and standardized practices and also details possible solutions.

An important issue when dealing with health records is how to handle terminology, especially when interconnecting with systems that use a different human language. The section *Translating Logical Observation Identifiers Names and Codes* deals with how to translate the LOINC laboratory codes into a language in which they are not yet available. Immediately afterwards, *Matching ICD-10 Codes Using Full-Text Search Engines* deals with another very important standard, ICD-10 for diagnostic codes. This standard has a more widespread use and is translated into numerous languages. However, searching precise codes is not straight-forward and the section compares ways of using open-source full-text search engines for matching ICD-10 codes.

The last section, *Achieving Interoperability* describes how an electronic health record system, built in a simple manner to solve specific immediate issues at creation, can be extended to link with medical terminologies and to benefit from openEHR specifications and existing artifacts built around these specifications. The section also deals with information exchange using the Clinical Document Architecture (CDA) standard.

4.1. Translating Logical Observation Identifiers Names and Codes

Electronic methods of communicating medical information like HL7 messages or HL7 CDA documents contain a lot of coded information. If each producer of such messages (e.g. laboratories) were to use their own internal codes, then understanding the message by the receiver would be problematic without adopting the sender's codes. The situation then becomes even more complicated if there are multiple sources [28].

This situation is fixed by the introduction and usage of medical coding and terminology systems. One such system is LOINC® (Logical Observation Identifiers Names and Codes) which "provides a set of universal names and ID codes for identifying laboratory and clinical test results" [28].

As LOINC contains textual descriptions for each code, a prerequisite for using LOINC in a country is the availability of translations for the terms used by LOINC in the country's native language. D. Vreeman describes the challenges of translating LOINC into Italian [29], placing emphasis on breaking up terms into separate pieces for independent translation.

This section presents a research into ways to find better, more detailed ways of reducing the effort required for translating LOINC into other languages. The results have been published under [64].

The following sections present some considerations on medical terminology and then go on to detail the process of analyzing the content of the LOINC data table with the goal of creating an efficient translation strategy.

4.1.1. Analyzing Medical Terminology

An interesting topic when dealing with medical terminologies in multiple languages is the question of how many times are terms used in their canonical form and how many times are they derived? In order to answer this question, a study has been conducted which compares the incidence rate of canonical vs. derived medical terminology written in English and Romanian. The findings have been published in a paper [65].

Performing the study required the design and development of a dedicated service, *text4all Term Analysis and Stats*[1]. This service takes the address of a web page as input and outputs a report about the medical terminology used in the web page. It performs the following steps: [65]

1. Downloads the HTML document of the web page in question, from the URL provided by the user
2. Extracts the meaningful text from the document
3. Normalizes the text by converting it to lower-case
4. Compares each word to a terminology dictionary in order to identify medical terms
5. Searches for medical terms that are derived.

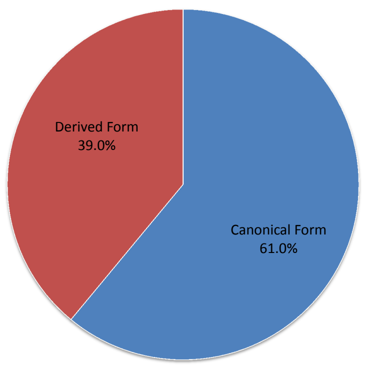
Identifying derived medical terms is done by using a technique called *fuzzy matching*, comparing words by using the Levenshtein or Hamming distance. The matching techniques have incorporated by V. Topac into a dedicated data structure called *FuzzyHashMap* [66].

The end report present the total number of words found on the given web page, the number of terminology found in canonical form (unique and repeating occurrences) as well as a list of all approximate matching terms identified [65].

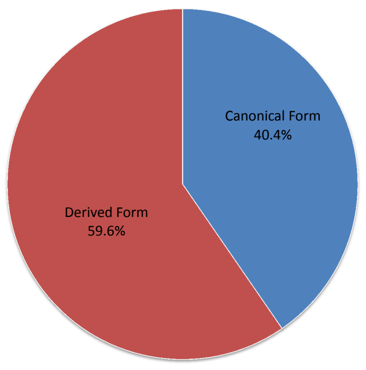
Analyzing the content of multiple English web sites which present medical information for the general public has revealed a ratio of 61.0% terms in canonical form to 39.1% in derived form (terms are counted multiple times if they appear many times throughout the text). Performing the same analysis on similar web sites which present medical content in Romanian leads to results which are almost the opposite: a ratio of 40.4% terms in canonical form to 59.6% in derived form [65]. The results are also presented in figure 4.1.

This study shows an important situation to be aware of when setting of to perform translations of medical terms. Although, in the case of the English language,

[1]Available under http://www.text4all.net/termanalysis.html



(a) English



(b) Romanian

Fig. 4.1: Incidence rate of canonical vs. derived medical terminology in natural language [65]. The results in English and Romanian are almost mirrored, with the canonical form being more commonly used in English while the derived form appears more often in Romanian.

canonical form of terms is more common, other languages, like Romanian, use far more derived terms.

4.1.2. Observed Data Patterns

An analysis was performed on the values of the LOINC data table with the goal of identifying patterns which can be used to reduce the number of works which require translation into other languages. This analysis was performed using version 2.46 of the English language version of the LOINC table, released on December 26th 2013 and available for download under the official website (https://loinc.org/downloads) in a variety of formats, including a simple comma-separated value (CSV) file.

Each row in the table is identified by a unique value stored in the first column, also called LOINC_NUM: e.g. 10092-5. The remaining columns store mostly strings (enumeration of terms or even sentences) but also numeric or date values. The content of the LOINC table is summarized in table 4.1.

Measurement	Value
Number of rows:	73,115
Number of columns:	48
Number of null (empty) values:	1,904,784
Number of non-string values:	497,721
Number of string values:	1,107,015
Number of unique string values:	261,115
Total length of unique string values:	23,785,626
Average length of unique string values:	91.09

Table 4.1: Summary of the data found in the LOINC table [64]

Because non-string values (e.g. numbers) do not require translations, one of the first steps taken was to separate them from ordinary string values. After identifying duplicates, only 261,115 string values would require translation, a number is still pretty high, keeping in mind that the average length of such an entry is 91.09 characters [64].

The next step involved identifying various patterns that can help in breaking up larger cells into smaller elements. Because larger cells don't include complete sentences but rather enumerations of different elements, they can be split up, translated individually and reassembled without being dependent on rules of the English grammar.

Examples of such enumerations are:

- *Person name; Point in time; Random; Cardiac rehabilitation Tx plan; Nominal*
- *Coxiella burnetii phase II; C burnet Ph2; Q fev; C burnetii; Qfever; Coxiella burnettii; Q fever; Dilution factor; Titer; Titre; Ttr; Titered; Point in time; Random; Serum; SR; Quantitative; QNT; Quant; Quan; ACIF; FA; Fluorescent antibody; Immune fluorescence; Fluoresent; Immunoflour; Immunofluor; Immunofluorescence; IFA; Time Resolved Fluorescence; Anticomplement Immunofluorescence; TRF; ABS; Antibodies; Autoantibody; Antibody; Autoantibodies; Antby; Aby; Anti; Microbiology*
- *Dx.primary; Dx; Interpretation; Interp; Impression; Impressions; Point in time; Random; Alcohol- substance abuse rehabilitation Tx; Alcohol-substance abuse rehabilitation Tx; Narrative; Report*
- *Immune globulin G; Immunoglobulin G; Arbitrary concentration; Point in time; Random; Serum; SR; Quantitative; QNT; Quant; Quan; ABS; Antibodies; Autoantibody; Antibody; Autoantibodies; Antby; Aby; Anti; species; spp; Microbiology*
- *Immune globulin G; Immunoglobulin G; B burgdor41kD; B burg; B burgdorferi; Lymes; Burgdorf; Lyme; Bb; Lyme disease; Arbitrary concentration; Point in time; Random; Cerebral spinal fluid; Spinal Fluid; Cerebrospinal Fl; Spinal Flu; Spinal Fld; Ql; Ordinal; QL; Qualitative; Qual; Screen; Western blot; WB; Immune blot; Immunobl; Blt; WBLOT; West blt; West blot; RIBA; Immunoblot; ABS; Antibodies; Autoantibody; Antibody; Autoantibodies; Antby; Aby; Anti; Microbiology*

As these terms are not related to each other, from a translations point of view, splitting each value up into its individual atoms results in translating much smaller terms that also have a bigger chance of being repeated. Cells containing narrative text can also benefit from being split up, although in this case only at sentence level.

Some values do not require any translation because they only represent various identifiers or value/measurement unit pairs [64], e.g.:

- *25 mg/dL*
- *30 mg/dL*
- *mU/mL;mcU/mL*

- *umol/L*
- *mm[Hg]*

Various columns have lead the way to the discovery of more complex patterns. One of these consists in specifying what has been detected/observed, where, the unit of measurement, and, optionally, the number of the specimen [64]:

- *Lutropin [Units/volume] in Serum or Plasma –1st specimen*
- *Thyrotropin [Units/volume] in Serum or Plasma –4th specimen*
- *Adenosine deaminase [Enzymatic activity/mass] in Chorionic villus sample*
- *Beta cortolone/Cortisol [Mass Ratio] in Urine*

4.1.3. Applying Patterns

By splitting the text values apart, chances are that some values will repeat themselves and thus contribute to the reduction of the amount of text that needs to be translated, as duplicates only need to be translated once. For the purpose of splitting the data apart into atomic parts, a C# application has been developed which applies patterns in a recursive manner.

The first type of patterns to be applied are common delimiters such as ;, – or :\t. A total of 18 such delimiters where implemented [64]. Among these patterns, one on them required more attention: the dot symbol, normally used to delimit sentences. Should entries have simply been split any time a dot was encountered, it would have generated unacceptable false-positives, like in E. coli [64]. To overcome this, a rule has been implemented which only performs a split on a dot symbol if the word before the dot is not an abbreviation (contains at least 4 characters) and the dot is followed by white-space [64].

All the others patterns are more complicated and require the use of regular expressions. As shown in figure 4.2, expressions split up the text into groups. In contrast to simply splitting by a delimiter, which allows the values to be recombined by joining them using the same delimiter, using regular expressions makes joining the values back more difficult.

In order to be able to rejoin the values after they have been translated, each expression is accompanied by a string template which uses %s as a placeholder for the actual values [64]. This template was chosen as it commonly encountered in the standard libraries of many languages that deal with string formatting (like the printf() function of the C standard library).

In the case of figure 4.2, the regular expression splits up 4 individual groups using mixed delimiters. Each group of text can then be individually translated. After that, the translations are reassembled using the %s [%s/%s] %s template.

A total of 25 regular expressions[2] have been used, like for example: [64]

- ^(.*)([\t]+)(PhenX)$

[2]Regular expressions form patterns which can be used to search text. The patterns are expressed with the help of characters and symbols.

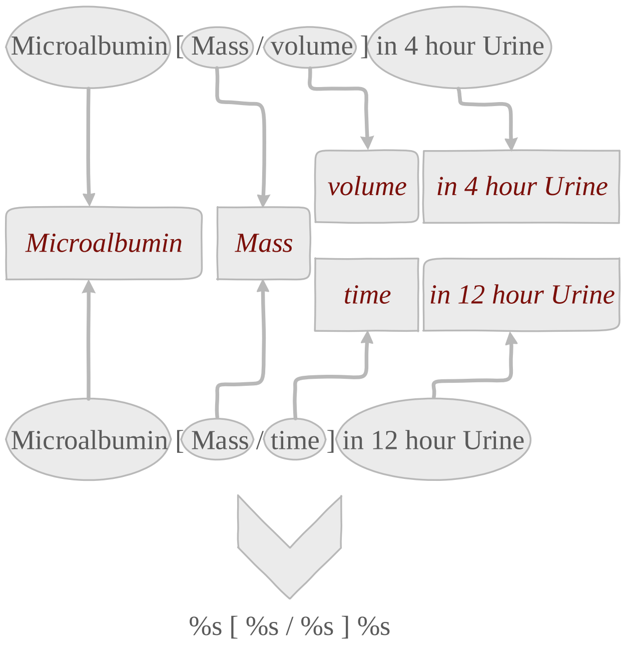

Microalbumin [Mass / volume] in 4 hour Urine
volume
in 4 hour Urine
Microalbumin
Mass
time
in 12 hour Urine
Microalbumin [Mass / time] in 12 hour Urine
%s [%s / %s] %s

Fig. 4.2: Splitting a LOINC value into patterns [64]. This shortens the average length of each time that must be individually translating, also allowing the identification of duplicates and providing a format used to reassemble the items after translation.

- ^(.*)([ ]*[\t]+)(NEGATIVE)$
- (.*) \((.*)\) (Ig[A-Z][0-9]*) ([A-Z][a-zA-Z]+)$
- ^(.*) ([0-9.]+) ([^ -:]*)/([^ -:]*)([^a-zA-Z0-9]*)$

In order to benefit the most from using these regular expressions, they are applied in a recursive manner [64]. Thus, after a pattern splits the text into, for example, 3 parts, each of these parts is further individually processed using all the patterns, often resulting into it being split into even smaller parts.

Implementing the patterns in code is simplified thanks to a programming technique called Metaprogramming [67], using the following steps: [64]

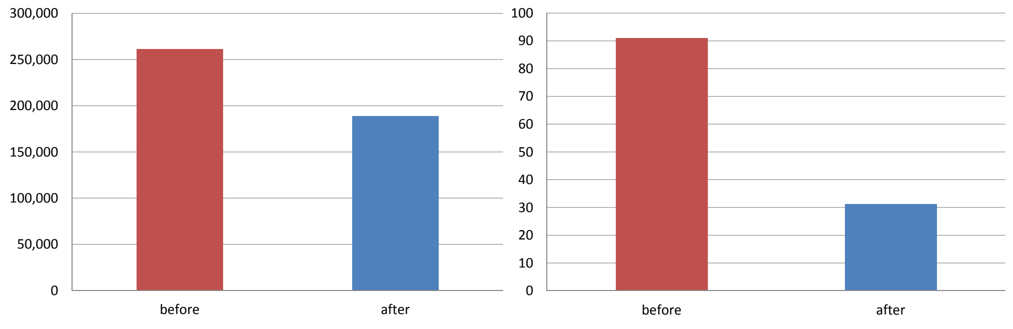
- All the patterns and regular expressions are filled into a C# code template
- Visual Studio's *TextTemplatingFileGenerator* is called to process the template and produce a C# class [68]
- The resulting class is included and compiled along with the rest of the project.

After the values have been split, another filter is applied to separate those that no longer require a translation. These values are identified by not having more than one consecutive letter (e.g. 10.5%, A10) [64].

Applying patterns in order to split the text entries into smaller, atomic parts has resulted into having fewer values that require translations, with these values also having a much lower average length. The actual numbers are presented in table 4.2. A comparison between the number of values to be translated and their average length before and after applying the split operating using the patterns is presented in figure 4.3.

Measurement	Value
Number of values that do not require translation:	7,661
Number of values to translate:	188,646
Total length of values to translate:	5,894,023
Average length of values to translate:	31.24

Table 4.2: Result of applying patterns to the LOINC data [64]



(a) String values to translate (b) Average length of string values

Fig. 4.3: Comparison of the number of values to be translated and their average length before and after applying patterns [64]

In order to check that errors have not shown up while performing the text splits, an application has been developed which reassembles all the text entries and compares them to the original LOINC data table, similar to a unit-test.

Thus, this section shows that using patterns to divide text entries into smaller parts which can be individually translated and reassembled is both doable and provides good results, reducing the amount of text that needs to be forwarded to translators.

4.1.4. Building a Web Application for Performing Translations

After having reduced the number of terms that require translation, it was time to pilot the translation process. To this end, a web application has been developed which allows users, mostly medical staff, to contribute translations.

The application relies on a very simple user interface which presents the user with a term that requires translation. The user is given three choices, as shown in figure 4.4: [64]

- *Save* – Provide a translation for the current term

- *Does not require translation* – Indicate that the current term can be used as-is without translating (perhaps it's a world recognized abbreviation)
- *Change term* – Skip translating the current term, display another one.

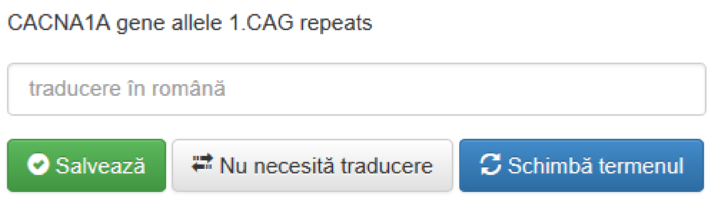
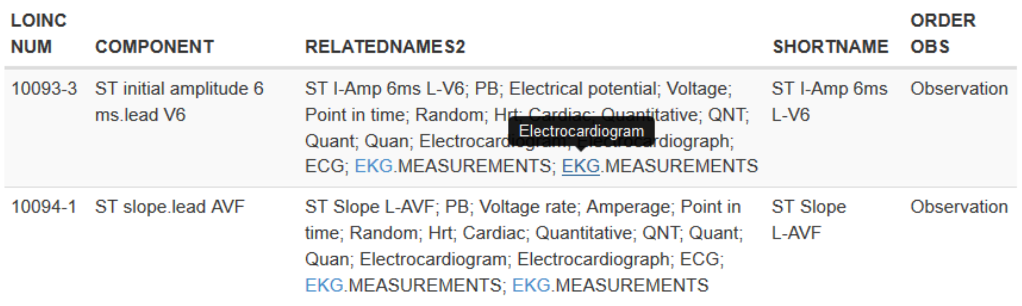


Fig. 4.4: Input area for the user's LOINC translation – localized to Romania [64]

As the terms are chosen at random, eventually, multiple users will translate the same term. This redundancy is considered a benefit as it can be used to improve precision and detect any errors. Should a translation for a term have already been submitted, it is shown to the user together with a rank indicating how many others have provided the same translation.

In order to make the users more familiar with LOINC, the web application also allows the users to browse through all the LOINC records. This is done via an interactive table displaying 50 rows/page and allowing the user to select the columns of interest. Various abbreviations present are highlighted and expanded on mouse-over, as shown in figure 4.5.



LOINC NUM	COMPONENT	RELATEDNAMES2	SHORTNAME	ORDER OBS
10093-3	ST initial amplitude 6 ms.lead V6	ST I-Amp 6ms L-V6; PB; Electrical potential; Voltage; Point in time; Random; Hrt; Cardiac; Quantitative; QNT; Quant; Quan; Electrocardiogram; Electrocardiograph; ECG; EKG.MEASUREMENTS; EKG.MEASUREMENTS	ST I-Amp 6ms L-V6	Observation
10094-1	ST slope.lead AVF	ST Slope L-AVF; PB; Voltage rate; Amperage; Point in time; Random; Hrt; Cardiac; Quantitative; QNT; Quant; Quan; Electrocardiogram; Electrocardiograph; ECG; EKG.MEASUREMENTS; EKG.MEASUREMENTS	ST Slope L-AVF	Observation

Fig. 4.5: LOINC table displayed with highlighted abbreviations [64]

The first step in the creation of the web application was to create a relational database that stores the results of applying the previously described patterns on the LOINC table. MySQL was used with the schema described by figure 4.6.

The main database table is called TableValues and is responsible for storing the content of each non-empty cell in the LOINC data table [64]. Each cell is identified by the LOINC_NUM, stored as a two-part integer and by the column index from the original LOINC table, stored also as an integer [64].

Cells that contain only non-translatable data such as numbers, dates or boolean values have the value stored in the row using a corresponding column. Cells that contain string values which where not further split up simply store a reference to the actual value found in the StringValues table.

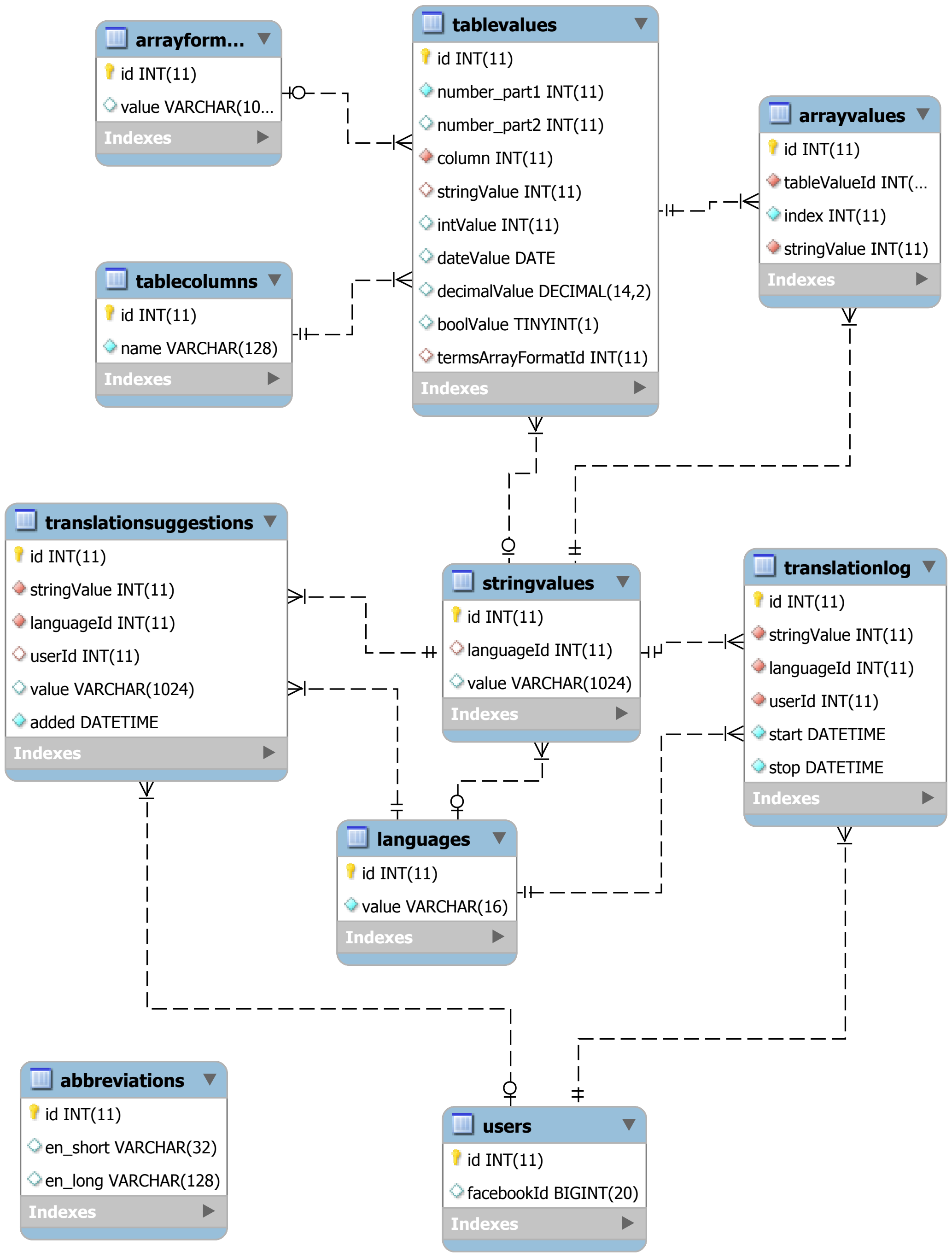


Fig. 4.6: The database schema used for the translation of LOINC terms [64]

In the case of cells that contain a value which was split using a specific pattern, the values table stores a reference to the format used to reassemble the original value and makes use of an $N-N$ relationship with the StringValues table, via the intermediate ArrayValues table, to store the actual atomic values found by the patterns [64]. All individual string values are stored in the StringValues table.

The database also contains a table called Users which is referenced by TranslationSuggestions, storing all submitted translations, and TranslationLog, storing metrics regarding when each translation was provided and an estimate on how the translation process tool [64].

For display purposes, the database also contains a table used for expanding abbreviations. The data inside was manually collected from the LOINC documentation [28].

The web application was coded using PHP as the server-side language together with their PDO classes for accessing the MySQL database in an object-oriented manner [69]. Development of the website was focused more on the client-side programming which makes use of asynchronous JavaScript with the help of jQuery in order to provide a rich and smooth client interface [70, 71]. RequireJS [72] has been used for managing client-side dependencies between software modules. By caching string values, the web client minimizes the number of server requests performed while browsing of the data table.

4.1.5. Results of Piloting the Translation

A number of 12 medical staff and students have accepted an invitation to pilot the website and translate LOINC terms into Romanian. They have translated a total of 179 terms, table 4.3 summarizing their results [64].

The time measurement is approximate and is calculate as the difference between the moment the user is shown the term to translate and the moment the translation is submitted.

Measurement	Value
Number of translators:	12
Number of terms translated:	179
Total time spent translating:	02:31:52
Average time spent translating / user:	00:12:39
Average length of translated English terms:	29.15
Average length of resulting Romanian terms:	35.46
Average time spent translating / English character:	1.74 seconds

Table 4.3: Summary of translating LOINC terms into Romanian [64]

By extrapolating, a team of 20 translators would require about 1154 hours to translate the entire LOINC table, estimating that each term would be translated by an average of 2 people [64]. This is an important advantage over simply translating all table entries individually.

4.2. Matching ICD-10 Codes Using Full-Text Search Engines

Although International Classification For Diseases (ICD) codes are freely available for download in the form of human readable documents, finding the precise code one is looking for can be a challenging task. In contrast to specialized medics which only deal with specific types of diseases and can easily get to know most common ICD codes specific to their field, trying to automatically choose the right ICD diagnostic codes from large amounts of clinical free text is more complicated.

Both automatic and manual assigned of codes can benefit from computational ways of matching ICD codes. This section presents a study performed with the goal of analyzing how existing open-source full-text search engines perform when querying the content of ICD-10 diagnostic codes. The study has resulted a in paper, published as part of the *14th International Conference on Informatics, Management and Technology in Healthcare* [73].

The study was performed on the tabular format of the *International Classification of Diseases, Tenth Revision, Clinical Modification (ICD-10-CM)* version 2016 [34]. When flattened, this document contains a number of 43 028 entries [73].

4.2.1. Analysis Methods

As shown by [74], natural language processing techniques play an important role in performing text search queries that are more complicated then finding a simple word in a sentence. Such techniques are made available for use in the form of full-text search engines, available either as stand-alone products or integrated in various databases, commercial or open-source.

One very popular open-source text engine is Apache Lucene™. Its feature richness and good integration in enterprise Java environments are being adopted in medical systems and play an important role even when it comes to retrieving medical images [75].

Using a full-text search engine involves writing a query, either in natural language or in the form of multiple words accompanied by Boolean operators, e.g. *X and (Y or Z)*. The engine then executes the query and returns a list of results composed of the actual text that was matched together with a score which indicates how well the current result matches the query. The absolute value of the score is not relevant as it differs from implementation to implementation. The score is only used for comparing/sorting results. In contrast to normal database string look-ups, full-text search engines recognize different forms of the same word and are also able to deal with stop-words [73].

The study aims to minimize the chance of wrongfully choosing between very similar ICD-10 codes. It analyses which combination of words leads to the entry in question being matched with a higher relative score than all others. Just looking at the top two results of each query is enough to identify that one result is being distinguished by a higher priority.

The analysis was performed using the following engines:

- *Apache Lucene* – a feature rich, open-source text-search engine library written in Java

- *lunr.js* – a full-text search engine written in Javascript which can run inside a web browser
- *PostgreSQL* (9.4) and *MySQL* (5.6) – relational database management systems which offer full-text search features.

Table 4.4 presents how the same query needs to be constructed for each of the 4 systems analyzed. The dedicated text engines, Apache Lucene and lunr.js provide object-oriented API's for populating the data store and executing queries. In the case of the relational databases, the query is still issued using the SQL language, although with extra non-standard constructs specific to each database engine.

As an important note, all these full-text search engines analyzed are general purpose ones. They are not specific to/optimized for medical terms.

Engine	Query
Apache Lucene	indexSearcher.search('**anemia and kidney**', 2);
Lunr.js	index.search('**anemia and kidney**');
PostgreSQL	SELECT id, ts_rank(search, to_tsquery('**anemia & kidney**')) AS rank FROM entries_indexed WHERE search @@ to_tsquery('**anemia & kidney**') ORDER BY rank DESC LIMIT 2
MySQL	SELECT id, MATCH(text) AGAINST('**anemia kidney**' IN NATURAL LANGUAGE MODE) AS rank FROM icd10.entries ORDER BY rank DESC LIMIT 2

Table 4.4: Example queries for D63.1 – Anemia in chronic kidney disease [73]

4.2.2. Results

The full-text search engine of PostgreSQL requires that the text to be searched be split into a data structure called *tsvector*, a "sorted list of distinct lexemes, which are words that have been normalized to merge different variants of the same word" [76].

Performing such a split provides valuable insight in the way a full-text search engine internally deals with the text. Analyzing these lexemes reveals which are the most common words used in the ICD-10-CM entries, as can be seen in table 4.5 [73].

The total number of lexemes found is 6 561 with 2 202 ($\sim$33%) of them only encountered once and "unspecifi" appearing most often, in 10 730 ($\sim$24%) of all entries [73]. On average, each ICD-10-CM entry contains 5.33 lexemes [73].

Because the number of entries in ICD-10-CM is quite substantial, an initial analysis was first performed on a list of common codes as identified by [77]: 456 entries in total, out of which 437 contain more than one word. The analysis involves extracting the words that compose the description of each code and generating all combinations of these words. These combinations are then used to construct queries which are executed against the full-text engine. Each query returns which entries it matches, together with a numeric score. For the combination of words to be considered

Lexeme	Count	Lexeme	Count
unspecifi	11664	Hand	1482
Left	4870	Bodi	1478
Right	4797	Level	1458
fractur	2390	Due	1430
Injuri	2212	Foot	1406
disord	1938	muscl	1340
specifi	1704	without	1316
lower	1636	effect	1306
eye	1570	injur	1279
diseas	1500	neoplasm	1217

Table 4.5: Most frequently encountered lexemes in ICD-10-CM (computed from data found in [34]): *unspecifi*, obtained from *unspecified*, appears most often as it is used at the end of most hierarchies as a placeholder for diagnostics that don't exactly match any of the other entries.

successful, it should match the current entry in question with the highest score, a score higher than that assigned to any other entry.

For each common ICD-10-CM entry (excluding those that are only based on one word), the analysis searches for the minimum amount of words that, when combined in a search query, cause the full-text search engines to list the desired entry at the top of the search results, with a greater score than any of the other entries [73]. Because the score provided for each result by the full-text search engines is a floating-point value, they were rounded to only 2 decimal places for comparison purposes.

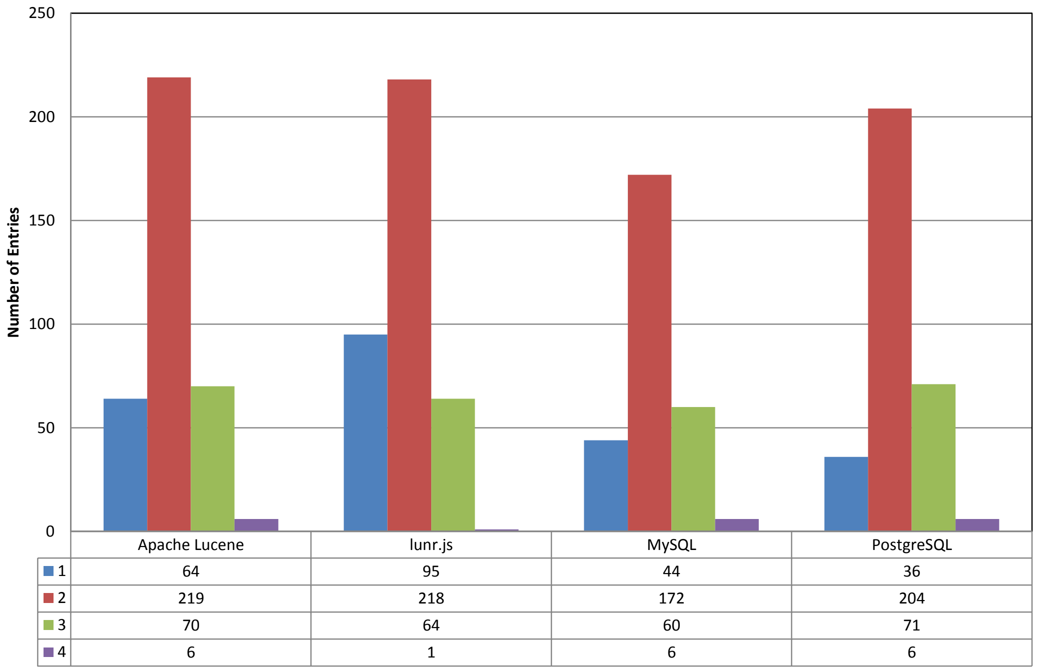
The results are presented in figure 4.7a. On average, about two words are needed by the search engines to isolate each entry. For example, in the case of *R25.2 – Cramp and spasm*, both "cramp" and "spasm" must be provided [73].

There are however cases of highly similar entries where the engines fail to identify a distinct match and end up assigning the same score to multiple multiple entries, even if all the words of the entries are used to form the query. An example of this is *M62.838 – Other muscle spasm* and its parent entry *M62.83 – Muscle spasm* [73]. Table 4.6 details how many entries failed to be distinctly identified.

Engine	Indistinct Matches	
	Number	Percent
Apache Lucene	78	18 %
Lunr.js	59	14 %
MySQL	155	35 %
PostgreSQL	120	27 %

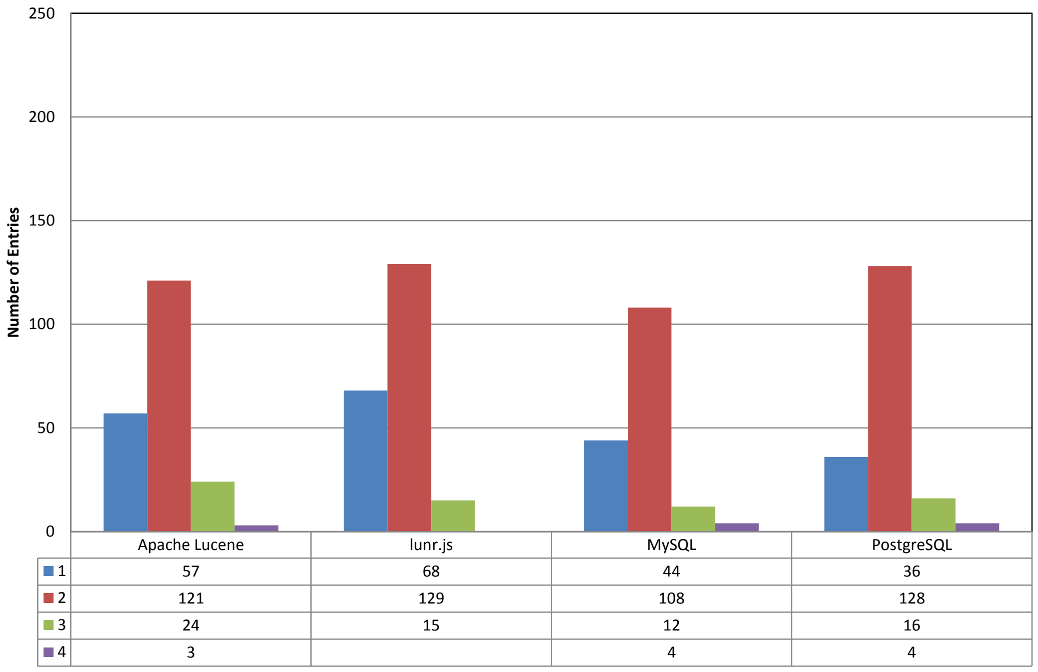
Table 4.6: Number of common entries not distinctly identified

A further analysis on the number of common entries that could not be distinctly identified reveals that 131 (∼78%) contain either the word "other" (∼32%)



	Apache Lucene	lunr.js	MySQL	PostgreSQL
🟦 1	64	95	44	36
🟥 2	219	218	172	204
🟩 3	70	64	60	71
🟪 4	6	1	6	6

(a) Including entries that contain *unspecified* or *other*



	Apache Lucene	lunr.js	MySQL	PostgreSQL
🟦 1	57	68	44	36
🟥 2	121	129	108	128
🟩 3	24	15	12	16
🟪 4	3		4	4

(b) Excluding entries that contain *unspecified* or *other*

Fig. 4.7: Amount of entries together with the minimum number of words (left) required for distinct matches in the case of common entries

or “unspecified” (∼48%). In the case of 43 entries, none of the engines was able to produce a distinct match [73].

Based on these findings, the analysis was performed again, but only on entries that do not contain the words “unspecified” or “other”. As shown in figure 4.7b, this has lead to a decrease in the number of entries that require a query composed of at least 3 words.

After performing the initial analysis on only a small subset of common ICD-10 entries, it was time to expand it on all entries. This analysis ran into the following issues with introduces some limitations in the scope of the analysis:

- Some ICD-10 entries contained a lot of words. This lead to a big increase in the number of unique combinations that needed to be checked for each entry, slowing down the analysis considerably. As such, entries with more than 15 words were skipped.
- The MySQL search was very slow for full-text searches, even after tuning the database configuration parameters. After ten hours of running continuously on the test machine, the progress was weak so it was skipped.
- Entries containing just one word are not counted as they trivially match.

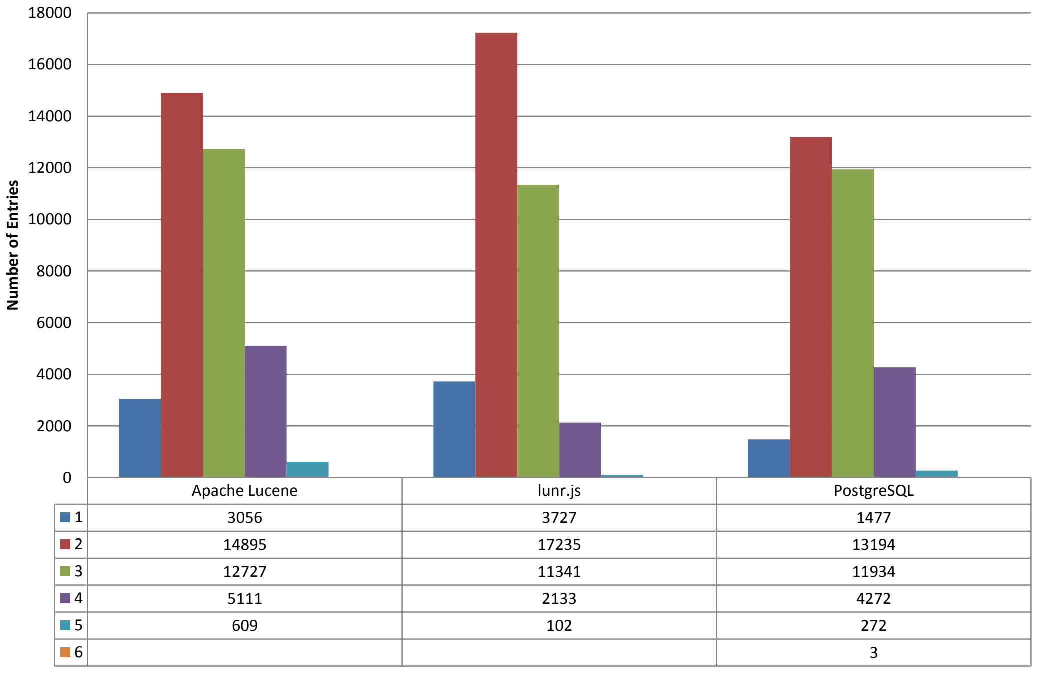
Engine	Average number of words needed with/without “unspecified” or “other”	
	with	without
Apache Lucene	2.60	2.38
Lunr.js	2.35	2.24
PostgreSQL	2.64	2.49

Table 4.7: Average number of words needed/match

Engine	Indistinct Matches with/without “unspecified” or “other”	
	with	without
Apache Lucene	14 %	18 %
Lunr.js	18 %	21 %
PostgreSQL	26 %	32 %

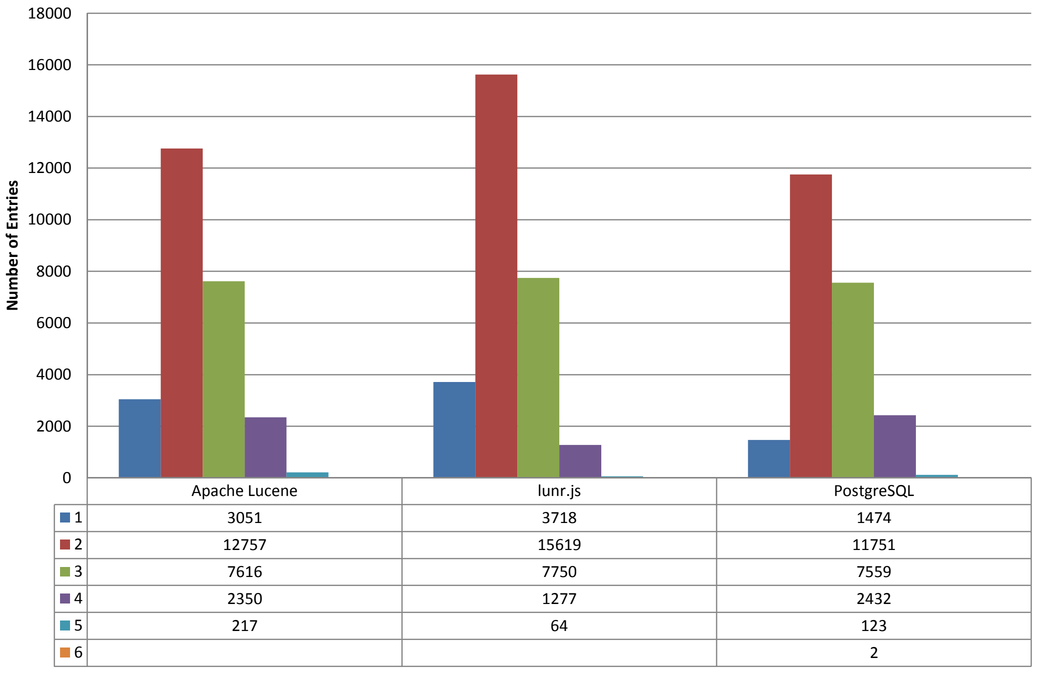
Table 4.8: Number of common entries not distinctly identified

As shown in figure 4.7, the results of Apache Lucene and PostgreSQL are quite close, with an average of ∼2.60 words needed/match in the case of Apache Lucene and ∼2.64 words/match for PostgreSQL. Lunr.js seems to perform better and requires an average of only ∼2.35 words/match. Excluding the entries that contain “unspecified” or “other” reduces these averages.



	Apache Lucene	lunr.js	PostgreSQL
🟦 1	3056	3727	1477
🟥 2	14895	17235	13194
🟩 3	12727	11341	11934
🟪 4	5111	2133	4272
🟦 5	609	102	272
🟧 6			3

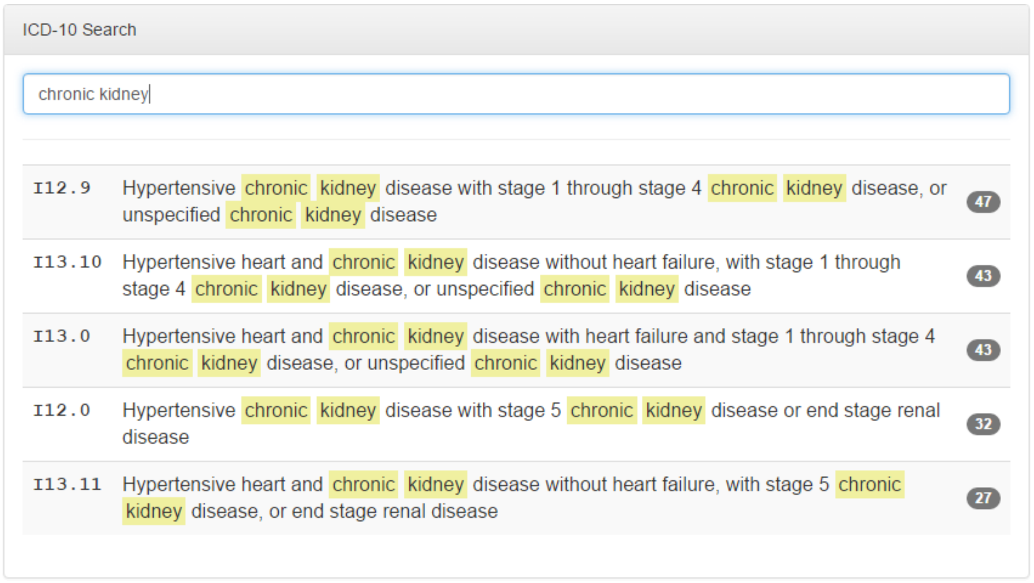
(a) Including entries that contain *unspecified* or *other*



	Apache Lucene	lunr.js	PostgreSQL
🟦 1	3051	3718	1474
🟥 2	12757	15619	11751
🟩 3	7616	7750	7559
🟪 4	2350	1277	2432
🟦 5	217	64	123
🟧 6			2

(b) Excluding entries that contain *unspecified* or *other*

Fig. 4.8: Amount of entries together with the minimum number of words (left) required for distinct matches in the case of all entries which contain between 2 and 15 words


ICD-10 Search

chronic kidney

I12.9	Hypertensive chronic kidney disease with stage 1 through stage 4 chronic kidney disease, or unspecified chronic kidney disease	47
I13.10	Hypertensive heart and chronic kidney disease without heart failure, with stage 1 through stage 4 chronic kidney disease, or unspecified chronic kidney disease	43
I13.0	Hypertensive heart and chronic kidney disease with heart failure and stage 1 through stage 4 chronic kidney disease, or unspecified chronic kidney disease	43
I12.0	Hypertensive chronic kidney disease with stage 5 chronic kidney disease or end stage renal disease	32
I13.11	Hypertensive heart and chronic kidney disease without heart failure, with stage 5 chronic kidney disease, or end stage renal disease	27

Fig. 4.9: Example of an ICD-10 search form showing results in descending order based on the matching score

Analyzing the number of entries that could not be distinctly matched revealed that Apache Lucene performed the best with only ∼14% non-distinct matches as opposed to ∼18% in the case of lunr.js. PostgreSQL was unable to produce a distinct match in about twice as many number of entries than Apache Lucene. The results are summarized in table 4.8. In the case of 2 376 entries was neither of the engines able to produce a distinct match.

The analysis shows that existing open-source full-text search engines provide good results when searching the ICD-10 codes of medical diagnostics, in English [73]. With an average of about two and a half words required per query, the users' effort is reduced and so is the chance of choosing the wrong code [73]. The biggest factor which explains why some entries were not being distinguishable from others was found to be the presence of the words "unspecified" and/or "other".

The engine written in JavaScript, lunr.js, shows good results. Taking into account that the list of ICD-10-CM codes is only a few megabytes large, this opens the possibility of performing the search in the client-side code, providing a richer user-experience by reducing the delay caused by calls to the server [73]. Figure 4.9 presents a user interface in which the user is querying ICD-10 codes by using a natural language and is displayed results sorted in descending order by their score.

4.3. Achieving Interoperability

This section discusses the implications of taking an electronic health records solution which is not designed with specific international standards in mind and trying to make it interoperable with solutions using openEHR or HL7 CDA.

Designing a solution fully according to the openEHR methodology requires dedicated developers that are familiar with it's approach and which might not be available in a team that is only requested to implement a specific subset of EHR functionality.

The prototype solution described in chapter 3 is an excellent candidate for analyzing what small changes would be necessary to bring it on the right track of medical interoperability. The solution is designed to solve a specific use case, registering health records of patients with chronic illness, in a simple manner.

As forms can easily change, the prototype described takes input forms and it's constituent elements into account and models them in a reusable manner. This is in contrast with simply adding a new column for each new type of data, a practice which can result in large, inefficient and more and more difficult to work with databases.

4.3.1. Linking openEHR archetypes

Even if an electronic health record solution is not built around the methodology described by openEHR, accessing openEHR's rich and freely available archetypes still provides important advantages both when building a new system and when integrating it with others:

- Archetypes are meant to be created by domain experts which have a much better understanding of medical terms than developers
- A lot of archetypes are already available at http://www.openehr.org/ckm/
- Translations into multiple languages are available
- Where appropriate, archetypes contain bindings to standard medical terminologies.

Ocean Informatics provides an application called *Archetype Editor* which enables domain experts to create and edit archetypes, as shown in figure 4.10. It supports both the Archetype Definition Language (ADL) and it's XML representation.

An example archetype relating to blood pressure is openEHR-EHR-OBSERVATION.blood_pressure.v1. It defines the systolic and diastolic measures as:

```
ELEMENT[at0004] occurrences matches {0..1} matches {  -- Systolic
   value matches {
      C_DV_QUANTITY <
         property = <[openehr::125]>
         list = <
            ["1"] = <
               units = <"mm[Hg]">
               magnitude = <|0.0..<1000.0|>
               precision = <|0|>
            >
         >
      >
   }
}
ELEMENT[at0005] occurrences matches {0..1} matches {  -- Diastolic
```

```
  value matches {
    C_DV_QUANTITY <
      property = <[openehr::125]>
      list = <
        ["1"] = <
          units = <"mm[Hg]">
          magnitude = <|0.0..<1000.0|>
          precision = <|0|>
        >
      >
    >
  }
}
```

The full paths to the actual quantities are data[at0001]/events[at0006] /data[at0003]/items[at0004]/value/magnitude and data[at0001]/events[at0006] /data[at0003]/items[at0005]/value/magnitude with the following English translations:

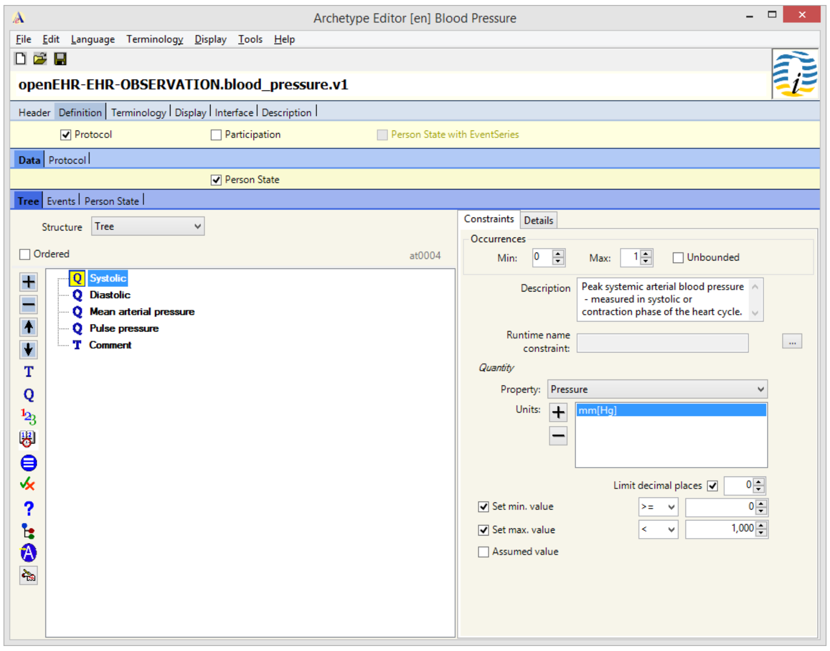
- at0001 – History
- at0006 – Any event
- at0003 – Blood pressure
- at0004 – Systolic
- at0005 – Diastolic

Because the database schema presented in section 3.3.3 is designed around input element semantics, these can easily be extended by adding columns that link them to openEHR archetypes. Three columns are necessary: the archetype id, the archetype version and the path to the value, as shown in figure 4.11.

As in the case of blood pressure, openEHR archetypes usually include limits for quantity values. Applying such limits would normally be implemented as constraints in the database. However, because such constraints affect all rows, implementing them is not that straight-forward.

The official documentation regarding the Archetype Definition Language [42] lists multiple ways of expression constraints for integer or real numbers: length matches |0>..<1000| (allow 0> x <1000), length matches |100+/-5| (allow 100 +/- 5, i.e. 95 - 105). These inputs can be expressed as intervals or lists of discrete values. The allowed values for the diastolic blood pressure, expressed in ADL syntax as magnitude = <|0.0..<1000.0|> are expressed in the XML representation of the archetype as an interval structure:

```
<magnitude>
  <lower_included>true</lower_included>
  <upper_included>false</upper_included>
  <lower_unbounded>false</lower_unbounded>
  <upper_unbounded>false</upper_unbounded>
  <lower>0.0</lower>
  <upper>1000.0</upper>
</magnitude>
```


Archetype Editor [en] Blood Pressure
File Edit Language Terminology Display Tools Help
Header Definition Terminology Display Interface Description
Data Protocol
Tree Events Person State
Constraints Details

openEHR-EHR-OBSERVATION.blood_pressure.v1

[✓] Protocol [ ] Participation [ ] Person State with EventSeries

[✓] Person State

Structure: Tree

[ ] Ordered at0004

- Systolic
- Diastolic
- Mean arterial pressure
- Pulse pressure
- Comment

Occurrences
Min: 0 Max: 1 [ ] Unbounded

Description: Peak systemic arterial blood pressure - measured in systolic or contraction phase of the heart cycle.

Runtime name constraint:

Quantity

Property: Pressure

Units: mm[Hg]

Limit decimal places [✓] 0

[✓] Set min. value >= 0

[✓] Set max. value < 1,000

[ ] Assumed value

Fig. 4.10: Using the Archetype Editor developed by Ocean Informatics to edit the *blood pressure* archetype [78]

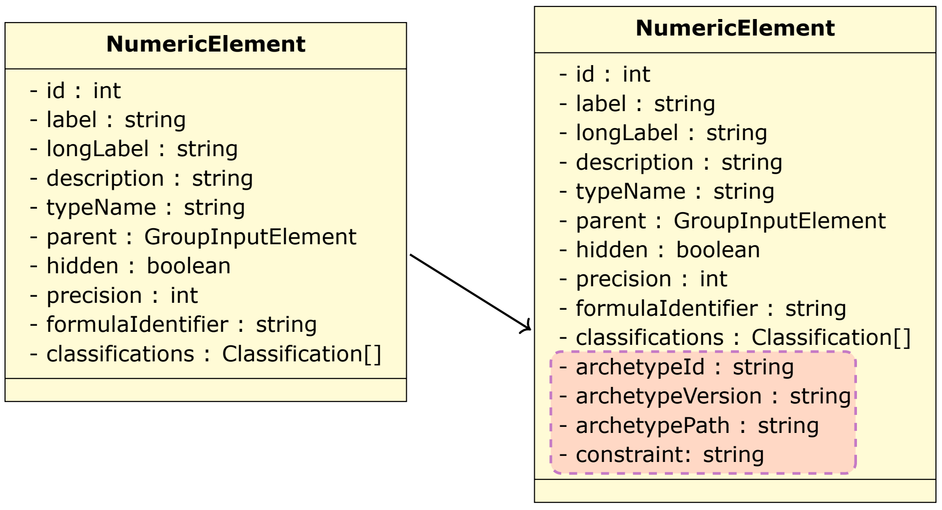


Fig. 4.11: Expanding a numeric input element with information necessary to link it to an openEHR archetype. The additional information required includes only the archetype's id, version and path. An optional constraint string ca also be used to enforce medically relevant values.

As shown in figure 4.11, the constraint can be stored as a string property of the entity. As the ADL syntax is well documented and easily parsed, the constraint could be stored in the exact same way as in an archetype. Evaluating and applying the constraint is a task for business logic in the application, and not, although tempting, in the database. By not writing stored procedures or functions that execute code in the database context, the solution stays free from being locked up in a specific vendor's technology and is also more easily migrated to different data stores, like NoSQL, which do not offer such programmatic features.

The major advantage of creating a link between an EHR system and openEHR archetypes is the ability to respond to Archetype Query Language (AQL) queries with little effort. A query which requests the systolic and diastolic blood pressures that exceeds a specific amount looks very similar to a SQL query, as shown in figure 4.12.

	SELECT
Archetype path	obs/data[at0001]/events[at0006]/data[at0003] /items[at0004]/value/magnitude **AS** systolic,
Archetype path	obs/data[at0001]/events[at0006]/data[at0003] /items[at0005]/value/magnitude **AS** diastolic
	FROM
EHR id	**EHR** [ehr_id/value='1234']
Archetype predicates indicating what archetype instances are relevant to this query	**CONTAINS COMPOSITION** [openEHR-EHR-COMPOSITION.encounter.v1] **CONTAINS OBSERVATION** obs [openEHR-EHR-OBSERVATION.blood_pressure.v1]
	WHERE
Condition	systolic >= 140 **OR** diastolic >= 90

Fig. 4.12: Archetype query requesting blood pressure over a specific amount [79]

Such a query is easily converted to a SQL query which returns information from the EHR database. Thus, the party that issued the archetype query is not required to have any knowledge of the specific schema used by the EHR repository. This allows the schema to evolve freely, without breaking existing clients.

```
SELECT *
FROM (
  SELECT
  CASE [InputElementID] WHEN 95001 THEN [DecimalValue] ELSE 0 END AS systolic,
  CASE [InputElementID] WHEN 95002 THEN [DecimalValue] ELSE 0 END AS diastolic
  FROM [InputElementValues]
  WHERE [InputFormValueID] = 1234
  ) AS temp
WHERE systolic >= 140 OR diastolic >= 90
```

4.3.2. Linking Medical Terminologies

Linking EHR data to well known and commonly used medical terminologies is an important step towards achieving interoperability. Medical terminologies map various data such as laboratory results or disease diagnostics to simple codes which allow other entities to easily recognize the semantics of the values, independent of the language used.

The prototype solution described in chapter 3 records individual forms in a table called *InputFormValues* which is linked through a parent-child relationship to *InputElementValues* which stores all the entries recorded for each form for a patient at a specific time. As shown in figure 4.13, linking the values to a specific terminology requires the addition of two extra field in the entity/two extra columns in the database: the terminology identifier (LOINC, ICD-10, SNOMED, ...) and the terminology code to which the value is mapped.

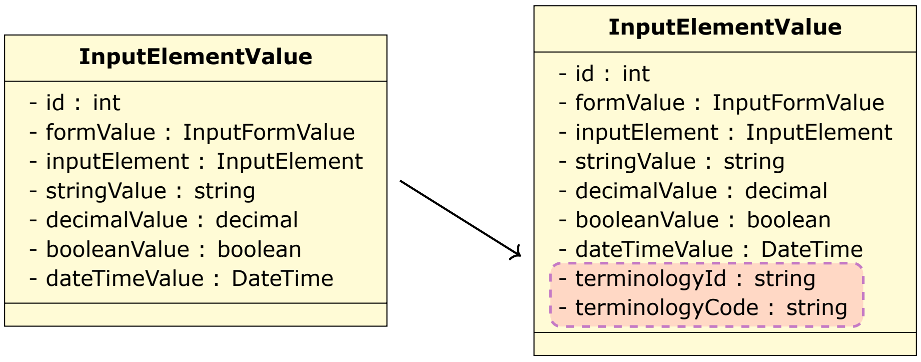


Fig. 4.13: Expanding an input element value with information necessary to link it to a medical terminology. The only additional fields are the terminology's id and code.

There is a reason for choosing to store the terminology codes for each value instead of only storing them on the input elements: some terminologies are very specific about the circumstances in which a value is obtained, allocating multiple codes for distinguishing on the way a sample is taken, where it was taken from and even at what moment in time. LOINC is such an example, allocating quite a few codes just related to the systolic blood pressure, as show in table 4.9.

Performing the actual mapping requires writing code that handles the specifics of each terminology entry that the system is to support. Because this issue is not part of the core functionality of the EHR system (it was doing fine recording health records even without dealing with standard terminologies), it can be viewed as a cross-cutting concern, similar to logging functionality found in most system. This is important because it allows using techniques such as meta-programming to introduce the feature of mapping to terminologies in legacy systems that might otherwise be to complex or fragile to allow such an extension.

A programming technique called Aspect-Oriented Programming (AOP) enables the separation of general code from aspects that cross the boundaries of a layer or object [81]. Aside from simple use cases like implementing logging throughout an application, the main strength of AOP is that it allows the developer to change the behavior of methods. Technically, AOP implementations rely on: [81]

- Pre-processing the source code
- Post-processing to add instructions to the compiled binary code

LOINC_NUM	COMPONENT	SYSTEM
11378-7	Intravascular systolic	Arterial system
20185-5	Intravascular end systolic.XXX	Circulatory system.XXX
20186-3	Intravascular peak systolic.XXX	Circulatory system.XXX
24370-9	Intravascular peak systolic^during MR max vel measurement	Arterial system.XXX
50402-7	Blood pressure systolic & diastolic^post transfusion	Arterial system
50403-5	Blood pressure systolic & diastolic^pre transfusion	Arterial system
8420-2	Intravascular systolic	Aorta.abdominal.distal
8421-0	Intravascular systolic	Aorta.abdominal.proximal
8422-8	Intravascular systolic	Aorta.thoracic.proximal ascending
8423-6	Intravascular systolic	Aorta.thoracic.ascending
8424-4	Intravascular systolic	Aorta.thoracic.descending
8430-1	Intrachamber systolic	Heart.ventricle.left
8431-9	Intrachamber systolic	Heart.ventricle.left.outflow tract
8432-7	Intrachamber systolic	Heart.ventricle.right

Table 4.9: Example LOINC codes related to the systolic blood pressure [80]

- Special compilers
- Code interception at run-time

AOP has been successfully employed in healthcare systems, as demonstrated by the Healthlink project in Ireland where developers use the AOP approach to implement creating and parsing HL7 messages [82]. The design aims to make use of XML DOM handler classes and introduce the required behavior at appropriate execution points in the base application. A comparison performed on an application called *HL7Browser* shows a reduction in size by 70% (lines of class code) after refactoring the HL7 functionality, which was scattered throughout the application, to use AOP [82].

In order to map a newly added health record values to a terminology, an approach would be to intercept the repository method that is responsible for storing the data. Upon interception, the entire form data is available and is passed to the library dealing with finding the right terminology codes. After the codes are found, the library populates the corresponding fields of the *InputElementValue* class and returns, allowing the store method to pass updated class instances to the ORM layer for storing in the database. Figure 4.14 shows the sequence of events.

By linking openEHR archetypes, one gets the benefit of being to use the archetype *term_bindings*. Parsing archetypes from an openEHR repository (https://github.com/openEHR/adl-archetypes) has resulted in identifying multiple terminology codes, mostly SNOMED, mapped to available archetypes, as displayed

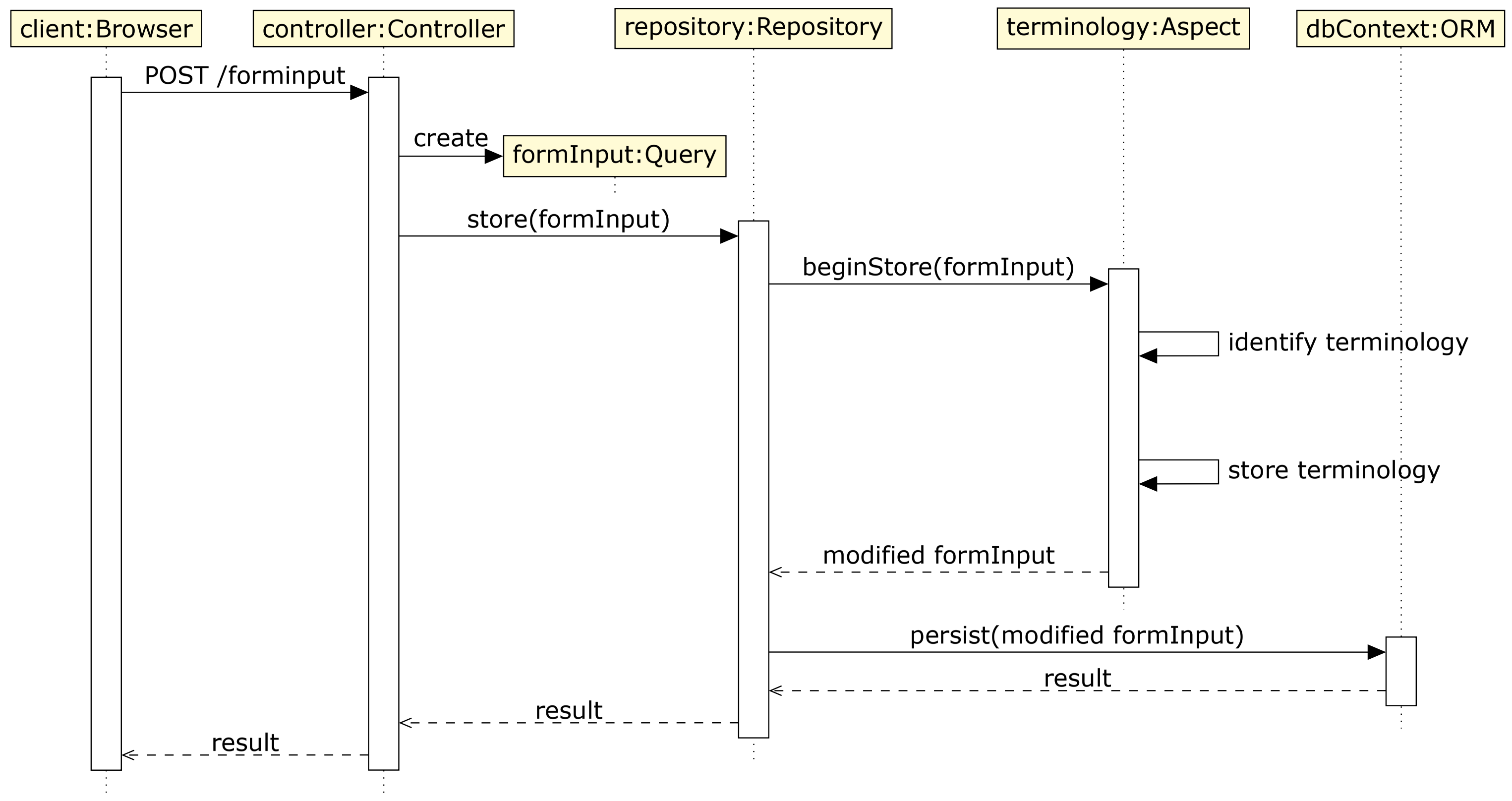


Fig. 4.14: Applying AOP inside the repository for matching terminologies. When persisting a form input, the repository has no knowledge of terminology. This is injected into the entity to be persisted with the help of aspect-oriented programming which allows intercepting the call inside the repository, just before the entity is passed to the database context.

in table 4.10.

4.3.3. Clinical Document Architecture

D. Baranov has developed an open-source library for creating, parsing and storing clinical documents, called HL7 SDK and available under Microsoft .NET and COM technologies [83]. It provides full support for the HL7 CDA Release 2 standard and was creating mostly by means of code generation from the HL7 CDA R2 XML schema [83].

The HL7 SDK provides classes that express the structure of HL7 CDA documents. The classes can be created/composed manually and then serialized as an XML document which becomes a valid CDA document, or they can be imported automatically by parsing an XML CDA document. The advantages of using such a library when dealing with CDA documents are plenty:

- Compile time checks for any errors in the naming of elements
- No need to process XML by hand
- No need to constantly check XML schemas in order to figure out the details of the document structure.

Figure 4.15 shows an example of how a section of a CDA document, which records *Asthma* as part of the *Past Medical History*, is represented using HL7 SDK classes.

The common approach, in the case of openEHR solutions, to outputting documents of different standards, like HL7 CDA, is to make use of openEHR templates. Figure 4.16 shows Ocean Informatic's (an openEHR vendor) approach to using openEHR to produce CDA R2 artifacts. By means of XSL transforms they manage to generate CDA or CCD documents.

A more generic approach, similar to using openEHR templates, but not restricting the usage to a system that is built around openEHR, involves the creation of *connectors*, as illustrated by figure 4.17. Connectors, in this scenario, are software components built specifically to consume or generate HL7 CDA documents that are coming from or are destined for external systems. These documents either contain external medical data that is to be inputted into the current system or exported information from the current database for external destinations.

Setting of to create connectors for an arbitrary EHR system must account for the following:

- A good understanding of the current system, especially software modules which deal with the underlying database and whether or not these modules are friendly to extensions
- Solid requirements on what data needs to be exchanged
- Identifying medical terminologies that will be used
- A good understanding of the HL7 CDA standard

LOINC	LA120-8	No limitation	SNOMED-CT	60621009	Body Mass Index	SNOMED-CT	271922009	Overall Interpretation
LOINC	LA6742-6	Bedfast	Snomed	61490001	Class I	SNOMED-CT	272741003	Side
LNC205	8310-5	History/Any event/Tree /Temperature	SNOMED-CT	68027008	Modification	SNOMED-CT	273903006	Visual Analogue Score
LOINC	LA8913-1	Adequate	Snomed	85284003	Class III	SNOMED-CT	362502000	Right eye
LOINC	LA9206-9	Excellent	Snomed	89323001	Class IV	SNOMED-CT	362503005	Left eye
LOINC	LA9603-7	Completely limited	SNOMED-CT	113147002	Direct Light Reflex	SNOMED-CT	363953003	Measured Size
LOINC	LA9605-4	Very limited	SNOMED-CT	117364006	Description	SNOMED-CT	363955005	Symmetry
LOINC	LA9605-2	Slightly limited	SNOMED-CT	128975004	At rest	SNOMED-CT	363983007	Interpretation
LOINC	LA9606-0	No impairment	SNOMED-CT	128976003	During exertion	SNOMED-CT	366060000	Overall Interpretation
LOINC	LA9607-8	Constantly moist	SNOMED-CT	128978002	Post-exertion	SNOMED-CT	370996005	Has resolved
LOINC	LA9608-6	Very moist	Snomed	134438001	Angina symptom Classification (CCS)	SNOMED-CT	372278000	Total Gleason score
LOINC	LA9609-4	Occasionally moist	SNOMED-CT	162408000	Clinical Description	SNOMED-CT	384994009	Primary Gleason grade
LOINC	LA9610-2	Rarely moist	SNOMED-CT	162442009	Duration	SNOMED-CT	384995005	Secondary Gleason grade
LOINC	LA9611-0	Chairfast	SNOMED-CT	162465004	Severity	SNOMED-CT	385002007	Tertiary Gleason grade
LOINC	LA9612-8	Walks occasionally	SNOMED-CT	162466003	Trivial	SNOMED-CT	385633008	Improving
LOINC	LA9613-6	Walks frequently	SNOMED-CT	162468002	Mild	SNOMED-CT	392017002	Frequency Doubling Perimetry (FDP)
LOINC	LA9614-4	Completely immobile	SNOMED-CT	162469005	Moderate	SNOMED-CT	392132006	FASTPAC automated standard perimetry
LOINC	LA9615-1	Very poor	SNOMED-CT	162470006	Severe	SNOMED-CT	397258008	Interpupillary Distance
LOINC	LA9616-9	Probably inadequate	SNOMED-CT	162471005	Very severe	SNOMED-CT	397282003	Reading Addition Power
LOINC	LA9617-7	Problem	SNOMED-CT(2003)	163020007	Blood Pressure	SNOMED-CT	397524001	Retinoscopy
LOINC	LA9618-5	Potential problem	SNOMED-CT(2003)	163030003	Systolic	SNOMED-CT	400913005	Snellen chart
LOINC	LA9619-3	No apparent problem	SNOMED-CT(2003)	163031004	Diastolic	SNOMED-CT	400914004	ETDRS chart
LOINC	38222-6	Sensory perception	SNOMED-CT	230993007	Worsening	SCT	413139004	BradenScale
LOINC	38223-4	Activity	SNOMED-CT(2003)	246153002	Cuff size	SNOMED-CT	419161000	Left
LOINC	38224-2	Mobility	SNOMED-CT	246223004	Prism Base Direction	SNOMED-CT	419465000	Right
LOINC	38225-9	Nutrition	SNOMED-CT	251762001	Prism Strength	Snomed	420300004	Class I
LOINC	38226-7	Friction and shear	SNOMED-CT	251795007	Power of Sphere	Snomed	420816009	NYHA Heart failure classification
LOINC	38227-5	Braden total score	SNOMED-CT	251797004	Power of Cylinder	Snomed	420913000	Class III
LOINC	38229-1	Moisture	SNOMED-CT	251799001	Axis of Cylinder	SNOMED-CT	421140005	Visual Field Index
LOINC	39156-5	Body Mass Index	SNOMED-CT	251802005	Intermediate Distance Power	SNOMED-CT	421362004	Pattern Standard deviation
SNOMED-CT	19019007	Symptom	SNOMED-CT	251894003	Exercise intensity	Snomed	421704003	Class II
SNOMED-CT	22253000	Pain	SNOMED-CT	252124009	Testing Distance	SNOMED-CT	422256009	CF - Count fingers
Snomed	41334000	Class II	SNOMED-CT	252806005	Humphrey	SNOMED-CT	422673001	US Snellen
SNOMED-CT	50121007	Spectacles	SNOMED-CT	252886007	Refraction assessment	SNOMED-CT	422859007	Mean deviation
SNOMED-CT	57368009	Contact lenses	SNOMED-CT	260369004	Increasing	SNOMED-CT	423083007	Glaucoma Hemifield Test (GHT)
SNOMED-CT	57465006	Precipitating factors	SNOMED-CT	260371004	Decreasing	Snomed	443428004	Braden total score
SNOMED-CT	58158008	Stable	SNOMED-CT	260908002	Course			

Table 4.10: Terminology found in archetypes from the openEHR repository https://github.com/openEHR/adl-archetypes

```
<observation classCode="COND" moodCode="EVN">
   <code code="195967001" codeSystem="2.16.840.1.113883.6.96" codeSystemName="SNOMED CT" displayName="Asthma">
      <originalText>
         <reference value="#a1"/>
      </originalText>
   </code>
   <statusCode code="completed"/>
   <effectiveTime value="1950"/>
   <reference typeCode="XCRPT">
      <externalObservation>
         <id root="2.16.840.1.113883.19.1.2765"/>
      </externalObservation>
   </reference>
</observation>
```

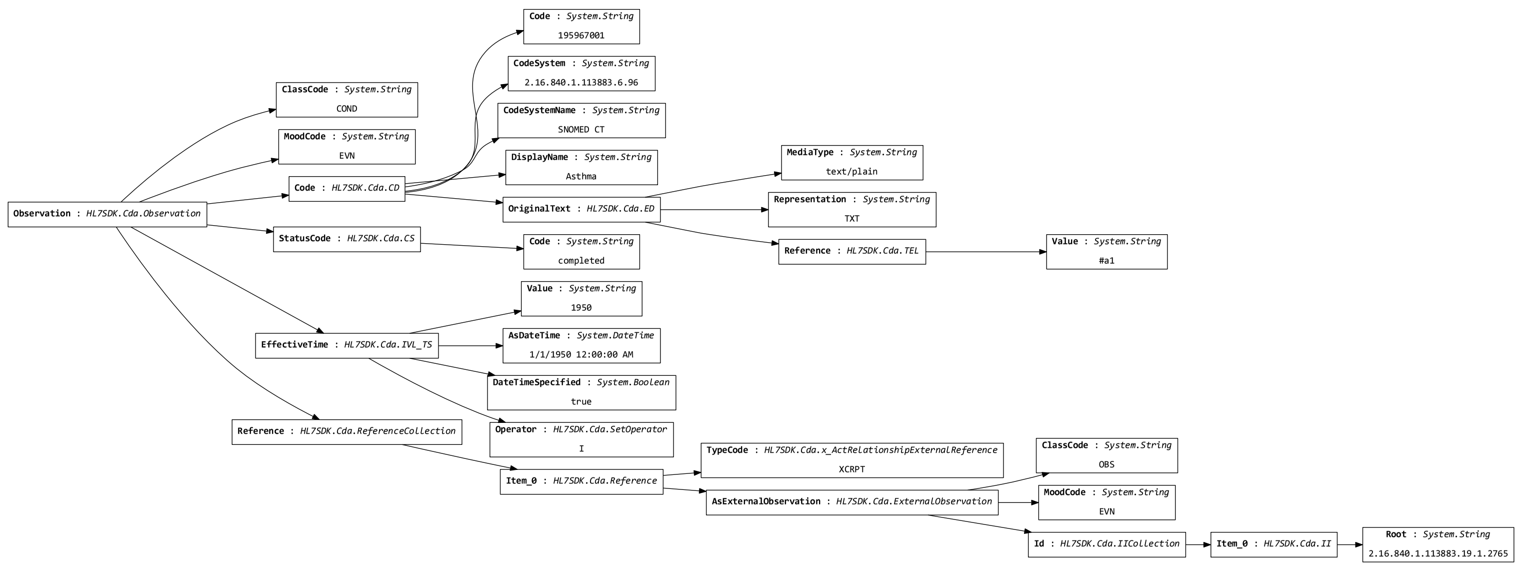

Observation : HL7SDK.Cda.Observation
ClassCode : System.String
COND
MoodCode : System.String
EVN
Code : HL7SDK.Cda.CD
Code : System.String
195967001
CodeSystem : System.String
2.16.840.1.113883.6.96
CodeSystemName : System.String
SNOMED CT
DisplayName : System.String
Asthma
OriginalText : HL7SDK.Cda.ED
MediaType : System.String
text/plain
Representation : System.String
TXT
Reference : HL7SDK.Cda.TEL
Value : System.String
#a1
StatusCode : HL7SDK.Cda.CS
Code : System.String
completed
EffectiveTime : HL7SDK.Cda.IVL_TS
Value : System.String
1950
AsDateTime : System.DateTime
1/1/1950 12:00:00 AM
DateTimeSpecified : System.Boolean
true
Operator : HL7SDK.Cda.SetOperator
I
Reference : HL7SDK.Cda.ReferenceCollection
Item_0 : HL7SDK.Cda.Reference
TypeCode : HL7SDK.Cda.x_ActRelationshipExternalReference
XCRPT
AsExternalObservation : HL7SDK.Cda.ExternalObservation
ClassCode : System.String
OBS
MoodCode : System.String
EVN
Id : HL7SDK.Cda.IICollection
Item_0 : HL7SDK.Cda.II
Root : System.String
2.16.840.1.113883.19.1.2765

Fig. 4.15: Example object hierarchy using the HL7 SDK [83]

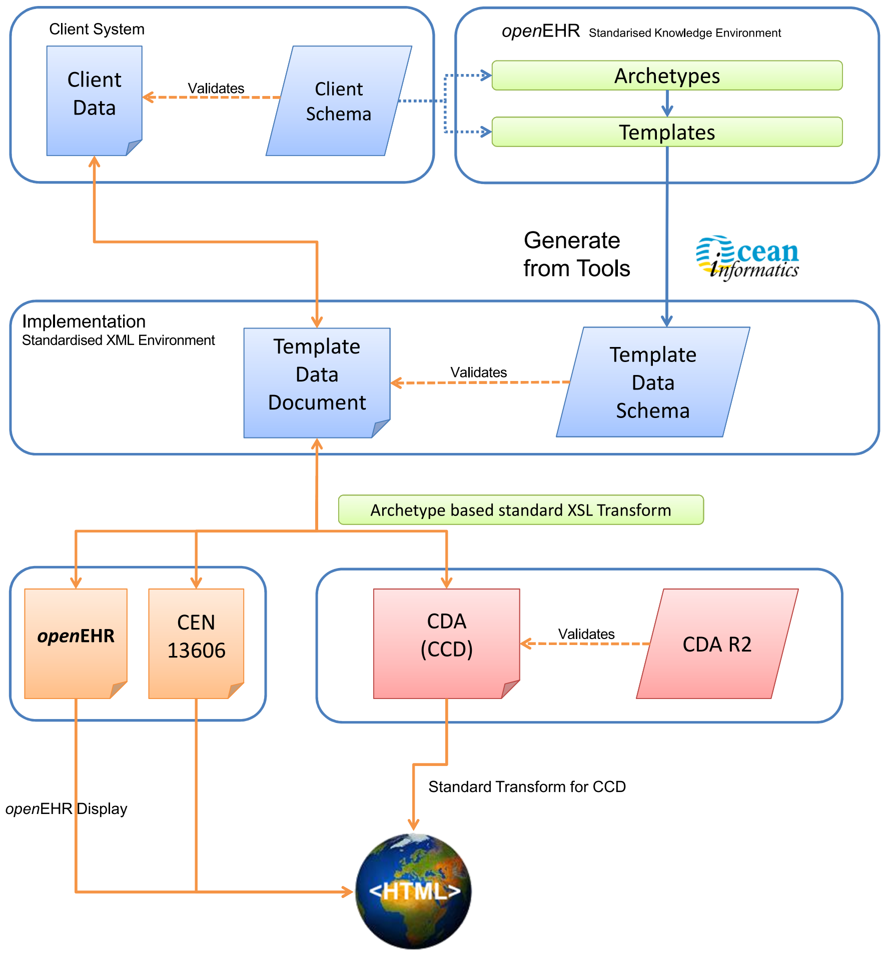


Fig. 4.16: Ocean Informatics' approach to using openEHR to produce CDA/CCD artifacts [84]. Their approach makes heavy use of the Extensible Stylesheet Language to transform XML data.

- Manual adjustments from domain experts may be needed where automatic mapping fails.

As a lot of medical information needed to implement systems might already be available in the form of openEHR archetypes, a good understanding of openEHR specifications might prove valuable. As shown in the previous section, using openEHR archetypes also helps when it comes to linking existing terminologies. Knowledge of other approaches, such as that of openEHR, is thus helpful even if working with HL7 Clinical Document Architecture documents.

After the inputs are in order, code templates (like Microsoft's Text Template Transformation Toolkit (T4)) can be used to generate software components which will handle specific use cases. Each component will provide a bridge between the external system and the data store of the current one, either a NoSQL document database, or a relational database, in which case an object-relational mapper is normally also used. By using an SDK like HL7 SDK for creating an object model from CDA documents, the connectors will be able to completely bypass dealing with any XML data directly, thus focusing on the domain classes.

Although it requires more time to set up, generating connectors by using code templates instead of simply creating XSL transforms that map XML documents from one format into another is a more flexible approach and allows for more complex rules to be easier expressed in a language that is more familiar to developers.

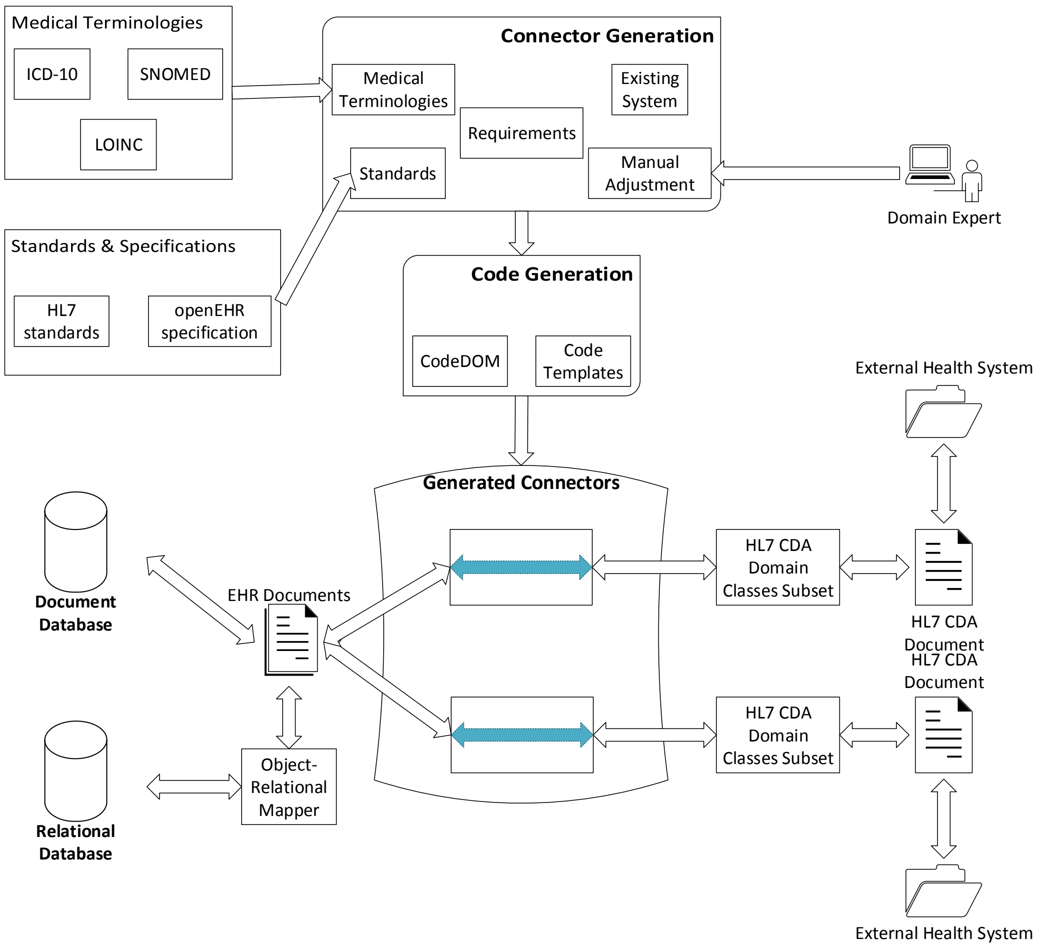


Fig. 4.17: Connectors linking systems via CDA documents. Connectors are small software components built specifically to transform data from one precisely defined source into another. Building connectors that understand HL7 CDA documents coming from external systems allows the data to be transformed into the format required by the target EHR system.

Conclusions

This chapter provided important contributions which link the development of a custom-made electronic health record solution with to existing medical standards and terminologies.

The chapter began with a study on medical terminology, showing that the incidence rate of canonical vs. derived medical terminology in natural language is 61/39% in English while the percentages is almost inverted when it comes to Romanian. Given the fact that the LOINC codes are not translated into Romanian, the chapter introduced an original study on how to translate such codes more easily, resulting in a decrease of 65% in the average length of each value that requires translation, together with a reduction by about 27% of the total number of unique values that need to be translated.

A further original contribution concerning medical codes consisted of a study on the possibility of using full-text search engines to match ICD-10 diagnostic codes. With lots of numeric results, the study showed that search queries were found, consisting of about 2.6 words/query, for distinctly matching between 74 and 86% of the codes analyzed, depending on the engine used.

The chapter continued with creating a bridge between chapter 3 and chapter 2 by contributing ways of integrating an existing EHR application with medical standards, specifications and terminologies. This part focused on linking openEHR archetypes with the existing model based on input types. Such a link allowed the import of terminologies directly referenced in archetypes found in open repositories and also allowed an implementation of the Archetype Query Language using mapping to SQL so that openEHR aware systems are able to execute queries against the presented EHR solution.

5. DEVELOPING A WEB FRONT-END FOR ELECTRONIC HEALTH RECORDS

After laying out important considerations on the core architecture of a flexible approach in building a solution for electronic health records, this chapter deals with the somewhat different, but equally important aspects of developing the front-end part of an EHR application.

As web technologies have evolved quite a lot in recent years, allowing the development of rich web clients that run on a multitude of operating systems and devices, from personal computers to portable tablets or mobile phones, it is only natural for such a front-end to be developed as a web application.

The first section, *Developing the Web Application* starts by identifying the required UI elements that the application should support. Afterwards it goes on analyze ways on implementing the user interface. The section compares multiple approaches and details why developing a rich single-page application is better than using other approaches such as XForms, or openEHR templates. The main design consideration in building this as a single-page application is the way the meta-structure of the input elements in stored and forwarded to the client-side application, which then uses it build the necessary UI elements on the fly.

The sections that follow deal with evaluating the user interface according to user-experience guidelines and procedures. A user activity tracking framework is developed and used to monitor how medics actually interact with a prototype application. Section *Analyzing the Users' Activity* than goes into detail of what can be learned from the users' interaction, leading the way for *Improving the User Interface* to provide improvements.

5.1. Developing the Web Application

In order to efficiently manage the medical data, the employees (mostly medics, but also system administrators) are presented with a rich web application, designed by modern standards. This allows them to always securely connect to the latest version of the software, from any device inside the medical network, running any modern operating system.

This section explores how such a web application—which makes use of everything presented in the previous sections—can be built, taking the user interface, client-side and server-side programming into account.

5.1.1. Designing the Web Interface

Contrary to most web sites, where a user mostly reads information, web applications are more concerned with gathering input from users. A web application that focuses on collecting medical data about patients makes no exception, medics having to fill in a lot of data for each patient form.

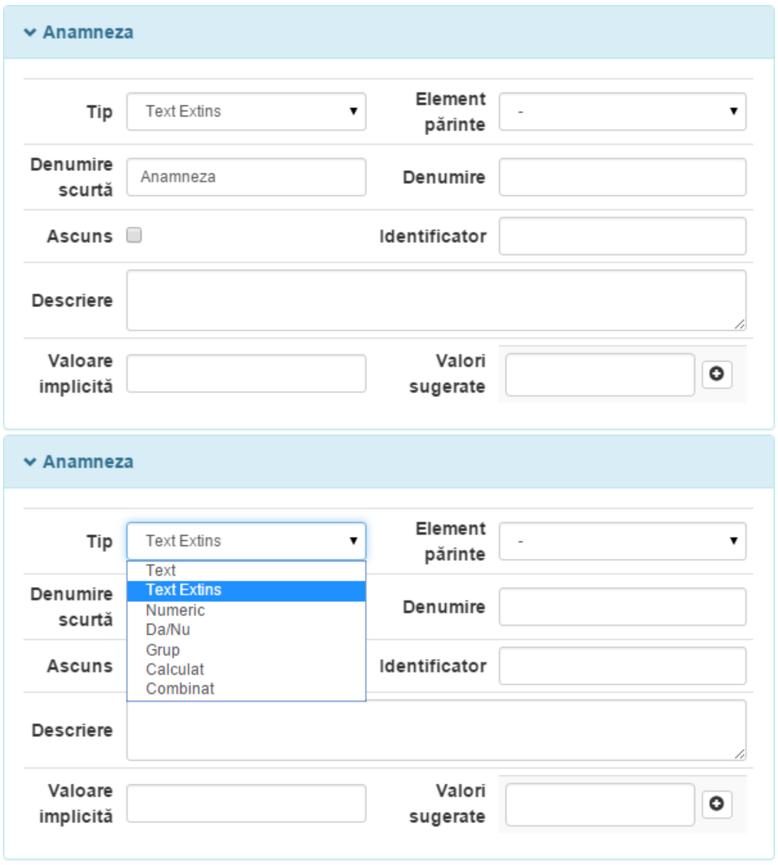
In order to provide a good user experience, the following requirements have been identified after consulting literature and talking with a medical professional:

- Focus on the medical, clinical aspects and not the software itself as shown by usability studies comparing the behavior of IT students and medical residents [85]
- Administrators need to be able to update input forms and their constituent elements through the web interface, without resorting to writing code
- Input forms must be easy to locate and select
- Inputs must be clearly visible and semantically grouped
- Each form needs to provide a good overview of all inserted values
- Text boxes need to be large enough to allow multi-word, even multi-sentence inputs
- Numeric inputs should easily be differentiated from text ones, and clearly specify the unit of measurement required
- Boolean inputs should allow distinction between a *no* value and a missing (not entered) value
- Duplicate information should be entered, but filled in automatically if possible
- Contextual help information and longer descriptions should be available where needed
- Initial form inputs should be easily distinguished from active ones
- Where possible, analytical interpretations of the values entered should be displayed
- The data input application must be easily integrated into a parent, more larger application.

The following paragraphs present a proposed web page layout aimed at dealing with the above issues. The users' needs are taken into account so as to provider them with a better efficiency and satisfaction [86].

The pages make use of a popular framework called Bootstrap[1]. Because this web application is aimed at delivering practically in an intranet environment (it is not a public facing website), Bootstrap's default theme, although less original, provides a more than adequate choice.

[1]http://getbootstrap.com/



Anamneza

Tip	Text Extins	Element părinte	-
Denumire scurtă	Anamneza	Denumire	
Ascuns	☐	Identificator	
Descriere			
Valoare implicită		Valori sugerate	

Anamneza

Tip	Text Extins	Element părinte	-
Denumire scurtă		Denumire	
Ascuns		Identificator	
Descriere			
Valoare implicită		Valori sugerate	

Fig. 5.1: Editing the *Anamnesis* (family history) input element. The first required field is the type of the input element, followed by it's parent, name and description. Input elements can also have an implicit value or multiple, commonly encountered, suggested values.

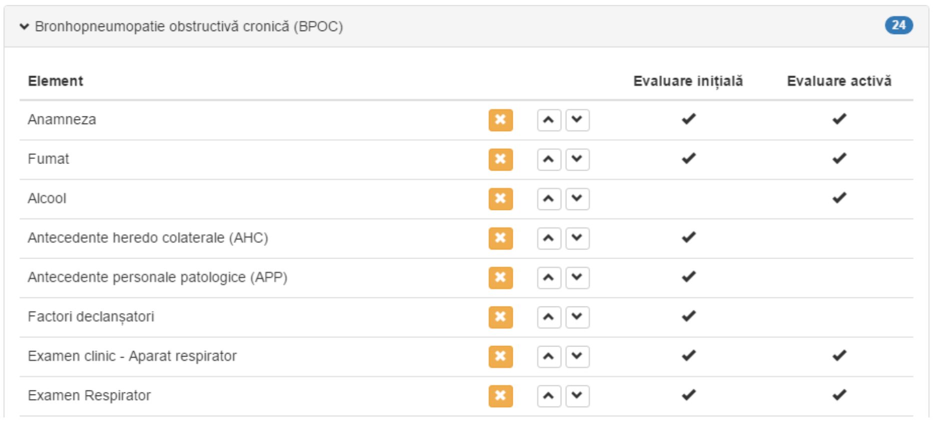
Administrating Elements

The major goal of this solution, when it comes to flexibility, is to allow the systems administrators to easily update the set of input forms and elements from an easy to use interface, without having to resort to writing code.

This goal is accomplished by providing an administrative interface composed of two parts:

- Manage Input Elements
- Manage Input Forms

The first part presents the administrator with all the input elements and allows for their creating/editing/deletion. Figure 5.1 lists the fields that can be edited when it comes to adding a new input element. To save space, the fields are placed in two columns, with the exception of the *description* which can contain more text.



Bronhopneumopatie obstructivă cronică (BPOC) 24

Element	Evaluare inițială	Evaluare activă
Anamneza	✓	✓
Fumat	✓	✓
Alcool		✓
Antecedente heredo colaterale (AHC)	✓	
Antecedente personale patologice (APP)	✓	
Factori declanșatori	✓	
Examen clinic - Aparat respirator	✓	✓
Examen Respirator	✓	✓

Fig. 5.2: Configuring the elements that make up an input form, specifying whether they should appear in the initial and/or active part and also choosing their relative order

Editing the input forms is more straight-forward as it only involves defining a new/selecting an existing form and choosing the right elements that make it up, together with the order in which they will be displayed, as can be seen in figure 5.2.

Presenting Input Forms

In order to be easily selectable, the input forms are displayed as tabs at the top of the page. Upon selecting one, the selected form is distinguishable from the non-selected ones and it presents the initial and active elements below in a two-column layout, as can be seen in figure 5.3. The values for the initial evaluation are always on the left side and those of the active one are on the right.

Each column then displays the appropriate input elements in a vertical layout, with the label on top of the actual input. This provides more then adequate space for each input. Scrolling will be necessary as each form contains many inputs, however this approach was preferred over pagination because if gives a much better overall view of the data entered.

Input groups can easily be distinguished by their header which uses a darker background and by the specific icon placed to the left of the name. As the icon indicates, groups can be collapsed and restored by simply clicking anywhere on their header, thus allowing medics to easily skip over optional elements, for which they have no value.

Should the web page be viewed on a low-resolution device, such as a mobile phone or tablet, the layout automatically changes to better fit the content, thanks to Bootstrap's responsive design. As shown in figure 5.4, the layout now uses only one column, make uses of the entire width of the mobile device. The menu is also laid out vertically as opposed to horizontally in case of the layout used on desktop displays.

Presenting Input Elements

Text elements can have value suggestions. These are presented in the form of a drop-down button at the right side of the text input. Should the input have a default

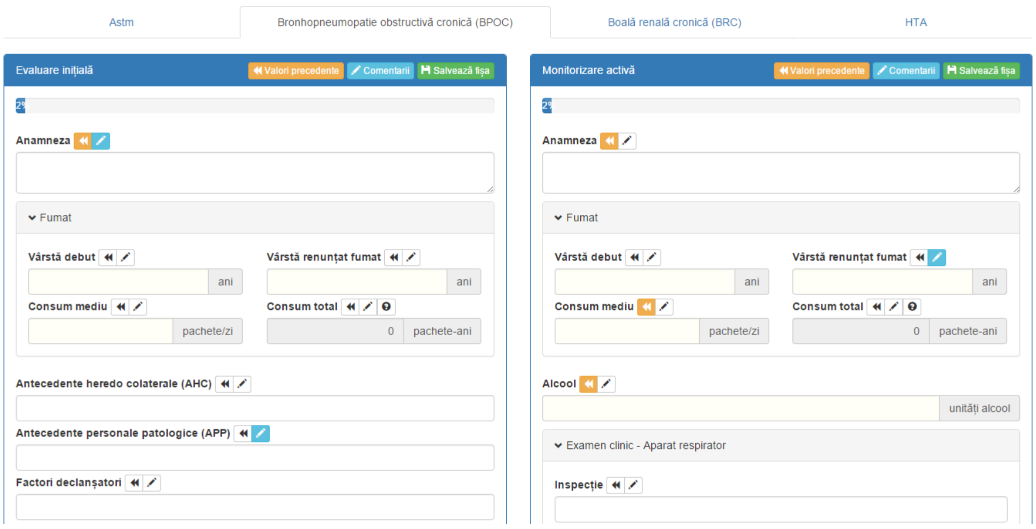


Fig. 5.3: Two-column layout showing the selected input form. The elements for the initial evaluation are displayed on the left and those of the active one are on the right.

value, a button is added also to the right of the input, accompanied by a tool-tip saying that it adds the default value and what the default really is.

In contrast to text elements which are implemented using an HTML <input /> tag, large text elements are implemented with the help of <textarea></textarea>, allowing the text to span multiple rows and providing a scroll-bar, if necessary.

Numeric elements can be distinguished first by a different background color and then by the measurement unit which accompanies most of them. The measurement unit is displayed as a label at the right of the input, which also has the text right-aligned, thus easily indicating that only the numeric part of the value must be provided by the medic. Because numeric values need less space than text ones, two of them can share the same row.

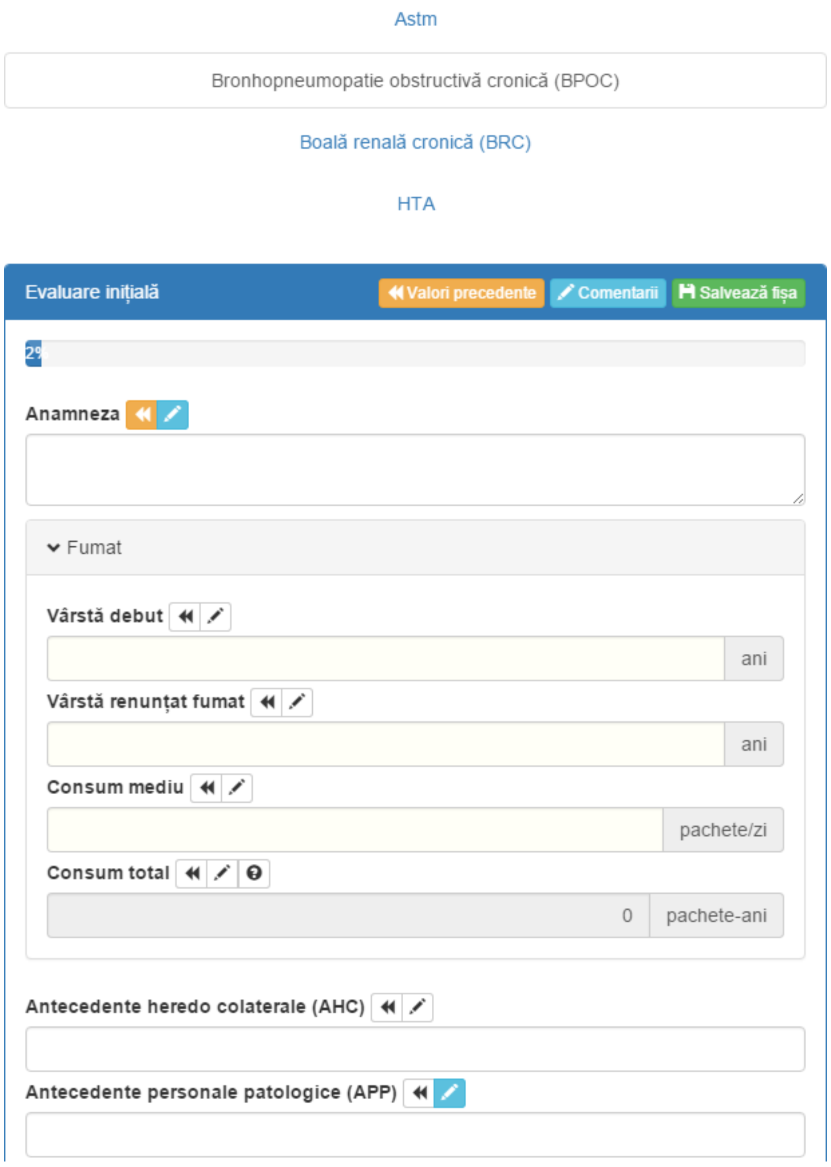
Similar to numeric elements are the calculated ones, the only differences between them being that they are not editable and that most of them are accompanied by a detailed description on the formula used.

Boolean elements use a simple <select></select> tag with three possible choices: blank (not specified), *yes* and *no*. As is the case for numeric, calculated elements, boolean elements can also be combined into two/row to save vertical space.

Combined elements present the usual label and then list all their child elements in the row below using a different background color. This proximity indicates that the elements are connected but does not draw too much attention.

Providing Feedback

Certain numeric inputs can be automatically analysed by simply comparing their value to different intervals or tables (e.g. the heart rate). Where such classification data is provided, the interface automatically updates the result when



Astm

Bronhopneumopatie obstructivă cronică (BPOC)

Boală renală cronică (BRC)

HTA

Evaluare inițială

Anamneza

Fumat

Vârstă debut ani

Vârstă renunțat fumat ani

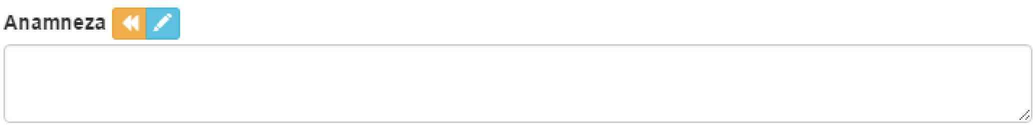
Consum mediu pachete/zi

Consum total 0 pachete-ani

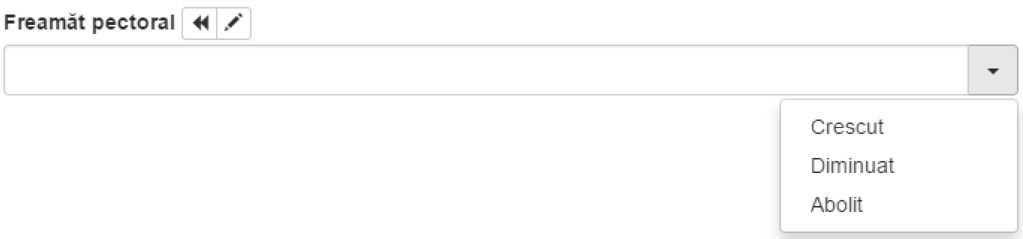
Antecedente heredo colaterale (AHC)

Antecedente personale patologice (APP)

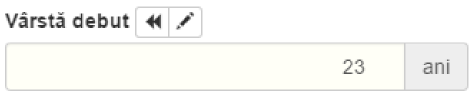
Fig. 5.4: Layout used on mobile devices for showing the selected input form



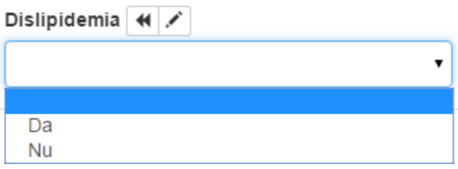
(a) Large text input element



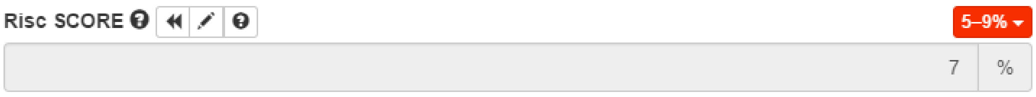
(b) Text input element with value suggestions



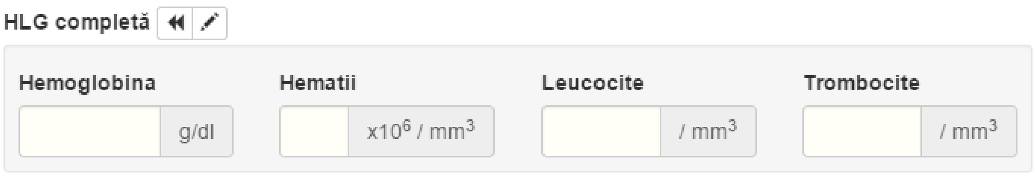
(c) Numeric input element also showing the measurement unit



(d) Boolean input element



(e) Calculated input element



(f) Combined input element with 4 numeric sub elements, each with measurement unit

Fig. 5.5: Various input elements

the user changes the value. Should this result also depend on other inputs (e.g. is the patient also a smoker), than the update is performed automatically also after editing each of the dependent input.

The feedback displayed is not only as text, but also in the form of a color to immediately signal if a value is normal or abnormal, even dangerous. This instant feedback also helps avoid typing errors as the medic can automatically tell that an extra zero was added because a value, known to be normal, suddenly shows up as highly dangerous. Figure 5.5e shows the value of ten year risk of having a heart attack, automatically calculated based on other inputs. The resulting percentage is also accompanied by a colored background indicating it's severity. The formula and color are based on a table found in [54].

Some inputs can have multiple classifications based on different other inputs (e.g. look in that table if the patient is a teenager or in the other one if the patient is old). A drop-down allows the medic to select between various classifications. The system goes even further than that and automatically disables various classification choices based on the available data (e.g. the system knows the patient is 40, so the medic won't select a classification only valid for teenagers).

Action Buttons

In order for users to be able to save the provided inputs, both the left and the right column have a save button. The buttons are located both at the top and at the bottom of the page to prevent the user from having unnecessarily scroll to the top just to press the button.

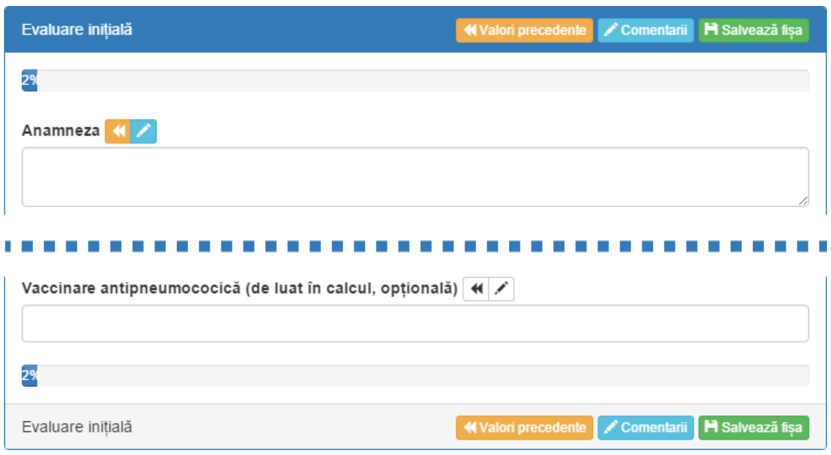


Fig. 5.6: Action buttons displayed at the top and bottom of the input form. This saves the user from having to scroll to the top to save changes when they have just reached the end of the form.

Each input element is designed to allow the user to view it's previous value and to store various notes or comments. Because this feature is optional, in order not to pollute the view, it is provided on demand with the help of two buttons placed after each input element's label.

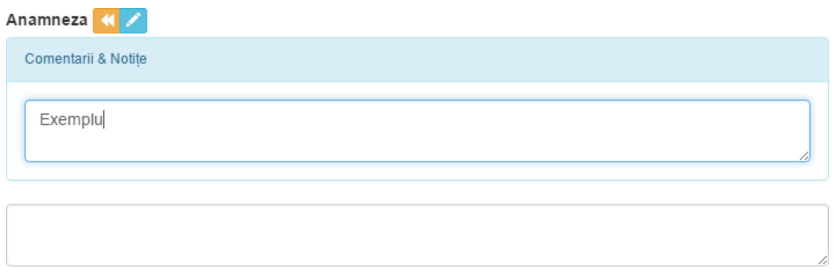


Fig. 5.7: Comments input for an input element triggered by an icon near the label. The icon's background signals that a comment is present.

Clicking the buttons triggers the display of new HTML elements containing the previous value and/or the comments. The buttons change their background from white to a specific color to indicate that such values are available. In order to expand all the inputs that contain a previous value and/or comments, appropriate buttons are placed at the top and bottom, near the *save* buttons.

5.1.2. Client-Side Development – Single Page Applications

Each web browser makes use of three types of inputs in order to control what it displays on screen and how the user can interact with it: [71, 87]

1. *HyperText Markup Language* – contains text in a structured manner,
2. *Cascading Style Sheets* – define rules for positioning and styling textual elements,
3. *JavaScript* – a scripting language which allows coding of interactions between the web page and the user.

Initial approaches on building web applications involved a 1 to 1 relationship between each screen displayed to the user and HTML documents (static or dynamic). Thus, each user interaction with the web page triggered a request to the server which in turn provided a new HTML document. Although this new document differs only slightly from the previous one, the entire document still had to travel from the server to the client each time, generating a lot of round trips. As the HTTP protocol is stateless, the developer also had to maintain a way of persisting the client state across calls.

The development of client-side scripting languages such as JavaScript, now supported by every major browser, together with the introduction of asynchronous calls (commonly referred to as AJAX) have enabled a better approach in which the client-side scripts request HTML fragments from the server and use them to modify only part of the HTML document, avoiding a full round trip [71].

This approach has been taken even further to produce Single-Page Applications (SPA), web pages that request the full HTML document only once and then rely of heavy use of scripts to update the content displayed to the user. Choosing this path for building the medical data collection application provides the following advantages over the classic approach: [88, 89]

- Better separation of concerns – The server-side code handles the business logic and the client-side code is the only one responsible for the user interface. As such, UI templates are no longer processed on the server. Furthermore, as long as the services exposed to the client remain the same, the server code can even be migrated from one development platform to another without the need to change the client code
- Client state is much easier to maintain across calls by simply storing the information as JavaScript variables
- The user experience is enriched by reducing the time the users need to wait between requests
- Performance improvements
- Ability to go offline for short periods of time without the user noticing
- Client-side logic is easier to test
- Server-side logic is easier to test

The generic advantages presented above also apply to a web application dealing with medical input forms. The primary one is that performing network requests in the background greatly improves the user experience by reducing the amount on time in which the interface is unavailable. The user can simply click an action button and then continue to perform more work, while the action is performed in the background.

The fact that the page is loaded only once and not entirely refreshed on every call allows for client-side state to be preserved as long as the user does not close the tab hosting the page. This offers the advantage of preserving information between calls without requiring extra round-trips to the server. This is particularly of use when dealing with temporary information that is not yet persisted: the medic can enter some details on one form, navigate to another one and come back to the original form without loosing the information previously entered, although the *save* action was not triggered.

Single page applications also have two important weaknesses:

- The client must have JavaScript enabled
- Search engines expect the content to be already rendered on the server in order to crawl the web pages

In the context of medical applications, these issues are non-existent. First of, search engine optimizations are not performed on intranet applications because they are not available to the general public, as they contain sensitive information. Secondly, JavaScript is enabled by default on all major browsers and studies show that, in recent years, only about 1% of users [90] explicitly disabled it. In an intranet environment, local software policies can also be enforced to always allow the execution of JavaScript on specific pages.

In the case of solutions based on openEHR, one of the ideal goals is to make use of operational templates in order to generate artifacts that can be used in building UI forms [91]. This approach brings the following challenges:

- A link needs to be maintained between form elements and the underlying template
- The system needs to closely follow the openEHR specifications
- Integration into existing applications not built according to openEHR is difficult
- Generating a new form implies modifying the application and requires executing a build step. In contrast, a rich single-page application receives all the data necessary to generate the form at run-time.

Various authors [17, 92] suggest implementing online health forms using XForms, "an XML markup for creating forms and form-like applications on the Web" [93]. Although a promising technology, XForms does not enjoy support from web browsers, requiring extra plugins or specialized software on the client side. Mozilla, for example, has removed support for XForms in their Firefox web browser starting with version 19[2].

HTML 5 forms together with a rich JavaScript client provide important features for displaying highly customizable forms inside web browsers, without requiring extra extensions.

Choosing AngularJS

One of the most famous JavaScript libraries for writing single page applications is AngularJS, developed by Google.

The library not only features a powerful template system, based on data binding and the Model View Controller (MVC) and Model View ViewModel (MVVM) patterns, but is also a framework that allows developers to build logical components and easily link them together with the help of Dependency Injection (DI) [88]. AngularJS also integrates well with the popular jQuery framework used for manipulating the Document Object Model (DOM) [88].

These features have made AngularJS a good choice for implementing the client-side code of the web application. The JavaScript code is organized in a way that allows it to preserve separation of concerns while benefiting from most of the framework's features: [88]

- *Controllers* – Create a bridge between the user interface and the business model, forwarding user actions to methods of the model and updating the view model when the business model signals changes.
- *Directives* – Encapsulate user interface components that can be reused in multiple page views.
- *Filters* – Provide reusable logic for enhancing the way textual elements are displayed and/or formatted.
- *Modules* – Define the AngularJS application objects.

[2]Removal note available at https://developer.mozilla.org/en-US/docs/Archive/Web/XForms

- *Services* – Provide the main business logic, often by calling API methods on the server side, but also performing actions such as processing results from the server to ease their use in the client-side code.
- *Helpers* – Contain auxiliary functions for processing arrays and strings in ways not provided by the standard JavaScript library.

Displaying Input Elements

One of the main challenges faced by the client-side development is that of displaying the appropriate input elements. After being provided by the server with details on each element that belongs to an input form, the client side JavaScript code is responsible for generating new DOM elements for displaying each input on the screen.

In the case of an architecture using AngularJS this can be accomplished by using *directives*. Directives are very similar to custom controls used in desktop applications. They allow the grouping of UI logic inside components that can be reused throughout the web page. This makes them a perfect fit for implementing input elements, as multiple instances of the same type of input element appear throughout the input forms.

An AngularJS directive can control its scope, has functions for defining the logic (before DOM elements are created – to customize what needs to be created, and afterwards – to set up event handlers and watches) and usually makes use of a template [88]. One such directive needs to be created for each type of input element.

The listing below illustrates the code and HTML template for the directive implementing the boolean input element, a simple directive which requires no extra logic apart from the model binding.

Implementation:

```
angular.module('monitoringApp')
.directive('inputBoolean', function () {

   return {
      templateUrl: urlLocation + 'Templates/inputBooleanTemplate.html',
      replace: true,
      scope: false,
      link: function (scope, element, attrs) {
      }
   };
});
```

Template:

```
<select>
   <option value=""></option>
   <option value="true">Da</option>
   <option value="false">Nu</option>
</select>
```

Using the directive involves placing a custom element in the DOM of the web page and letting AngularJS process it. The custom element also requires various attributes, the most important of which being the name of the model on which the two-way data binding operates. Each input element is also given a unique id, generated from the name of the actual input:

```
<input-boolean ng-model="inputsInitial[29].currentValue"
data-value-array="inputsInitial" data-value-id="29"
class="form-control ng-valid" id="inputHTAi" name="inputHTAi" />
```

Once all the required input directives are written, a façade is needed to isolate the choosing of the appropriate directive based on the input element entity. This façade is also implemented as a directive and leverages AngularJS's $compile service in order to obtain the actual input directive by processing a dynamically created HTML fragment in which the name of the desired input directive is inserted. Figure 5.8 illustrates the steps involved.

Processing Calculated Inputs

Some of the numeric inputs can be automatically calculated based on information provided in other inputs, thus relieving the medic from having to enter the same information multiple times. The single-page architecture allows for an instant update of the calculated value as soon as any of the dependent inputs changes.

These calculated inputs are read-only and, internally, store a mathematical formula which references other inputs. As an example, the following formula calculates how many years the patient has smoked by subtracting the age at which the patient has started smoking from the age at which the patient has quit smoking (or the patient's current age if still smoking):

```
(@smokingEndAge > 0 ? @smokingEndAge : @patient['age']) - @smokingStartAge
```

Because the client-side code is implemented using JavaScript, it is easier to structure these formulas in a way that they can be evaluated using JavaScript's built-in eval() function. This allows for simple constructs as well as more complicated ones, as seen above in the usage of the ternary operator.

In order to make the syntax as easy as possible, so that non-technical administrators can edit and understand these formulas, the references to other input fields look just like variables. They are in fact obtained by using the formula identifier of the input to be referenced and prefixing it with the @ character. This character is not valid in JavaScript variable names and thus prevents the accidental overlap of identifiers with built-in JavaScript objects. In a similar way, information about the current patient can also be referenced via the @patient identifier which behaves like an array, providing various properties of the current patient.

The process of evaluating the formulas is coded inside an AngularJS service and involves the following steps:

1. Enumerate all input elements and create hash map of all formula identifiers and their corresponding element id,

2. Enumerate all input elements and, for each formula identifier, store all elements that have a formula which references the identifier,

3. For each formula identifier, create an AngularJS scope watch which tracks the current value of the identifier's input element,

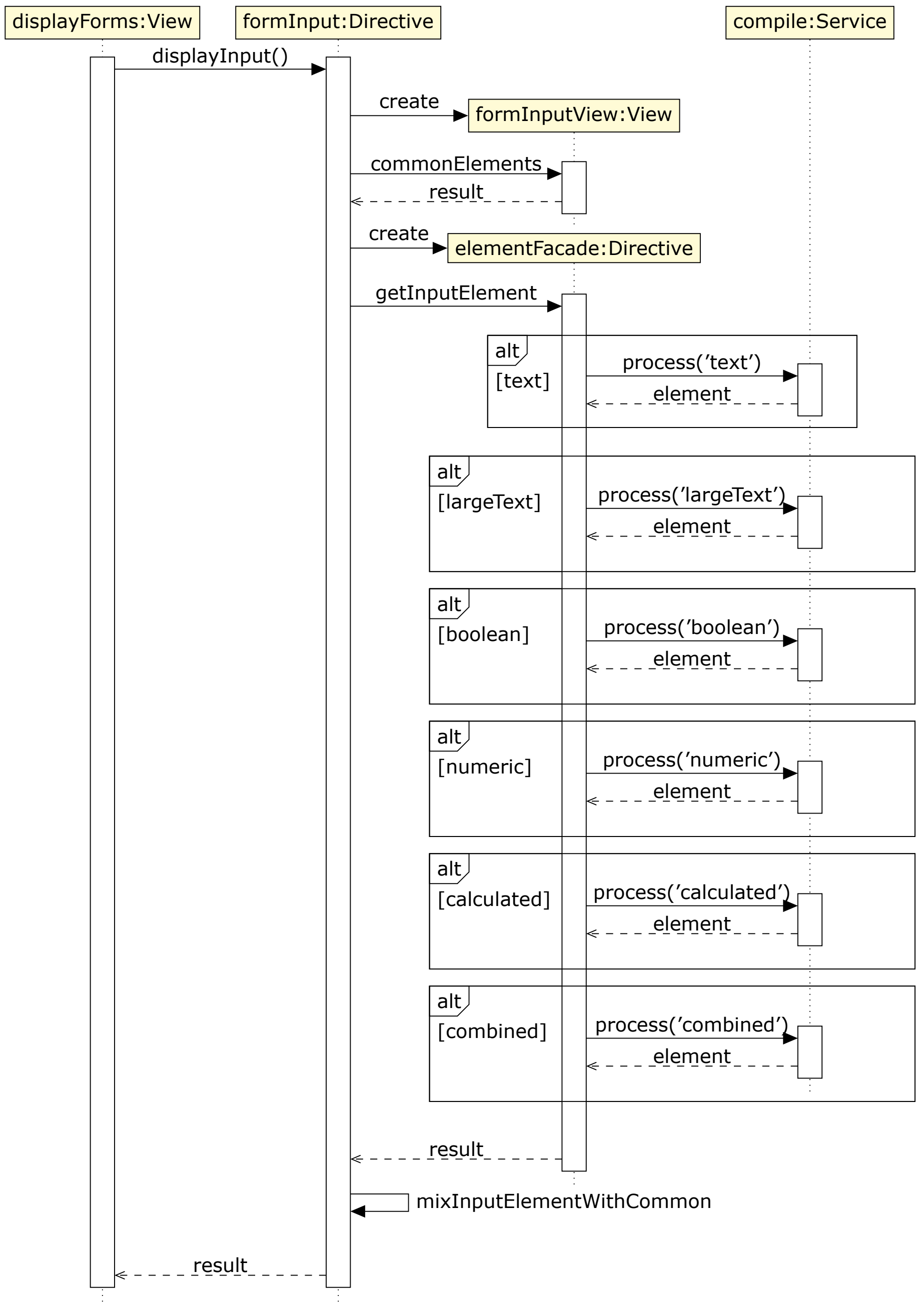


Fig. 5.8: Sequence of steps taken for displaying input elements. The client side code iterates through all input elements that need to be displayed and uses individual templates to render each type of element on-the-fly.

4. Whenever such a watch is triggered by the value being modified, iterate through all the calculated input elements that depend on the input whose value has just been changed and re-evaluate the formula.

When a formula is evaluated, it requires an extra pre-processing step before it is passed to eval(). This step extracts all formula identifiers that start with @, looks up the input element associated with each one (or special values such as @patient) and replaces them with the present value of the element. As a security precaution, the formulas are also cleaned for any function calls so as not to represent a security issue when passed to eval(). As such, the formula above is converted into e.g. (33 > 0 ? 33 : 50) - 19 which then leads to the final value of 14.

Evaluating formulas is not only useful in the case of calculated input elements, but also for classifications. These apply only to individual input elements and contain formulas which evaluate to a boolean value that determines classification to choose.

The flexibility of this formula system allows for instant feedback on data ranging from a simple interpretation of blood pressure to more complicated, even tabular, assessments of risks such as the 10-year risk of having a heart attack, described in figure 5.9. Automating the process of determining such a value helps the medic both by relieving the need to consult formulas or tables and by providing an instant alert should a wrong value be inserted into one of the fields, causing an unrealistic classification to be displayed.

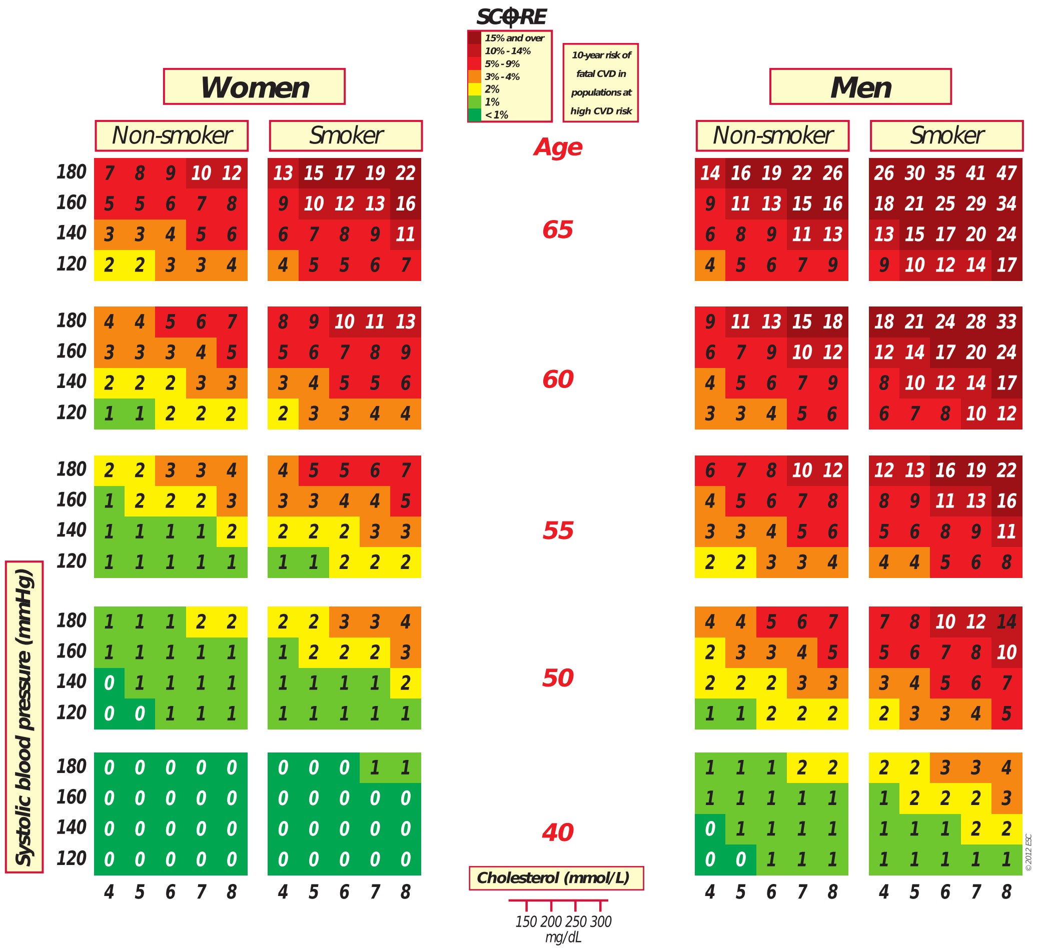
5.1.3. Server-Side Development – Web API

The most important actions of the web application happen on the server. Among them are:

- Authenticating & authorizing the client,
- Sending the required information to the client,
- Filtering the input from the client,
- Storing new data,
- Various cross-cutting concerns (logging, auditing, etc.).

Having implemented a rich, single page, client-side application changes the role of the server from a traditional web server (which outputs HTML to the client) to a service which can be interrogated by the client. This service still operates using the HyperText Transfer Protocol (HTTP), but only outputs data, leaving the client to format and display it in the best way it sees fit.

Because the client side code is written in JavaScript, formatting the server data as JavaScript Object Notation (JSON) makes it very easy for the client to consume. JSON is “a text format that facilitates structured data interchange between all programming languages” [95]. It is easy to work with from any language and it much less verbose than XML [96], thus requiring far less bandwidth. The listing below exemplifies how an input element object is serialized using JSON.



		Women										Men									
		Non-smoker					Smoker					Non-smoker					Smoker				
Age		4	5	6	7	8	4	5	6	7	8	4	5	6	7	8	4	5	6	7	8
65	180	7	8	9	10	12	13	15	17	19	22	14	16	19	22	26	26	30	35	41	47
	160	5	5	6	7	8	9	10	12	13	16	9	11	13	15	16	18	21	25	29	34
	140	3	3	4	5	6	6	7	8	9	11	6	8	9	11	13	13	15	17	20	24
	120	2	2	3	3	4	4	5	5	6	7	4	5	6	7	9	9	10	12	14	17
60	180	4	4	5	6	7	8	9	10	11	13	9	11	13	15	18	18	21	24	28	33
	160	3	3	3	4	5	5	6	7	8	9	6	7	9	10	12	12	14	17	20	24
	140	2	2	2	3	3	3	4	5	5	6	4	5	6	7	9	8	10	12	14	17
	120	1	1	2	2	2	2	3	3	4	4	3	3	4	5	6	6	7	8	10	12
55	180	2	2	3	3	4	4	5	5	6	7	6	7	8	10	12	12	13	16	19	22
	160	1	2	2	2	3	3	3	4	4	5	4	5	6	7	8	8	9	11	13	16
	140	1	1	1	1	2	2	2	2	3	3	3	3	4	5	6	5	6	8	9	11
	120	1	1	1	1	1	1	1	2	2	2	2	2	3	3	4	4	4	5	6	8
50	180	1	1	1	2	2	2	2	3	3	4	4	4	5	6	7	7	8	10	12	14
	160	1	1	1	1	1	1	2	2	2	3	2	3	3	4	5	5	6	7	8	10
	140	0	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	4	5	6	7
	120	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	3	3	4	5
40	180	0	0	0	0	0	0	0	0	1	1	1	1	1	2	2	2	2	3	3	4
	160	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	3
	140	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	2	2
	120	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Fig. 5.9: SCORE - European High Risk Chart. 10 year risk of fatal CVD in high risk regions of Europe by gender, age, systolic blood pressure, total cholesterol and smoking status [94]

```
{
  "id" : 152,
  "inputPrefix" : "inputANIDEFUMAT",
  "label" : "Ani de fumat",
  "longLabel" : "",
  "description" : "",
  "type" : "calculated",
  "parentId" : 148,
  "defaultValue" : "",
  "suggestions" : [],
  "precision" : null,
  "hidden" : true,
  "formulaIdentifier" : "smokingYears",
  "formula" : "(@smokingEndAge > 0 ? @smokingEndAge : @patient['age']) ...",
  "formulaLabel" : "",
  "combinedParentId" : 0,
  "combinedChildren" : [],
  "classifications" : null
}
```

Having described the core business logic in section 3.3, what is basically left for the server-side logic to do is create a bridge between the client and the core business logic, interpreting the client's requests and forwarding them to the business layer.

In contrast to traditional web services such as those using the Simple Object Access Protocol (SOAP), single page applications are easier to integrate with Representational State Transfer (REST) services. Such services characterize themselves by being lightweight [97], using unique Uniform Resource Identifiers (URI) to identify resources and common HTTP verbs to manipulate them [98], as can be seen in table 5.1.

REST services make use of the fact that the same entity can be represented in multiple ways and thus gain flexibility by decoupling the representation of an entity when sent over the wire from the representation of the same entity when stored in a database [99]. Not suprisingly, REST services are seeing an increasing adoption in web platforms for healthcare [45, 46, 96, 97, 100, 101, 102, 103, 104].

Verb	*URI*	*Result*
GET	/inputforms	Retrieves all input forms
GET	/inputforms/123	Retrieves the input form with id=123
POST	/inputforms	Stores a new input form
PUT	/inputforms/234	Modifies an existing input form (id=234)
DELETE	/inputforms/345	Deletes the input form with id=345

Table 5.1: Example of a RESTful service. Simplicity is achieved by using suggestive URLs to identify resources and standard HTTP verbs to perform actions against them.

As do most of the technology stacks which target web applications [97], the Microsoft .NET Framework offers an easy way to create RESTful services in the form of a framework called *ASP.NET Web API*, with brings in features such as: [98]

- Convention-based CRUD Actions
- Built-in Content Negociation
- Support for OData
- Self-hosting

The *ASP.NET Web API* framework allows the creation of controllers which subclass ApiController and implement methods for each supported HTTP verb. The methods receive parameters and return various objects which are then serialized to standard open formats such as JSON or XML [96, 102], depending of the client's request.

Figure 5.10 shows the methods implemented by the API controller which deals with input forms. This class also makes use of data transfer objects in order to isolate

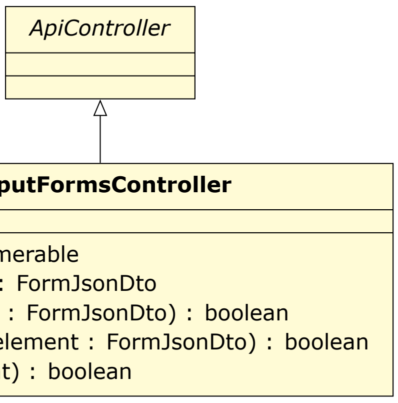

ApiController
InputFormsController
+ Get() : IEnumerable
+ Get(id : int) : FormJsonDto
+ Post(element : FormJsonDto) : boolean
+ Put(id : int, element : FormJsonDto) : boolean
+ Delete(id : int) : boolean

Fig. 5.10: UML class diagram showing the input forms API controller. The controller contains methods for each HTTP verb that it handles. Request data is provided as method arguments with the help of model binding.

the class that gets serialized to/from JSON (FormJsonDto) from the business layer class. The serialization library uses reflection to determine the properties to serialize and also allows extra configuration with the help of annotations.

Figure 5.11 presents a simplified view of the sequence of operations that occur when a medic enters a new form for a patient. The client code requests a list of all input forms. These are presented to the medic which chooses one of them. The client then requests information about the structure of each input element contained in the selected input form. Having received a reply, the elements are rendered on the screen and the medic can start adding data. After the data is added, the medic clicks *save* which triggers a POST request that sends the medic’s input to the server.

```
public class FormJsonDto
{
   [JsonProperty(PropertyName = ”id”)]
   public int Id { get; set; }
   [JsonProperty(PropertyName = ”name”)]
   public string Name { get; set; }
   [JsonProperty(PropertyName = ”longName”)]
   public string LongName { get; set; }
   [JsonProperty(PropertyName = ”inputs”)]
   public IEnumerable<FormInputJson> Inputs { get; set; }
}
```

Keeping a clean architecture when it comes to implementing the web service help keep a good separation of concerns between the domain and the application services. An architecture based on open, standardized interaction protocols is also implementable in any type of technology stack, preventing vendor lock-in.

5.2. Evaluating the User Interface

Once an electronic health records application is in production, medical professionals which interact with it are not exposed to it’s inner workings, but instead

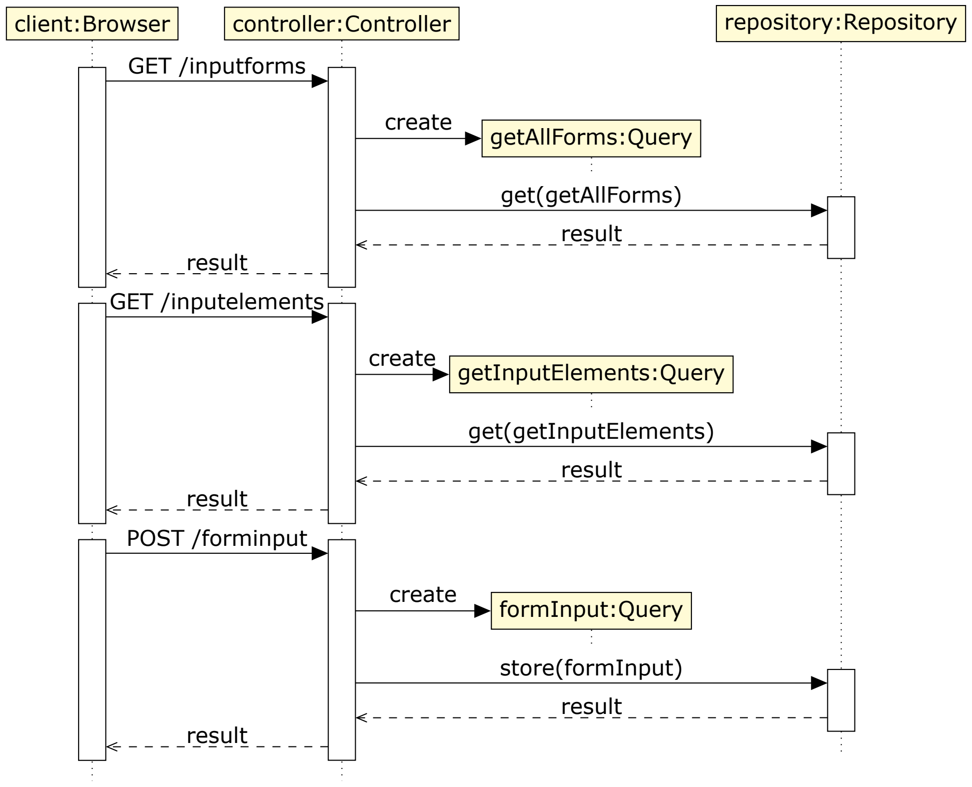


Fig. 5.11: Sequence of RESTful API calls. The client code first requests a list of all available forms. After the client chooses a specific form, the code requests all the inputs elements of that form. Later, when the form has been filled, the client provides the server with the form inputs.

interact with the user interface. As such it is essential to provide a good user experience in order to improve productivity by making it easier for users to do their job.

When talking about user experience, the theory of interaction design distinguished between: [105]

- *Usability goals* – concerned with meeting specific usability criteria
- *User experience goals* – largely concerned with detailing the quality of the user experience

In contrast to back-end systems, user interfaces often benefit from simple considerations like, for example, improving the screen clarity and readability by making screens less crowded. Studies show that this has been proven to increase productivity by about 20% [106]. Avoiding crowded screens also applies in the case of medical informatics where "complex tables, lists, charts, and diagrams are inevitable" [107].

Previous studies show that the "lack of good user interfaces has been long recognized as a major impediment to the acceptance and routine use of clinical informatics applications" [108, 109]. In the case of early user interfaces, a common cause for complaint arose from the lack of a guidance to the desired workflow [109].

One of the solutions found was to migrate from the classical tabular way of organizing forms to a design that preserves the look-and-feel of traditional paper charts, to the extent that scrolling using the mouse wheel simulates the paper-flipping behavior. [109].

An important step in the creation of a quality user interface is to perform a proper evaluation. The usability of a product should be tested early in the development process in accordance with modern usability practices, such as the user centered design approach [110].

Among the multiple possible types of evaluations are, as described in [111]:

- *Expert evaluation* – involves analysts systematically stepping through a user interface
  - Heuristic evaluation – the interface is compared against a set of usability guidelines and heuristics
  - Cognitive walk-through – the user interface is being stepped through for a task, noticing problems or responses
- *Usability testing* – involves observing representative end-users while interacting with the system to carry out tasks
- *Clinical simulation* – is a type of usability testing conducted in a simulation laboratory or in real setting to ensure a higher fidelity of the evaluation
- *Post implementation surveillance* – is a high fidelity evaluation based on the actual usage of a live system.

In order to learn the navigation behavior of clinicians, studies used "a sequential pattern analysis method to analyze actual usage recorded in the computer logs that contain time stamped events" [109]. This method allows for the discovery of hidden and recurring pattern within large sequences of events [112]. Emphasis is also placed on making use of actual usage data from the interaction with a live system, as opposed to laboratory exercises [109].

From a technical point of view, there is an issue with using server logs for gathering usage information: "data left by many interactive applications in the server's log file is minimal and not sufficient for extracting detailed information about the actual usage of the application" [113].

Information recorded in such logs only normally presents error events or details about when a specific page was accessed by a user. Should information be present about various forms that have been filled, it will not include specific details such as the order in which the inputs have been filled out or whether the user has encountered any difficulties in doing so [114].

A solution which enables tracking the users' activity involves recording every action that the users makes such as where and when the user moves the mouse or what keyboard keys are clicked. This results in a large amount of data that doesn't directly provide any usability information. More abstract information needs to be inferred from it, while also taking various events into consideration which may account for the user making various pauses [113].

A distinction can be made between two types of interactions with a website: [113]

- *Explicit interaction* – the actions performed of which the user is aware of (e.g. entering text in a form field)
- *Implicit interaction* – unconscious actions like hesitation before filling a form field due to uncertainty about the correct answer.

The following sections present ways of performing a qualitative and quantitative evaluation which, together with the results of applying them on the developed prototype, have been published under [114].

5.2.1. Heuristics

Usability consultant Jakob Nielsen lists the following 10 general principles for interaction design: [115]

- Visibility of system status
- Match between system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Aesthetic and minimalist design
- Help users recognize, diagnose, and recover from errors
- Help and documentation

Table 5.2 details how the current application design takes these principles into consideration.

Principle	*Adoption*
Visibility of system status	The user can assess which inputs are filled and which not at any moment Each form also presents a progress indicator both on the top of the page and on the bottom for better visibility
Match between system and the real world	The system is modeled on real-world concepts The user interface presents medical information The user interface doesn't leak technical details
User control and freedom	The use can change any input on the current form All inputs are presented on the same page, without the need for pagination

	The *save* button can be clicked when the user is finished and does not need to be used after each individual input
Consistency and standards	The meaning of words is consistent throughout the interface The medical forms presented are based on domain standards and legislation
Error prevention	Different types of input (text, numeric, yes/no) are clearly differentiated in order to quickly indicate to the user what is requested Numeric inputs signal an error if textual data is entered Numeric inputs are usually accompanied by measurement units to prevent confusion and increase consistency Numeric inputs support automatic classifications with color codes to indicate the semantic of the value entered
Recognition rather than recall	By keeping the entire form on screen, the user can quickly navigate back to a previously filled input, without requiring pagination Each type of input has distinguishing visual features that make it easy to recognize
Flexibility and efficiency of use	Accelerators are available to support keyboard navigation Emphasis is places on the user having to supply as little information as possible (e.g. the measurement unit is already shown and the user must only enter the value in question)
Aesthetic and minimalist design	Textual information is kept at a minimum Details and help messages are only shown on demand to prevent visual pollution when not needed Aesthetic considerations are further details in section 5.2.2
Help and documentation	Inputs support displaying long names to expand abbreviations, help messages and even detailed mathematical formulas for calculated items

Table 5.2: Respecting Jakob Nielsen's principles of interaction design [115].

According to [116], the following criteria of user interface quality can be considered:

- Speed of user's work
- Complexity of user's work

- Quantity of user's mistakes
- Speed of studying
- Subjective user's satisfaction

5.2.2. Aesthetics

Given the fact that people prefer attractive user interfaces, a solution for obtaining a good design on a tighter budget involves the use of automatic tools for the evaluation of interface aesthetics [117]. Existing studies have introduced mathematical formulas that enable an objective treatment of the aesthetic issues of graphical user interfaces [118, 119, 120].

The listing below presents such formulas according to [119]:

- **Balance**

$$\mathrm{BM} = 1 - \frac{\left|\frac{\sum_i^{n_L} a_{iL}d_{iL} - \sum_i^{n_R} a_{iR}d_{iR}}{\max\left(\left|\sum_i^{n_L} a_{iL}d_{iL}\right|, \left|\sum_i^{n_R} a_{iR}d_{iR}\right|\right)}\right| + \left|\frac{\sum_i^{n_T} a_{iT}d_{iT} - \sum_i^{n_B} a_{iB}d_{iB}}{\max\left(\left|\sum_i^{n_T} a_{iT}d_{iT}\right|, \left|\sum_i^{n_B} a_{iB}d_{iB}\right|\right)}\right|}{2}$$

  L, R, T, B – Left, Right, Top, Bottom; a_{iX} – area of object i on side X; d_{iX} – distance between the central lines of the object i on side X and the frame; n_X – total number of objects on the side X

- **Equilibrium**

$$\mathrm{EM} = 1 - \frac{\left|\frac{2\sum_i^n a_i(x_i - x_c)}{w_{\text{frame}} \sum_i^n a_i}\right| + \left|\frac{2\sum_i^n a_i(y_i - y_c)}{h_{\text{frame}} \sum_i^n a_i}\right|}{2}$$

  (x_i, y_i) – coordinates of the center of object i; (x_c, y_c) – coordinates of the center of the frame; w_{frame} – width of the frame; h_{frame} – height of the frame

- **Unity**

$$\mathrm{UM} = \frac{\left|1 - \frac{n_{\text{size}} - 1}{n}\right| + \left|1 - \frac{a_{\text{layout}} - \sum_i^n a_i}{a_{\text{frame}} - \sum_i^n a_i}\right|}{2}$$

  a_i – area of object i; a_{layout} – area of the layout; a_{frame} – area of the frame; n_{size} – number of sizes used; n – number of objects in the frame

- **Proportion**

$$\mathrm{PM} = \frac{\left|\frac{1}{n}\sum_i^n \left(1 - \frac{\min(|p_{j_1} - p_i|, |p_{j_2} - p_i|, \ldots)}{0.5}\right)\right| + \left|1 - \frac{\min(|p_{j_1} - p_{\text{layout}}|, |p_{j_2} - p_{\text{layout}}|, \ldots)}{0.5}\right|}{2}$$

$$p_i = \begin{cases} \dfrac{h_i}{w_i}; h_i \leq w_i \\ \dfrac{w_i}{h_i}; h_i > w_i \end{cases}$$

$$p_{\text{layout}} = \begin{cases} \dfrac{h_{\text{layout}}}{w_{\text{layout}}}; h_{\text{layout}} \leq w_{\text{layout}} \\ \dfrac{w_{\text{layout}}}{h_{\text{layout}}}; h_{\text{layout}} > w_{\text{layout}} \end{cases}$$

$$\{p_{j_1}, p_{j_2}, p_{j_3}, p_{j_4}, p_{j_5}\} = \left\{\frac{1}{1}, \frac{1}{\sqrt{2}} \approx \frac{1}{1.414}, \frac{1}{\frac{1+\sqrt{5}}{2}} \approx \frac{1}{1.618}, \frac{1}{\sqrt{3}} \approx \frac{1}{1.723}, \frac{1}{2}\right\}$$

w_i – width of the object i, h_i – height of the object i, w_{layout} – width of the layout, h_{layout} – height of the layout

- **Simplicity**

$$\text{SMM} = \frac{3}{n_{\text{vap}} + n_{\text{hap}} + n}$$

n_{vap} – number of vertical alignment points; n_{hap} – number of horizontal alignment points; n - number of objects in the frame

- **Density**

$$\text{DM} = 1 - 2 \left| 0.5 - \frac{\sum_i^n a_i}{a_{\text{frame}}} \right|$$

a_i – area of the object i; a_{frame} – area of the frame; n – number of objects in the frame (the authors assume 50% as the optimum screen density level)

- **Economy**

$$\text{ECM} = \frac{1}{n_{\text{size}}}$$

n_{size} – number of sizes used

All of these aesthetic measures are confined "to examining only the dimension and position of rectangular regions in order to control content effects and to facilitate interpretation of the data analyses" [119]. The result of applying the formulas is a numeric value between 0 (worst) and 1 (best) [119].

Aesthetics Results

A first step in calculating the measures presented above was to render a screenshot of the web application showing all the inputs of one input form. This resulted in a 1643x3265 bitmap which was then processed to extract the coordinates of all its inner rectangles [114].

As the amount of text displayed along side an empty form is quite low and text is normally present in the form of labels for each input, the analysis was conducted once without taking input labels into account and once with them in mind as small bounding rectangles for each individual word [114]. The results are presented in table 5.3.

Measure	*With Input Labels*	*Without Input Labels*
Balance	0.7704	0.7706
Equilibrium	0.9790	0.9842
Unity	0.4797	0.4776
Proportion	0.7344	0.5601
Simplicity	0.0051	0.0192
Density	0.9353	0.9745
Economy	0.0227	0.0833

Table 5.3: Analysis of aesthetics measures [114]. The two-column layout provides a good score for a number of measures. *Simplicity* and *Economy* are quite low as they advocate displaying only a few elements on screen, a situation which cannot be accomplished due to the complexity of medical inputs.

A quick overview shows that the values describing *economy* and *simplicity* are low. This however is of no surprise, as these two metrics decline rapidly as the number of objects on the screen grows, a situation with is inevitable due to the complexity of medical inputs.

The score for *proportion* is above 0.5, however it shows room for improvement as the value is negatively influenced by the high width/height ratio of most input forms. *Unity* is another measure which can be improved, in this case by increasing the margin of the layout in regards to the whole page and, at the same time, reducing the space between individual elements, compacting the screen [114].

The chosen layout shows a good *balance*, avoiding the placement of heavy objects on only one side. Further improvements can be made by reducing the size of the header shown at the top of the page, or by adding a footer to balance it better.

Using a two-column layout to present the initial recordings at the same time as the active ones leads to a very good *equilibrium* and also contributes to *density* having a similar high score.

With the exception of *propertion*, taking the input labels into account or not when computing the scores brings very little differences.

Section 5.5.1 discusses ways of improving the aesthetics scores.

5.2.3. Goals, Objects, Methods and Selection Rules

A method for performing quantitative analyses of user interface design is the classic model of goals, objects, methods and selection rules (GOMS) presented by [121]. This method enables the analysis of various operations on a specific user interface in order to find out how much time an experienced user would require to perform them [122].

The keystroke-level model, one of simpler yet still valuable, GOMS method, relies on the observation that "the time it takes the user-computer system to perform a task is the sum of the time it takes for the system to perform the serial elementary gestures that the task comprises" [122].

The authors of the GOMS method have come up with timings for different gestures, by means of laboratory experiments. When a keyboard and graphical input device are involved in solving the task, they found these typical timings to be enough for conductive comparative analyses as opposed to the much more demanding task of having to measure the exact time it takes each individual [122].

Each type of action was assigned a specific one-letter mnemonic, as can be seen in table 5.4. When performing an evaluation, after all the actions needed to perform a task are identified, each is associated with the specific mnemonic. The resulting mnemonics are then concatenated to form a string and their durations are summed up.

Mnemonic	*Duration*	*Description*
K	0.2 sec	The time it takes to tap a key on the keyboard
P	1.1 sec	The time it takes to point to a position on the display
H	0.4 sec	The time it takes the user's hand to switch from the keyboard to the graphical input device or vise versa
M	1.35 sec	The time it takes the user to prepare mentally for the next step
R		The time it takes the computer to respond to input

Table 5.4: Timings used by the GOMS model [121]. The actual values differ from person to person, however the average values found in literature can be used when comparing the interactions required before and after changing a specific user interface.

Table 5.5 shows the results of performing an analysis of the web application prototype using the GOMS model for the following inputs:

1.	Anamneză	Expectorație în cantități reduse
2.	Hemoglobina	11 g/dl
3.	Leucocite	10 000 /mm^3

4.	Trombocite	265 000 /mm^3
5.	Vaccinare antigripală	Neefectuată

Input	*GOMS Representation*	*Duration*	
		Total	*Actual Input*
#1	HPKHKKKKKKKKKKKKKKKKKKKK KKKKKKKKKKKK	8.5 sec	6.4 sec
#2	HPKHKK	2.5 sec	0.4 sec
#3	HPKHKKKKK	3.1 sec	1.0 sec
#4	HPKHKKKKKK	3.3 sec	1.2 sec
#5	HPKHKKKKKKKKKKK	4.3 sec	2.2 sec

Table 5.5: Quantitative analysis using the GOMS model [114]. As more text needs to be provided, the time spent preparing for this task becomes negligible.

The results of the analysis indicate an efficient user interface as the user does not waste time with secondary actions, spending the most amount of the time entering the actual information in the input fields. The amount of time spent before starting to enter data in a field (e.g. required to navigate to a desired input) is constant and becomes ever more insignificant the more information the user needs to input [114].

A constant delay of 2.1 seconds has been identified when navigating from one input field to another using the mouse. By displaying all the inputs on the same page and not hiding them in tabs or various sub window, this delay is kept under control. In the case of small inputs, it might take longer to navigate to a field using the mouse than to actually provided the necessary input. The solution to this is to benefit from the fact that numeric inputs are often placed close to one another and can be reached via keyboard navigation. Thus, a single key press (taking 0.2 sec) is often sufficient to move to the next input, resulting in a significant reduction of the delay [114].

5.3. Developing a Framework for Automated User Activity Tracking

The first step towards towards analyzing the user experience provided by an interface is to record and log the interactions between the user and the application. To this purpose, this section presents a developed web framework which transparently tracks the medic’s actions while using the application and provides information such as: [114]

- How was the web page rendered on the client machine?
- Where and how often did the user move the mouse?
- What did the user type using the keyboard and in which field?

5.3.1. Existing Tracking Solutions

Traditionally, web pages behave by requesting a new web page from the server every time a link was clicked. This behavior has led to the development of industry standard analytics frameworks like Google Analytics[3]. Such frameworks record when and by whom a web page is accessed and also provide an the interpretation of the data to the web site administrator.

Such tracking solutions, however, come with the major drawback of being designed for public facing web-sites and not intranet applications: [114]

- The recording is handled by an external server that must be reachable from the client machine. This can be a problem especially if, from a security point of view, one does not want external services to come in possession of information on the users' activities.
- The analytics provided (what pages does a user navigate to, how often does one return to the website) are mostly targeted at marketing departments, not user-experience engineers.

The introduction of single-page applications complicates matters quite a bit, making the task of recording relevant information harder. This is due to the face that, from the browser's perspective, the user is still on the same page even if, after clicking a link, the content of that page changes almost entirely [114].

The solution for this in the case of scripts such as the one provided by Google Analytics has been to allow the developer to write code that manually notifies the tracking service that a new page view is being shown to the user: "To track dynamically loaded content as distinct pageviews you can send a pageview hit to Google Analytics and specify the Document Path by setting the page field" [123]. The downside to this approach is that the need to explicitly provide this kind of information increases the development time proportionally to the number of distinct views.

Other companies offer commercial online analytics that are more centered on detecting user experience issues: ClickTale®[4], ExtraWatch™[5], Mouseflow[6], Seevolution®[7]. Their serivices include: visualizing heat maps showing mouse movement, replaying the actions in a session and analyzing the completion of forms.

5.3.2. Creating a Custom Solution

Instead of opting for an already available solution, a decision was made to create a custom one for recording and analyzing the data. This provides the following benefits: [114]

- The recordings can be stored on the same server used to host the web application being tested. This in turn:

[3]http://www.google.com/analytics/
[4]http://www.clicktale.com/
[5]http://www.extrawatch.com/
[6]https://mouseflow.com/
[7]https://www.seevolution.com/

  - Protects the privacy of the data which no longer travels to external servers
  - Has the same up/downtime as the web application
  - Allows for lower latency and lower bandwidth usage

- The type of data being collected is not limited to presets

- The way in which the tracking information is being analyzed can be improved without having to wait on 3rd parties to update their code.

Creating a framework for tracking user activities on a web page involves two major steps: recording the events and sending the records to a server for storage [114]. The sequence of events that take place is presented in figure 5.13.

Gathering Event Data

The main requirement for the custom framework was to correctly handle single-page applications. As such a decision was taken to perform all data gatherings on the client side.

Implementing such a feature is quite trivial as current web browsers already offer JavaScript API's for listening for a variety of events [113]. The listing below shows how event handlers registered using jQuery receive information on any mouse movement, click or key press on a page:

```
$('body').mousemove(function (event) {
   eventTracker.mousePosition = {
      x: event.pageX,
      y: event.pageY
   };
});

$('body').click(function (event) {
   eventTracker.logClick(event);
});

$('body').keypress(function (event) {
   eventTracker.logKeyPress(event);
});
```

Because the moving the mouse creates a lot of events, especially if the users moves it from one part of the screen to another in a very short amount if time, recording the position of mouse cursor is only performed every 50ms.

Reading that the mouse cursor was as position (40, 100) does not reveal any information, by itself. In order for it to be useful, the position needs to be correlated with whatever element the cursor was hovering upon. The first solution that comes to mind in this regards is to superimpose the position of the cursor on a screenshot showing the web page in question. This approach however must be pursued with caution as the way a web page is rendered depends heavily on the client screen resolution and also the browser used.

To overcome the problem of where exactly is the cursor positioned in relation to the web page, the developed framework includes a "screenshot" of how the page is actually rendered on the client's browser. This is technically accomplished by using an open-source library called *html2canvas*[8] to render the current page on a canvas. The content of the canvas is then exported to a bitmap and sent to the server.

As single-page applications can display multiple views in what is still, technically, the same web page, a solution was needed to transparently detect when such a view changes in order to avoid hard coding events. Because the web application under analysis mostly changes its view when a medic navigates to a different input form, and different forms contain different amount and type of inputs, the solution found was to rely on the change of page height to identify a change in the current view.

As such, whenever the total page height changes, a new recording is started for the current user. This recording contains the screenshot of the current layout and also the coordinates of all the visible DOM elements in the current view. In order to account for the time it takes the browser to actually render the page, time in which the page height progressively grows until all elements are rendered, a two second delay is used from the first page height change event to the moment the actual recording is started.

Sending the Tracking Data to the Server

After the client script gathers information on the user's actions, the tracking data needs to be uploaded to the server for persistence.

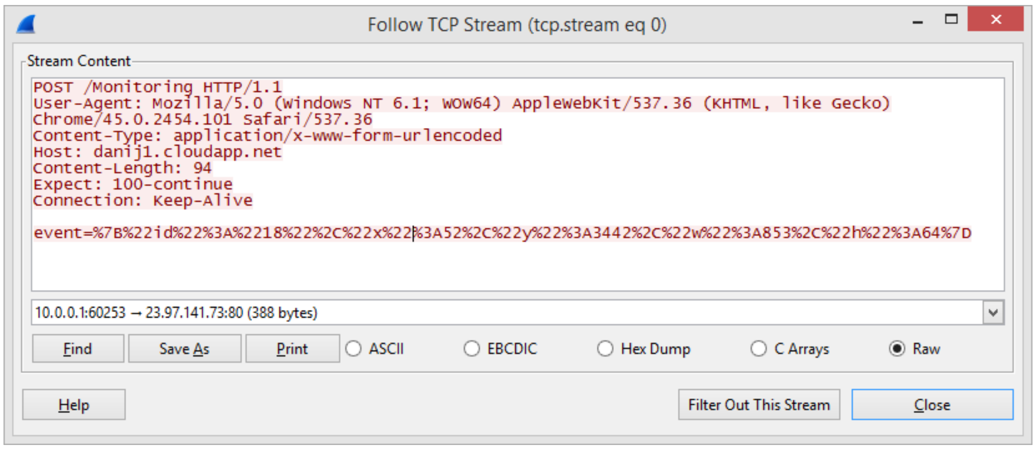
An easy approach would be to send traditional HTTP requests on each event with a small payload detailing the action taken by the user (e.g. click x=815;y=231 target=id:SPAN16 [113]). However, because the HTTP protocol is stateless [124], this approach ends up creating a lot of requests in the process, thus requiring the transfer of much more data (TCP handshakes, HTTP headers) than the actual intended payload [114]. Figure 5.12 shows how a 94 byte payload actually requires 388 bytes to be transferred due to the HTTP headers.

A better solution found involves the use of the newer *Web Sockets* standard which "enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code" [125]. This solution works much like a normal TCP connection, keeping a channel open between the client and the server for the duration of the page being displayed in the browser.

The web socket protocol is much more efficient when it comes to the data transferred, adding very little extra overhead when transferring the data after the initial handshake is completed. This brings the benefit of not affecting the user's experience while using the web application while the activity tracking takes place at the same time.

The records containing the client's actions are serialized using the JSON format, easy for both the client-side JavaScript code to create, and for the server-side code to interpret. Each event is an entry in an array, with properties describing at least its type of event and the time stamp (milliseconds elapsed since the recording was started) [114].

[8] https://html2canvas.hertzen.com/



```
POST /Monitoring HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/45.0.2454.101 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Host: danij1.cloudapp.net
Content-Length: 94
Expect: 100-continue
Connection: Keep-Alive

event=%7B%22id%22%3A%2218%22%2C%22x%22%3A52%2C%22y%22%3A3442%2C%22w%22%3A853%2C%22h%22%3A64%7D
```

10.0.0.1:60253 → 23.97.141.73:80 (388 bytes)

ASCII EBCDIC Hex Dump C Arrays [✓ Raw]

Fig. 5.12: Packet capture showing the overhead of HTTP headers. Sending small amounts of data at a time requires considerably more bandwidth due to the size of the headers.

As can be seen in the listing below, property names are kept short to preserve bandwidth, a mouse move (“m”) event contains the new coordinates, a key input (“k”) event contains the id of the DOM element where the key was pressed, along with the key’s character code (100 – *d*); and a screen content changed (“s”) event contains the new size of the page and the DOM elements that comprise it (each with id and bounding rectangle). Assigning id’s to each distinct input element has proven to simplify identifying them [114].

```
{
   ”t” : ”m”, ”ts” : 1673959, ”x” : 92, ”y” : 1603
}, {
   ”t” : ”k”, ”ts” : 1674762,
   ”c” : 100, ”id” : ”inputMURMURVEZICULARi”,
}, {
   ”t” : ”s”, ”ts” : 1675815,
   ”w” : 1349, ”h” : 3137,
   ”e” : [{
         ”id” : ”labelinputANAMNEZAi”,
         ”x” : 36, ”y” : 343, ”w” : 71, ”h” : 20
      }, {
         ”id” : ”buttonPrevinputANAMNEZAi”,
         ”x” : 111, ”y” : 348, ”w” : 12, ”h” : 18
      }, {
         ”id” : ”inputANAMNEZAi”,
         ”x” : 36, ”y” : 373, ”w” : 582, ”h” : 20
      }
   ]
}
```

The amount of bandwidth required for transferring the recordings is further reduced by compressing the (text) JSON documents using the deflate algorithm[9]. As compression is more effective, the greater the amount of data that can be scanned for

[9]https://tools.ietf.org/html/rfc1951

potential duplicates, the JSON entries are stored in a buffer where they accumulate for 5 seconds before the buffer is compressed and sent over the wire. The compression ratio obtained is about 23% for the average entries and 56% for the initial entries which contain the screen elements and the screenshot inserted as a base64 string in the JSON document [114].

If making small changes to a live application (in order to also include the scripts required by the tracking framework) is not possible, [113] presents a solutions that uses an HTTP proxy in order to modify HTML pages before passing them on to the client browser. In their scenario, the proxy also intercepts the tracking data and logs it; however this method leads to more connections and more bandwidth being used then my solution presented above.

5.4. Analyzing the Users' Activity

In order to evaluate the users' experience in using the resulting web application, I have uploaded a prototype on a publicly accessible web site and created a medical scenario detailing information, about a fictitious patient consultation, that needs to be inputted into the forms.

The scenario, a screenshot of which is shown in figure 5.14, contains information relevant to two chronic diseases, ranging between textual, numeric and even boolean inputs. The same scenario was presented to all candidates and it also included an introductory guide on how to use the application [114].

Twenty medical doctors and residents have accepted the invitation to pilot the prototype. Their interactions with the web application has been logged in detail and later analyzed using various methods [114].

5.4.1. Heat Maps

Tracking the users' eye movements, analyzing what they are focusing at, is an important method in understanding how a user interacts with something shown on a computer screen.

Normally, performing such an analysis is difficult as it implies the use of expensive monitoring equipment and a controlled environment. However, studies have found that plotting the users' mouse movements gives a good indication of what elements the user's eyes were focusing at [126].

As presented in section 5.3.1, a popular method for visualizing mouse movement offered by commercial solutions is in the form of heat maps. Heat maps are 2D plots, overlaid upon the screenshot of the web page, which indicate the relative time the mouse pointer has hovered over specific areas using color hues.

Based on the position of the mouse, logged by the tracing framework, and the screenshot of how the web page got rendered on each client browser, heat maps were plotted by varying the colors' hue from 0°(red) to 240°(blue) and using a radius of 40 pixels.

When using the default, linear scale, one issue that became immediately apparent was that a lot of users left the mouse cursor in a stationary point for a longer period of time [114]. This, as can be seen in figure 5.15a, led to mouse movement

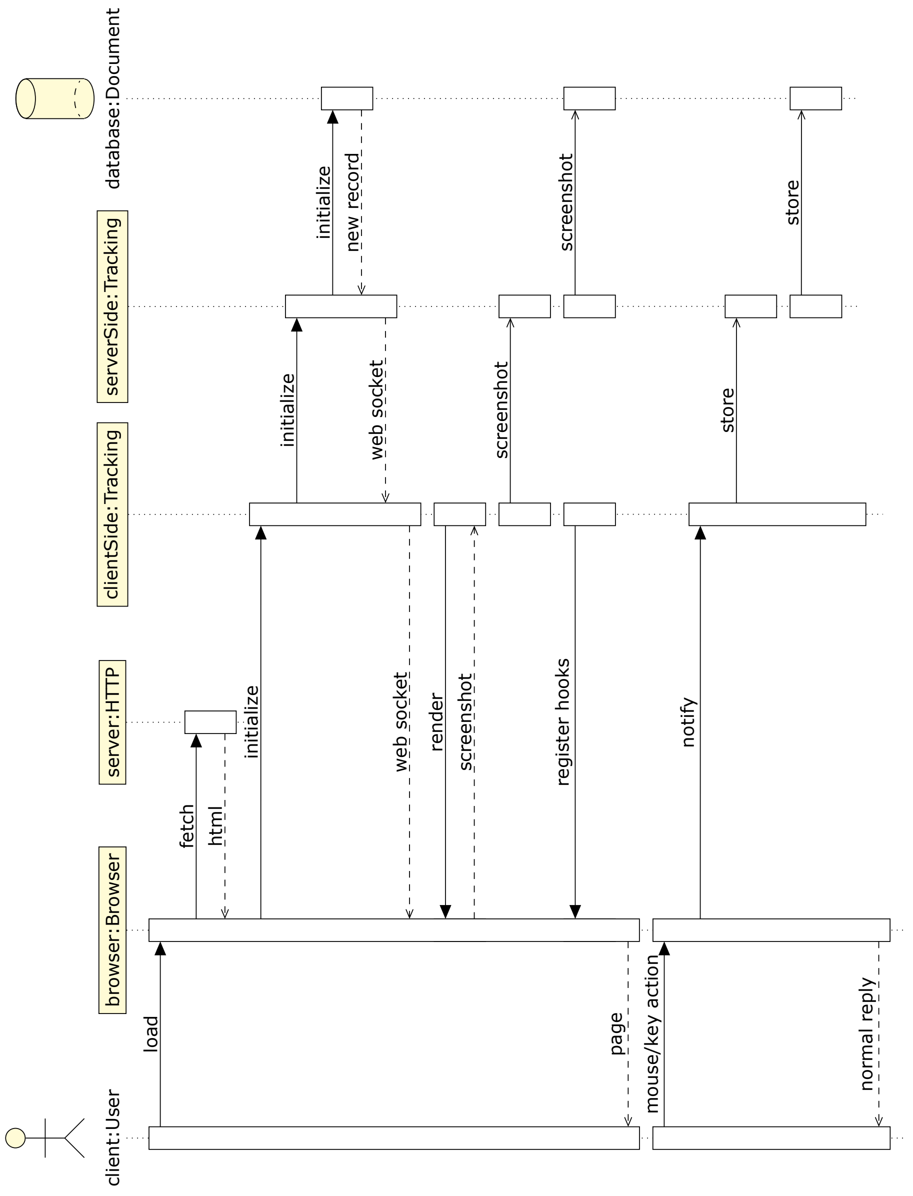
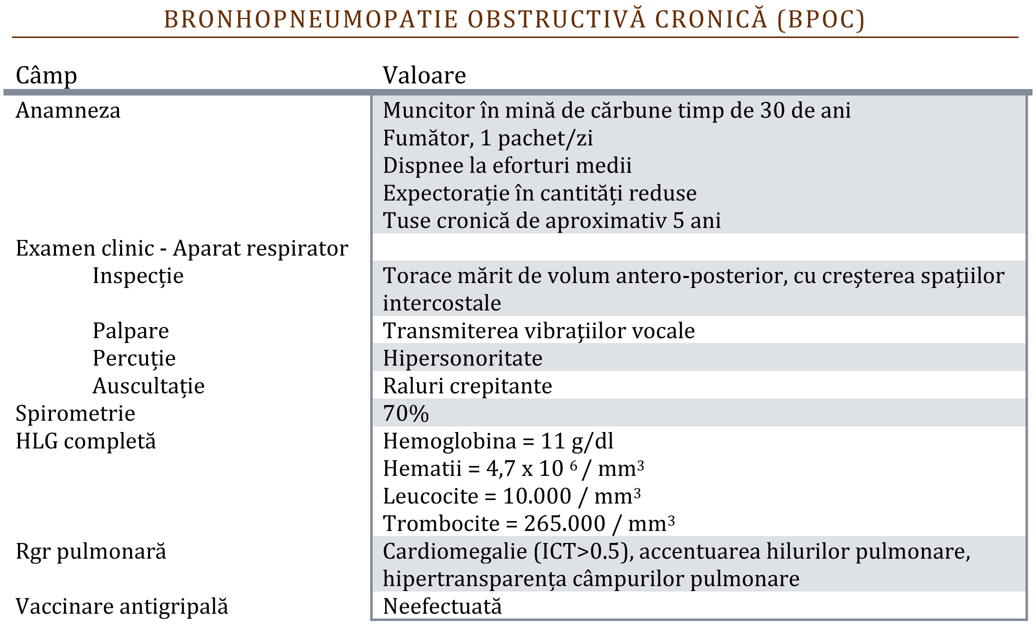


Fig. 5.13: Sequence of events describing the tracking framework



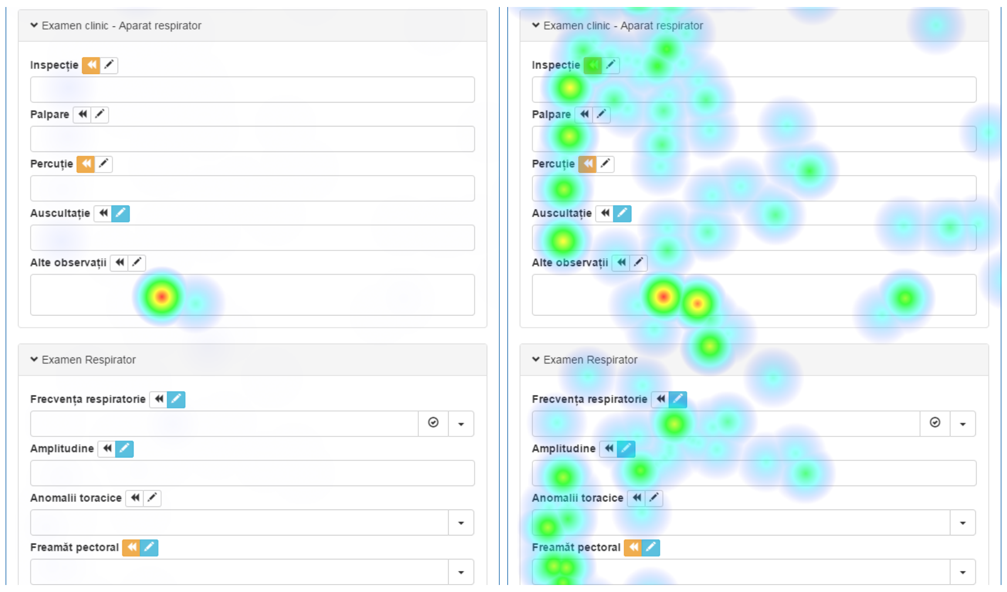
BRONHOPNEUMOPATIE OBSTRUCTIVĂ CRONICĂ (BPOC)

Câmp	Valoare
Anamneza	Muncitor în mină de cărbune timp de 30 de ani Fumător, 1 pachet/zi Dispnee la eforturi medii Expectorație în cantități reduse Tuse cronică de aproximativ 5 ani
Examen clinic - Aparat respirator	
Inspecție	Torace mărit de volum antero-posterior, cu creşterea spaţiilor intercostale
Palpare	Transmiterea vibraţiilor vocale
Percuţie	Hipersonoritate
Auscultaţie	Raluri crepitante
Spirometrie	70%

HLG completă	Hemoglobina = 11 g/dl
	Hematii = 4,7 x 10 ^{6} / mm^{3}
	Leucocite = 10.000 / mm^{3}
	Trombocite = 265.000 / mm^{3}

Rgr pulmonară	Cardiomegalie (ICT>0.5), accentuarea hilurilor pulmonare, hipertransparenţa câmpurilor pulmonare
Vaccinare antigripală	Neefectuată

Fig. 5.14: Example information shown in the user experience evaluation scenario [114]. All users are requested to input the same information in the pilot, in order to be able to compare results.

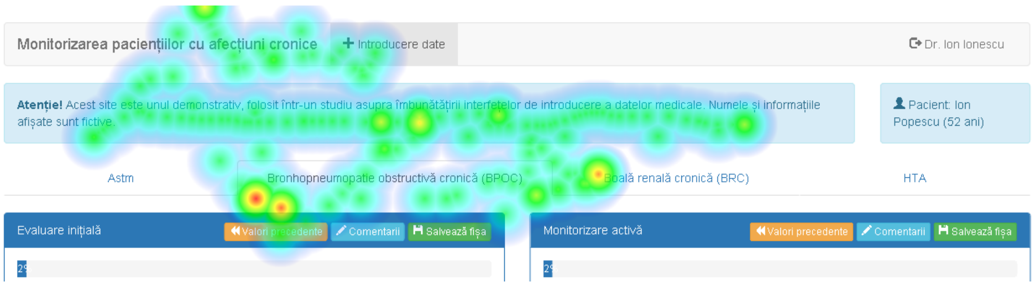


(a) Linear scale (b) Logarithmic scale

Fig. 5.15: Heat maps using a linear and logarithmic scale. They show where the mouse has remained stationary for the longest periods.

barely visible in other areas of the page, because the cursor stationed for far less in those areas. A solution was to use a logarithmic scale which, as shown in figure 5.15b, enables an easier distinction between areas in which the cursor was stationed a lot, was just passing through or did not reach at all.

After analyzing multiple user sessions and comparing the generated heat maps, it became obvious that users hovered a lot over click-able areas like the navigation links at the top of the page, or the actual input forms. Figure 5.16 shows the link between the eye and mouse movement as the user moves the cursor along a line of text while reading it [114].



Atenție! Acest site este unul demonstrativ, folosit într-un studiu asupra îmbunătățirii interfețelor de introducere a datelor medicale. Numele și informațiile afișate sunt fictive.

Fig. 5.16: Heat map showing the cursor following a text while the user reads [114]

The observation that the mouse is used for navigation from one input to another, as indicated by the concentration of spots on the heat maps, prompted a further analysis by super-imposing each heat map with the location of recorded mouse clicks. As can be seen in figure 5.17, areas where the mouse is stationed for a longer time are also accompanied by mouse clicks. The users activate the inputs with their mouse which then remains stationary while the keyboard is used to enter data [114].

A more detailed analysis shows that 78% of inputs are activated using mouse-clicks with only 22% using the keyboard by means of the TAB key [114]. Figure 5.17 also shows that, by placing the measurement unit on the right side of the numeric inputs, the user was able to correctly deduce that the text of numeric inputs is aligned to the right [114].

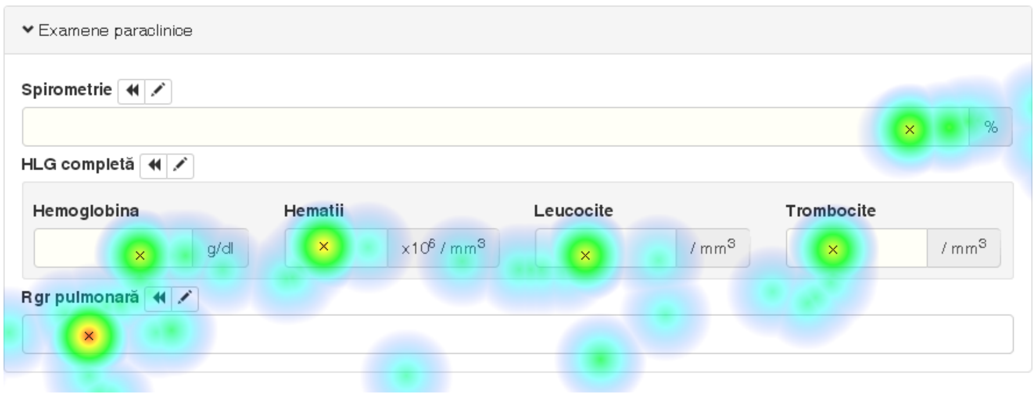
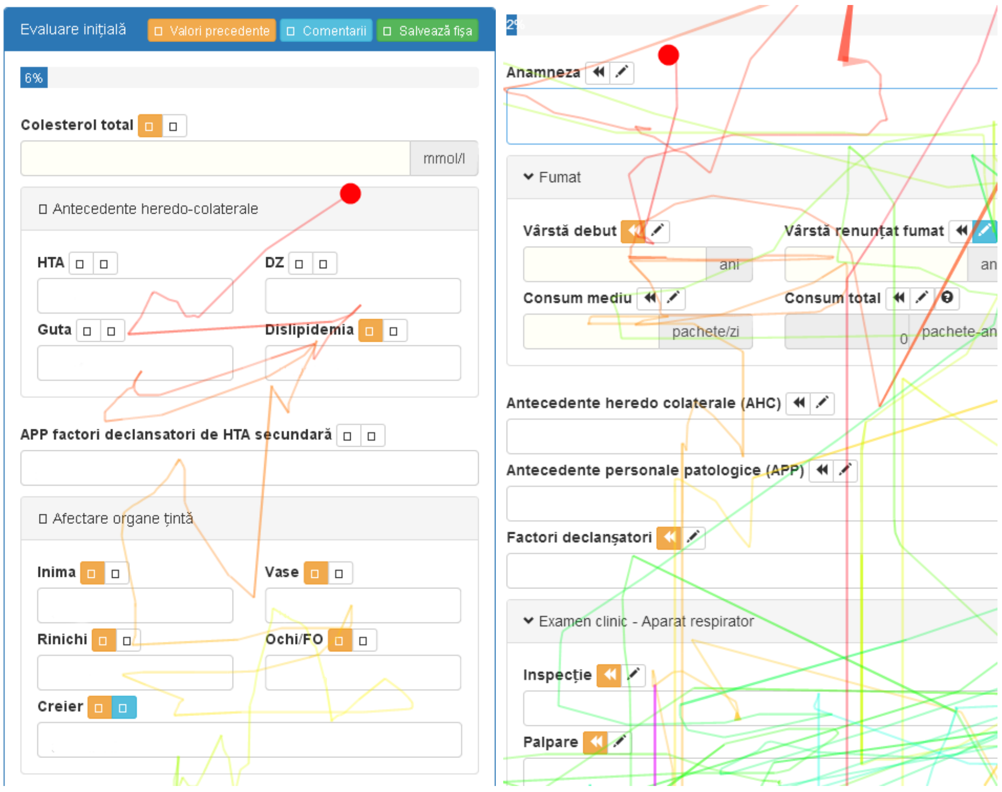


Fig. 5.17: Heat map also highlighting mouse clicks [114]

5.4.2. Mouse Movement

One type of information that is not visible on heat maps is the chronology of the mouse movement. Although this is usually viewed as an animation, I was able to plot the chronological movement in the form of a single image by again using different hues to indicate the movement's direction. Apart from that, the starting position is marked with the big, filled, red circle and the amount of time the mouse is stationed in a single point is indicated by varying each segment's width.



(a) Easy to follow (b) Hard to follow

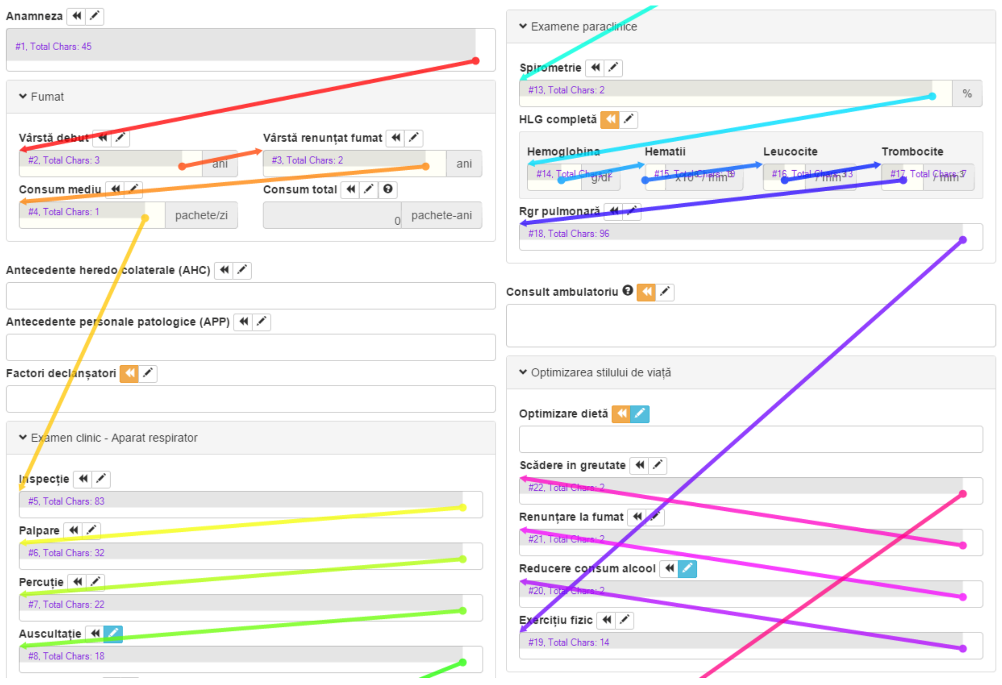
Fig. 5.18: Chronology of the user's mouse movements

One disadvantage of this method, as highlighted by figure 5.18, is that it can quickly become to congested and thus hard to read. However, it still managed to show important information on the flow of the users' activity on the page.

5.4.3. Input Analysis

As expected and also shown by the GOMS analysis, the majority of time spent in the application's user interface involves providing textual input. While the speed of typing letters is heavily dependent on the user's dexterity, the order in which the user fills the inputs, along with the ease to find the exact input one searches is important for boosting productivity by reducing the time spent navigating around the page [114].

In order to visualize the order in which inputs were filled, the analysis looked at the chronology of each keyboard click and plotted arrows that indicate the temporal relation between inputs.



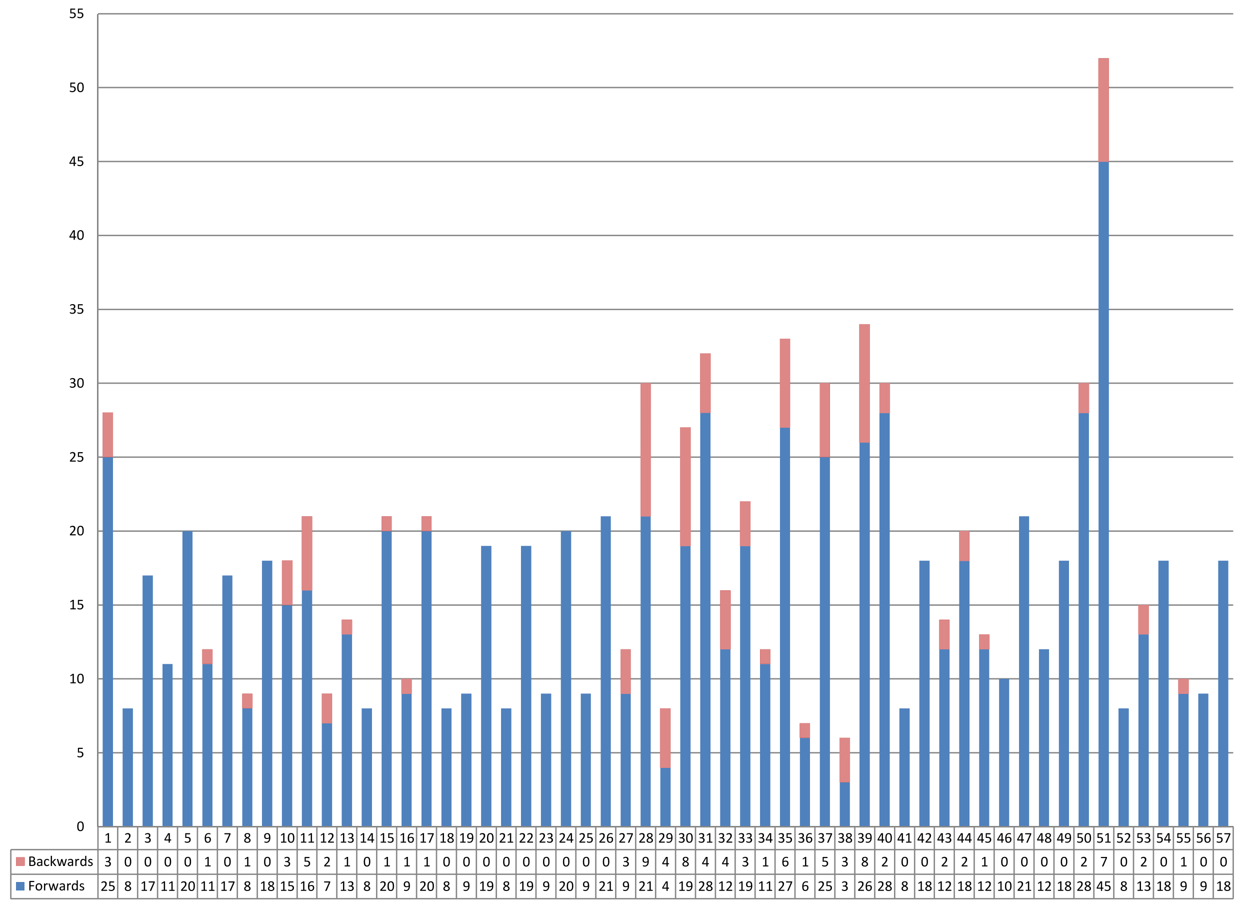
(a) Left-to-Right, Top-to-Bottom (b) Mixed order

Fig. 5.19: Order in which inputs are filled [114]

Figure 5.19 presents such plots. In order to indicate the chronological order, the arrows are colored using different hues, in ascending order (red, yellow, green, blue, violet). The plots also contain the number of characters entered in each text box. The majority (90%) of inputs are entered in a left-to-right, top-to-bottom order, as is natural for the Romanian (and/or English) language [114]. Figure 5.20 details this percentage. Figure 5.19b presents an example in which some inputs are skipped and then went back on.

When it comes to the typing speed of the users, calculations where performed on the duration of each keyboard click as the difference in the time stamp of two consecutive entries. These results, as exemplified in figure 5.21, also highlight an important aspect: occasionally, the duration spikes because the user stops to think or navigate to a different input.

In order to get a better overview of which data entries can be discarded, a histogram was plotted showing how many occurrences there are of various durations between inputs, taking the entries for all users into account. Based on it, one can safely consider that entries taking longer than 2500 ms are caused by other factors



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
🟥 Backwards	3	0	0	0	0	1	0	1	0	3	5	2	1	0	1	1	1	0	0	0	0	0	0	0	0	0	3	9	4
🟦 Forwards	25	8	17	11	20	11	17	8	18	15	16	7	13	8	20	9	20	8	9	19	8	19	9	20	9	21	9	21	4

	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57
🟥 Backwards	8	4	4	3	1	6	1	5	3	8	2	0	0	2	2	1	0	0	0	0	2	7	0	2	0	1	0	0
🟦 Forwards	19	28	12	19	11	27	6	25	3	26	28	8	18	12	18	12	10	21	12	18	28	45	8	13	18	9	9	18

Fig. 5.20: Direction of navigation between adjacent inputs [114]. The majority of inputs are filled in a left-to-right, top-to-bottom order. Numbers indicate how many transitions have occurred from an input to another one which is to the right/bottom (forwards) or to the left/top (backwards).

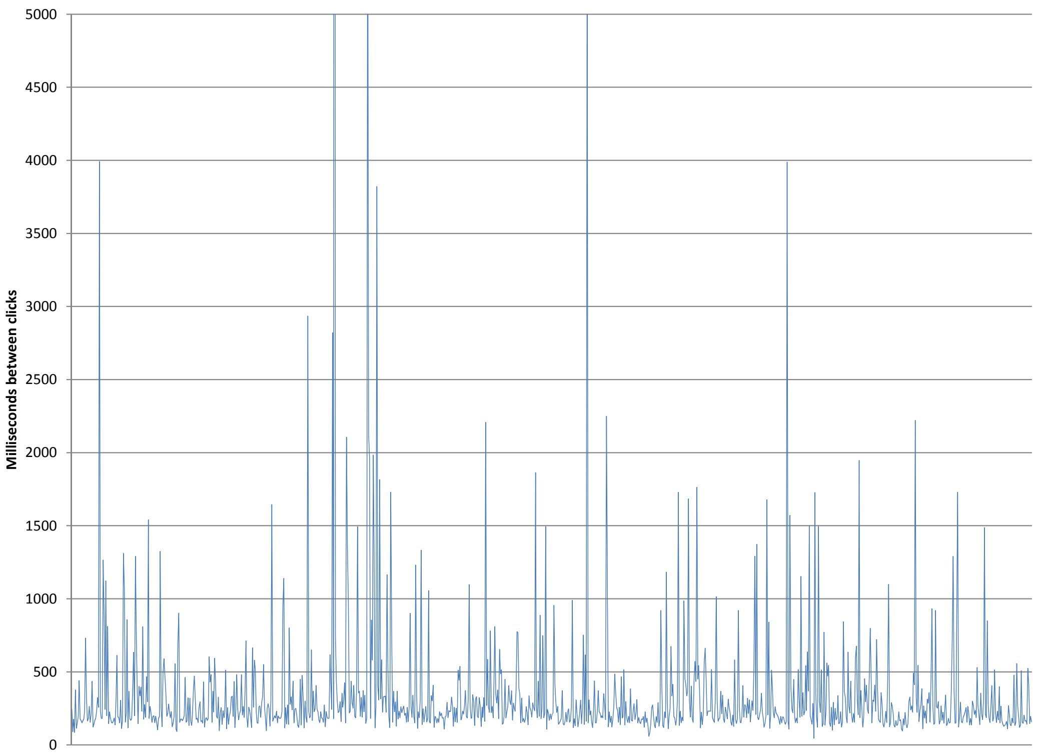


Fig. 5.21: Spikes indicating pauses when writing. Longer pauses are often an indication of the user switching to a different input.

and can be excluded from calculating the average writing speed [114].

The results of analyzing the writing speed, when expressed as characters/minute are shown in table 5.6.

Measure	*CPM*
Minimum	135.0
Maximum	245.3
Mean	175.2

Table 5.6: Writing speed per minute of all users [114]

5.5. Improving the User Interface

The results of the user experience analysis has highlighted aspects and use cases that can benefit from improvements. The following sections details ways in which the issues can be addressed.

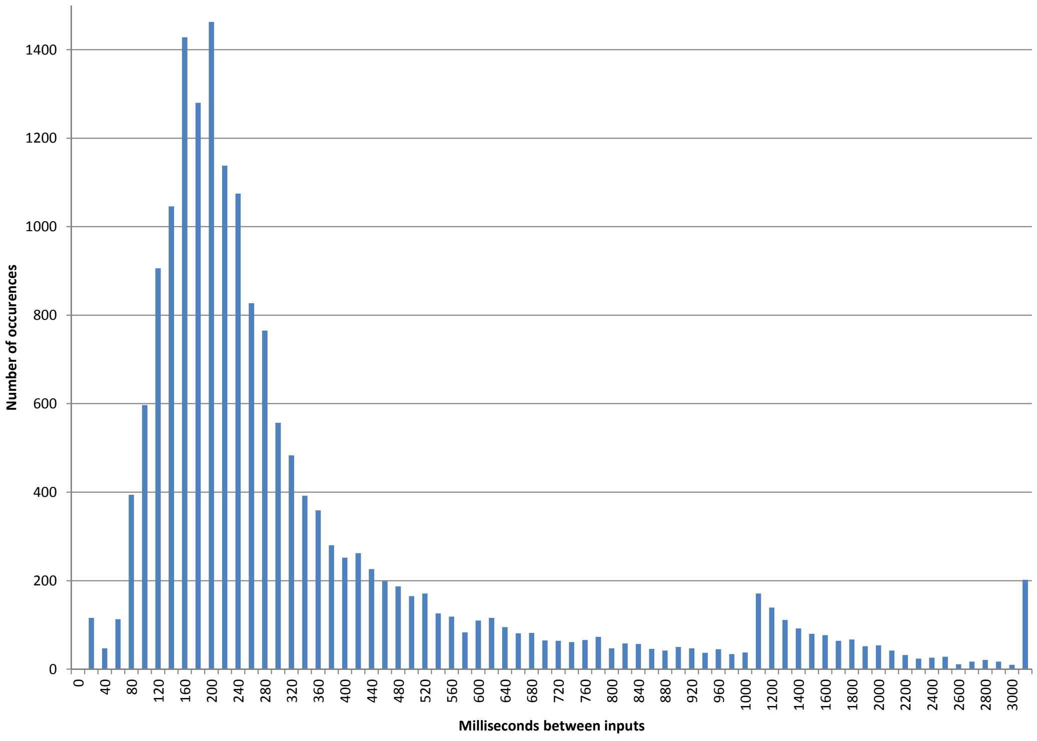


Fig. 5.22: Histogram of the duration between user key presses [114]

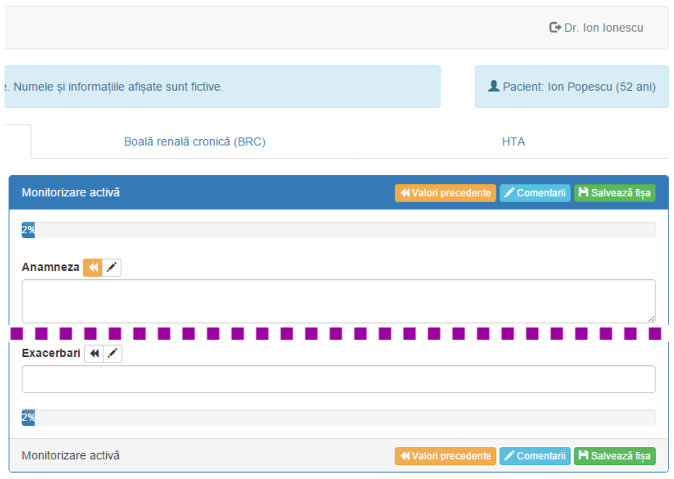
5.5.1. Improving Aesthetics

The aesthetic measures results, presented in section 5.2.2 presented a below 50% score for *unity*, a measure that "reflects coherence, a totality of elements that constitute visually one piece" [119]. By increasing the page margin, as the authors of the measure suggest, from 20 pixels to 100 pixels, the measure of unity is increased to 0.5672.

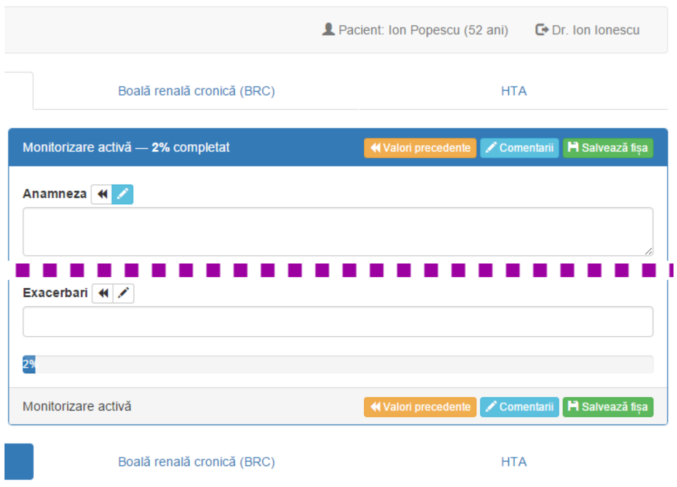
The measure of *balance* presents an above average score of 0.7706. However, a closer inspection shows that the two components that make up the measure, the horizontal and the vertical balance, have quite different scores. The vertical balance has an almost maximum score, due to the two-column layout of the web page, while the horizontal balance is slowed down by the important amount of space occupied by the page header.

A solution to this involved compacting the header, as can be seen in figure 5.23, while also duplicating the form selection tabs at the bottom of the page. To make the selector tabs at the bottom of the page stand out, the same blue background is used for the selected item, as the background for the panel header, thus creating symmetry. This also serves as a shortcut for moving on to the next form without needing to scroll back to the top.

In order to compact the header even more, the progress bar indicator, originally placed just before the first input, is merged into the panel header. This helps fill up unused space in the header while also reducing the number of different lines of information the user is first shown before the first actual input.



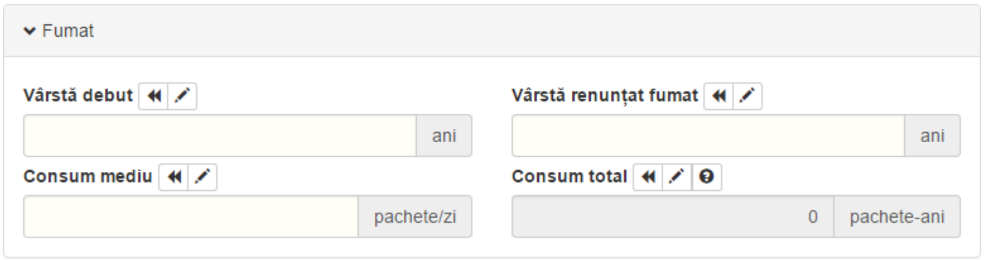
(a) Before



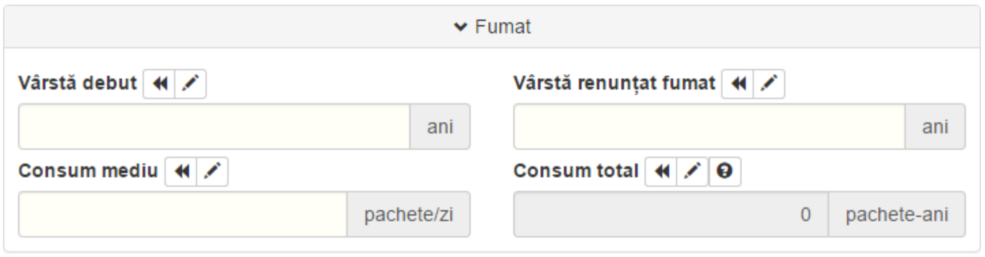
(b) After

Fig. 5.23: Compacting the header and duplicating the form selector. This enables the navigation to a different form when the user is at the bottom of the page.

The header indicating the name of an input group has been found to be too large, leading to unnecessary white space between inputs. Due to it's styling, similar to labels of inputs, and the left alignment of the text, it has also generated confusion. The solution found was to reduce the header and center the text. Figure 5.24 shows a before and after comparison.



(a) Before



(b) After

Fig. 5.24: Centering the input group header. This results in a better visual distinction between the header and the input labels above and/or below.

5.5.2. Increasing Flexibility

As presented in earlier chapters, one of the main goals of the chosen software design was to allow non-technical administrative users to easily configure and adapt data input forms.

This approach can be taken one step further by giving the user the possibility to rearrange the inputs of a form in a specific manner. By implementing this feature, the medics no longer need to adapt their way of work to that of the system, but can also apply changes to the system in order to make it more fit for day-to-day activities.

One easy way to implement the rearrangement of input elements is to use drag&drop. As shown in figure 5.25, the user hovers the mouse pointer over an input element or input group and then, while clicking the left mouse button, drags move to a new location. The web page provides guidance by using a colored horizontal line to indicate the destination placement. Once the user releases the mouse button, the input is moved to the new location.

By using GOMS this approach can be represented as: HPKPK with a duration of 3 seconds.

The new placement of elements can then be persisted using multiple approaches:

- *Temporary, until the page changes* – nothing is persisted and the advantages of single page applications not requiring page reloads so often allows for this temporary storage of element order by just keeping the variables of the current page in scope

- *Temporary, but until the end of the session* – this can be implemented by storing the customization data in a temporary server-side session
- *Persistent, but just on the current machine* – the user might want different settings for different work environments. In order to distinguish between such environments, the customizations can be stored in *localStorage* by means of the Web Storage API[10].
- *Persistent* – should there be a desire to apply the customizations on any login session, independent of the browser used, the data can be stored in a server-side database along side other information relevant to the user.

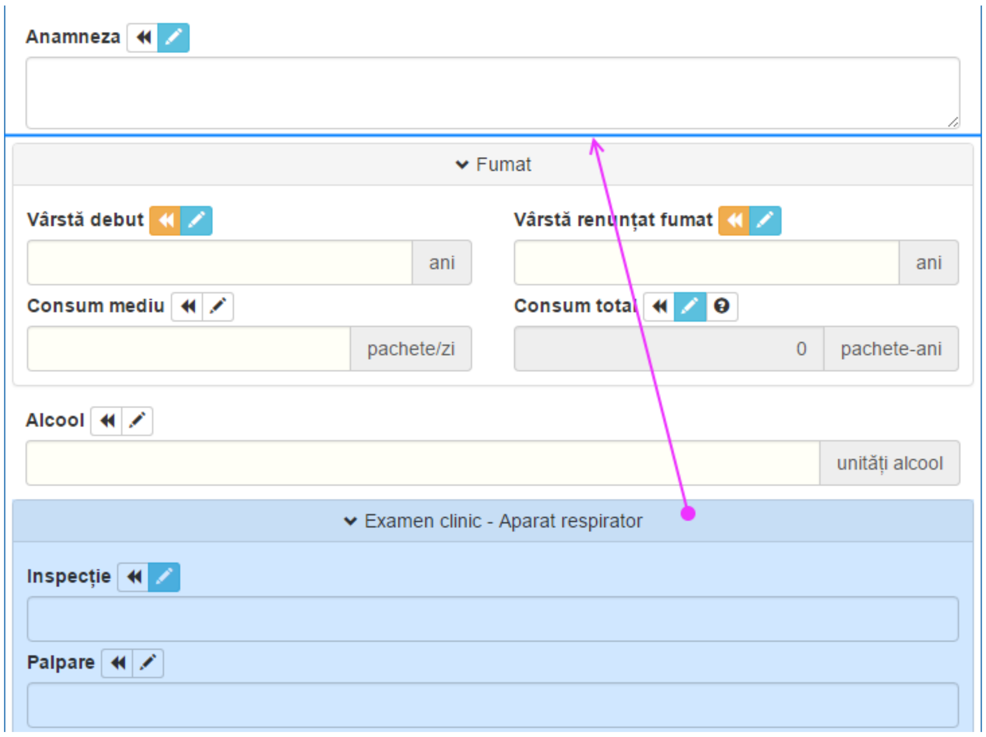


Fig. 5.25: Using drag&drop to rearrange input groups

[10]https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

Conclusions

This chapter treated a different tier when it comes to electronic health record application development, the presentation tier or the user interface, providing important contributions to architecting this part of the application as well as evaluating it.

The first major contribution of this chapter involved extending the previously presented architecture built around input types to support generating a user-interface automatically from the same model, combining web services and modern single-page applications as technologies. The types of inputs for which user interface elements are generated range from simple text boxes to complex combinations of elements with suggested values, measurement units and automatic interpretation of inputs based on complex rules and linking multiple provided values.

The chapter continued with contributions on evaluating the user interface of an application prototype that was built. At first, heuristic and aesthetic measures were applied to the prototype providing reassurance as well as indicating areas that can benefit from improvements. Later on, a usability evaluation was contributed with a detailed analysis on the findings discovered after medics and medical students piloted a test scenario on the application prototype. Measuring the activity of the users that interacted with the system posed another challenge which lead to an original contribution regarding the development of an efficient activity tracking framework.

6. CONCLUSIONS

In conclusion, the thesis has proved to be a success, providing important and original contributions to methods regarding the way electronic health record systems can be designed at both the business and data layers as well as concerning the user interface.

At the beginning, chapter 2, *Medical Standards and Terminologies*, presented an extensive literature study on current methods and standards created to support developing health related applications.

The chapter started with an analysis of Health Leven Seven's Clinical Document Architecture, a standard focused on structuring electronic health information for the purpose of information exchange. After pointing out the key points of HL7 CDA, the study also cited various literature opinions on disadvantages and short-comings of using that standard.

In an attempt to identify other approaches, the literature study continued with analyzing the methodology of openEHR. This approach favors reusable content created by medical domain experts. From the technical point of view, it presents how software should interact and make use of such content (archetypes, templates) but doesn't introduce constraints into how the implementation should be created (frameworks, persistence methods, etc.).

Besides standards and methodologies for managing electronic health records, the chapter also looked at medical coding systems that are freely available: the Logical Observation Identifiers Names and Codes (mostly used for coding laboratory results) and International Classification for Diseases, both used in multiple countries.

Chapter 3, *Designing a Solution for the Structured Collection of Medical Data*, presented an original contribution consisting in a new architecture for the business and data layer of an electronic health record application based on placing the main focus on modeling the types of inputs that can occur in a medical input form and building the solution around them.

Such an approach allows focus on the present requirements and still manages to provide a good separation of what parts of the application require a developer to work on the code and what parts allow a non-technical domain expert, with medical training, to perform tweaks and customizations on the business data.

When it comes to persistence solutions, the chapter presented a classical relational database example, coupled with an object-relational mapper to ease the link between relational data and object-oriented entities. Besides that, another example was given which uses a more modern, NoSQL, approach for storing EHR records in document databases.

The methods described in chapter 3 can benefit from using the standards and specifications described in chapter 2. In this regard, chapter 4, *Integrating Medical Standards*, brought contributions on how an existing electronic health record system can be integrated with openEHR archetypes for extracting predefined medical rules and terminology links. It also suggested an architecture for generating code that makes use of clinical documents to transfer medical information between electronic health record systems.

Chapter 4 also emphasized the role of medical coding and terminology standards. In the case of Logical Observation Identifiers Names and Codes, a system which has only been translated into few languages, the chapter brought an important original contribution by means of a study into how these codes can be more easily translated. The study presented a method which resulted in a decrease of 65% in the average length of each value that requires translation, together with a reduction by about 27% of the total number of unique values that need to be translated.

The same chapter also brought an original contribution in regards to International Classification for Diseases diagnostic codes. In this case it analyzed how ICD-10 codes could be matched by name using four full-text search engines (two dedicated ones and two part of relational databases). The results showed the viability of using open-source full-text search engines for finding precise matches when searching such codes, indicating the possibility of using about 2.6 words, on average, for a query that uniquely identifies a code in 86% of cases (best-case scenario).

This thesis also treated an aspect that is quite often overlooked by developers: building an efficient front-end for an electronic health record application. Chapter 5, *Developing a Web Front-End for Electronic Health Records*, compared multiple approaches and proposed and implemented a prototype of a flexible solution based on passing the information about the structure of each input form and its constituent input elements from the server to the client side code. The client side code, a single-page rich-client application, would then use that information to generate the required input elements on the fly and also link them to the required behavior.

In the case of usability, the chapter contains contributions that analyze multiple documented ways of user interface evaluation, from heuristics, to aesthetics calculated from the geometry of a web page. Another original contribution is presented in the form of describing how to build a simple yet powerful framework that tracks the users' actions. This framework is put to work in a user evaluation involving multiple medical doctors and residents, an evaluation which reveals important ways in which the web application is used and which are the areas for improvement.

The goals set out in the first chapter have been accomplished. With the help of rapid prototyping, types of inputs have been stored in XML documents as soon as they have been identified, following the analysis of medical forms. The thesis proposes a flexible architecture for storing electronic health record inputs and also presents an implementation of the business and data layer. Another important goal that the thesis deals with is identifying ways of including medical standards, terminologies and specifications into an existing solution. When it comes to user interaction, an entire chapter is dedicated to building and evaluating the usability of a web application aimed at electronic health records.

On the basis of those presented, the list of claimed original contributions is as follows:

- A new architecture for the business layer of an electronic health record application based on placing the main focus on modeling the types of inputs
- Considerations on persisting electronic health records based on the above architecture using both relational databases and NoSQL

- Contributions on how an existing electronic health record system can be integrated with openEHR archetypes for extracting predefined medical rules and terminology links
- An architecture for generating code that makes use of clinical documents to transfer medical information between electronic health record systems
- A study on how to make use of patterns in order to reduce the amount of work required for translating LOINC codes
- An analysis on how ICD-10 codes can be matched by name using full-text search engines
- A prototype web application that uses metadata transferred from the server to generate EHR forms on-the-fly
- A heuristic and aesthetic analysis of a prototype EHR web application
- Ways of building a simple yet powerful framework for tracking user activity for the purpose of usability analysis
- Usability evaluation on a EHR web application combined with improvement suggestions

As with any research, various directions have been identified which warrant further studies. Among the first is a deeper dive into ways of automating the analysis of electronic health data/documents with the goal of being able to automatically generate connectors that import or export specific information. When it comes to medical codes, studies into optimal ways of matching them can be extended and compared for multiple languages besides English. In the case of usability evaluations of health record applications, recording the activity of users using a production system can reveal important new insights.

REFERENCES

[1] J. H. van Bemmel and M. A. Musen. *Handbook of Medical Informatics*. Bohn Stafleu Van Loghum, Houten, The Netherlands, 2000.

[2] Health Level Seven International. Introduction to: HL7 Reference Information Model (RIM). https://www.hl7.org/documentcenter/public_temp_0E9D15E0-1C23-BA17-0C66E77606FCB023/calendarofevents/himss/2011/HL7%20Reference%20Information%20Model.pdf, 2011. Accessed: May 2016.

[3] Ji Hyun Yun and Il Kon Kim. Processing HL7-CDA Entry for Semantic Interoperability. In *2007 International Conference on Convergence Information Technology*, pages 1939–1944, 2007.

[4] Tuncay Namli, Gunes Aluc, and Asuman Dogac. An Interoperability Test Framework for HL7-Based Systems. *IEEE Transactions on Information Technology in Biomedicine*, 13(3):389–399, 2009.

[5] Mihaela Vida, Oana Lupșe, Lăcrămioara Stoicu-Tivadar, and Vasile Stoicu-Tivadar. ICT solution supporting continuity of care in children healthcare services. In *2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pages 635–639, 2011.

[6] George I. Mihalas, Dan D. Farcas, Diana Lungeanu, and Mircea Focsa. Building eHealth National Strategies – The Romanian Experience. In *Medical Informatics in a United and Healthy Europe*, pages 33–37. IOS Press, 2009.

[7] Naeem Khalid Janjua, Maqbool Hussain, Muhammad Afzal, and Hafiz Farooq Ahmad. Digital health care ecosystem: SOA compliant HL7 based health care information interchange. In *2009 3rd IEEE International Conference on Digital Ecosystems and Technologies*, pages 329–334, June 2009.

[8] Rita Noumeir and Jean-François Pambrun. Hands-on approach for teaching HL7 version 3. In *Proceedings of the 10th IEEE International Conference on Information Technology and Applications in Biomedicine*, pages 1–4, 2010.

[9] HL7 International. Clinical Document Architecture (CDA). https://www.hl7.org/fhir/comparison-cda.html, 2015. Accessed: March 2016.

[10] Alshuler Associates LLC. HL7 Implementation Guide: For Simple CDA Release 2 Documents. http://wiki.siframework.org/file/view/CDAQuickStart.doc, 2007. Accessed: January 2014.

[11] Mihaela Vida, Valentin Gomoi, Lăcrămioara Stoicu-Tivadar, and Vasile Stoicu-Tivadar. Generating medical computer-based protocols using standardized data transmission. In *2010 4th International Workshop on Soft Computing Applications (SOFA)*, pages 155–158, 2010.

[12] Marten Smits, Ewout Kramer, Martijn Harthoorn, and Ronald Cornet. A comparison of two Detailed Clinical Model representations: FHIR and CDA. In *European Journal for Biomedical Informatics*, volume 11, pages 7–17. EuroMISE s.r.o, Czech Republic, 2015.

[13] René Spronk (Editor). HL7 version 3: Message or CDA Document? http://www.ringholm.de/docs/04200_en.htm, 2007. Accessed: May 2016.

[14] Grahame Grieve. Health Intersections. http://www.healthintersections.com.au/wp-content/uploads/2011/05/RIM.png,

2011. Accessed: March 2015.

[15] Mustafa Yuksel and Asuman Dogac. Interoperability of Medical Device Information and the Clinical Applications: An HL7 RMIM based on the ISO/IEEE 11073 DIM. *IEEE Transactions on Information Technology in Biomedicine*, 15(4):557–566, 2011.

[16] Patricia A. H. Williams, Sarah Gaunt, Grahame Grieve, Vincent McCauley, and Hugh Leslie. The Development of a National Approach to CDA: Successes, Challenges, and Lessons Learned from Australia. In *European Journal for Biomedical Informatics*, volume 8, pages 37–44. EuroMISE s.r.o, Czech Republic, 2012.

[17] Thilo Schuler, Martin Boeker, Rüdiger Klar, and Marcel Müller. A Generic, Web-based Clinical Information System Architecture Using HL7 CDA: Successful Implementation in Dermatological Routine Care. In *Medinfo 2007: Proceedings of the 12th World Congress on Health (Medical) Informatics; Building Sustainable Health Systems*, pages 439–443. IOS Press, Amsterdam, Netherlands, 2007.

[18] Health Level Seven, Inc. HL7 Implementation Guide for CDA Release 2: NHSN Healthcare Associated Infection (HAI) Reports, Release 2. http://www.cdc.gov/nhsn/PDFs/CDA/CDAR2L3_IG_HAIRPT_R2_D2_2009FEB.pdf, 2009. Accessed: January 2014.

[19] Naoki Mihara, Kanayo Ueda, Shirou Manabe, Toshihiro Takeda, Yoshie Shimai, Hiroyuki Horishima, Taizo Murata, Ayumi Fujii, and Yasushi Matsumura. Cross-institutional document exchange system using clinical document architecture (CDA) with virtual printing method. In *Digital Healthcare Empowering Europeans. Proceedings of MIE 2015*, pages 444–448. IOS Press, Amsterdam, Netherlands, 2015.

[20] P. H. Cheng, C. H. Yang, H. S. Chen, S. J. Chen, and J. S. Lai. Application of HL7 in a Collaborative Healthcare Information System. In *Engineering in Medicine and Biology Society, 2004. IEMBS '04. 26th Annual International Conference of the IEEE*, volume 2, pages 3354–3357, Sept 2004.

[21] Barbara Giannini, Roberta Gazzarata, Patrizia Orcamo, Caterina Merlano, Giovanni Cenderello, Alberto Venturini, Antonio Di Biagio, Giovanni Mazzarello, Marcello Montefiori, Marta Ameri, Maurizio Setti, Claudio Viscoli, Giovanni Cassola, and Mauro Giacomini. IANUA: a regional project for the determination of costs in HIV-infected patients. In *Digital Healthcare Empowering Europeans. Proceedings of MIE 2015*, pages 241–245. IOS Press, Amsterdam, Netherlands, 2015.

[22] Barry Smith and Werner Ceusters. HL7 RIM: An Incoherent Standard. In *Ubiquity: Technologies for Better Health in Aging Societies, Proceedings of MIE2006*, volume 124, pages 133–138, 2006.

[23] Eduardo Fernandez and Tami Sorgente. An Analysis of Modeling Flaws in HL7 and JAHIS. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, pages 216–223, New York, NY, USA, 2005. ACM.

[24] Eric Browne. openEHR Archetypes for HL7 CDA Documents. https://openehr.atlassian.net/wiki/display/stds/openEHR+Archetypes+for+HL7+CDA+Documents, 2008. Accessed: March 2016.

[25] M. Schweitzer, N. Lasierra, and A. Hoerbst. Observing health professionals' workflow patterns for diabetes care – First steps towards an ontology for EHR services. In *Digital Healthcare Empowering Europeans. Proceedings of MIE 2015*, pages 25–29. IOS Press, Amsterdam, Netherlands, 2015.

[26] Barry Smith. The Rise and Fall of HL7. http://hl7-watch.blogspot.com/2011/03/rise-and-fall-of-hl7.html, 2011. Accessed: May 2016.

[27] Philip J. Kroth, Shamsi Daneshvari, Edward J. Harris, Daniel J. Vreeman, and Heather J.H. Edgar. Using LOINC to link 10 terminology standards to one unified standard in a specialized domain. *Journal of Biomedical Informatics*, 45(4):674 – 682, 2012.

[28] Clem McDonald, Stan Huff, Jamalynne Deckard, Katy Holck, and Daniel J. Vreeman. *Logical Observation Identifiers Names and Codes (LOINC®) Users' Guide*. Regenstrief Institute, Inc., 2013.

[29] Daniel J. Vreeman, Maria Teresa Chiaravalloti, John Hook, and Clement J. McDonald. Enabling international adoption of LOINC through translation. In *J BIOMED INFORM 45*, pages 667–673, 2012.

[30] Regenstrief Institute, Inc. International – LOINC. https://loinc.org/international, 2016. Accessed: May 2016.

[31] Hyeoneui Kim, Robert El-Kareh, Anupam Goel, FNU Vineet, and Wendy W. Chapman. An approach to improve LOINC mapping through augmentation of local test names. In *J BIOMED INFORM 45*, pages 651–657, 2012.

[32] Christian Zunner, Thomas Bürkle, Hans-Ulrich Prokosch, and Thomas Ganslandt. Mapping local laboratory interface terms to LOINC at a German university hospital using RELMA V.5: a semi-automated approach. *Journal of the American Medical Informatics Association*, 20(2):293–297, 2013.

[33] World Health Organization. International Classification of Diseases (ICD). http://www.who.int/classifications/icd/en/. Accessed: February 2016.

[34] National Center for Health Statistics. International Classification of Diseases, Tenth Revision, Clinical Modification (ICD-10-CM). http://www.cdc.gov/nchs/icd/icd10cm.htm, 2016. Accessed: March 2016.

[35] Hari Nandigam. Facilitating Quick and Better Text Searching for ICD-10-CM Codes. *iProc*, 1(1), 2015.

[36] Shunsuke Doi, Takashi Kimura, Takahiro Suzuki, and Katsuhiko Takabayashi. Development of Doctors Search Engine based on ICD-10. In *2012 Joint 6th International Conference on Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 795–798, Nov 2012.

[37] Koray Atalag, Thomas Beale, Rong Chen, Tomaž Gornik, Sam Heard, and Ian McNicoll. openEHR – a semantically-enabled health computing platform. http://www.openehr.org/resources/white_paper_docs/openEHR_vendor_independent_platform.pdf. Accessed: March 2016.

[38] Sergio Miranda Freire, Erik Sundvall, Daniel Karlsson, and Patrick Lambrix. Performance of XML Databases for Epidemiological Queries in Archetype-Based EHRs. In *Scandinavian Conference on Health Informatics 2012*, pages 51–57, 2012.

[39] Thomas Beale, Sam Heard, et al. openEHR Architecture – Architecture Overview. http://www.openehr.org/releases/1.0.2/architecture/overview.pdf, 2008. Accessed: April 2016.

[40] Sergio Miranda Freire, Douglas Teodoro, Fang Wei-Kleiner, Erik Sundvall, Daniel Karlsson, and Patrick Lambrix. Comparing the Performance of NoSQL Approaches for Managing Archetype-Based Electronic Health Record Data. *PLoS ONE*, 11(3):1–20, 2016.

[41] Thomas Beale, Sam Heard, et al. openEHR Architecture – Archetype Definitions and Principles. http://www.openehr.org/releases/1.0.2/architecture/am/archetype_principles.pdf, 2007. Accessed: March 2016.

[42] Thomas Beale, Sam Heard, et al. The openEHR Archetype Model – Archetype Definition Language ADL 1.4. http://www.openehr.org/releases/1.0.1/architecture/am/adl.pdf, 2007. Accessed: March 2016.

[43] Jon Patrick, Richard Ly, and Donna Truran. Evaluation of a Persistent Store for openEHR. In *HIC 2006 Bridging the Digital Divide: Clinician, consumer and computer*. Health Informatics Society of Australia Ltd, 2006.

[44] Li Wang, Lingtong Min, Rui Wang, Xudong Lu, and Huilong Duan. Archetype relational mapping - a practical openEHR persistence solution. *BMC Medical Informatics and Decision Making*, 15(1):1–18, 2015.

[45] E Sundvall, M Nyström, M Sandström, M Eneling, H Örman, and D Karlsson. REST Based Services and Storage Interfaces for openEHR Implementations. http://www.imt.liu.se/~erisu/2010/EEE-Poster-multipage.pdf, 2010. Accessed: April 2016.

[46] Erik Sundvall, Mikael Nyström, Daniel Karlsson, Martin Eneling, Rong Chen, and Håkan Örman. Applying representational state transfer (REST) architecture to archetype-based electronic health record systems. *BMC Medical Informatics and Decision Making*, 13(1):1–25, 2013.

[47] Thomas Beale. openEHR – Integrating with data source systems. http://www.mz.gov.si/fileadmin/mz.gov.si/pageuploads/eZdravje/Novice/gradiva_predstavitve_dogodkov/Open_EHR/9_integration-non_EHR.pdf, 2007. Accessed: April 2016.

[48] Diego Bosca, David Moner, Jose Alberto Maldonado, and Montserrat Robles. Combining Archetypes with Fast Health Interoperability Resources in Future-proof Health Information Systems. In *Digital Healthcare Empowering Europeans. Proceedings of MIE 2015*, pages 180–184. IOS Press, Amsterdam, Netherlands, 2015.

[49] Ursula Hübner, Georg Schulte, Björn Sellemann, Matthias Quade, Thorsten Rottmann, Matthias Fenske, Nicole Egbert, Raik Kuhlisch, and Otto Rienhoff. Evaluating a Proof-of-Concept Approach of the German Health Telematics Infrastructure in the Context of Discharge Management. In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, pages 492–496. IOS Press, Amsterdam, Netherlands, 2015.

[50] Pernille Bertelsen and Lone Stub Petersen. Danish Citizens and General Practitioners' Use of ICT for their Mutual Communication. In *MEDINFO 2015:*

eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics, pages 376–379. IOS Press, Amsterdam, Netherlands, 2015.

[51] William Van Woensel, Patrice C. Roy, Samina R. Abidi, and Syed SR Abidi. A Mobile and Intelligent Patient Diary for Chronic Disease Self-Management. In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, pages 118–122. IOS Press, Amsterdam, Netherlands, 2015.

[52] Joseph M.J. D'Souza and Inga Hunter. Why should I? – Acceptance of Health Information Technology Among health professionals. In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, page 962. IOS Press, Amsterdam, Netherlands, 2015.

[53] Ministerul Sănătății, Casa Națională de Asigurări de Sănătate. Referat de aprobare a Ordinului ministrului sănătăţii şi al preşedintelui Casei Naţionale de Asigurări de Sănătate privind aprobarea Normelor metodologice de aplicare în anul 2015 a Hotărârii Guvernului nr. 400/2014. http://www.cnas.ro/media/pageFiles/norme_metodologice_cnas_2015.pdf, 2015. Accessed: September 2015.

[54] Formare Medicala. Fișa de evaluare inițială/monitorizarea pac. cu BPOC. http://www.formaremedicala.ro/wp-content/uploads/2014/09/FISA-BPOC-evaluare-initiala-si-monitorizare-activa.pdf, 2014. Accessed: September 2014.

[55] American Heart Association. Understanding Blood Pressure Readings. http://www.heart.org/HEARTORG/Conditions/HighBloodPressure/AboutHighBloodPressure/Understanding-Blood-Pressure-Readings_UCM_301764_Article.jsp, 2015. Accessed: August 2015.

[56] Kordon F. et al. An introduction to rapid system prototyping. In *Software Engineering, IEEE Transactions on (Volume: 28, Issue: 9)*, pages 817–821. IEEE, 2002.

[57] Albrecht Schmidt and Kjetil Nørvåg. Rapid XML Database Application Development. In *6th International Conference on Enterprise Information Systems*, 2004. Accessed: September 2014.

[58] Formare Medicala. Fișa de evaluare inițială/monitorizare activă a pacientului hipertensiv sau/și diabetic cu/fără dislipidemie. http://www.formaremedicala.ro/wp-content/uploads/2014/09/FISA-HTA-DZ-evaluare-initiala-si-monitorizareH-activa.pdf, 2014. Accessed: September 2014.

[59] Martin Fowler, Dave Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford. *Patterns of Enterprise Application Architecture*. Addison-Wesley, USA, 2003.

[60] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled*. Addison-Wesley, Upper Saddle River, NJ, USA, 2013.

[61] Julia Lerman. *Programming Entity Framework*. O'Reilly, Sebastopol, CA, USA, 2nd edition, 2010.

[62] Daniel-Alexandru Jurcău and Vasile Stoicu-Tivadar. Modern Technologies for Improving Interoperability in Health Information Systems. In *Buletinul*

Științific al Universității Politehnica Timișoara, pages 53–58, 2014.

[63] Eizen Kimura and Ken Ishihara. Virtual file system on NoSQL for processing high volumes of HL7 messages. In *Digital Healthcare Empowering Europeans. Proceedings of MIE 2015*, pages 687–691. IOS Press, Amsterdam, Netherlands, 2015.

[64] Daniel-Alexandru Jurcău, Vasile Stoicu-Tivadar, and Alexandru Șerban. Using Modern Technologies to Facilitate Translating Logical Observation Identifiers Names and Codes. In *Proceedings of the 6th International Workshop Soft Computing Applications (SOFA 2014)*, volume 1, pages 219–229, Switzerland, 2014. Springer.

[65] Vasile Topac, Daniel-Alexandru Jurcau, and Vasile Stoicu-Tivadar. Incidence Rate of Canonical vs. Derived Medical Terminology in Natural Language. In *Digital Healthcare Empowering Europeans. Proceedings of MIE 2015*, pages 5–9. IOS Press, Amsterdam, Netherlands, 2015.

[66] Vasile Topac. *Improving Text Accessibility and Understanding of Domain-Specific Information*. Editura Politehnica, Timisoara, RO, 2014.

[67] Kevin Hazzard and Jason Bock. *Metaprogramming in .NET*. Manning, Shelter Island, NY, USA, 2013.

[68] Design-Time Code Generation by using T4 Text Templates. http://msdn.microsoft.com/en-us/library/dd820620.aspx, 2013. Accessed: April 2014.

[69] The PHP Group. PHP Data Objects. http://www.php.net/manual/en/book.pdo.php, 2014. Accessed: April 2014.

[70] The jQuery Foundation. jQuery. http://jquery.com/, 2014. Accessed: February 2014.

[71] Adam Freeman. *Pro jQuery 2.0*. Apress, New York, NY, USA, 2013.

[72] RequireJS – A Javascript Module Loader. http://requirejs.org/, 2014. Accessed: February 2014.

[73] Daniel-Alexandru Jurcău and Vasile Stoicu-Tivadar. Evaluating Open-Source Full-Text Search Engines for Matching ICD-10 Codes. In *14th International Conference on Informatics, Management and Technology in Healthcare*, 2016.

[74] Ping Chen, Araly Barrera, and Chris Rhodes. Semantic analysis of free text and its application on automatically assigning ICD-9-CM codes to patient records. In *9th IEEE International Conference on Cognitive Informatics (ICCI)*, pages 68–74, July 2010.

[75] Wencheng Cui, Mengjia Xu, Huayu Sun, and Hong Shao. Research on application of Lucene in medical image retrieval system. In *2011 International Conference on Computer Science and Network Technology (ICCSNT)*, volume 2, pages 661–664, Dec 2011.

[76] The PostgreSQL Global Development Group. PostgreSQL 9.4.7 Documentation. 8.11. Text Search Types. http://www.postgresql.org/docs/9.4/static/datatype-textsearch.html, 2016. Accessed: March 2016.

[77] Quest Diagnostics™. Diagnostic Services ICD-10 Common Codes. https://questdiagnostics.com/dms/Documents/Other/ICD-10-CPT-2012/37783-v1-37783-ICD_10_Common_Codes_-_MI3862-ONLINE-051115.pdf, 2015. Accessed: March 2016.

[78] The openEHR Foundation. Archetype Editor Home. http://www.openehr.org/downloads/archetypeeditor/home, 2015. Accessed: April 2016.

[79] Chunlan Ma, Heath Frankel, and Thomas Beale. openEHR – Archetype Query Language (AQL). http://www.openehr.org/releases/QUERY/latest/docs/AQL/AQL.html, 2016. Accessed: April 2016.

[80] Regenstrief Institute, Inc. LOINC® from Regenstrief. https://loinc.org/, 2013. Accessed: March 2014.

[81] Bruno Sonnino. Aspect-Oriented Programming : Aspect-Oriented Programming with the RealProxy Class. https://msdn.microsoft.com/en-us/magazine/dn574804.aspx, 2014. Accessed: April 2016.

[82] Jennifer Munnelly and Siobhan Clarke. HL7 healthcare information management using aspect-oriented programming. In *Computer-Based Medical Systems, 2009. CBMS 2009. 22nd IEEE International Symposium on*, pages 1–4, 2009.

[83] Dmitry Baranov. HL7 SDK - Open Source CDA R2 Implementation for .NET and COM. https://hl7sdk.codeplex.com/, 2013. Accessed: May 2016.

[84] Ocean Informatics. Automating the production of CDA R2 artefacts using openEHR Archetypes and Templates. https://www.hl7.org/documentcenter/public_temp_DFE89F2D-1C23-BA17-0CFCEFA5EBE5E9C0/wg/patientcare/minutes/NEHTA_CDA_Seminar_05122007.pptx, 2007. Accessed: April 2016.

[85] Aviv Shachak, Sharon Domb, Elizabeth Borycki, Nancy Fong, Alison Skyrme, Andre Kushniruk, Shmuel Reis, and Amitai Ziv. A Pilot Study of Computer-Based Simulation Training for Enhancing Family Medicine Residents' Competence in Computerized Settings. In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, pages 506–510. IOS Press, Amsterdam, Netherlands, 2015.

[86] Enrique Stanziola, María Quispe Uznayo, Juan Marcos Ortiz, Mariana Simón, Carlos Otero, Fernando Campos, and Daniel Luna. User-Centered Design of Health Care Software Development: Towards a Cultural Change. In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, pages 368–371. IOS Press, Amsterdam, Netherlands, 2015.

[87] Luís A. Bastião Silva, Carlos Días, Johan van der Lei, and José Luis Oliveira. Architecture to Summarize Patient-Level Data Across Borders and Countries. In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, pages 687–690. IOS Press, Amsterdam, Netherlands, 2015.

[88] Adam Freeman. *Pro AngularJS*. Apress, New York, NY, USA, 2014.

[89] Parth Ghiya. Single Page Applications. http://www.knowarth.com/single-page-applications/, 2014. Accessed: August 2015.

[90] Peter Herlihy. Government Digital Service – How many people are missing out on JavaScript enhancement? https://gds.blog.gov.uk/2013/10/21/how-many-people-are-missing-out-on-javascript-enhancement/, 2013.

Accessed: February 2016.

[91] Thomas Beale, Sam Heard, et al. The openEHR Archetype Model – openEHR Templates. http://openehr.org/releases/trunk/architecture/am/tom.pdf, 2014. Accessed: March 2016.

[92] Rabea Krexner and Georg Duftschmid. Plug-and-play Integration of dual-model based Knowledge Artefacts into an Open Source EHR System. In *e-Health – For Continuity of Care. Proceedings of MIE 2014*, pages 101–105. IOS Press, Amsterdam, Netherlands, 2014.

[93] Erik Bruchez et al. XForms 2.0. https://www.w3.org/community/xformsusers/wiki/XForms_2.0, 2016. Accessed: April 2016.

[94] European Guidelines on CVD Prevention in Clinical Practice. SCORE - European High Risk Chart. http://www.escardio.org/static_file/Escardio/Subspecialty/EACPR/Documents/score-charts.pdf, 2012. Accessed: September 2014.

[95] Ecma International. ECMA-404 Standard – The JSON Data Interchange Format. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, 2013. Accessed: September 2015.

[96] Stefan Kropf, Claire Chalopin, and Kerstin Denecke. Template and Model Driven Development of Standardized Electronic Health Records. In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, pages 30–34. IOS Press, Amsterdam, Netherlands, 2015.

[97] Valerio De Luca, Italo Epicoco, Daniele Lezzi, and Giovanni Aloisio. A Web API Framework for Developing Grid Portals. In *International Conference on Computational Science, ICCS 2011*, pages 392–401. Procedia Computer Science, 2011.

[98] Jamie Kurtz. *ASP.NET MVC 4 and the Web API*. Apress, New York, NY, USA, 2013.

[99] Sam Newman. *Building Microservices*. O'Reilly, USA, 2015.

[100] Nafees Qamar, Yilong Yang, András Nádas, Zhiming Liu, and Janos Sztipanovits. Anonymously Analyzing Clinical Datasets. *CoRR*, abs/1501.05916, 2015.

[101] Halgeir Holthe and J. Artur Serrano. ePoint.telemed – An Open Web-based Platform for Home Monitoring of Patients with Chronic Heart Failure. In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, pages 74–78. IOS Press, Amsterdam, Netherlands, 2015.

[102] Pablo Pazos Gutiérrez. Towards the Implementation of an openEHR-based Open Source EHR Platform (a vision paper). In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, pages 45–49. IOS Press, Amsterdam, Netherlands, 2015.

[103] Mani Abedini, Stefan von Cavallar, Rajib Chakravorty, Matthew Davis, and Rahil Garnavi. A Cloud-Based Infrastructure for Feedback-Driven Training and Image Recognition. In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, pages

691–695. IOS Press, Amsterdam, Netherlands, 2015.

[104] Diego Garcia, Claudia Maria C. Moro, and Lilian Mie M. Cintho. Bridging the Gap between Clinical Practice Guidelines and Archetype-Based Electronic Health Records: A Novel Model Proposal. In *MEDINFO 2015: eHealth-enabled Health; Proceedings of the 15th World Congress on Health and Biomedical Informatics*, page 952. IOS Press, Amsterdam, Netherlands, 2015.

[105] Yvonne Rogers, Helen Sharp, and Jenny Preece. *INTERACTION' DESIGN*. John Wiley & Sons, Inc., USA, 2002.

[106] Wilbert O. Galitz. *The Essential Guide to User Interface Design*. Wiley Computer Publishing, USA, 2002.

[107] Aaron Marcus, Karl Wieser, John Armitage, Volker Frank, and Edward Guttman. User-Interface Design for Medical Informatics: A Case Study of Kaiser Permanente. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*. IEEE, 2000.

[108] Constance M. Johnson, Todd R. Johnson, and Jiajie Zhang. A user-centered framework for redesigning health care interfaces. In *Journal of Biomedical Informatics 38*, pages 75–87. Elsevier, 2005.

[109] Kai Zheng, Rema Padman, and Michael P. Johnson. User Interface Optimization for an Electronic Medical Record System. In *Proceedings of the 12th World Congress on Health (Medical) Informatics*, pages 1058–1062. IOS Press, 2007.

[110] Diana Rueda, René Hoto, and Andrés Conejero. *Human Factors in Computing and Informatics: First International Conference, SouthCHI 2013, Maribor, Slovenia, July 1-3, 2013. Proceedings*, pages 122–136. Springer Berlin Heidelberg, Germany, 2013.

[111] Romaric Marcilly, Andre W. Kushniruk, Marie-Catherine Beuscart-Zephir, and Elizabeth M. Borycki. Insights and limits of usability evaluation methods along the health information technology lifecycle. In *Digital Healthcare Empowering Europeans. Proceedings of MIE 2015*, pages 115–119. IOS Press, Amsterdam, Netherlands, 2015.

[112] Alan L. Montgomery, Shibo Li, Kannan Srinivasan, and John C. Liechty. Modeling Online Browsing and Path Analysis Using Clickstream Data. In *Marketing Science Volume 23, Issue 4*, 2004.

[113] Richard Atterer, Monika Wnuk, and Albrecht Schmidt. Knowing the User's Every Move: User Activity Tracking for Website Usability Evaluation and Implicit Interaction. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 203–212, New York, NY, USA, 2006. ACM.

[114] Daniel-Alexandru Jurcău and Vasile Stoicu-Tivadar. Evaluating the User Experience of a Web Application for Managing Electronic Health Records. In *Proceedings of the 7th International Workshop Soft Computing Applications (SOFA 2016)*, volume 1, Switzerland, 2016. Springer.

[115] Jakob Nielsen. 10 Usability Heuristics for User Interface Design. https://www.nngroup.com/articles/ten-usability-heuristics/, 1995. Accessed: October 2015.

[116] Olga A. Belaya. The metrics for quantitative evaluation of user interface usability construction methodology. In *SPECOM'2004: 9th Conference Speech and Computer*. ISCA Archive http://www.isca-speech.org/archive, 2004.

[117] Aliaksei Miniukovich and Antonella De Angeli. Computation of Interface Aesthetics. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI ’15, pages 1163–1172, New York, NY, USA, 2015. ACM.

[118] David Chek Ling Ngo, Lian Seng Teo, and John G. Byrne. A Mathematical Theory of Interface Aesthetics. In *Visual Mathematics*, number 8. Mathematical Institute SASA http://eudml.org/doc/256723, 2000.

[119] David Chek Ling Ngo and John G. Byrne. Another look at a model for evaluating interface aesthetics . In *International Journal of Applied Mathematics and Computer Science*, pages 515–535. University of Zielona Gora Press, 2001.

[120] Patrick Mbenza Buanga. *Automated evaluation of graphical user interface metrics*. Université catholique de Louvain, Louvain-la-Neuve, Belgium, 2011.

[121] Stuart Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1983.

[122] Jef Raskin. *The Humane Interface. New Directions for Designing Interactive Systems*. Addison-Wesley, USA, 2000.

[123] Google Developers. Analytics for Web (analytics.js) – Single Page Application Tracking. https://developers.google.com/analytics/devguides/collection/analyticsjs/single-page-applications, 2015. Accessed: September 2015.

[124] Internet Engineering Task Force (IETF). Request for Comments: 7230 – Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. https://tools.ietf.org/html/rfc7230, 2014. Accessed: September 2015.

[125] Internet Engineering Task Force (IETF). Request for Comments: 6455 – The WebSocket Protocol. https://tools.ietf.org/html/rfc7230, 2011. Accessed: September 2015.

[126] Mon Chu Chen, John R. Anderson, and Myeong Ho Sohn. What Can a Mouse Cursor Tell Us More?: Correlation of Eye/Mouse Movements on Web Browsing. In *CHI ’01 Extended Abstracts on Human Factors in Computing Systems*, CHI EA ’01, pages 281–282, New York, NY, USA, 2001. ACM.