FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Entwurf und Entwicklung der klinischen Anwendung eines EHR-basierten Krankenhausinformationssystems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Harald Köstinger
Matrikelnummer 0625775

an der

Fakultät für Informatik der Technischen Universität Wien

Betreuung

Betreuer: Thomas Grechenig

Mitwirkung: Wolfgang Schramm

Wien, 17.10.2011 _____ _____

(Unterschrift Verfasser) (Unterschrift Betreuer)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Design and Implementation of a Clinical Application Using an EHR Based Hospital Information System

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Medical Informatics

by

Harald Köstinger

Registration Number 0625775

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Thomas Grechenig

Assistance: Wolfgang Schramm

Vienna, 17.10.2011 _____ _____

(Signature of Author) (Signature of Advisor)

Entwurf und Entwicklung der klinischen Anwendung eines EHR-basierten Krankenhausinformationssystems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Harald Köstinger

Matrikelnummer 0625775

ausgeführt am

Institut für Rechnergestützte Automation

Forschungsgruppe Industrial Software

der Fakultät für Informatik der Technischen Universität Wien

Betreuung:

Betreuer: Thomas Grechenig

Mitwirkung: Wolfgang Schramm

Wien, 17.10.2011

Love all, trust a few, do wrong to none.

William Shakespeare

Declaration of Authorship

I, Harald Köstinger, declare that this thesis titled, ”Entwurf und Entwicklung der klinischen Anwendung eines EHR-basierten Krankenhausinformationssystems” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Vienna	October 17, 2011	
....	.....	.....
Place	Date	Signature

Acknowledgements

I wish to express my sincere gratitude to my assistant advisor, Dr. Wolfgang Schramm for his inspiration, guidance and continuous support without which this thesis would have never been possible.

I would like to thank my colleagues Klaus Bayrhammer and Michael Fiedler for the invaluable discussions, ideas and for their constructive criticisms.

Also thanks to my parents and my whole family for always supporting me and believing in my ways I am going.

And I would love to thank all my friends for the patience they displayed when I was busy writing this thesis and for the many hours they just took me away to get my head free while climbing in the beautiful nature with them. You are the best!

Abstract

The basic aim of this work was the design and implementation of a medical health care application for a Hospital Information System (HIS) based on the Open Electronic Health Record (openEHR) standard, which extends an existing integration architecture for openEHR and enriched this with additional functionality.

In the area of a HIS many different individual systems are used to help and support physicians and health care personnel during their work in a hospital. One important component of a HIS is a centralized stored and maintained Electronic Health Record (EHR). There are many different standards available for electronic health records, having different approaches in how to store and visualize medical data. In the course of this work the most common and widespread standards and their characteristics are discussed and compared.

A patient synchronized application responsible for the dynamic visualization of the medical data coming from the openEHR based back-end was developed as part of this work. The implementation focused the design of a component capable of dynamically rendering archetype based health care records defined in the openEHR standard. Furthermore, additional services to the existing integration architecture were introduced to prototype e.g. anonymization components for medical data, system independent interfaces for patient synchronized applications and a Patient Master Index (PMI).

The developed front-end for maintaining medical data implements most of the defined and necessary visual input elements of the openEHR standard and supports the visualization of so called Operational Templates (OPT).

The service oriented architecture is capable of maintaining and processing patient synchronized application context data within a HIS in a system independent manner and assists the integration of web based, Intra- and Internet based solutions for medical health care applications.

The dynamic visualization of archetype based electronic health records can be achieved using the introduced front-end and patient synchronized applications can be integrated into existing HIS environments. Although the use of archetype based electronic patient records raises the complexity of the individual developed systems it has main advantages such as flexibility and therefore justifies their application.

Kurzfassung

Heutzutage sind im Bereich eines Krankenhausinformationssystems (KIS) viele verschiedene Einzelsysteme im Einsatz, die den Arbeitsalltag im Krankenhaus erleichtern sowie das Personal in ihrer Arbeit unterstützen. Ein wichtiger Bestandteil solcher KIS sind zentral gespeicherte elektronische Patientenakten (EPA, EHR), wobei es in diesem sensiblen Bereich viele verschiedene Ansätze der Datenhaltung, Speicherung und Darstellung sowie unterschiedlichste Standards gibt. Zunächst werden im Rahmen der vorliegenden Arbeit die aktuellen Trends in diesem Forschungsbereich sowie verbreitete Standards und deren Eigenschaften beleuchtet und diskutiert.

Im Bereich der KIS existieren heute praktisch keine Open Source Software Pakete, die für den wissenschaftlichen Bereich - speziell für die Evaluierung von neuen Workflow Konzepten - geeignet sind. Das Ziel dieser Arbeit ist somit der Entwurf und die Entwicklung einer klinischen Anwendung eines Electronic Health Record (EHR)-basierten KIS auf Basis von openEHR, die auf eine existierende Integrationsarchitektur für openEHR aufbaut.

Im Rahmen der Umsetzung wurde eine patientensynchronisierte Anwendung erstellt, die die Darstellung medizinischer Daten aus dem openEHR Backend ermöglicht. Der Fokus hierbei lag auf der Entwicklung einer Komponente, die dynamisch die Darstellung der archetypbasierten Patientenakten, wie sie im openEHR Standard definiert sind, übernimmt. Zugleich wurde aufbauend auf die existierende Integrationsarchitektur und deren Services eine serviceorientierte Komponente entwickelt, die exemplarisch einen Anonymisierungsservice für die medizinischen Daten anbietet, systemunabhängige Schnittstellen für eine patientensychronisierte Anwendung zur Verfügung stellt und einen Patienten Master-Index (PMI) implementiert.

Das entwickelte Frontend für die medizinischen Daten implementiert einen Großteil der in openEHR vorgesehenen Eingabeelemente und unterstützt die Darstellung von Templates. Die implementierte serviceorientierte Architektur ermöglicht die systemunabhängige Verwaltung und Verarbeitung von patientensynchronisierten Anwendungen innerhalb eines KIS und stellt Methoden bereit, welche Anbindungen für webbasierte, Intra- und Internet basierte Software bieten.

Es wurde gezeigt, wie die dynamische Darstellung von archetypbasierten elektronischen Patientenakten gehandhabt werden kann und wie patientensynchronisierte Anwendungen in einem solchen Umfeld eingebunden werden.

Terms and Abbreviations

ADL Archetype Description Language 35, 51, 52, 55, 56

AM Archetype Model 32–35, 65

AOM Archetype Object Model 65, 66

API Application Programming Interface 30, 85

ATNA Audit Trail and Node Authentication 76

CCOW Clinical Context Object Workgroup 77, 78

CDA Clinical Document Architecture 3, 19, 21–23, 38, 72, 75

CEN European Committee for Standardization 3, 4, 6, 16, 22, 26, 30, 38

CT Computed Tomography 5

CT Consistent Time 76

ECG Electrocardiogram 5

EDIFACT Electronic Data Interchange For Administration, Commerce and Transport 30

EHR Electronic Health Record x, 2–5, 7, 9–20, 22, 26–30, 33, 35, 37, 39, 41, 42, 44, 47, 49, 50, 53, 54, 57, 58, 72, 74, 83

ERP Enterprise Resource Planning 5

ESB Enterprise Service Bus 44, 47, 49–51

EUA Enterprise User Authentication 75

GUI Graphical User Interface 7, 8, 53, 64, 67, 68, 70

HIS Hospital Information System x, 2, 3, 5–7, 11–13, 17, 18, 30, 41–43, 46, 49, 53, 79, 81

HL7 Health Level 7 3, 19–22, 30, 38, 72, 77, 78

IHE Integrating the Health Care Enterprise 4, 24, 25, 72, 74–76

IoC Inversion of Control 56, 68

ISO International Organization for Standardization 4, 16

JPEG Joint Photographic Experts Group 75

MRI Magnetic Resonance Imaging 5

MSMQ Microsoft Message Queuing 82

MST Minimal Standard Terminology for Digestive Endoscopy 8, 64

MVVM Model-View-ViewModel 67

openEHR Open Electronic Health Record x, 2–4, 6–9, 29–39, 41, 42, 44, 46, 49, 51, 53–56, 64–66, 83, 86, 88

Opereffa openEHR REFerence Framework and Application 8

OPT Operational Template x, 42, 52–54, 64, 65, 68

PACS Picture Archiving and Communication System 5

PCP Patient Context Participant 78, 79, 85

PDF Portable Document Format 75

PDQ Patient Demographic Query 76

PHR Personal Health Record 14

PIX Patient Identifier Cross-referencing 75, 84, 85

PMA Patient Mapping Agent 85

PMI Patient Master Index x, 9, 42, 54, 56–58

PSA Patient Synchronized Application 2, 5, 6, 9, 10, 41–45, 53, 54, 56–63, 72, 74–79, 81–85

PWP Personnel White Pages 76

RID Retrieve Information for Display 75

RIM Reference Information Model 20–23, 38

RIS Radiology Information System 5

RM Reference Model 32–35, 64, 65, 68, 70

SDE Structured Data Entry Component 8, 64

SM Service Model 33

SNOMED-CT Systematized Nomenclature of Medicine - Clinical Terms 30

URL Uniform/Universal Resource Locator 51

WCF Windows Communication Foundation 42–44, 60, 61, 82

WPF Windows Presentation Foundation 45, 59, 64, 65, 67

XDS Cross-Enterprise Document Sharing 4, 76

XML Extensible Markup Language 22, 23, 72

Contents

Start

CHAPTER 1

Problem Statement and Basic Idea

Main goal of this work is to design and implement an application on top of an openEHR infrastructure (back-end) which is capable of dealing with archetypes and operational templates, meaning capable of rendering such EHR data, allowing the user to edit and save data. The implemented solution should be a Patient Synchronized Application (PSA) allowing the application to be part of a synchronization context as required for PSA systems.

1.1 Problems of Current HIS

The introduction of health record standards is improving HIS and therefore the quality of medical health care [1, 2]. Care usually requires more and more clinicians to access shared information about patients which is detailed and complete [3]. So far, there does not exist "the" electronic health record and therefore a system [4] supporting the aforementioned things yet. However, health care systems grew very fast in the last few years and they started to be "solutions which support a continuous medical process" [5].

Proprietary HIS so far mostly deal with their own EHR implementation, leading to the fact that the "EHRs differ from application to application and from country to country" [5]. Taking into account that it is not only the EHR itself which differs but as well the methods involved to access, edit and save data, this becomes more and more a problem. A possible solution for

this is to build systems upon an agreed standard for EHRs, such as openEHR. Such a standard however has to describe the structure, the content of the EHR and the methods of exchanging data, meaning interoperability [5]. All those requirements are met by the use of openEHR.

Problems with the Structure of Electronic Health Records

Different (already existing) health care systems are usually dealing with different structuring of data when it comes to the involved EHR. Especially when used in a productive environment, problems such as a not flexible and limited data structure arise which lead to scalability problems. Systems based on relational database approaches without meta models are bound to such fixed and not flexible data structures and are therefore not scalable.

The structure of data in the back-end usually is closely related to the actual design and implementation of the user interface. Dealing with a fixed data structure means dealing with a fixed set of user interfaces, especially designed for this purpose. So not only the data structure in the background limits the possibilities but the graphical user interface does too.

To overcome these problems, modern HIS usually build upon archetype based solutions (see Section 2.4) to ensure interoperability, another problem in health care systems, discussed in the next section.

Problems with Interoperability of Electronic Health Records

Another big problem in the area of healthcare informatics is the missing interoperability in between standards and applications and therefore the inability to share EHRs across different domains and enterprises. Although several standards such as Health Level 7 (HL7) Clinical Document Architecture (CDA) [6], European Committee for Standardization (CEN) EN 13606 [7] and the openEHR [8] aim to clearly structure medical information, it becomes harder to achieve interoperability between different customized systems, since all of the standards are following their own specifications and not much effort is put into the interoperability part when building systems. [9]

As well as for the EHR itself, there are several definitions for the interoperability of health records. Begoyan [5] defines the interoperability according to the International Organization for Standardization (ISO) (ISO TC 215, ISO/TR 20514, 2005) [10] as

> "the ability of two or more applications being able to communicate in an effective manner without compromising the content of the transmitted EHR"

In their work Schloeffel et al. [11] state that the development and adoption of national and international standards for EHR interoperability is essential for

- sharing patient health information,
- interoperability between organizations within an enterprise, and for the future across national borders,
- and supporting interoperability of software from different vendors.

Standards and recommendations such as the Integrating the Health Care Enterprise (IHE) Cross-Enterprise Document Sharing (XDS) (see Section 6.1) describe ways of how to overcome such issues and build up flexible and inter operable systems which aim to exchange medical information.

The demo system presented in Chapter 3 deals with medical data in the openEHR format and therefore deals with a standard for which solutions to exchange data with other systems exist. Since the CEN EN 13606 is a subset of the full openEHR standard, the developed system clearly has advantages in the interoperability with health care systems present and being developed throughout Europe [11].

The specifications and models of the openEHR standard as described in the architecture [12] are prepared to be functional and semantic inter operable [13]. Whereas functional interoperability means that humans can read transmitted health information, the semantic interoperability is far more important. It states, that actually computers can understand and automatically process health information. It is essential to do automatic processing and therefore is a prerequisite of an intelligent decision support system for care planing. [11]

System Integration Aspects of Modern Hospital Information Systems

In an existing HIS there are many other systems which have to be integrated to enable seamless working for clinicians and high quality health care. Despite the typically used (standard) EHR backend, other internal systems from different departments such as billing systems, administration and Enterprise Resource Planning (ERP) systems, chemical laboratory systems or radiology systems as well as systems for cardiology such as Electrocardiogram (ECG)-, Magnetic Resonance Imaging (MRI)- or Computed Tomography (CT)-systems are usually present and need to be integrated. But there are not only internal systems which need to be integrated. Nationwide and other environmental applications such as insurance systems are also needed within the clinical context. [14]

To explain integration problems, the sample of a Radiology Information System (RIS) and Picture Archiving and Communication System (PACS) is taken which are existing in every digital radiology department. Whereas the RIS is taking care of text-based functionalities such as billing and shift plans, the PACS provides the image based functionalities to actually perform medical image acquisition, storage and distribution. The PACS requires many other components (such as PACS servers, viewing workstations, etc.) to integrate with in order to perform medical imaging and can be a hospital-integrated or an enterprise system. Standards for integrating such systems are needed. [15, 16]

A typical task in PACS environments is the actual image acquisition for several patients and later on the analysis and interpretation of the images on another workstation. Pictures are taken, sent and stored on a PACS server and then evaluated later on. Hence, a recurring process is the selection of a patient and her related pictures and the evaluation of them. A common mistake and problem when performing this task over and over again is the fact, that one can select the wrong patient, evaluate the wrong images and store diagnostic findings in the wrong patient's EHR.

To minimize this problem and to overcome the recurring task of selecting patients one can integrate the existing PACS into the HIS by the use of a Patient Synchronized Application (PSA). By enabling the single systems to communicate changes about the active and selected patient to a central server— such a PSA server then can later on distribute this message to other

active clients— the recurring task of switching the view to the currently selected patient can be eliminated. Especially when doing batch work as usually done in radiology departments (taking pictures of several patients and then later on diagnosing them), such a PSA can maintain the order of patients and support the user in selecting the correct patients.

The introduced system provides the possibility for client applications to take part in a PSA context to get notified about changes from a central server. The solution is generic and so far a prototype implementation for communicating changes in active and selected patients across all active workstations. Section 5.2 deals with the actual implementation aspects and Chapter 6 focuses on the main advantages of such a solution.

The developed backend core system furthermore provides an easy to extend enterprise integration architecture to seamless incorporate different systems present in a HIS context as stated above. By the use of a flexible plugin system and an enterprise service bus technology, such integrated systems can share and access information which is persisted in the backend. The typical problem of how to exchange data in between such systems is thus minimized. To access nationwide information not present in the backend system, additional plugins can be integrated to access this information making it later on available for the other integrated applications. Chapter 4 deals with the backend and its architectural design and outlines the communication ways within the already implemented and existing plugins.

1.2 Dynamic and Archetype-Based Approach with openEHR

The system to develop should overcome the problems outlined before by setting up an architecture and system upon the archetype-based openEHR standard. Since this standard is a superset of the CEN EN 13606 standard as used in Europe [11], the so build system is capable of dealing and inter operating with data coming from third party systems based on either of these standards.

Rather than building a system for a small and fixed set of archetypes, the introduced applications uses a dynamic way of parsing archetypes and rendering information on user interfaces. This means, that the introduced system is very flexible when it comes to the visualization and processing of archetype based data.

1.3 Extending the Integration Architecture

In order to be able to deal with the archetype based approach as presented by openEHR and to overcome problems such as system integration into existing HIS, the developed application makes use of an existing integration architecture which provides back-end services for openEHR data such as simple storage solutions and versioning. One aim was to show that the existing integration solution can be used to build new applications upon and that the communication paths and interfaces to this back-end are defined correctly and working.

1.4 Related Work

Besides a lot of research in the field of archetype based electronic health records and the openEHR standard itself, there are some projects and related work making actually use of the already existing openEHR reference implementations in several programming languages. Many of them deal with the same problems and try to solve them by using standards and dynamic approaches. To name just a few ongoing and active projects the following list briefly describes them.

- **Implementations**

  There are several implementations available from the openEHR standard. Basically they aim the fields of archetype parsing problems and the reference implementation of the standard itself. So far, there are implementations available for programming languages such as Java, .NET, Eiffel, and Python.

- **Modelling Tools**

  Since the modelling task for archetypes is not only complex when it comes to the semantic validity but also to the technical and syntactical validity, there exist several modelling tools for them, again realized in different programming languages. To mention just a few of them, there are tools available from the university of Linköping in Sweden [17] including an archetype editor and an EHR Graphical User Interface (GUI). Another project focusing on the archetype modelling is the .Net Knowledge Tools project by openEHR itself.

- **Projects on openEHR**

  Currently there are a few ongoing projects available. The most relevant one as related work are the Opereffa Project and the GastrOS implementation.

  Opereffa Project

  The openEHR REFerence Framework and Application (Opereffa) [18] is an open source clinical application. It is build on top of Java and the existing Java reference implementation using several other open source projects. It "aims to bring together the openEHR clinical and technical communities on a common platform" [18]. It is not yet a fully implemented tool but rather a proof-of-concept. The existing implementation is a web based approach using before transformed GUI artifacts to display data. Rather than a dynamic approach this tool deals with a fixed GUI which has to be transformed before.

  GastrOS

  Another ongoing open source project is GastrOS [19, 20, 21]. GastrOS is an application used for endoscopic reporting and based on open standards such as openEHR and the Minimal Standard Terminology for Digestive Endoscopy (MST). The tool itself is part of a research project at the University of Auckland to investigate software maintainability and interoperability [19].

  This tool focuses on a GUI which is driven by archetypes and templates as defined in the openEHR standard. The later on presented implemented demo HIS as part of a research project for this thesis based some functions on the existing open source framework from GastrOS - the MST-Structured Data Entry Component (SDE).

Further related work, articles and academic research about this topic can be found on the official home page of openEHR[1].

[1] `http://www.openehr.org/shared-resources/usage`, last accessed: 24.07.2011

1.5 Methodological Approach

Area of this work was to build a prototype application which supports the rendering of openEHR data using the archetypes and templates meta information. To furthermore enrich the front-end application with some more complexity, it was extended by the PSA feature, allowing clients to take part in a context sharing the currently selected patient information among all connected clients within the same context.

The front-end services are based upon an existing integration architecture for openEHR in the background. The introduced system uses the provided services for actually storing and retrieving patient related data in the openEHR format but also introduced several new services and server components.

Developing the prototype thus includes:

- Implementation of service consumers for the provided openEHR back-end services.
- Implementation of a server component which is capable of dealing with the openEHR back-end callbacks.
- Extending this server component with additional features to fetch and cache archetype and template defintions from openEHR.
- Implementation of a small PMI service allowing EHR data to be associated with patient specific information.
- Design and development of corresponding anonymization services to hide EHR identifiers from the actual viewing applications.
- Analyze the features of PSA applications and implement them into the server component to allow for PSA clients to connect and take part in a context.
- Design and implementation of the actual EHR viewing applications which serve as PSA context clients as well.

The following chapters of the thesis first deal with a general problem statement and the basic idea behind the prototype. A special chapter about medical informatics and the use of EHRs follows. The middle part mostly deals with the technical concepts of the back-end and the front-end starting with an architectural overview of the whole system. Later chapters then deal with the special topic of PSA applications and the profile itself and outline further extension possibilities to the introduced systems.

CHAPTER 2

Medical Informatics: the Use of EHRs

When it comes to information systems for medical health care there is currently no such thing like standard software or of-the-shelf software available for health care environments or a Hospital Information System (HIS) which is capable of sufficiently supporting clinicians at their work—existing software is usually sub-optimal and does not fulfill all the requirements of a HIS. Seen as specialized software in the field of medical informatics, a HIS supports clinicians in their work. One important component of such a HIS is a centralized access to an Electronic Health Record (EHR) which might be stored in a central manner or is distributed over several distinct locations. There are many different standards available for EHRs, having different approaches concerning the storage of medical data and the actual representation of it.

This chapter provides an overview and introduction into existing EHRs, discusses the need for them, their definition and how the single standards are dealing with the aforementioned problems of storing information and displaying it later on. Further details of the most commonly used standards are described and the interoperability of them is discussed in the end.

2.1 Why do we need an Electronic Health Record?

Good patient care nowadays requires physicians to have access to all the relevant and related health care information about the patient at the point of care, no matter if this information is

spread and distributed across multiple sites, available in paper form or electronically. But it is not only the physician who has to have access to all the information. As nowadays medicine is becoming more inter discipline, different actors may have access to the EHR—or part of it—of a patient: physiotherapist, people working in care and nurses. All such carers extend and complete the EHR of a patient, thus making the record a full longitudinal chronicle providing more and detailed information about a patient's characteristics. Furthermore, patients nowadays also want to have access to their own personal EHR too, thus allowing them "to play an active role in their health management" [3].

The use of well designed and distributed EHRs thus may enhance the quality of health care within the medical environment. But there are still problems with the introduction of electronic based records. A lot of information is still present in paper form, thus not available throughout the EHR environment [3, 22]. Moreover, even good and modern HIS storing all the information in an electronic format encounter problems (e.g. interoperability issues) when it comes to information exchange with other, probably different, EHR systems. Kalra [3] claims that such systems "limit the ability of users to extract clinical details in a form that can be communicated" and that just very few systems even can receive such extracted information.

It is not only the need for a good and standardized EHR system which becomes more and more important to ensure good health care, but also the need for the ability to share stored information across several different platforms and HIS and thus EHR systems. This requires more work when it comes to the point of system integration and interoperability. Shortliffe explained this as:

> "System integration has emerged as a key element in the reinvention of environments for patient data management and health promotion." [23]

Making relevant information available at the point of care to physicians and patients, even across distributed and probably different EHR systems, sharing and exchanging clinical information without the loss of actual meaning in the context of clinical care makes the need for a standardized, inter operable and exchangeable EHR even bigger, thus helping to ensure the

quality an accuracy of medical care and "preserving the original clinical meaning intended by the author" [7].

2.2 The Definition of an EHR

Electronic health records follow different definitions, have different names and behavior. But a thing they all have in common is that they are the important components behind each single HIS. According to Nordberg [24] "an EHR is considered the core application of modern healthcare and welfare processes". Oliver Bott [4] outlines this as

> "... electronic health record systems (EHRS) are a key component of current and coming health telematic platforms. ... the main objective is clearly to support the treatment of patients by provision of information needed for decisions by health care professionals."

According to Haas et al. [22] an electronic health record stores part or all medical information of a patient on electronic or digital devices and provides ways and methods to access this information so that it can be viewed, edited or navigated. It can be seen as a lifelong record for this very patient which is extended with information every time a patient visits a care institution or provides information to it himself.

Having a more practical view on this topic, one can obtain the following characteristics of EHR systems [13]:

- **Patient-centered**

  The actual EHR present is patient-centered, meaning it consists of entries related to the subject of care and not to any other information about care for example of the executing institution.

- **Longitudinal**

  EHR systems are intended to keep track of all health care actions related to one subject over time, meaning from birth to death of the subject, and not just storing information

about a certain action. This is one of the most important points about EHR systems, since patient history and previous applied actions and events are of great value for future health care actions.

- **Comprehensive**

  Since the EHR is shared across multiple enterprises and accessible from different authorized users, it does not only contain health care events and actions from a specific institution but rather from all health carers. All stored events and actions belonging to one EHR have the same importance.

- **Prospective**

  The build EHR should not be considered as a kind of diary, reflecting all actions and events which took place in the past, but should rather be used as instruction plan for further and upcoming health care events and actions, such as plans, medications or evaluations.

Apart from the actual concept and characteristics of electronic health records there are different names for it. Haas et al. bring up a few different names for it [22]: electronic health records (EHR), electronic patient record (EPR), computerized patient record (CPR), electronic medical record (EMR), computerized medical record (CMR), electronic health care record (EHCR), and continuous electronic care record (CECR).

This thesis uses to the nomenclature of an Electronic Health Record (EHR) and will use this name as a synonym for all the others.

In contrast to the mentioned EHR, a Personal Health Record (PHR) is a health record where data is entered and maintained by the individual and not by a professional within a health care providers facility. The aims of EHR and PHR systems are different: whereas EHR systems tend to provide full information about the patient's treatments etc. within medical institutions, the latter stores information which comes from the patient itself, meaning, which was entered by the user and not by a health care professional. Such data may include data from external devices like electronic weighing scales or pulse watches which a user collects out of personal interest. Despite the data entered in the health record, another major difference is the ownership of the data. A PHR is, in contrast to the EHR, completely controlled by the individual.

Since the built system is being used by health care professionals within a health care institutions, the thesis deals with EHRs.

Problems Arising with Lifelong Patient Records

Conventional case records usually contain all relevant information for the current treatment and should provide the ability to reproduce the current state of treatment every time. Core functionality thus is to provide ways to document which medical treatments have taken place at which time by which physician, why this treatment was performed and which result was the actual outcome of the treatment. Only focusing on the current case, such records do not keep track of before performed treatments and are usually not available the next time the patient is being examined. [22]

In contrast to such case records, the lifelong EHR keeps track of all performed actions and treatments for a patient. Hence, all information concerning one patient is available when opening his EHR. However, having stored all the information at one place relying on a certain EHR standard, the system is dependent on its single components to actually be able to display and edit information within a patient record and it is dependent on the underlying EHR standard.

Implications are, that data probably becomes unavailable and not accessible or readable anymore, when underlying EHR standards or the systems change or are completely discontinued. This means that not only data for just one case is not accessible anymore, but rather all patient related data, thus the whole lifelong patient record.

The online health record Google Health[1] for example started up with the intention to make health care better by providing better information. It allows the user to "organize, track, monitor, and act" (Google Health, 2011) on health information. Information then is stored on central servers making it possible to access it from anywhere and anytime and to share information with other people. However, the services are now to be discontinued. People using Google Health are now facing the stated problems: the application is discontinued and the data is not accessible anymore. Offering ways to export and download data Google makes data still available, but data cannot be easily imported and further used by other existing systems.

[1] `http://www.google.com/health`, last accessed 15.08.2011

Interoperability standards and definitions thus are important to allow other systems to process and interpret data coming from different vendors without loss of meaning and structure.

There are many different definitions, characteristics and even names for an EHR. Gök [25] outlines, that depending on the point of view, these terms define the aspects of an EHR. However, it is hard to give a clear and formal definition of such standards since one cannot encapsulate all the varieties of the EHR into a single definition [26]. There are a few definitions which actually show more "similarities than differences with respect to the purpose, functions and goals of electronic records" [26].

The ISO/TR 20514:2005 defines the EHR as follows:

> "... a repository of information regarding the health of a subject of care in computer processable form, stored and transmitted securely, and accessible by multiple authorized users. It has a commonly agreed logical information model which is independent of EHR systems. Its primary purpose is the support of continuing, efficient and quality integrated health care and it contains information which is retrospective, concurrent and prospective." [10]

In contrast to the definition given by the ISO, the European standard CEN 13606 EN gives the following definition of an EHR:

> "This European Standard considers the EHR to be the persistent longitudinal and potentially multi-enterprise or multi-national record of health and care provision relating to a single subject of care (the patient), created and stored in one or more physical systems in order to inform the subject's future health care and to provide a medico-legal record of care that has been provided." [7]

Although the definition itself is not the same, the intended meaning is. Both definitions mention a record which is related to one subject (the patient) and which can be accessed by different authorized users from different enterprises (meaning, it is independent from the actual EHR implementation and system). Furthermore, both of them take into account the time line of

such an electronic record, that consists of already provided health care actions, the current and the future actions, making the EHR a longitudinal record.

2.3 Advantages and Drawbacks of Electronic Health Records

Besides the actual definition and characteristics of EHR systems there are many ongoing discussion about the actual benefits and drawbacks of electronic based record systems. But how do EHR systems improve the quality of care and what are the actual benefits?

The typical benefit of an EHR pointed out first is the possibility to access the data anywhere and anytime without being bound to a certain place. This is obviously one of the major drawbacks of the paper based records. Not that it can get easily lost (whereas this is unlikely the case for the electronic records), but they are not shareable and one cannot easily search for them nor sort or filter entries. Moreover, considering the fact that those paper based records are not easily shareable, one has to deal with several different records, actually belonging to the same subject, thus it is hard for physicians to have a good overview of all the actions and events of a subject. [27, 28, 29]

The possibility to access and share EHR data anywhere and anytime however has certain prerequisites. A given and well defined HIS IT infrastructure is important in order to allow sharing and accessing of medical data. One has to take into account that the amount of—especially medical imaging—data over several years increases drastically. Scalable and extendable storage solutions have to be defined in order to face the enormous amount of data. Moreover, to ensure the persistence of electronic data and its consistency, backup and archiving systems have to be present. Archiving systems and mechanisms need to be capable of storing and accessing medical information over time, meaning, not that the information can be read some time in the future, but also keeps its semantic and structural interoperability.

Menachemi et al. [28] state, that advantages include the "improvement in the quality of medical care, a reduction in medical errors, and other improvements in patient-level measures". Especially the point of improving the quality of care and enhancing patient safety is of course considered as one of the major benefits of EHR systems.

Drawbacks and disadvantages for EHR systems are seen in the potential financial issues, the changes in workflows in the clinical environment thus reducing productivity of physicians, inflexibility, security and safety issues and increasing costs in the medical infrastructure itself.

As financial issues, one can consider the costs of analyzing, implementing and maintaining HIS and EHR systems and the probably ongoing costs for adoption of existing systems to new ones. The introduction of such systems may change the workflows in clinical environments, physicians and health care personnel is used to. Not does it only affect the productivity itself but rather does it increase costs for medical health care. [28]

Although the introduction of EHR systems may increase the costs related to medical health care, some others argue that shared care is less expensive than specialists-only care [13, 30]. They argue that information technology has the potential to reduce costs [31] and improves the quality of care [32]. On the other side, failures in the IT infrastructure and problems with the maintenance of the systems can reduce the quality of care, since patient records might not be available due to missing or faulty infrastructure. [13]

Whereas most of the disadvantages can be considered as being of financial nature, some are more enhanced privacy issues. Those are the major drawbacks of EHRs: patient privacy violations. Since more and more health care information is shared and exchanged throughout different enterprises and different health care providers [33, 34], the potential issues with patient privacy are increasing.

2.4 Existing EHR Solutions and Standards

Nationally as well as globally health care providers are dealing with integrated health care environments and therefore are trying to establish health care standards for EHR based systems. EHR systems have some clear benefits, namely enhanced quality in patient care and enhanced efficiency. To assure that EHR systems meet such characteristics and requirements, institutions and organizations are working on standards.

The following sections deal with the major standards and high level definitions of EHRs and should provide a rough overview of existing standard solutions. Later on, the closer relationship in between those standards is described.

HL7 - Health Level 7

The standard HL7 is being developed by Health Level Seven International, a non-profit organization founded in 1987 in the USA, and is "dedicated to providing a comprehensive framework and related standards for the exchange, integration, sharing, and retrieval of electronic health information that supports clinical practice and the management, delivery and evaluation of health services" [35]. Earlier focusing on interface requirements for messaging systems between large healthcare enterprises, HL7 now releases standards for actual health records as well [3].

The vision statement of this organization is to develop the best and most widely used standard in the sector of medical health care. They focus on standards to ensure and enhance interoperability, workflows in medical health care to improve the actual care delivery and to raise the quality of knowledge transfer between the single stakeholders in a medical environment (such as health care providers, government, vendors, etc.). [35]

Usually HL7 is known as an organization developing messaging standards [36] such as the HL7 Version 2 and HL7 Version 3 standards, but moreover, they are developing messaging standards such as the Clinical Document Architecture (CDA). The following short sections provide an overview about these standards.

HL7 Version 2

Version two of the messaging standards produced by HL7 and formally known as "Application Protocol for Electronic Data Exchange in Healthcare Environments" [37] is now the "most widely used protocol" [5, 37] for exchanging messages between different health care providers and medical information systems. Usually, this version of the standard is known as Version 2.x, since there have been several releases so far, indicating that the standard itself is a growing and refining one.

Since version two of the standard was not developed systematically, it is lacking of consistency but on the other hand allows a big flexibility within implementing applications. Another big problem arises since it is not based on any underlying reference model [11] thus not supporting interoperability between different health care systems from different vendors in a good way [5]. The missing of a precise model in the background led to inconsistent implementations of the standard thus forcing applications using the standard to rely on additional agreements in order to be able to ensure inter operable applications [5].

HL7 Version 3 and HL7 v3 RIM

Some of those problems and many more new features were solved and introduced in the new version of the standard, which was started to be developed in 1997. Rather than just keeping the old standard and remodeling it, they began to apply object-oriented modeling to the standard [38] which led to the Version 3 Message Development Framework. More than the previous standard, version three is an interoperability specification defining communications produced and received by computer systems [37].

It introduces a Reference Information Model (RIM), which is one of the key features of version three, to enhance the interoperability and get rid of the problems existing in version two. However, on its own it is not a full specification for an EHR system [11]. "HL7's RIM is a comprehensive, nondiscipline specific, object-oriented information model of patient care and of the providers, institutions, and activities involved" [39]. According to HL7 the definition for the ANSI approved standard, RIM is "a large, pictorial representation of the HL7 clinical data (domains) and identifies the life cycle that a message or groups of related messages will carry".

The RIM itself as the object-oriented core of the standard represents several classes and attributes, which are used by the messages defined in the standard [11, 39, 40]. Since the model is defining all classes and their attributes, the RIM is considered to be a one model approach. Problems arising with that kind of methodology are that for extensions to the model itself, classes and attributes sometimes have to be renamed or moved [2]. Considering this, the applications developed according to this standard are hard to maintain, since changes in the standard have to be applied to the actual applications to keep interoperability and consistency high.

As a standard, RIM introduces several core classes. Those classes are then to be used by the applications and can be mapped to objects in the health domain. The following list provides a short overview on the existing classes [2]:

- **Entities**
  are the actual physical information objects such as an organization, the subject of care, materials used for actions and events on the subject.
- **Roles**
  are assigned to entities and thus providing them with more information and granting them actions. Roles can be: patient, provider, physician, etc.
- **Participations**
  are entities in specific roles and acts, such as a performer or an author.
- **Acts**
  are the actual actions/events such as observations or medications.
- **Role Links**
  are keeping the relationships between roles and the linked entities.
- **Act Relationships**
  allow the chaining of acts, thus modeling flows.

Since the two standards (HL7 Version 3 and HL7 v3 RIM) are lacking good documentation and are often systematically ambiguous, Smith et al. [40] outline them as "incoherent standard". They are arguing that there has to be a "coherent, clear and implementable" uniform information representation. Thus, together with the problems of renaming and moving classes and attributes, the standard is not as successful as version two [2, 40].

HL7 Clinical Document Architecture (CDA)

In contrast to the two messaging standards of HL7 the CDA focuses a different part in medical health care. The named RIM can be seen as the "proposal for the Clinical Document Architec-

ture" [5]. On a very high level CDA provides medical documents with structure and semantics and defines how such documents are to be exchanged by the use of the data types and classes defined in the RIM. It is the current strategy of HL7 to ensure EHR interoperability and is mainly defined in a singe Extensible Markup Language (XML) schema [11].

"The HL7 Clinical Document Architecture (CDA) is a generic message structure for the communication of a clinical document, derived as a message model from the HL7 RIM" [3]. It is "a document markup standard that specifies the structure and semantics of clinical documents" [36]. There are so far two releases of this standard. Whereas release one basically defines the structure and semantics of clinical documents, version two of it subsequently adds fine-grained information to the document itself and introduces some concepts for structuring the document body and its elements [41]. Version two thus is now closer to other standards and their hierarchies such as the CEN 13606, which is discussed later on, thus allowing a better cross-mapping of information [3].

Defined as standard of HL7 it gets its content from the HL7 RIM and integrates perfectly within the applications and architectures build upon HL7 technologies [41].

The basic structure of a CDA document is as follows [42]:

- Consisting of a document header and a body, the CDA document is defined in its very basic form.
- The header contains information about the identity of the CDA document, includes information about authentication, the actual subject of care (the patient), the involved providers and some more information.
- The body itself contains the actual report wrapped up in sections.

As defined above, the actual resulting documents are represented in XML data and thus easy to understand, evaluate and process. Blobel et al. state that "CDA documents are human readable, machine processable, persistent, legally binding, and valid" [2].

The standard furthermore defines several levels at which a CDA document can be defined [2, 4, 5]:

- **Level 1**
  is the simplest level and only requires a valid document header and a body included into the document. The body itself contains clinical data.
- **Level 2**
  defines observations and instructions in the header and constrains the structure and content. It therefore increases the interoperability.
- **Level 3**
  provides completely structured entries and full compliance to the RIM. It provides medical concepts and can be validated against the XML schema.

The following two figures (Figure 2.1 and Figure 2.2) show the very basic structure of a CDA document and how entries can look like. As stated before, Figure 2.1 shows the major components of a CDA document: the header and the actual structured body. Figure 2.2 shows a very small example of a simple observation, which is the actual content of a CDA document. Encoded as a `section`, it defines basic attributes such as a title and a basic description text, a reference to a code system and the code used and the actual entry.

The current release is a stable platform which can be used to exchange clinical information and documents across different health care providers and vendors. The different levels of compliance enable the implementers to choose the 'amount of implementing' the standard. For simply exchanging documents without any further XML structure one can choose the simplest level one. Furthermore, the introduction of several levels allow the vendors to just implement a specific level and later on adding more detailed information and probably upgrading to the next level without any big problems, since the basic infrastructure for sending and receiving documents is defined in the header and does need to be changed.

```
<ClinicalDocument>
  ... CDA Header ...
  <structuredBody>
    <section>
      <text>(a.k.a. "narrative block")</text>
      <observation>...</observation>
      <substanceAdministration>
        <supply>...</supply>
      </substanceAdministration>
      <observation>
        <externalObservation>...
        </externalObservation>
      </observation>
    </section>
    <section>
        <section>...</section>
    </section>
  </structuredBody>
</ClinicalDocument>
```

Figure 2.1: CDA Major Components (taken from Figure 1, [42])

```
<section>
  <code code="8716-3" codeSystem="2.16.840.1.113883.6.1"
   codeSystemName="LOINC"/>
  <title>Vital Signs</title>
  <text>Temperature is 36.9 C</text>
  <entry>
    <observation classCode="OBS" moodCode="EVN">
      <code code="386725007" codeSystem="2.16.840.1.113883.6.96"
       codeSystemName="SNOMED CT" displayName="Body temperature"/>
      <statusCode code="completed"/>
      <effectiveTime value="200004071430"/>
      <value xsi:type="PQ" value="36.9" unit="Cel"/>
    </observation>
  </entry>
</section>
```

Figure 2.2: CDA simple observation example (taken from Figure 4, [42])

IHE Profiles

Founded in 1998, the Integrating the Health Care Enterprise (IHE) is an initiative of healthcare professionals and an industry sponsored organization which aims the improvement of healthcare standards in order to enhance interoperability of computer systems [43], and to improve the quality, efficiency and safety of clinical care [44]. IHE profiles try to fill the gaps between the actual implementation of a standard and its formal definition. The reason for that is, that standards usually are highly abstract definitions which are actually too complex and too much of information engineers would need to build a system.

By bringing together clinical and technical experts in the field of medical health care, IHE profiles define critical use cases for information sharing and tend to optimize already established standards in specifications, industry implementers later on can follow. Such profiles are then summarized and collected and the so called IHE technical framework is build out of it, which functions as catalog of how to solve integration problems with health care standards [45].

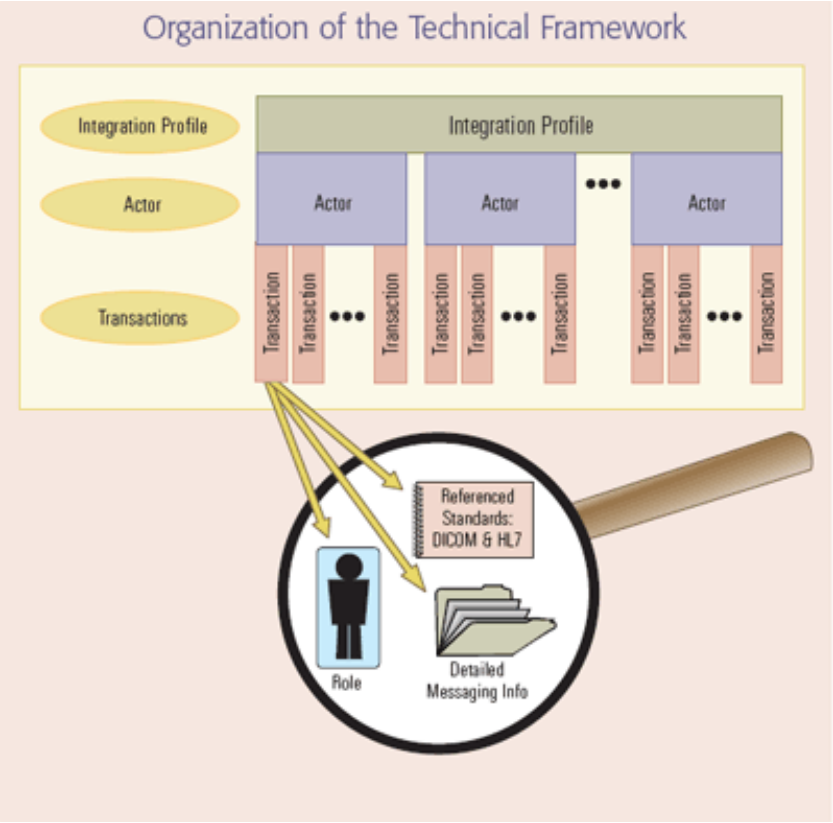

Figure 2.3: IHE Profiles: Technical Framework (taken from [44] - About)

The profiles are always defined upon the following basic design principles and conventions [44, 46] (compare Figure 2.3):

- they describe the solution to specific integration problems,
- make use of a specific actor and document the system roles,
- are related to standards and design details for implementers.

Actors (such as the Department System Scheduler) are in charge of producing, managing and acting on information in the context of profiles and are assigned specific requirements within each profile. Actors might be reused within several profiles. Transactions specified within such profiles should always complete specific tasks and are usually related to a single standard. They can be used within multiple profiles and on multiple actors.

CEN EN-13606

In its first version published in 1999-2000, the ENV 13606 was a four-part pre-standard. Since by then it was hard to implement the given standard in actual health care applications, CEN made a decision to revise the pre-standard and update it to a full standard by the end of 2006. By then, the now final standard EN 13606 consisting of five parts was released. But rather than a full standard for EHR systems the CEN 13606 is a specification for EHR Extracts only [7, 11].

The goal as defined by this European standard is "to define a rigorous and stable information architecture for communicating part or all of the Electronic Health Record (EHR) of a single subject of care (patient)" [7]. It is a standard build to:

- support the interoperability of systems and components communicating EHR data,
- to preserve the original meaning of data while transferring as intended by the author,
- and to reflect the confidentiality of data.

The European standard may be a useful contribution for a full EHR system design, but on its own is not intended to specify architecture nor database designs for such systems. It is only "used to define a message, an XML document or schema, or an object interface" [7].

Parts of the Standard

As described before, the multipart standard consists of the following five parts [3, 5, 2]:

- **Part 1: Reference Model**

  This is the actual scalable generic information model used to represent and communicate virtually any EHR information of any patient. It defines the generic building blocks used to later on define the characteristics of health records.

- **Part 2: Archetypes Interchange Specification**

  This is again a generic information model and language "for representing and communicating the definition of individual instances of archetypes" [2]. Archetypes define or constrain the legal combinations of the reference model objects.

- **Part 3: Reference Archetypes and Term Lists**

  This is a basic set of standard archetypes and terms which are typically used in the health care environment. Furthermore, it defines data objects for describing rules for the distribution or the sharing of EHRs. It can be seen as illustration how one can define clinical content.

- **Part 4: Security**

  Defines concepts that need to be implemented in the individual system in order to allow a suitable interaction with security components to introduce data safety and security in the context of data exchange.

- **Part 5: Messages for exchange / Exchange Models**

  Forming the basis for the message-based communication, this part is still under development.

EHR Extract Record Hierarchy

The standard tries to reflect the hierarchical structure and organization of files and medical records in the original clinical context to ensure that the meaning is preserved when records

are transferred, since the information in a health record itself is inherently hierarchical [7]. Entries such as observations, medications and intentions usually have a simple or complex inner structure, but are organized within headings in documents, which are then filed in folders, related to a patient. A patient can have more folders within a health care environment.

All those parts can be again found in the definition of the standard in its single parts. The most important (and biggest components) of such an EHR Extract are described in the following [7]:

- **EHR_EXTRACT**

  This is the top-level container related to a patient and serves as communication object between health care providers

- **FOLDER**

  Used for organizing elements within an extract, e.g. 'Diabetes care', 'Hospital', etc.

- **COMPOSITION**

  A composition is the actual information committed to the EHR by one agent as a result of a clinical encounter.

- **SECTION**

  This is basically the data within the composition, belonging to a clinical heading, reflecting the flow of information gathering during the encounter. Typical entries are: 'Family history', 'Symptoms', 'Objective findings', etc.

- **ENTRY**

  Is the actual recorded information in the EHR at a clinical action, an observation, etc.

- **CLUSTER**

  Organizing nested data elements into groups and structures, such as time series and data tables.

- **ELEMENT**

  The most outer leaf node in the hierarchy containing a single data value, e.g. the systolic blood pressure.

The above described parts of the standard can be seen in Figure 2.4 and Figure 2.5, making the actual relation between all those components more clear.

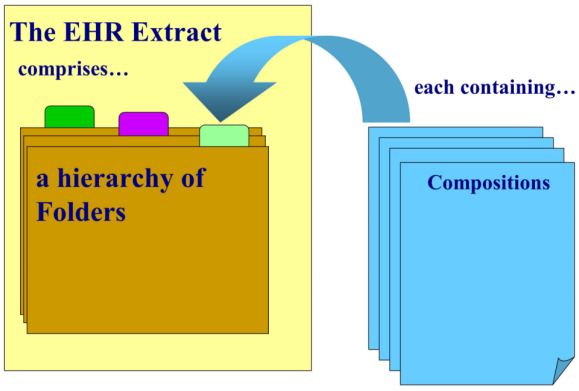

Figure 2.4: EHR Extracht hierarchy (part 1) (taken from [7], Figure 1)

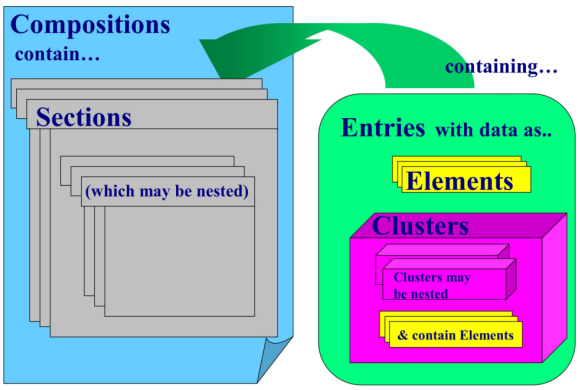

Figure 2.5: EHR Extracht hierarchy (part 2) (taken from [7], Figure 2)

openEHR

Being developed by the openEHR Foundation, an independent and non-profit organization and community founded in the year 2000 by the company OceanInformatics[2] and the University College London [8], openEHR evolves as standard for EHRs and related systems and is becoming the most complete and validated EHR architecture worldwide [3]. As a foundation its aims

[2] http://www.oceaninformatics.com/, last accessed 24.07.2011

are "the development of an open, interoperable health computing platform, of which a major component is clinically effective and interoperable electronic health care records (EHRs)" [8]. For its development, people from the foundation work closely together with developers from other standards such as HL7 or the CEN 13606. This first of all brings a higher interoperability and secondly, since more experience is shared, a better standard itself. (More on interoperability of standards and their relationships in Section 2.5)

According to the head developers of the standard, openEHR is "an open, detailed, and tested specification for a comprehensive interoperable health information computing platform for the EHR and other major services such as terminology" [11]. People working on this standard have experience in health informatics for more than 15 years and try to design it in a way, that it fulfills as many requirements typically present in a medical environment as possible. So far, openEHR and its specification support requirements such as [8]:

- recording **clinical information**, which is actually the most important requirement, workflow-based instructions, imaging data, diagnoses, and many more,
- **archetype- and template-enabling** of all clinical systems, allowing professionals to actually define and model clinical content, the semantics and user-interfaces,
- supporting **terminology** systems to integrate, such as Systematized Nomenclature of Medicine - Clinical Terms (SNOMED-CT),
- allowing the systems to be able to communicate via **messaging** systems, such as HL7 v2 or Electronic Data Interchange For Administration, Commerce and Transport (EDIFACT),
- making it easier to integrate with an existing **Hospital Information System (HIS)**,
- providing an Application Programming Interface (API),
- and allowing of **distributed versioning** of EHR data.

To achieve all those outlined requirements and to be able to build scalable and maintainable EHR systems, openEHR follows a so called 'two level methodology'. Whereas 'single-level

methodologies' build in the actual information and the knowledge concepts into one level of object and data models, the two level approach separates them to ensure that highly complex scenarios and large number of concepts still can be maintained. In two-level approaches, systems are built only from the information model, providing the basic infrastructure to deal with knowledge, and are driven by the knowledge concepts (or 'archetypes') which are authored directly by the domain specialists. [47]

This is the key concept openEHR is following. To better understand the upcoming sections, three major definitions and explanations are to be made: the defintion of **information**, **knowledge** and **archetypes**. Thomas Beale, Chief Technology Officer[3], in his work about archetypes and constraint-based domain models give definitions for those terms as follows [47]:

- **Information**

  Are statements about specific entities, e.g. "Gina Smith (2y) has an atrial septal defect, 1cm x 3.5cm". This is actual information about a subject (a patient).

- **Knowledge**

  Are statements which apply to entities of a certain class, e.g. "the atrial septum divides the right and left atrial chambers of the human heart". This is information which comes from a medical knowledge-base, but does not apply to a specific subject in general.

- **Archetype**

  This term is introduced to denote "a model defining some domain concept, expressed using constraints on instance structures of an underlying reference model". In other words, archetypes define valid information structures and enable users to express concepts in a formal way. Moreover, they enable systems to validate user input against and ensure interoperability.

The afore described tow-level approach can be seen in Figure 2.6. Knowledge is separated from actual information. Knowledge and terminology or ontology information is used by domain experts to build archetypes from, resulting in an archetype library. Such archetypes define

[3]taken from `http://www.oceaninformatics.com/`, last accessed 24.07.2011

which information is collected and how this information is validated. The actual information is built upon such archetypes. The implementation then has to be done in background, to first implement a reference model, which is used to express information, and second to develop an archetype model language in order to be able to express and define archetypes. Implementation therefore limits to those parts and has not to be redone all the time, whenever concepts or archetypes change. This is the main factor for better maintainability of such systems compared to one-level approach systems.

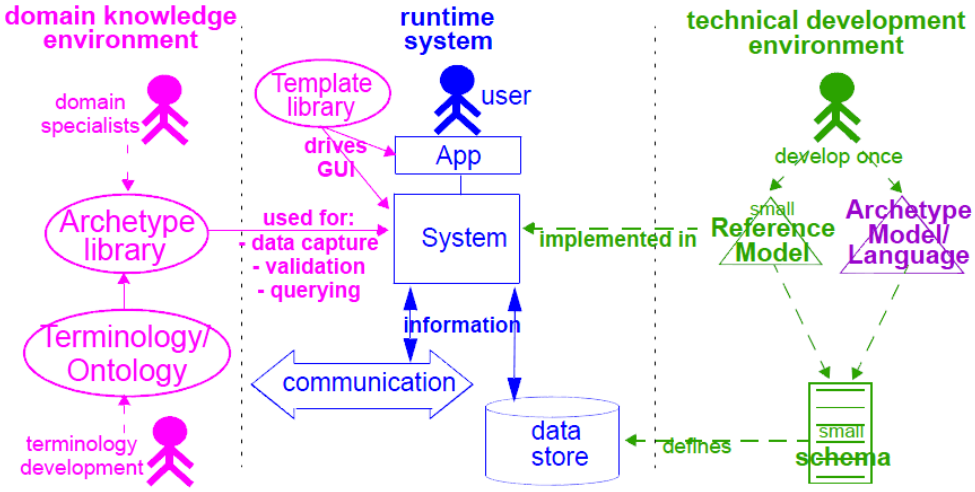

Figure 2.6: openEHR's two-level methodology (taken from [12], Figure 6)

The following sections describe the basic architecture of openEHR and the single specifications. Starting with a basic overview, then the two major parts of the standard are discussed: the reference model and the archetype model. Later on, features such as versioning and historization are outlined.

Architecture Overview

The single architecture specification documents are built upon the specification project, which was first of all realized to get a clear definition of the single parts, the standard is going to describe. Figure 2.7 shows the relationship of all the single parts of the standard. The core of the specification is build by the Reference Model (RM). The Archetype Model (AM) and the

Service Model (SM) are then extensions of the RM and make use of the introduced classes there. The inner two parts (RM and AM) are defined corresponding to the ISO RM/ODP information and computational viewpoints. [12]

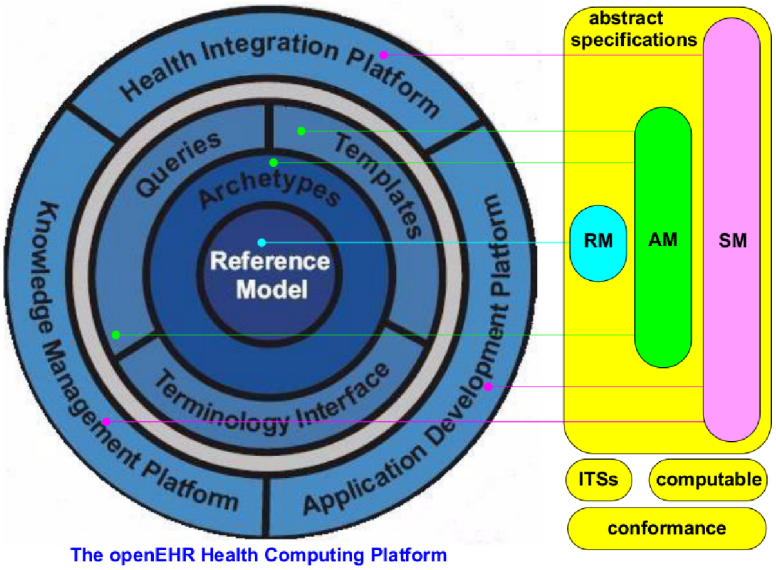

Figure 2.7: openEHR specification project (Figure taken from [12], Figure 1)

Having this abstract layer model in mind, one can build a model for the software engineering picture, meaning a figure showing the actual relationships between all those single layers and models. Figure 2.8 shows this relationships and draws a border between the actual information side and the knowledge side (two-level approach). Via the RM one can build actual instances for information. Furthermore, the AM makes use of it to get the semantics for constraints out of the RM. Via the AM one can build instances of archetypes. Such archetypes define and constrain at runtime instances of the 'information' side.

The following few sections now deal with the single models in more detail.

Reference Model

This openEHR specification is the very base for all other models and specifications and basically describes the general structure of EHRs. The packages introduced in this model are generic

and are then used by the models in outer packages. According to Beale et al. "they provide identification, access to knowledge resources, data types and structures, versioning semantics, and support for archetyping" [12].

The package consists of the following information models [12]:

- **Support**: is used for definitions, terminology relevant data, measurements and identification services and describes the most basic concepts required by all the other packages. [48]
- **Data Types**: the set of available and clear defined data types in an openEHR model, such as text, quantities, dates and times, basic data types such as booleans and state variables, etc. [49]
- **Data Structures**: provides generic data structures for the existing data types, such as lists, tables, trees, etc. [50]
- **Common**: defines classes which are used to link between the RM and AM, such as the `LOCATABLE` and `ARCHETYPED` classes. [51]
- **Security**: very basic semantics for access controls and privacy settings.

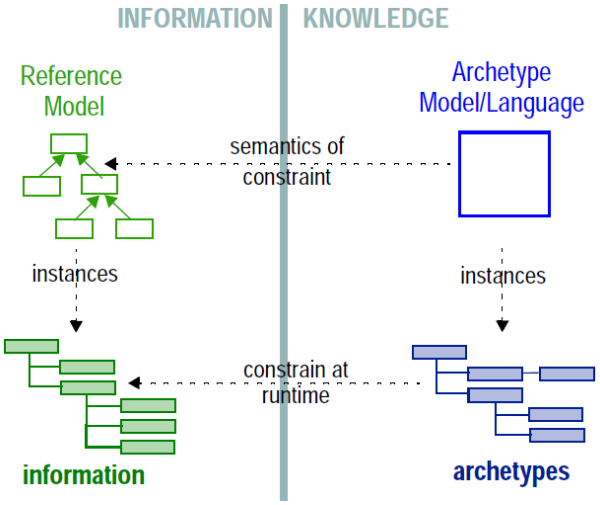

Figure 2.8: Software Engineering picture for the openEHR model layers (taken from [47], Figure 6)

- **EHR**: contains the actual definition of EHR data and related classes. More on this in Section 2.4.[52]
- **EHR Extract**: defines how an extract can be build from a given EHR.
- **Integration**: defines classes to represent free-form legacy or external data as a tree. [53]
- **Demographics**: defines generic concepts for demographic related data, such as a party, a role or other related details such as contact addresses.
- **Workflow**: not fully supported by the standard yet, this model will provide elements to describe the semantics of processes in clinical care.

Archetype Model

As one of the center pieces of the openEHR standard, this package contains models which are important to describe the semantics of archetypes and templates, whereas each archetype describes configurations of data instances whose classes are described in a reference model. (Templates are sets of archetypes which furthermore allow constraints across archetypes. Meaning, a template can define constraints in addition to the ones already defined by archetypes. e.g. a template consisting of archetype A and B, whereas both of them already define constraints on their own values, the template then can constraint things like 'if value X has been chosen in A, then value Y is not allowed on a certain field in B'). The package includes the language definition used to define archetypes, the Archetype Description Language (ADL), and the `archetype` and `template` packages. [12, 54, 55, 56]

The openEHR EHR Design

Considering all the basic infrastructure defined by the RM and AM, the actual EHR as defined in openEHR is build up very simple. Identified by its unique EHR ID, a record specifies references to a set of structured and versioned information, which together form the actual information present in an EHR. Figure 2.9 illustrates this.

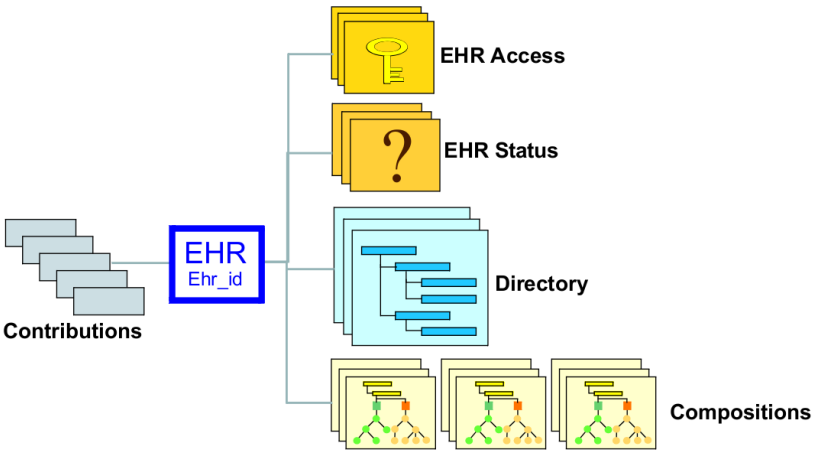

Figure 2.9: High-level Structure of the openEHR EHR (taken from [12], Figure 14)

As shown in Figure 2.9, belonging to the given record, the EHR Access components define a set of objects containing access control information and settings for the record. The reference to the EHR Status object contains status information about the record and optionally the identifier of the subject (the patient) currently associated with the record. The references to the Directory elements are for defining a logical hierarchical structure of compositions using folders. The containers of all the clinical and administrative content, the Compositions, are related to the EHR as well. The set of Contribution objects is used for every single change made to the record, and stores version information about the changed items (it can be seen as the history of the EHR element). [12, 52]

The actual content or information about a certain treatment is stored and versioned in compositions. Figure 2.10 shows the logical structure of a composition. A composition itself stores several sections (blue), each of them can itself consist out of several others, or can directly contain other entries (cyan) such as observations and administration entries. Each entry itself stores information about its history (orange) and the corresponding events and holds an item list (gray). Item lists use clusters (green) to organize their data elements (light green). An element itself then refers to one of the data types defined in the openEHR data types information model (e.g. a `DV_TEXT` or `DV_DATE`).

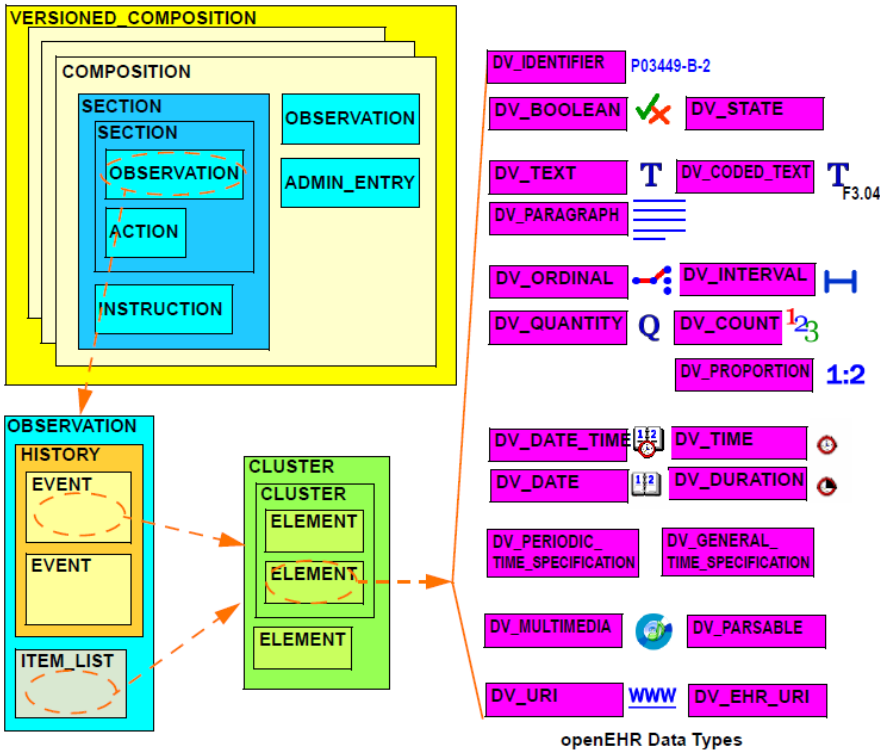

Figure 2.10: Composition in openEHR and its elements (taken from [12], Figure 15)

Versioning

The content of EHRs is versioned throughout the whole life-time of a record. Since this is one of the key requirements of the openEHR project, it is an integral part of the openEHR architecture. Figure 2.11 illustrates this concept. openEHR defines `VERSIONED_OBJECT` classes, which serve as version containers. Within them, single elements of the class `VERSION` are held, each of them representing a single version of the actual content. Each such single version then contains the actual versioned information, such as a composition. [12]

Having explained the most widespread and common standards for health record systems, the next section focuses on bringing them into a relationship. It shortly discusses the common concepts all of them have and argues on their interoperability.

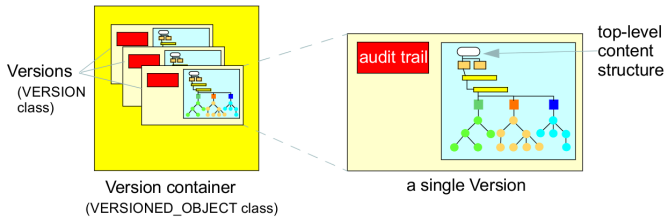

Figure 2.11: Version-control structures (taken from [12], Figure 23)

2.5 The Relationship of Standards

As discussed in the previous sections, the single standards are somewhat related to each other, since experts exchange knowledge and try to build interoperable standards. Bringing them closer together allows easier integration and much better interoperability. In fact, since the development of openEHR started, many of the related standards got influenced by the new approach and on the other side, openEHR took some of the best practice parts from other standards and integrated them.

Figure 2.12 shows a schematic drawing of how the single standards are related. openEHR can be seen as a superset for the CEN 13606 standard which is then a superset of the HL7 CDAv2 standard. Especially the archetype methodology of openEHR influenced the single standards on a large account. The HL7 v3 RIM as already described before serves as base for the HL7 CDA standard and also influenced the development of the CEN 13606 standard.

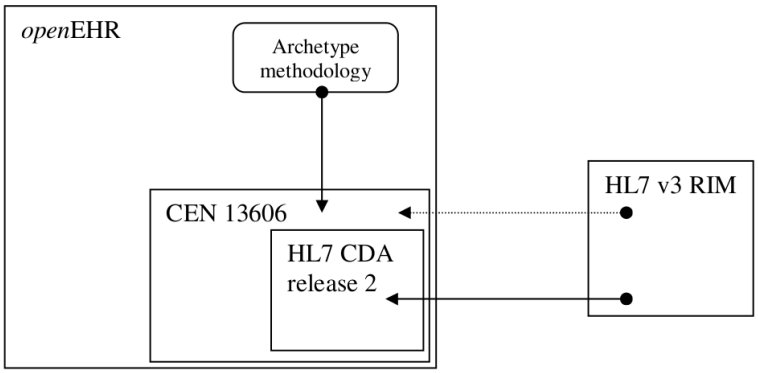

Figure 2.12: Interoperability of standards: schematic relationship between openEHR, CEN 13606 and HL7 CDA (taken from [11], Figure 1)

The later on presented work is based on the openEHR standard, since this standard can be considered as superset for all existing standards and is furthermore the only one specifying a full EHR system. Additionally, since it provides an archetype based approach, such a developed system is much more flexible and maintainable. As a benefit integrations into other systems or interoperability problems are easier to solve.

Middle

CHAPTER 3

HIS: Architecture Overview

Whereas the previous chapters in this work dealt with the theoretical background of EHRs and HIS applications, the following chapters present the actual realized prototype of a HIS using openEHR. This chapter gives a rough overview on the overall architecture of the complete system. Chapter 4 then deals with the back-end components, and Chapter 5 with the implemented front-end services and client views. Chapter 7 focuses on some possible extensions to the existing solution and discusses them.

3.1 Description

As part of a research project, a medical health care application for a HIS based on the openEHR standard has been developed. It extended an existing integration architecture for openEHR and enriched it with additional functionality. With focus on a centralized stored and maintained EHR in the back-end, the developed patient synchronized front-end realizes a dynamic approach for visualizing and editing archetype based EHR data coming from the back-end.

The implementation focused the design of a component capable of dynamically rendering archetype based health care records like defined in the openEHR standard. In addition to the existing integration architecture some more services were introduced to the prototype: for example anonymization components for medical data, system independent interfaces for PSAs and

a small PMI. The developed graphical user interface for maintaining medical data implements most of the defined and necessary input elements for the openEHR standard and supports the visualization of so called Operational Templates (OPT). The service oriented architecture is furthermore capable of maintaining and processing PSA context data within a HIS in a system independent manner and assists the integration of web based, Intra- and Internet based solutions for medical health care applications.

3.2 Goals of the Implementation

Goals were to implement a prototype for a HIS which allows easy integration of components and to build a PSA on top supporting archetype based EHR data visualization. As the implementation of an openEHR based system means implementing a superset of the European CEN 13606 standard, one big goal which was met was to ensure interoperability of components to existing HIS applications within Europe.

Another goal, which was immediately pursuited after closer analyzing the infrastructure and architecture, was to show that the already implemented back-end services are fully functional and supporting different platforms to receive calls from. Since the back-end is implemented in Java technology using a JBoss[1] server, an important fact was to show that calls from a .NET specific implementation can actually reach the back-end and use its services.

Considering the front-end service implementation, goals were to build and provide a system independent PSA context adapter which allows clients from different platforms and technologies to connect and use the PSA services. In this setup the existing prototype supports PSA clients running on web technologies communicating via HTTP (web-services), clients running on TCP/IP protocols or clients using the .NET specific Windows Communication Foundation (WCF) technology.

[1] `http://www.jboss.org/`, last accessed 24.07.2011

3.3 Architecture Overview

The evolved and designed allover architecture of the HIS prototype is rather complex. This has several reasons. First of all, the introduced back-end is designed to be an integration architecture, allowing other applications to be easily integrated, thus making use of several integration architecture paradigms. Upon the back-end services another service layer was added to support the aforementioned goals of a PSA and to provide the necessary callback interfaces for the back-end. Figure 3.1 shows this rather complex architecture.

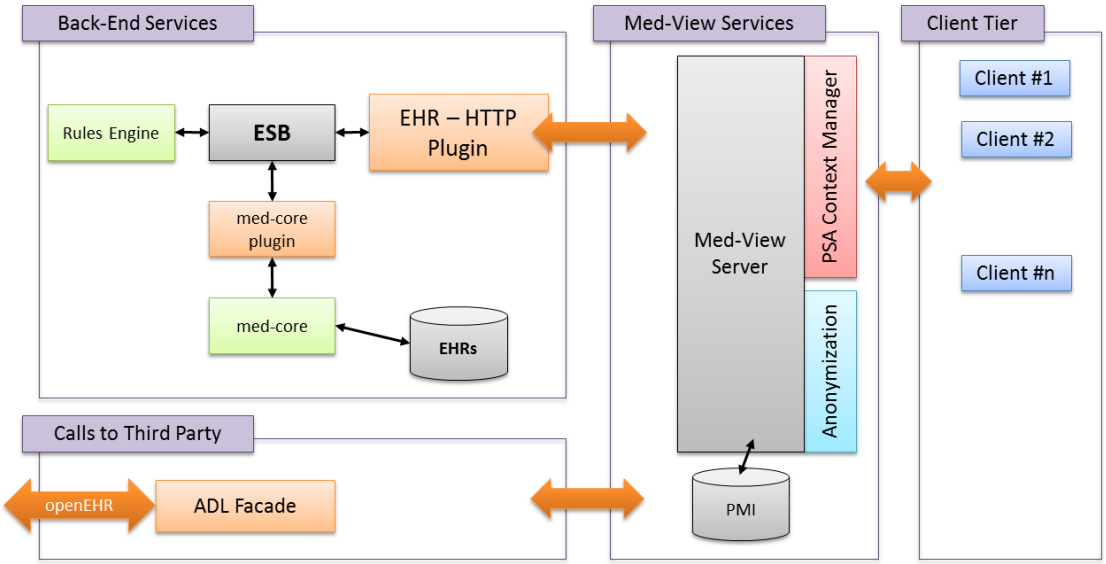

Figure 3.1: HIS Overall Architecture

Having a closer look at the architecture, Figure 3.1 shows, that there are distinct parts in the architecture. This allows the whole system to be deployed on different servers, for communication between the single parts is realized mostly as web-service calls (shown in orange, big double-sided arrows).

Since one goal was to build a scalable and maintainable software architecture, many design aspects focused on good and reusable design of components, typically known as design patterns [57]. To ensure interoperability and location transparency with other platforms and to allow calls from all environments, especially the front-end services made heavy use of the WCF technology - a type of a web service framework. WCF according to Löwy [58] "is a software development

kit for developing and deploying services on Windows". Another fact why WCF was taken as service development kit was the easy way of configuring services to use secure channels for their transmitting objects. Services can then make full use of the existing .NET framework security paradigms. In a clinical environment and especially when dealing with EHR data, security plays a major role. The used framework provides many different security models such as the role based security model, where each identity has several roles and therefore is granted permissions to objects or not [59].

Back-End Services

The back-end being fully developed provides all the necessary interfaces and services for the actual openEHR EHR standard implementation. By using an Enterprise Service Bus (ESB), plugins are integrated into the back-end to process service calls via HTTP and to store data to the database in the background. A detailed description of these services can be found in Chapter 4.

Third Party Interfaces

Some developed parts are performing calls to third party web-services, such as the openEHR Archetype Finder Bean[2], a web-service used to list and resolve existing archetypes by attributes such as the archetype ID, or by certain search patterns.

The Med-View Services can then make use of this service by wrapping its proxy into the ADL Facade to resolve archetypes which are missing in the local archetype store.

Med-View Services

The front-end which is actually in charge of rendering the archetype based information (namely the clients) is based upon services provided by the Med-View Services layer. Here one can find the basic infrastructure which is needed to communicate with the back-end and which is necessary to apply PSA context concepts and anonymization to the prototype. This part of the prototype is discussed in detail in Chapter 5.

[2] `http://www.openehr.org/wiki/display/healthmod/CKM+Webservices`, last accessed 24.07.2011

Client Tier

The client tier so far is implemented as Windows Presentation Foundation (WPF) application running on Windows® platforms. The WPF is a new unified programming model which was introduced in 2006 and can be used to develop Windows- and web-browser applications. [60, 61]

Clients support the rendering of the archetype based data and the participation in a PSA context provided by the Med-View Services. More details about the rendering techniques are discussed in Section 5.3.

CHAPTER 4

HIS: Back-End Services

The back-end follows a service oriented approach building up an integration architecture. This means, that the single components of a HIS can be deployed independently and that the services itself provide interfaces for other services to consume [62]. The presented back-end is capable of these requirements and is described in a short overview in the following chapter.

4.1 Description

With the openEHR platform as underlying standard for the medical content, we have to specify some requirements for the back-end to meet:

- The provided services have to have interfaces for SOAP web services and RESTful services to enable location independent calling of them.
- Additional services can be integrated into the system using a plugin system, thus allowing single deployment of components.
- Services for storing and receiving clinical data, namely openEHR objects such as EHR and compositions, have to be provided.
- The back-end has to take care of versioning and archiving features in order to meet the specification for the openEHR standard.

Other requirements which exist are requirements concerning computer based information systems in health care, for example that the interoperability between subsystems has to be possible or that the communication between such systems (meaning within the architecture) has to be persistent and reliable. [63, 62]

In order to build a flexible, scalable and maintainable integration architecture, the back-end was built using a JBoss ESB approach. This enables an easy to extend service oriented architecture. The following section deals with the basic architecture of the back-end.

4.2 Back-End Architecture

The build system is "based on a complex one-way-messaging event processor" and heavily uses loose coupling of components to guarantee interoperability and easy integration of subsystems [62]. Figure 4.1 illustrates the architecture by showing the flow of a sample request for retrieving an EHR from the back-end.

A caller uses the provided ehr-http plugin and its service interface to call the back-end services (1). The plugin then publishes the request as a message on the ESB core queue (2). After consuming the message (3), the rule engine applies content and header based rules to the message to actually find out, where the message is going to be routed (4). The corresponding receiving plugin pulls the message from the queue (5) and processes it in the core system (6). The response from the core system is then transferred back to the ESB core queue (7) and later on again processed by the rules engine (8). The rules engine forwards the message to the original plugin (9) which receives the message (10) and sends the response back to the caller (11).

Integrating more subsystems in this architecture means providing more plugins, which can be used in the ESB core system. After configuring the rules engine, such additional plugins are ready to communicate and integrate with the existing services.

Since they are the most important ones for the realization of front-end services and clients, the med-core component and the ehr-http component are going to be explained in detail in the following sections.

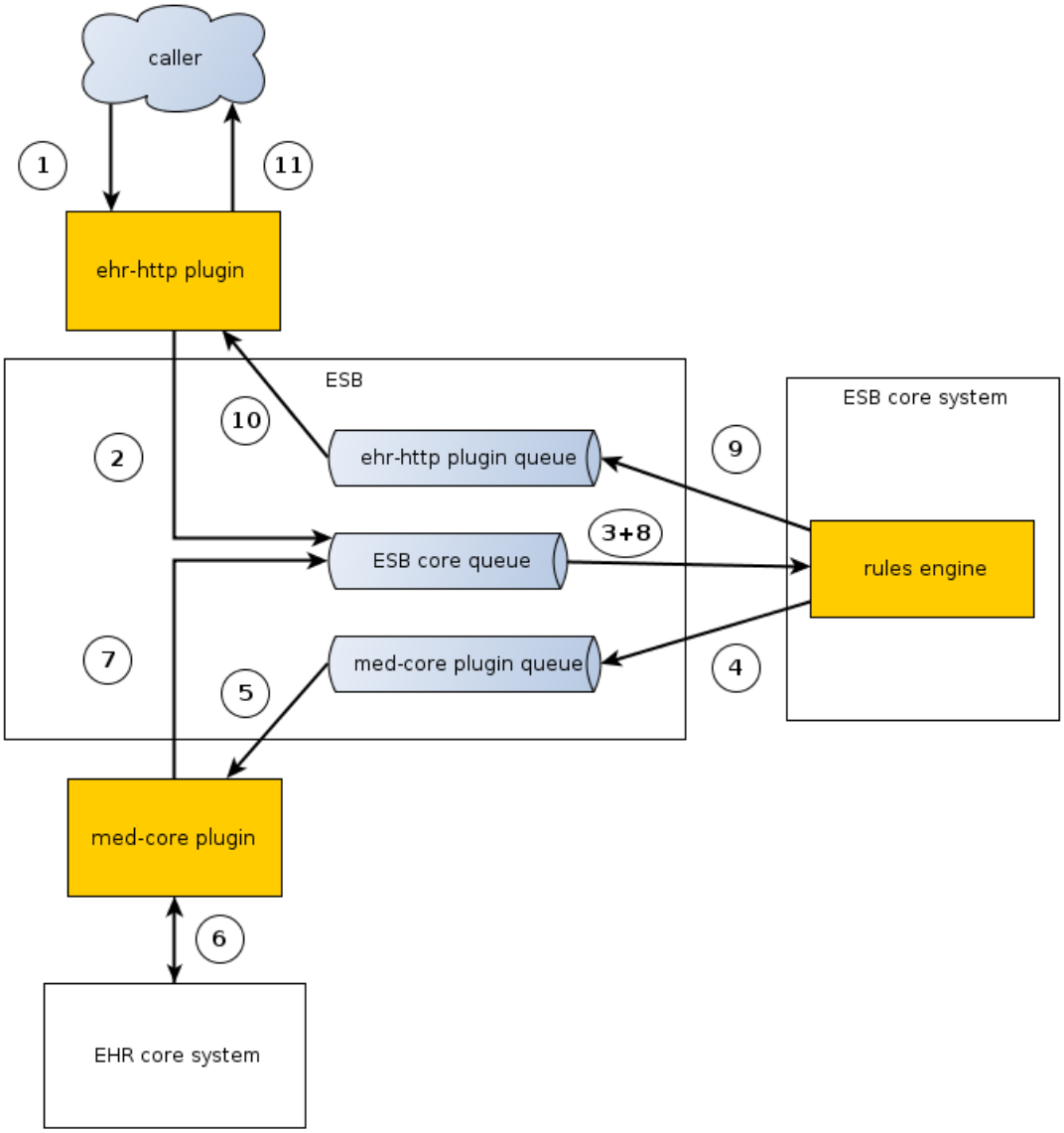

Figure 4.1: Architecture and full request workflow for retrieving an EHR (taken from [62], Figure 5.4)

The med-core Component

Building basically the 'heart' of the openEHR based patient-centered back-end services, this component is capable of storing, retrieving, archiving and versioning of clinical data, meaning EHR data such as an EHR itself, a composition or something else, specified in the openEHR standard.

The component itself is divided into three layers [62]:

- A model layer, which is basically responsible for storing, retrieving and versioning of all patient related data. By the use of a so called No-SQL database, the model layer uses Apache Jackrabbit[1] to perform the single tasks.
- The service layer which is providing the actual business logic services for the involved components, such as input validation or composition of data.
- A web service layer, which allows clients to call services via the exposed SOAP and RESTful service interfaces.

The exposed interfaces from this component include interfaces for EHR related functions such as creating a new EHR or saving an existing one. Additional interfaces for storing actual content to EHRs are also exposed. For finding data again with a corresponding identifier later on, an object reference resolver was added and exposed as well via interfaces. These functions build up the basic core component of the prototype.

The ehr-http Component

The med-core component described before is now being integrated into the ESB system and the rule engine is configured with corresponding rules to delegate messages (service calls) to the plugin. In order to invoke service methods from the outside of the ESB system, another component is needed: the ehr-http plugin. This plugin exposes the functionalities of the med-core plugin of the HIS.

[1] `http://jackrabbit.apache.org/`, last accessed 24.07.2011

Invoked methods on this plugin force the plugin architecture to create a corresponding message and enqueue it to the ESB. Handled and hand over to the related actual service by the rule engine, this message contains all the needed information on the occurred method invocation, such as the event type (creating an EHR, retrieving a composition, etc.). Identified by a unique key, this message is passed through the architecture, leading to an asynchronous processing of the actual service call. Having this in mind, callers actually cannot keep their HTTP request open that long to wait for the response. [62]

So, the callers get responded a callback ID for this service invocation and can then decide to either poll the service for the result to be present or can implement a callback interface and get automatically notified about finished requests by the ehr-http plugin. Fetching the result or waiting for the callback to occur always involves the callback ID, this way the client (caller) is able to identify the matching response to a request via this ID.

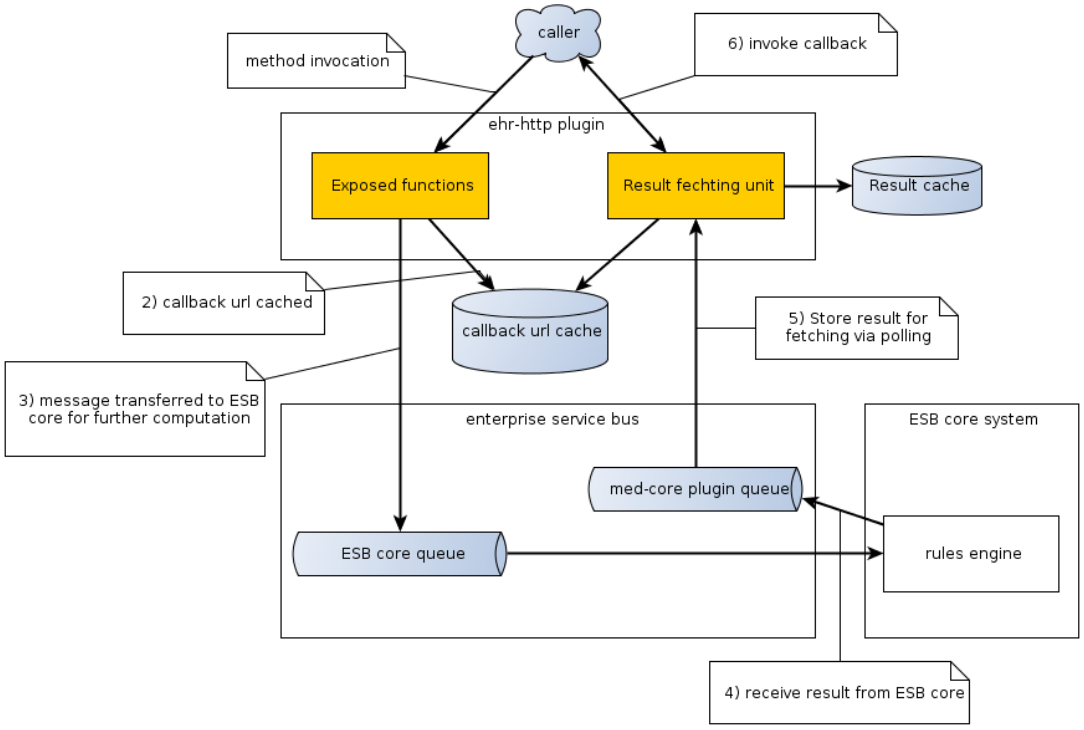

Figure 4.2: ehr-http plugin communication flow (taken from [62], Figure 5.3)

Figure 4.2 shows the process of a service call including the corresponding callback. A caller invokes a method on the ehr-http plugin passing a callback Uniform/Universal Resource Locator (URL). This URL is cached and the actual message for the service call is transferred to the ESB. The rule engine then evaluates this message and passes it on to the corresponding plugin. After finishing the service call, the chosen plugin stores the results, which are afterwards fetched via polling by the resource fetching unit. This part of the system uses the information of the callback URL in the cache and the corresponding message ID to correlate the result with the actual callback URL. After having the results ready, this unit performs the callback on the caller, handing the results over by invoking the given service method on the caller side. [62]

The later on presented front-end (and the corresponding implemented services) are using the back-end services via the callback functionality, meaning, they have a callback interface implemented. Since a polling mechanism would involve a lot of additional traffic and thus overhead, the way of getting notified about the finished call and getting provided the results is saving both resources and bandwidth.

4.3 Extensions for the Back-End

During the development of the front-end services and the related client application, some minor issues arose. Those issues are collected here and can be seen as a possible extension for the back-end services, since they are closely related to the openEHR service related methods.

ADL-Http Plugin

Already described in the architecture overview in Chapter 3, the implemented system uses a facade to call the ADL Finder Bean from openEHR to search for existing and published archetypes to make use of them in the application. Since this will be a recurring task for all applications dealing with archetype related data, the now implemented facade could be integrated into the ESB core system to better provide and share this service across all consuming applications.

Not only would this minimize code duplication and therefore related bugs in the applications, but also it would be possible to implement a good caching mechanism to store found ADLs in

the back-end. Furthermore this would enable the back-end to keep track of different versions of the stored ADLs thus leading to multi-versioned archetypes. Whenever an ADL gets replaced by a new version in the ADL Finder Bean, a new version is created in the back-end as well. Data stored to match previous versions of the ADL can in this case still be viewed without any further problems.

OPT Extensions Plugin

Addressing almost the same issue is the creation and usage of OPTs. Since the front-end client application is designed to work with OPT data, it has to load them somehow. So far, the implemented methods focus on loading them manually. An additional plugin in the back-end would again minimize code duplication and enable versioned and archived maintenance of OPT files. A possible extension to this would be a search engine for existing templates and the possibility to generate such templates on the fly.

CHAPTER 5

HIS: Front-End Services

Besides the already described back-end for the developed prototype of the HIS, a front-end and its services were developed to enable client applications to access the medical information from the underlying openEHR back-end. This chapter describes the actual developed front-end services including the client applications and the advantages and disadvantages of the chosen solution. Possible extensions to the system are shown later on in Chapter 7.

5.1 Description

Whereas the back-end services are capable of dealing with all the related medical data such as EHRs and compositions, versioning and archiving it, the front-end was designed to meet the following requirements:

1. Most important is the adequate visualization of medical data on the GUI. Therefore, the front-end has to implement certain rendering techniques in order to deal with the archetype based records and the OPTs.

2. The front-end has to provide a Patient Synchronized Application (PSA) adapter interface, such that different client applications can connect to it and share information within a PSA

context. The interface should be accessible platform independently and from any enabled technology.

3. The usage of a Patient Master Index (PMI) on the front-end side allows the maintenance of patient information. However, an anonymization has to be done in order to provide a safe and secure environment.

4. The built front-end services have to use the back-end services and make use of the callback functions. Therefore, a callback interface has to be provided to the back-end.

Requirement one appears to be trivial and only needs to be implemented on the client application. However, as described in Section 5.3, the actual process of rendering archetype based data and therefore user interfaces is not as trivial as it seems. Many constraints and limitations make it hard to build a flexible and maintainable framework to render user interfaces based upon OPTs.

Requirements two and four are related to implementing a service. For requirement two it is an active service being used by the client applications to take part in a PSA context, for requirement four it is more or less a passive service which gets invoked by the callbacks coming from the back-end services. Both requirements make the need for an additional service layer in the whole architecture. This additional layer provides immediate services for the clients such as the PSA related services and it provides the callback services for the back-end.

The introduction of a small PMI in requirement three involves storing data independently from the actual EHR data in the openEHR back-end. Considering anonymization, this leads to a distinct database for storing only patient related information and an (probably encrypted) identifier of the corresponding EHR in the back-end. A running anonymization service brings together the patient related information and the actual medical data and sends it to the clients. It never transmits the original identifiers of both, the patient data and the EHR data, but a generated session identifier. Thus the client never gets information about the real identifiers of the patient and the EHR but only combines a certain name and some medical data and visualizes it.

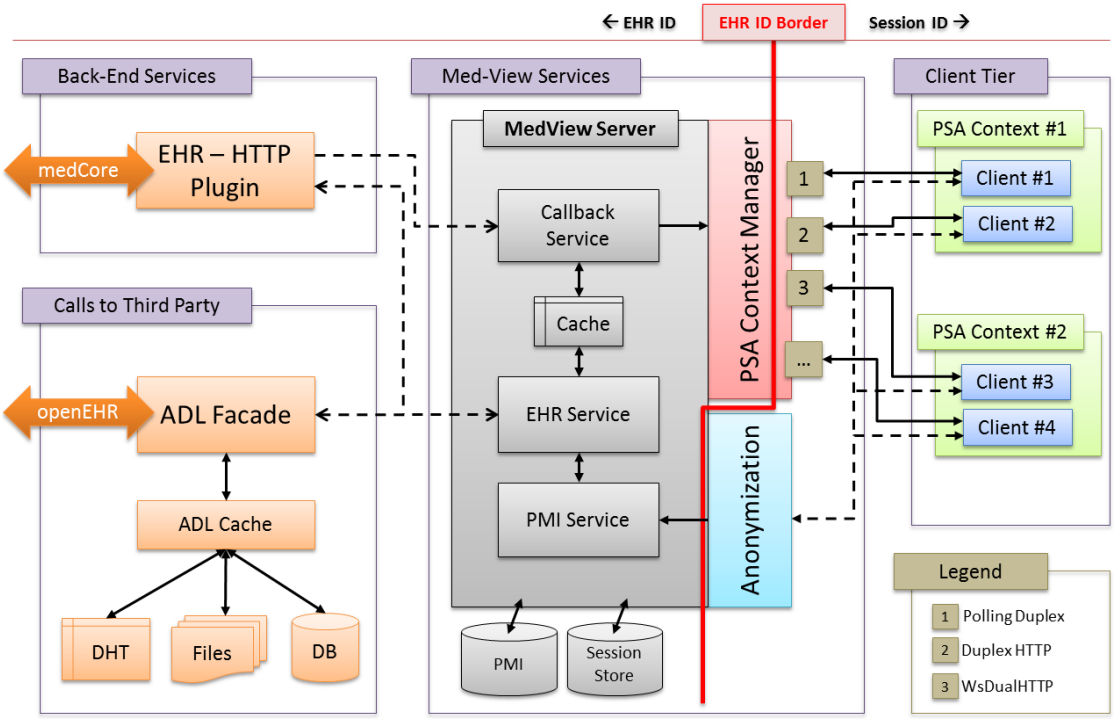

Figure 5.1: MedView System overall Architecture

5.2 Front-End Architecture

The designed application—because of the requirements—has a rather complex architecture, relying on many distinct services possibly running on different servers. This section provides a detailed description about the distinct involved services and parts of the architecture.

Figure 5.1 shows the introduced architecture for the front-end services and clients. The following sections describe the single parts of the architecture, starting from the back-end and third party services.

ADL Facade

The back-end so far does not provide services for searching and retrieving archetypes. Since the introduced applications need to load ADLs and render them, a facade was introduced to access the openEHR ADL Finder Bean web-service.

As shown in Figure 5.1, the ADL Facade performs calls to third party services, in this case the openEHR services. To minimize the overhead for calling this services constantly, the facade itself introduces several mechanisms to cache already loaded ADLs. For this, the ADL Facade uses a cache strategy, which can be configured using an Inversion of Control (IoC) container. By implementing certain interfaces, additional caching strategies can be implemented and then loaded into the configuration.

So far, the ADL Cache uses a simple strategy implementation storing cached items in a simple hash table in memory. To enhance the response time and to ensure that cached data does not get lost, strategies such as a distributed hash table solution or a database based solution should be implemented here.

EHR and Callback Service

The clients in the presented solution do not make use of the back-end services directly, but use several other services to delegate their calls. This has several reasons:

- First of all, the back-end services provide the functionality of callback functions. Since the clients are not polling the server for finished requests and therefore available results, a service has to be implemented serving as callback location. This cannot be done by the applications itself.
- Secondly, clients are usually not accessible via a public IP address or have a web-server running to actually implement such callback services.
- Furthermore, clients of the presented solution are part of a PSA context and thus connected to a central server for this purpose.
- Lastly, one requirement was the storage of a central PMI and to implement anonymization services, hence leading to a centralized service layer again.

For accessing the openEHR relevant back-end data and for providing a callback service, the MedView Server as shown in Figure 5.1 has two separate services running: the **EHR Service** and the **Callback Service**.

The **EHR Service** is called by the clients directly and then delegates the calls to the EHR-HTTP Plugin as provided by the back-end services, forwarding the messages to the med-core components. Listing 5.1 shows the offered interface for clients. Clients are allowed to create a new EHR and to save a composition belonging to an EHR. As seen in the listing, clients always call the service by providing a session identifier and a `pmiPatientId`, which is a generated identifier from the PMI. Clients because of this never know the actual EHR system identifier.

```
namespace hi5.medview.server.contracts.EHR
{
    [ServiceContract(SessionMode = SessionMode.Required)]
    public interface IEhrService
    {
        [OperationContract]
        Guid CreateEhr(Guid sessionId, Guid pmiPatientId);

        [OperationContract]
        Guid SaveComposition(Guid sessionId, Guid pmiPatientId, string composition);
    }
}
```

Listing 5.1: EHR Service interface definition

The **Callback Service** is used by the EHR-HTTP plugin to notify the MedView Server about finished requests and available results. After sending a request to the back-end using the EHR Service, the Callback Service is called. Whenever a call is received by the EHR Service, a unique identifier for this method call together with the session identifier of the client and the returned callback identifier from the back-end are stored in a cache. After notification by the back-end, the Callback Service looks up this information again from the cache, thus knowing to which client and which method call the callback results belong to. The results and the corresponding session identifier are passed on to the PSA context manager to notify any clients belonging to this session about the available results from the back-end.

In case of a store action on the back-end, the returned results usually only have to be passed on to the clients. However, there is one specific call forcing some more actions: the creation of a new EHR in the back-end. A new EHR for a patient means that the returned result from the EHR-HTTP plugin contains information about the given EHR system ID. This system identifier

is not passed on to the clients, but is encrypted and stored in the PMI. For this special case, the cache shared by the Callback Service and EHR Service provides a feature to insert a function callback, which is executed when the response from the back-end arrives. This function callback then searches for the corresponding patient in the PMI and stores the encrypted EHR identifier.

PMI Service

This service is used within the MedView Server components to resolve the EHR identifier for `pmiPatientId`s contained in the calls coming from the clients. Whenever an action involves the search for a patient or the act of resolving a `pmiPatientId` to the actual EHR identifier, the PMI Service gets invoked.

Anonymization Interface

Besides being used within the MedView Server components, the PMI Service gets called by the Anonymization Service. This service provides an interface to clients to search for patients in the database. Clients then get back a generated `pmiPatientId`, which is the identifier for a patient within a session. As long as the session is alive, the client can use this identifier to perform actions on a patients medical record via the EHR Service or can make use of the PSA Context Manager functions.

This service is being introduced due to security and safety issues. Without having a valid session and PMI patient identifier, a client cannot access nor modify patient related data. Furthermore, since this simple form of anonymization is done, the real given identifier for a patient record in the back-end is never sent to clients, thus not allowing to perform any actions on a 'personalized' record without notice.

PSA Adapter Services

One of the main parts of the front-end architecture is being represented by the PSA Context Manager providing all the necessary service methods for clients to take part in a PSA enabled context. As shown in Figure 5.1 (in red), the context manager sits on top of the MedView Server and offers several different adapters (shown in ocher) for the underlying PSA services to clients.

The so far implemented solution supports three different adapter services for the PSA context manager but can be easily extended by any other adapter.

- **PollingDuplex**: offered to support clients running on Silverlight[1]
- **DuplexHTTP**: can be used by all clients supporting the HTTP protocol and is a full duplex interface, meaning, the client does not have to poll for notifications but rather the server can actively push a message down to the client. This interface is offered to support scenarios where clients sit behind firewalls or proxy servers.
- **WsDualHTTP**: offered for WPF and all other .NET specific clients this endpoint uses web-services to communicate with the connected clients. This endpoint however has some limitations in reachability, meaning that clients sitting behind firewalls or somewhere in the internet cannot get notified.

As all of the adapters expose the same interface, a service has to implement the following interface `IContextManager` as shown in Listing 5.2.

```
namespace hi5.medview.server.contracts.PSA
{
    [ServiceContract(CallbackContract = typeof(IContextClient),
        SessionMode = SessionMode.Required)]
    public interface IContextManager
    {
        [OperationContract]
        ContextInformation JoinContext(Guid userId);

        [OperationContract(IsOneWay = true)]
        void ChangeContext(ContextInformation contextInformation);

        [OperationContract]
        void LeaveContext(ContextInformation contextInformation);

        void HandCallbackToClient(Guid sessionId, Guid callbackId, string payload);
    }
```

[1] http://www.silverlight.net/, last accessed 24.07.2011

```
    public interface IContextClient
    {
        [OperationContract(IsOneWay = true)]
        void FollowContext(ContextInformation contextInformation);

        [OperationContract(IsOneWay = true)]
        void NotifyAboutCallback(Guid callbackID, string payload);
    }
}
```

Listing 5.2: PSA adapter interface definition

As already described in Chapter 6 and its sections, a client can use the method `JoinContext` to take part in a PSA context. This means, that the client signs up at the server and then gets notified about changes in the context and incoming callback results by the server. The method `ChangeContext` can be used to switch the actual patient context to a different selected patient. All other clients taking part in the same context will get a notification about the change and then can decide what to do about the change. To leave a context again, a client can simply call the `LeaveContext` method. The server will remove the client from all the notification caches after this and no further messages will be sent to the client.

To ensure accessibility also the callback results of requests sent to the back-end are provided by the context manager exposing a method named `HandCallbackToClient` which enables the Callback Service to forward incoming messages from the back-end to the corresponding clients.

Adapter Endpoints using WCF

From a technologist point of view, the actual endpoints offered are not standalone implementations but rather a different endpoint offered via the same WCF service. "WCF is Microsoft's implementation of a set of industry standards defining service transactions, type conversions, marshaling, and the management of various protocols. Consequently, WCF provides interoperability between services" [58]. Such services are flexible and achieve compatibility by use of different binding modes. Furthermore, easy configuration allows to use secure channels for transmitting objects.

The following table shows the available combinations for build-in bindings and security modes:

Name	None	Transport	Message	Mixed	Both
Basic binding	Y (d)	Y	Y	Y	N
TCP binding	Y	Y (d)	Y	Y	N
IPC binding	Y	Y (d)	N	N	N
WS binding	Y	Y	Y (d)	Y	N
Dual WS binding	Y	N	Y (d)	N	N
MSMQ binding	Y	Y (d)	Y	N	Y

Y (d) ... Yes (default), Y .. Yes, N ... No

Table 5.1: Bindings and transfer security modes (taken from [58] Table 10-1)

Dual WS binding as shown in Table 5.1 offers security on message level by default. Since this one is the only build-in mode offering the duplex channel as well, it is exposed by the adapter. The exposed interface for Polling Duplex comes with the Silverlight framework and is specially made for Silverlight clients. The Duplex HTTP[2] interface is an extension to the existing binding modes in .NET enriching them with a special functionality: duplex over HTTP in WCF.

Extending the possibility of endpoints offered by the service means implementing further custom binding modes offering duplex functions and then including them into the service by simply configuring them on the PSA service adapter.

Implementation and Integration of PSAs

The previous section outlined the details of the server-side of the PSA context application. The client application is designed to use the offered services by signing up to the PSA context automatically while logging in. Besides signing up to the context, a client also has to fulfill a certain callback contract as specified by the WCF service contract on the PSA context manager. Listing 5.2 shows a second interface specifying the callback contract, a client has to implement.

[2] `http://archive.msdn.microsoft.com/duplexhttp`, last accessed 24.07.2011

Being used on the original contract as parameter for the `CallbackContract` value, the `IContextClient` interface specifies two methods:

- **FollowContext**: whenever another client belonging to the same context is switching to a different patient, the signed up client is getting notified about this change via this method. As parameter, the client gets handed the current `ContextInformation` object, which is responsible for storing parameters such as the current session identifier, the currently selected patient and additional information needed for the server communication.

- **NotifyAboutCallback**: having performed calls against the EHR Service in the MedView Server, a client gets notified about the results of the call from the PSA context manager via this method. Containing the callback identifier to correlate the result to a call, this method also gets passed the actual payload (if there is one). The client then can make use of this payload to update the user interface.

Figure 5.2 shows a schematic drawing of several running PSA contexts and signed up clients. A client (illustrated in blue) is used by a physician (e.g. a doctor). The doctor signs in at a client, meaning, the client is connecting to the PSA infrastructure in the background with the credentials of the connecting doctor. The client then gets assigned a unique callback identifier so that the context manager later on can notify this very client and gets back this information together with a session identifier.

The same doctor signing up at a different terminal (see Figure 5.2, Client #2) then connects to the same PSA context as before, meaning, every single step in switching to another patient on client #1 is propagated to client #2 and vice versa. Therefore the client currently signing up is getting assigned a different callback identifier, but the same session identifier.

Another doctor signing up (compare PSA context # k with doctor # n on client # m) creates a new context on the server and gets assigned completely new session and callback identifiers. A context's lifetime is limited to the lifetime of its belonging clients. As long as the clients are online, the context is kept alive. When the last client signs out from a session (context), the context is being destroyed on the server too.

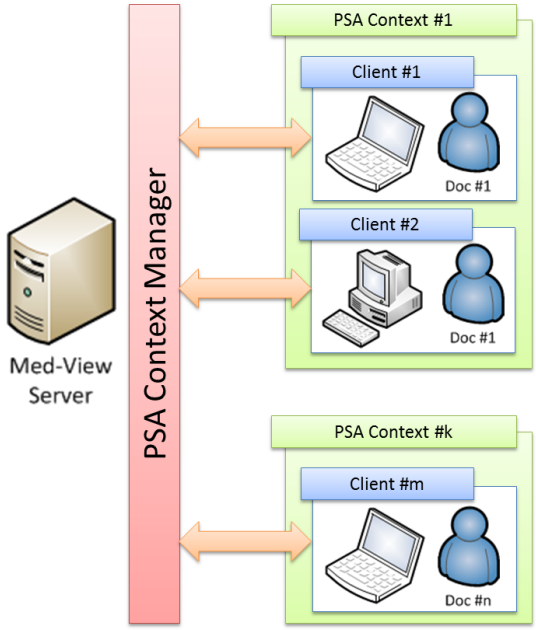

Figure 5.2: PSA Context Manager Implementation

Figure 5.3 shows the view of a signed in doctor onto the existing stored compositions of a selected patient in the list view. The left part of the window shows the currently logged in user, the currently selected patient details and the PSA context information which the client has available. Each client has its own unique session identifier but may belong to an already existing PSA context which is given by the name and the identifier value. The currently selected patient identifier (the anonymous one) and the resolved patient's name is displayed along with the other information.

One important implementation aspect about the PSA context clients is, that all the actions are performed in an asynchronous manner. Since it is not defined how long a call may last, the user interface is not blocked during the process is running on server-side. This does not only involve problems when thinking about a good user experience of applications but also involves certain implementation aspects about asynchronous callbacks. Whenever dealing with decoupled method calls on user interfaces, one has to implement certain dispatcher services to react on incoming method invocations.

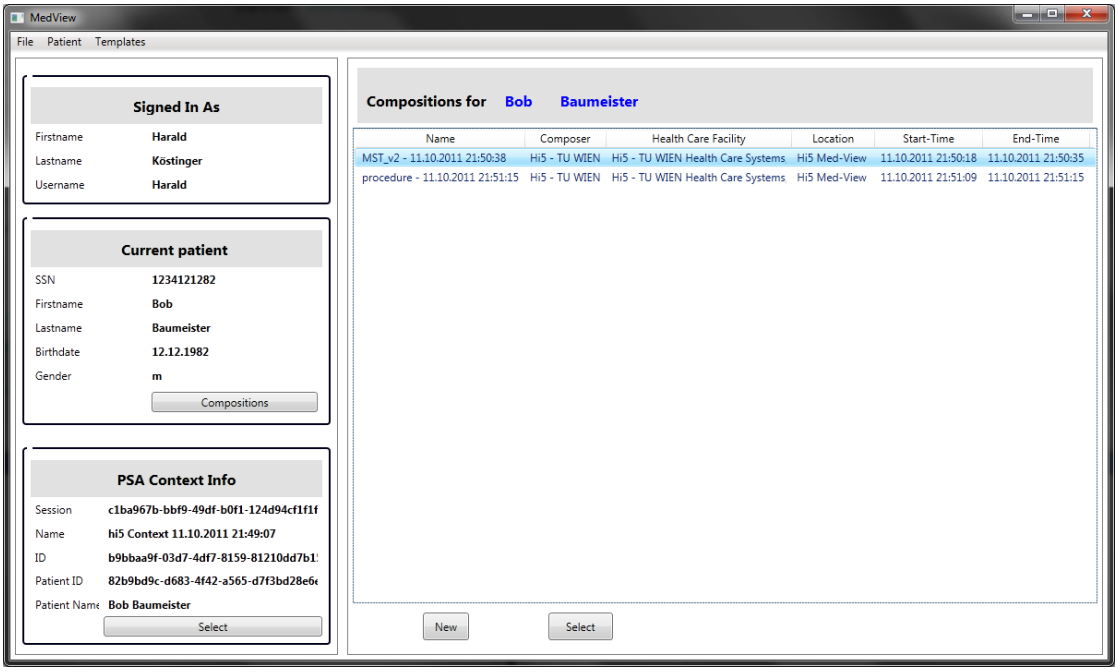

Figure 5.3: Med-View Front-End: Showing the selection of a patient's compositions and the PSA context information

5.3 Client: Dynamic Rendering of Archetypes

The rendering and therefore interpretation of archetypes and templates takes place on the client side, directly at the client application. After loading medical data, the client is going to fetch the corresponding OPT and its archetypes, to start processing it and rendering the GUI.

The way of processing the templates and rendering the GUI is not new, but rather a refactored and remodeled approach already used by GastrOS. As already explained in Section 1.4, the implemented solution is based on the existing open source framework from GastrOS called the MST-SDE [19]. This framework uses the existing .NET implementation of the openEHR standard and provides extensions to it to parse and load OPTs and to actually create instances of the containing archetypes using the openEHR RM.

Although this framework provides a lot of functionality, it cannot be used directly in the implemented solution, since it is based upon the .NET WinForms technology and closely coupled to it, thus making it hard to reuse the components in a WPF environment. Therefore, the basic

approach and its architectural design was taken and ported to a loosely coupled implementation in the .NET framework 4.0 and then realized using the WPF technology. The new designed framework allows a better reusability of the single classes and interfaces within different front-end technologies.

The framework has to be capable of the following things:

- Given an archetype or an OPT and the AM representation, the library has to parse it into an instance of the Archetype Object Model (AOM).
- It has to evaluate the inner structure of the AOM and generate the view elements.
- It has to consider constraints within the AOM which probably are present to limit the range of inputs (e.g. for quantity values there is a lower and upper limit, meaning a range in which the values are considered to be valid).
- It has to be capable of rendering single view elements such as text, ordinal, quantity or date-time elements and it has to be equipped with a mechanism to render clusters as well. (Clusters are a sort of grouping of elements, compare Figure 2.10)
- In case of an unknown element or structuring element, it has to render an appropriate control giving an information, that a certain type of control is not implemented yet.

To show the actual architecture of the components, the following figures should help to explain the connections between the single parts. Figure 5.4 illustrates some of the existing interfaces for the so called *ViewElements*. Each view element implements one of the shown interfaces, all of them deriving from `IViewelement<T>`. This interface is generic and allows for a model type specification. Each view element has a belonging model (a property where the actual values are coming from; in this case instances of the openEHR RM) and a description. The concrete interfaces for the special types of view elements have some more properties, such as a format and format string for the `IDateTimeViewElement`. A special case is the `IClusterViewElement` which allows to be a host for sub-view-elements, meaning a container control for them.

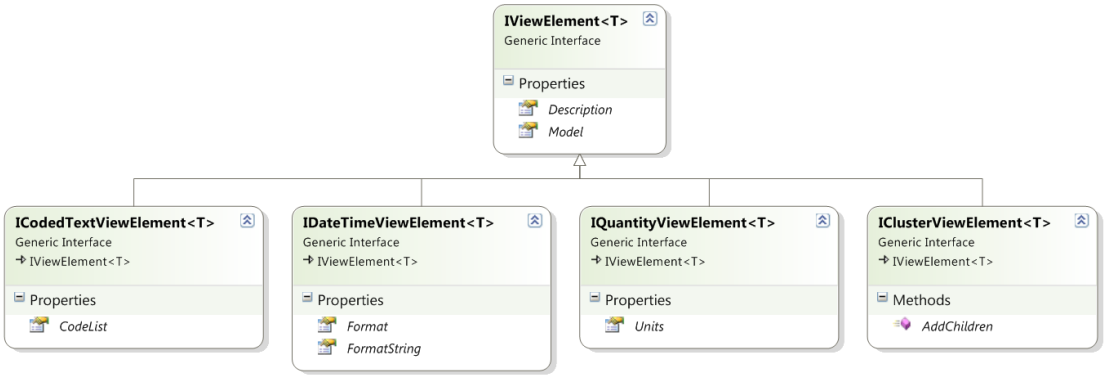

Figure 5.4: Interfaces for some of the existing and implemented ViewElements

Some of the concrete implementations of the interfaces are shown in Figure 5.5. A concrete view element is build upon a `Constraint` coming from the underlying AOM and is usually present as instance of type `CComplexObject`. This constraint keeps all relevant information about rendering the control, its type, its name, its description etc. All those single values can be extracted from it using extension methods from the openEHR reference implementation. Concrete view element classes are always implementations of their respective belonging interfaces and a concrete model class: e.g. the `DvQuantityViewElement` implements its interface and specifies the type `Element` for its model. A `ClusterViewElement` however is only a container control, thus its model type is a `Cluster`. In contrast to standard view element controls having a `Element` as model, a `ClusterViewElement` can have a certain amount of occurrences, meaning that the user can repeat this container several times.

Listing 5.3 shows how to extract certain constraints out of the AOM and how to use them later on. Since archetypes and templates can be translated into several languages, one can specify which language to extract with the method `ExtractOntology`. As default, the current language is taken.

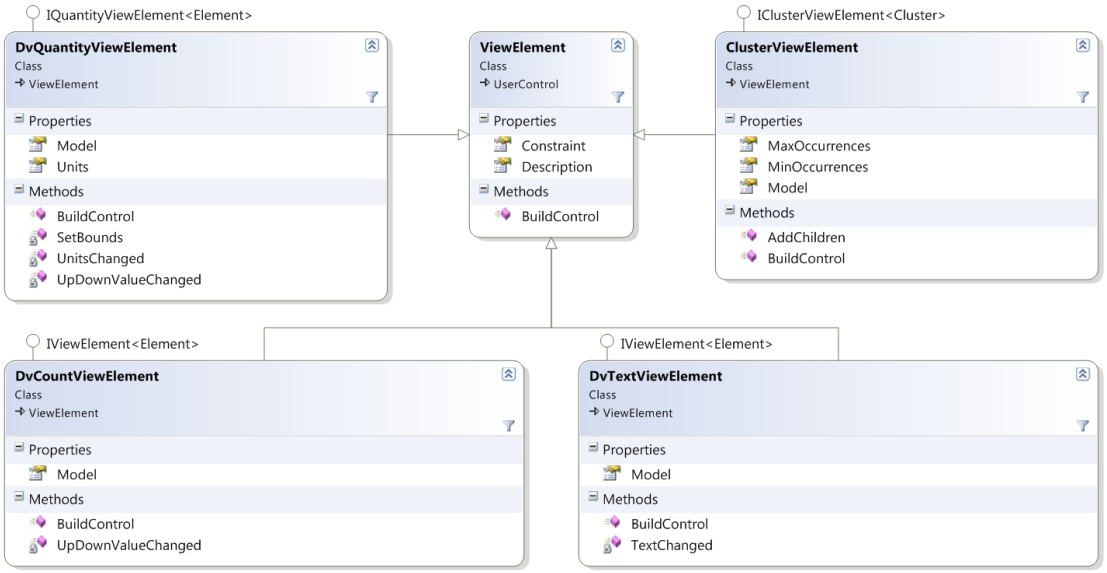

Figure 5.5: Extract of the actual implemented ViewElements

```
// Extract the description out of the constraint
Description = constraint.ExtractOntology();

// e.g. for a DvQuantity View Element
// extracting the quantity information out of the constraint (lower and upper bounds)
var quantityConstraints = Constraint.ExtractElemValueConstraint() as CDvQuantity;

// e.g. for a DvCodedText View Element
// extracting the single coded text elements (meaning the selection possibilities)
var valueConstraint = Constraint.ExtractElemValueConstraint() as CComplexObject;
AssumedTypes.List<string> codePhrases = valueConstraint.ExtractCodePhrase().CodeList;
```

Listing 5.3: Samples for extracting constraints and building controls

Since the implemented user interface is realized with the WPF technology, it follows a special design pattern for GUI development: the WPF Model-View-ViewModel (MVVM) design pattern. The MVVM[3] pattern is based upon the MVC design pattern which is usually used to develop user interfaces. In contrast to the MVC pattern, the MVVM pattern exchanges the controller part with a ViewModel part, since the controller is already implemented by default in .NET by the use of so called DataBindings. The ViewModel is a model exactly build for the

[3] http://msdn.microsoft.com/en-us/magazine/dd419663.aspx, last accessed 24.07.2011

view related parts, meaning, it is a subset of the data present in the actual model, specifically conditioned for the current view.

The so developed ViewModels are shown in Figure 5.6. Each ViewModel has the constraint, a reference to the belonging archetype in the background, a model and the corresponding ontology plus a reference to the actual view. This class furthermore is responsible for building the actual view element by invocation of the `BuildControl` method. It then extracts the constraints and ontology items out of the archetype, instances the corresponding model and builds up the related view. Thus, it stores a more precise set of data for the view in its model.

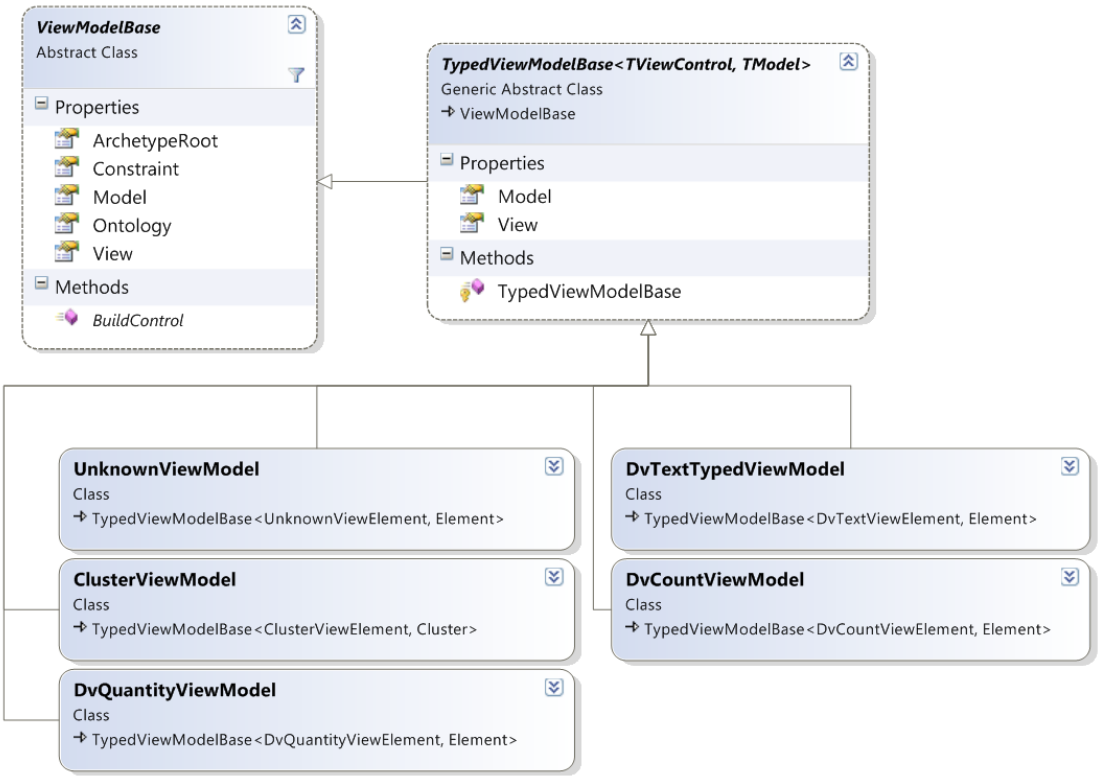

Figure 5.6: Extract of the corresponding model classes to ViewElements

Responsible for rendering the GUI out of the given OPT is the `IViewGenerator` and its implementing class the `ViewGenerator` as shown in Figure 5.7. Via an IoC container the `ViewGenerator` gets injected a specific type of `RmTypeViewGeneratorFactory` which is then capable of rendering views for certain RM types. The shown `RmTypeViewGe-`

`neratorFactory` for example can render elements and clusters. The instantiation of the needed models for the view models is done by the `RmInstanceFactory` which is capable of doing so for cluster and element types.

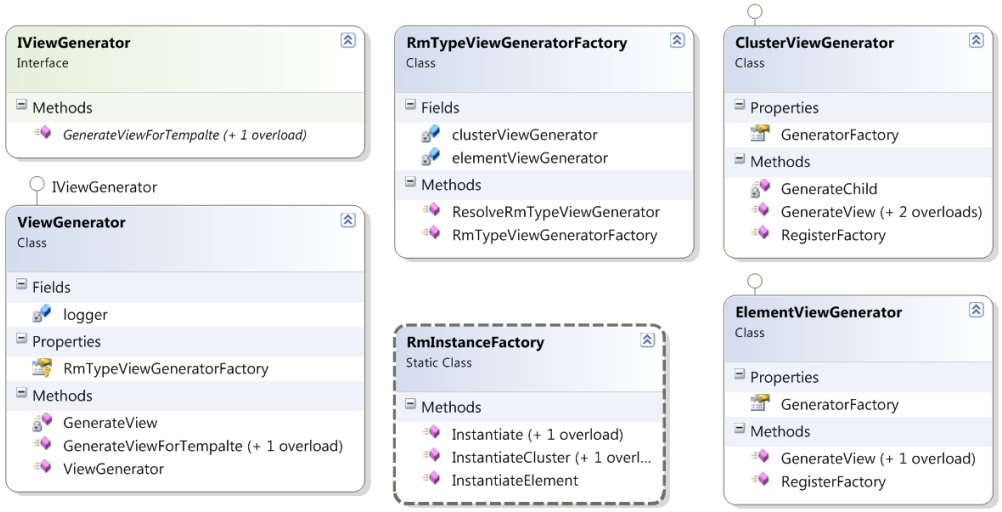

Figure 5.7: Implemented View Generators

Schematic Work Flow of the Rendering Engine

To see how all the single components and classes play along together, the list should provide an overview of how an archetype based view is actually being rendered. This now just takes into account the rendering process itself, not the data binding nor the loading of the data or the template from the back-end.

The rendering is done in the following steps:

1. The template is handed to the configured view generator and its corresponding method 'GenerateViewForTemplate' is called.

2. The single archetypes within the template are loaded. For each archetype, a separate tab-view control is created.

3. The view generator calls the 'RmInstanceFactory' to instantiate the currently loaded archetype.

4. The instance factory evaluates the type of the archetype and either instantiates a cluster or an element out of it.

5. The view generator calls the 'RmTypeViewGeneratorFactory' with the instance and the loaded archetype.

6. Again, the type of archetype is checked and the corresponding view generator is called: Cluster or Element.

7. The ElementViewGenerator directly builds up the associated view models and view elements by evaluating the RmType of the handed constraint.

8. In case of the ClusterViewGenerator, a container is built, and all its children are parsed in the same manner again, forcing this step to be a recursive one. The rendering process for children of a cluster start again by calling the 'RmTypeViewGeneratorFactory' in step 5.

9. The so built view model with its containing view element and model is then returned and added to the before created tab-view in step 2.

10. The whole process of rendering the view models and views for an archetype is done again for the next archetype within the template, starting at step 2.

Figure 5.8 shows the results of such a rendering process for a simple template for an address field. Each container within the template is rendered as a group-box on the GUI. Coded text elements in the archetype are rendered as drop-down lists and simple text entries are text-box elements. More structured entries are provided by e.g. the date-time controls, rendered when a `DV_DATE_TIME` element is found in the archetype. The results show the description of the archetype in German, since the ontology items for this archetype were only present in German.

Storing Medical Information as Compositions

The application now can render templates and their containing archetypes. Also, the corresponding instances for the underlying RM are created and are held in the model of the view controls. The remaining step is to actually store this entered medical data. As described in Section 2.4

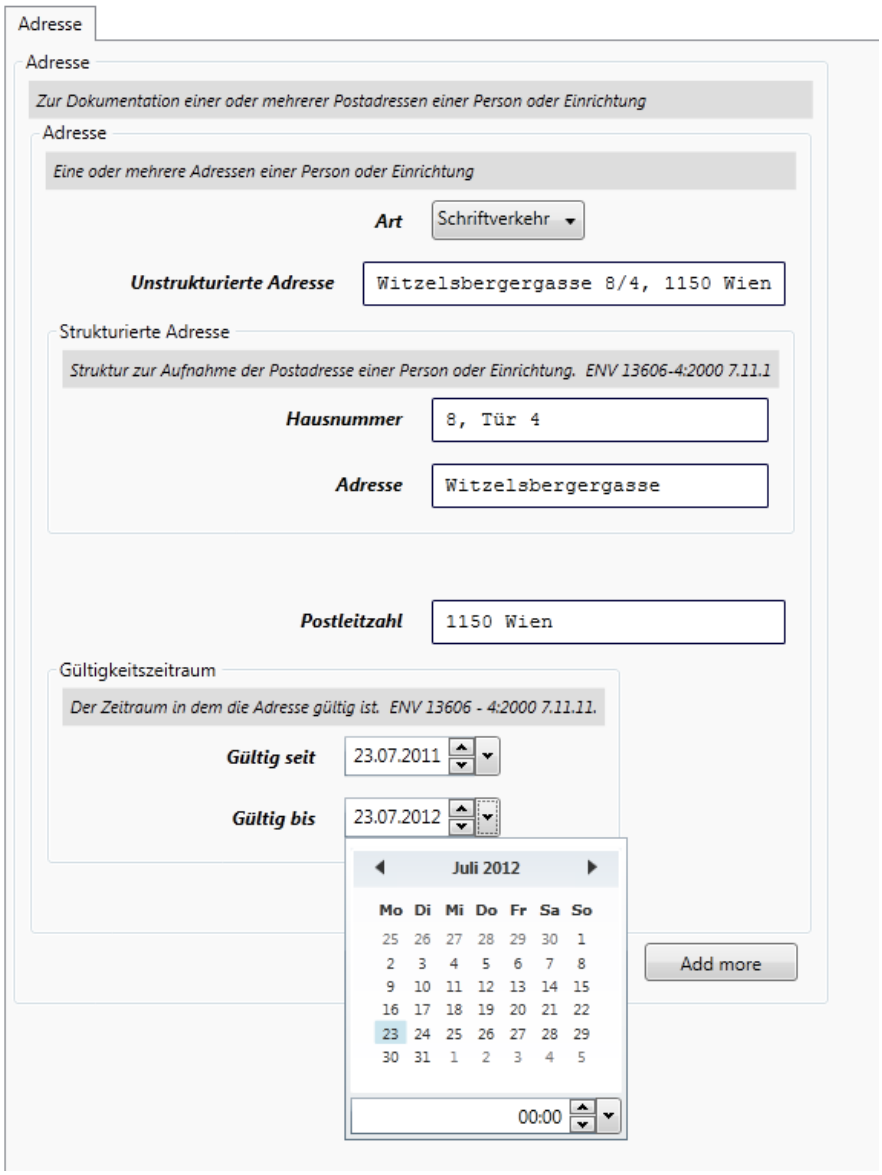

Figure 5.8: Rendered GUI for an address using the ontology for 'German'

and shown in Figure 2.10, the actual content is stored in compositions. Depending on the type of entry within the archetype (e.g. Observation, Action, Instruction, Admin_Entry, etc.), data is stored a little bit different.

Every entry type stores its content within and 'ITEM_STRUCTURE' [50]. The concrete classes for this type are:

- **ITEM_SINGLE**: used to encode objects as a single `ELEMENT`.
- **ITEM_LIST**: encoding `CLUSTER` elements containing `ELEMENT`s.
- **ITEM_TABLE**: rows in a table are encoded using `CLUSTER` objects which contain a number of `ELEMENT` objects.
- **ITEM_TREE**: are used to produce correct EN 13606 hierarchical formats.

To show how content is managed within such a structure, Figure 5.9 shows this in case of an `ITEM_TREE`. The `ITEM_TREE` class is compatible with the equivalent CEN class and can be constructed in HL7v3 by the use of a CDA 1.0 list [50]. As seen in the figure, the logical form (lower part) is a hierarchy of elements, whereas 'lipid studies' is a container for several other elements. Having a closer look at the physical form, which is constructed of the storing information, single elements in the logical form are converted to `ELEMENT` representations in the tree and container controls such as the 'lipid studies' are translated to `CLUSTER` objects.

Stored within the `data` property of an entry, such an `ITEM_STRUCTURE` is then part of the composition which is simply serialized into XML. This serialized format is sent to the back-end and stored as additional composition to an existing EHR.

The chapters presented before dealt with the technical parts of the system and its architecture. Since the front-end makes extensive use of the IHE PSA profile, the upcoming chapter deals with the IHE profiles in general and outlines the basic functions and limitations of the PSA profile. Later on, some possible extensions to the profile are given which would lead to enhanced quality in health care when dealing with workflows.

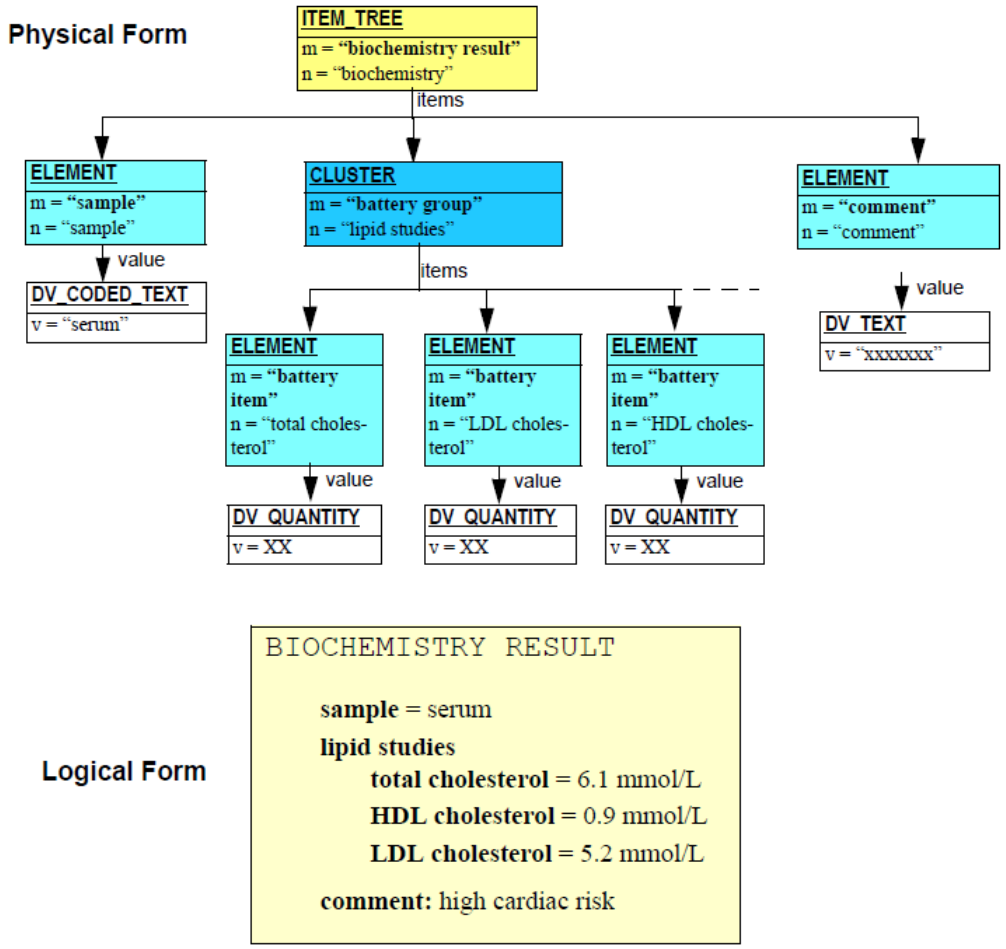

Figure 5.9: ITEM_TREE instance structure (taken from [50], Figure 6)

CHAPTER 6

Patient Synchronized Applications

In order to enhance the quality of modern health care systems and to stimulate the integration of such systems into health care institutions, the IHE International, Inc. designs and describes fundamental objectives to do so. Among several integration profiles, the profile for a Patient Synchronized Application (PSA) supports the clinical work with EHR data on workstations.

6.1 IHE Integration Profiles

The IHE IT infrastructure integration profiles (see Figure 6.1) define a common language in precise terms that health care professionals can use. They recommend the adoption of existing standards and state how applications should be designed to meet clinical standards and needs. As they define precise terms, health care professionals can exactly define what they need by the use of such terms and referencing the detailed specifications of such profiles from the IHE IT infrastructure. Moreover IHE specifies when to use which standard respectively profile and how. [64, 65]

Profiles are defined by the actors involved and a set of specific transactions which are exchanged by the actors. Vendors for health care software systems can then easily implement new products by following the specifications and by implementing the appropriate actor(s) and transactions as defined in the profiles. [64]

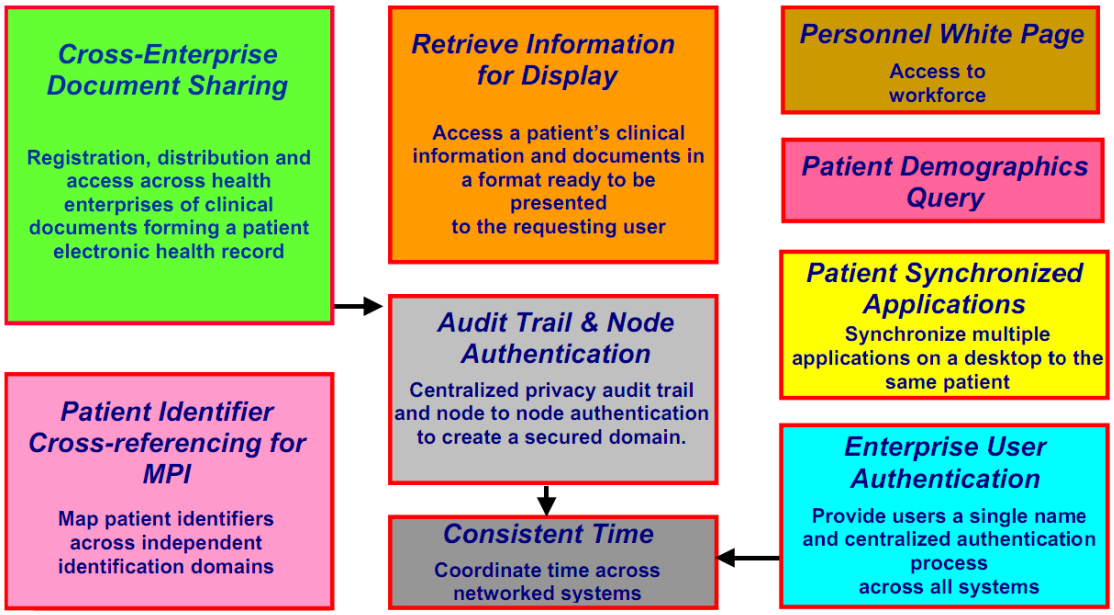

Figure 6.1: IHE IT infrastructure integration profiles (taken from [64], figure 2.1)

The currently defined set of profiles consists of the following (also shown in Figure 6.1) [64]:

- **Retrieve Information for Display (RID)**: used to define the simple and rapid access to patient information which can be stored in formats such as CDA, Portable Document Format (PDF), Joint Photographic Experts Group (JPEG), etc.
- **Enterprise User Authentication (EUA)**: supports a centralized user authentication management, single sign-on features for applications and so called 'one name per user' methods.
- **Patient Identifier Cross-referencing (PIX)**: allows to do cross-referencing among multiple Patient Identifier Domains by identifying a single patient in one system and storing information about the identifiers of this very patient in other domains.
- **Patient Synchronized Application (PSA)**: support viewing of the same patient among different independent devices, workstations and applications to reduce the tasks of selecting patients in different applications.

- **Consistent Time (CT)**: a definition for mechanisms to ensure a consistent time management between multiple actors. Many other integration profiles make use of this one to ensure their functionality.
- **Patient Demographic Query (PDQ)**: defines a standard of how to query patient information servers for patients according to a given search criteria and retrieve patient demographic information.
- **Audit Trail and Node Authentication (ATNA)**: mainly describes the security environment assumed for the systems, basic auditing and security requirements.
- **Cross-Enterprise Document Sharing (XDS)**: enables health care facilities and organizations belonging to an XDS Affinity Domain to share clinical records in form of documents. The profile is based on different standards for the message exchange.
- **Personnel White Pages (PWP)**: provides access to a user directory and has broad use among all different kinds of applications. It is used to enhance the clinical workflow on retrieval of contact information and can be integrated into user interfaces as well to provide user friendly names and titles for referenced actors of medical information.

All of the above discussed profiles can be implemented and integrated into existing health care systems. The upcoming sections deal with the integration profile for Patient Synchronized Applications and how this profile can be integrated and implemented.

6.2 Definition of a PSA

A PSA in terms of a medical health care integration profile is defined by the IHE International, Inc. [44] as follows:

> **Patient Synchronized Applications**
> a means for viewing data for a single patient using independent and unlinked applications on a user's workstation, reducing the repetitive task of selecting the same patient in multiple applications. [64]

PSAs allow users the selection of a patient in one application which causes all other running applications of this user (even on other workstations) to tune to the same patient. This not only reduces the work of manually selecting the same patient again but enhances the quality of the electronic data, since the failure of viewing and editing a different patient is minimized. Moreover, integrating PSA profiles into health care software is essential for multi system environments when dealing with several PCs running health care applications which have to be synchronized [66].

6.3 Features

Patient Synchronized Applications enable a so called 'single patient selection' for the user working on several applications or workstations [64]. Figure 6.2 shows the basic principles of the PSA integration profile, which recommends to use the HL7 Clinical Context Object Workgroup (CCOW) standard to integrate such systems. The CCOW are interoperability specifications for visual integration of applications that allows users to experience an integrated computer-user session on the desktop, meaning the automatic coordination and synchronization of disparate healthcare applications. Such applications enable the user to set a certain clinical context for a session. Every other application joining such a context is then automatically tuned to the same context. [67].

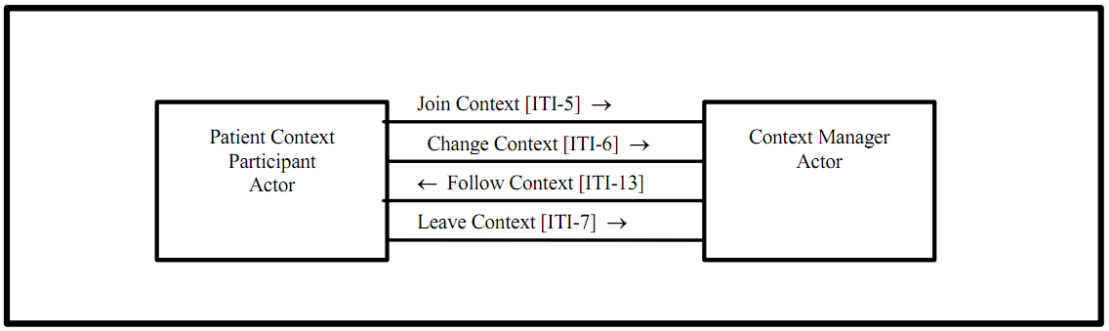

Figure 6.2: Patient Synchronized Applications Profile Actor Diagram (taken from [64], Figure 6.1-1)

As shown in Figure 6.2, a PSA should support four transactions in order to follow the recommendations according IHE [68]:

1. **`JoinContext`**: a Patient Context Participant (PCP) locates a responsible context manager instance, which is the server running the actual context maintaining all the necessary context and session information. If a context manager is successfully located, the PCP intends to join a context. The context manager assigns a unique context participant identifier to the PCP which is then used by the PCP and the context manager until the context session is terminated (either by the context manager or the PCP). After the context session is up and running, the actor shall periodically verify availability of the context manager by pinging it or sending is-alive packages. If a PCP joins an already running context which is tuned to a specific patient, an immediate response to the `JoinContext` method is the `FollowContext` invocation handing the current patient information to the just joined client.

2. **`ChangeContext`**: a client invoking this method on the context manager forces all the participants of the current context session to synchronize their values based on the new context values from the calling client. According to the HL7-CCOW standard, this method consists of several other transactions: a phase instigating the change, a phase surveying the other participants if such a change is possible now, and finally the publishing phase where the decision of the change is triggered to the other participants by invoking their `FollowContext` method. Participating clients return an acknowledgment for the changed data after receiving this call. [68]

   During the phase of surveying the clients it is possible that applications cannot accept the change immediately. This could happen because the user is active and working on a patient record. If this happens, the application will send a conditional acceptance message back to the server, indicating that the user can decide to follow the context or not. The user can suspend the context participation, cancel the pending change or continue with the change which will result in a context change. If the participant application does not respond after a certain period of time, the application will be considered as busy. Such

busy applications should be treated as applications, suspending their context change. As soon as they are responding again, a change is triggered for them as well to prevent them from being out of sync.

3. **`FollowContext`**: as for the `ChangeContext` method, this transaction consists of multiple phases too: surveying the participants, notifying them of the decision whether the context changed or not and informing them about the new context parameters, meaning the new patient information to tune to. After finally receiving the new information, the client tunes to the desired new patient information.

4. **`LeaveContext`**: this method is used to inform the context manager of a planned leave of the context. The context manager acknowledges the receipt of the message and disposes all relevant information stored about the leaving client in the current session. The leaving participant will not get any further notifications about changing context information anymore until he re-joins the context by invoking the corresponding `JoinContext` method again.

The following figure (Figure 6.3) illustrates the above mentioned methods in a workflow. PCP 1 joins the context on the context manager by sending a `JoinContext` message. After a while PCP 1 selects patient A and triggers the `ChangeContext` method. After the change, PCP 2 joins the context and immediately gets a `FollowContex` notification, since the context is already tuned to patient A. PCP 2 tunes to patient A automatically. PCP 2 then selects patient B triggering the `ChangeContext` method and additionally the notification of PCP 1 by the `FollowContext` method. PCP 1 would now tune to patient B as well but seems not to accept the change because it is already closing the application. A `LeaveContext` method is triggered to the context manager indicating, that this client is not available anymore. After a while, PCP 2 leaves the context as well.

The concept of PSAs is used heavily in the implemented demo HIS presented in Chapter 3. More details about the specific PSA implementation itself can be found in Section 5.2.

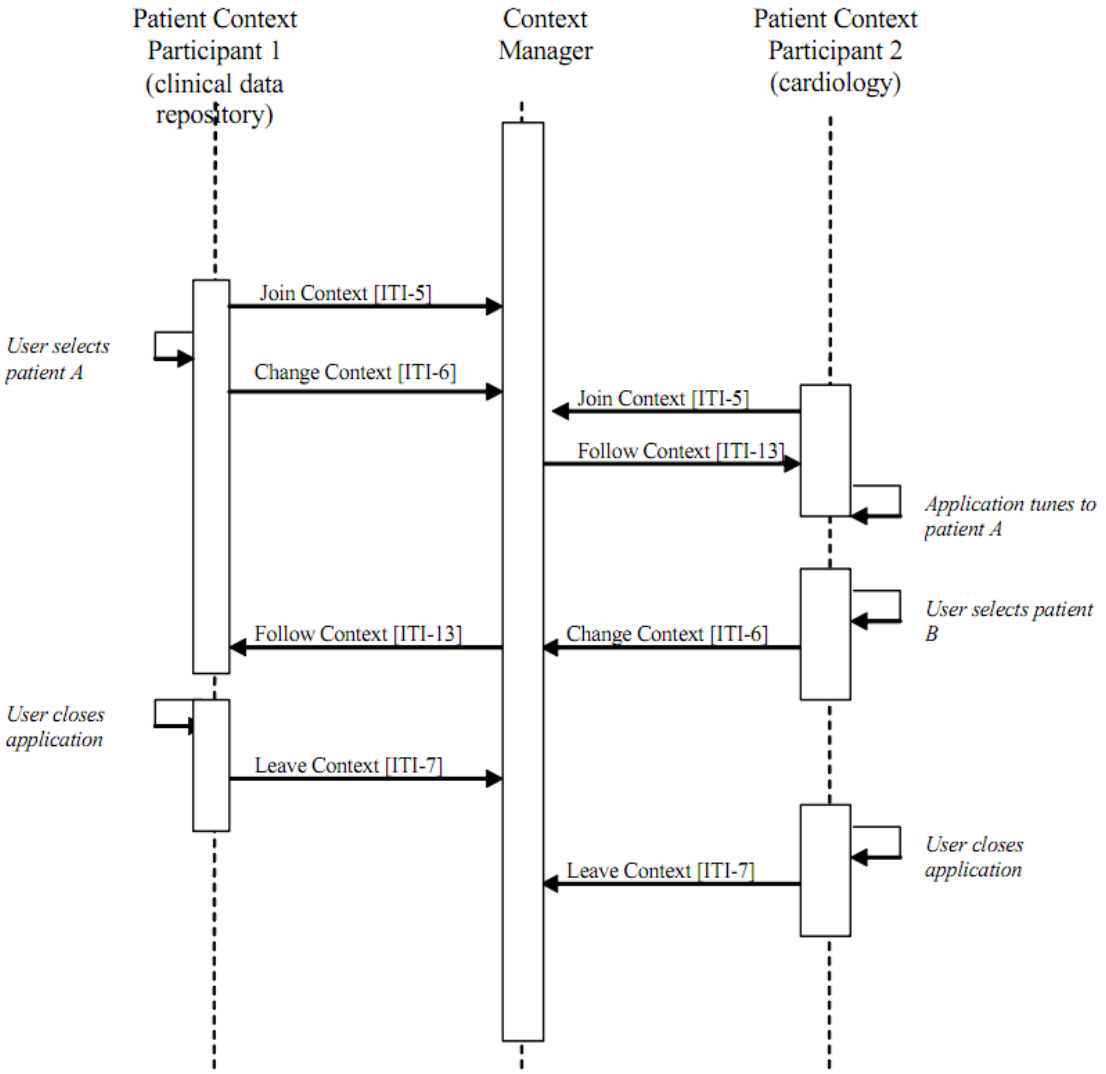

Figure 6.3: Simple Patient Switching Process Flow (taken from [64], Figure 6.3-1)

CHAPTER 7

HIS: Possible Extensions and Future Work

The chapters as presented before provide an overview as well as detailed information on the whole implemented HIS: the back-end services, the front-end services and user interface implementation of the actual applications. Although the presented work is coming up with a quite matured architectural design and implementation, there are some possible extension, especially for the front-end services. This chapter presents some of these possible extensions and discusses the benefit of them.

7.1 Dispatcher Services for Clients

As especially discussed in Chapter 5, service calls from clients to services on the MedView Server instance are performed asynchronously. So far, some of the most important service calls are thus dispatched to be executed in a different thread, hence not leading to a blocked application in the meantime.

The same applies to incoming messages on clients sent from the server via the PSA infrastructure. Calls have to be dispatched in order to build good non-blocking and responsive

user interfaces. The introduction of dispatching services on the client layer for incoming and outgoing messages and service calls would enhance the user interface experience.

By the use of a queue for messages and service calls, blocking calls could then just be enqueued and sent asynchronously in background. Critical operations in the user interface thus would not be blocking anymore.

7.2 Services using Pipelines and Queues

So far, incoming and outgoing messages on the MedView Server instance are directly processed, without any kind of buffering. When performing performance critical tasks on the server, the server might not be as responsive as expected then, thus, service calls from clients might time out or incoming callback messages with results from previous service invocations might get lost.

Moreover, if clients are not reachable (e.g. they are offline for a short period of time due to network problems), outgoing messages via the PSA system might get lost, the PSA context thus might not be in a consistent state.

The usage of message queues and pipelines for incoming and outgoing messages would help to minimize these problems. Since the server and its services are built using the WCF technology, some of the out-of-the-box solutions would be to enable 'NamedPipes' or the Microsoft Message Queuing (MSMQ) services. Both of them are available on the WCF system. Other possibilities would be to use third party libraries such as 'zeromq' [1] which provide interfaces for various platforms and programming languages.

Problems involved in the usage of queues and pipes are problems of concurrency and currently offline and therefore not reachable clients. Take for example two clients being part of a PSA context, one of them switching to another patient, the other one just being not reachable. Due to the queues and pipelines the message thus might reach the offline client some time later but might not be accurate anymore, since the other client already has switched to another patient. Good algorithms and protocols need to be introduced to handle such problems.

[1] `http://www.zeromq.org/`, last accessed 24.07.2011

7.3 Message System and Serverless Conferencing of Clients

Since clients are already connected together via the central MedView Server and can receive messages from this server, the question might arise, if a messaging system only between clients could be implemented. Such a system might enable a serverless conferencing between clients and the exchange of small messages between them, without having the server involved all the time.

With the use case, that two physicians are signed on on different terminals, physician A then could send an 'invitation' to physician B to help him diagnose a certain problem. This would be a small message between the two distinct clients allowing physician B to have a (probably) read only look at the patients data to help A diagnosing the problem. This would be just one possible use case for a serverless conferencing of clients and a message system between them.

7.4 Workflow Integration for openEHR

The so far existing openEHR specification in its revision 1.1. does not consider a workflow management, however, it states that the Workflow Information Model will be part of future releases. According to Beale et al. [12] a "workflow is the dynamic side of clinical care, and consists of models to describe the semantics of processes, such as recalls, as well as any care process resulting from execution of guidelines".

Currently, a lot of work is going on in the openEHR community about workflows[2]. In her work Barretto presents possible solutions for workflow integrated EHRs [69].

Sam Heard[3] in his work about work flow definitions in openEHR shows how to use openEHR `INSTRUCTION`s and `ACTION`s to model flows.

As shown in Figure 7.1, a workflow can be defined using the openEHR model as is. The presented front-end could be extended with a hook upon such defined workflows, since it can analyze all loaded and saved data. An event system in background can then notify corresponding actors about upcoming work to do via the PSA infrastructure.

[2] `http://www.openehr.org/shared-resources/publications/workflow.html`, last accessed 24.07.2011

[3] `http://www.openehr.org/wiki/display/spec/Instructions+and+Actions+-+work+flow+in+openEHR`, last accessed 24.07.2011

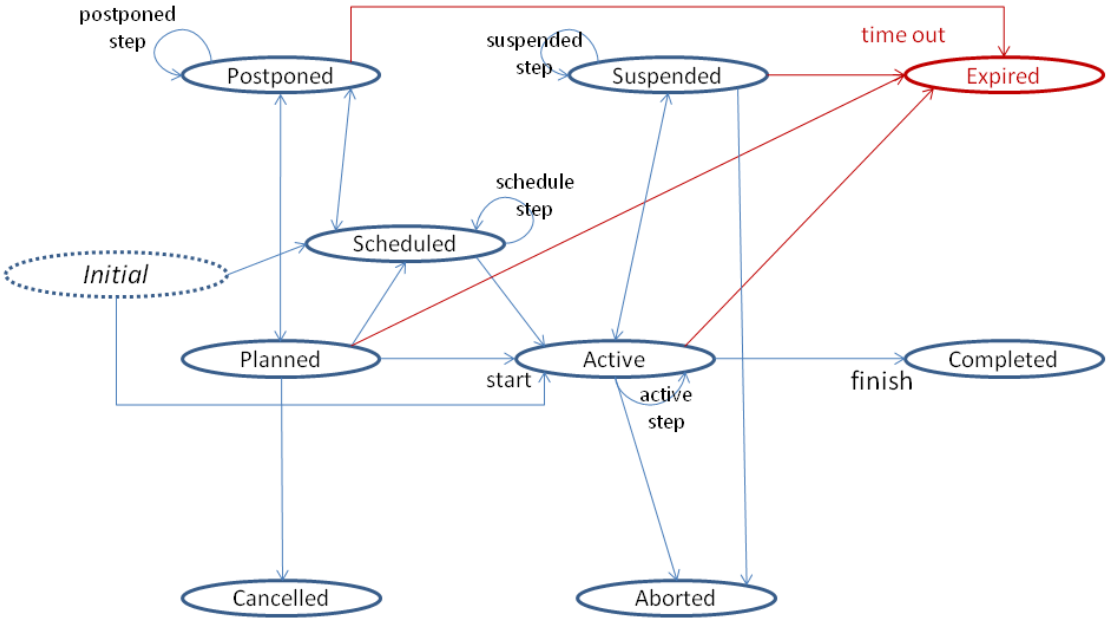

Figure 7.1: State machine defining a possible solution for work flows of an instruction (Source: Sam Heard, Instructions and Actions - work flow in openEHR)

7.5 Extension to the Patient Synchronized Application Concepts

The original concepts of PSAs allow users to follow context switches to other patients and therefore ensure and enhance the quality of care, since wrongly entered data for wrong patients is avoided respectively minimized. But the original concepts of PSAs can also be taken to enhance other parts of the medical health care as well. This section briefly mentions some possible further work and extensions for the PSA concepts.

Integrating PSAs with PIX Applications

When health care systems are integrated into an existing environment, Patient Identifier Cross-referencing (PIX) (see Section 6.1) systems are necessary to solve the issues when referencing patient data from other existing domains. Since the PSA concept also relies on referencing patients to change the context of applications, PSA integration profiles and the PIX system must identify patients in a constistent manner. Therefore, the context manager of the PSA and the PIX manager have to be grouped together to do a common patient mapping. Figure 7.2 shows this

concept. To integrate the PIX manager into the system, the Patient Mapping Agent (PMA) API of the context manager is used to access the patient ID consumer which acts as front-end for the PIX manager. This allows to access the same patient identifiers from within the PSA context as well as from within the PIX context. [68, 70]

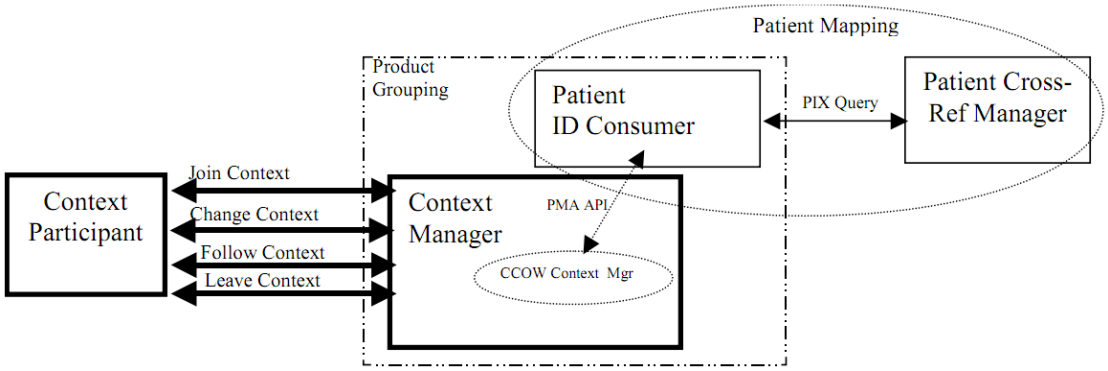

Figure 7.2: PSA and PIX Actor Grouping Diagram (taken from [70], Figure D-1)

Supporting Clinical Pathways

Applications running a patient context and connected to a common context manager have the ability to make further use of this concept and extend the concepts of the PSA to support clinical pathways.

Clinical pathways are defining optimal sequencing and timing of interventions by physicians, nurses, and other staff for a particular diagnosis or procedure. They are developed to support the people involved and to improve the quality of patient care. [22, 71]

Furthermore, clinical pathways can not only enhance the quality of care itself but reduce the length of hospital stay and save money involved in medical health care accordingly [72].

PSA systems would offer a good and stable background to implement a clinical pathway into systems since clients (PCPs) are connected to a common server. With the common context manager serving as clinical pathway master, this server could distribute messages from clients to other clients and inform them about upcoming examinations of patients.

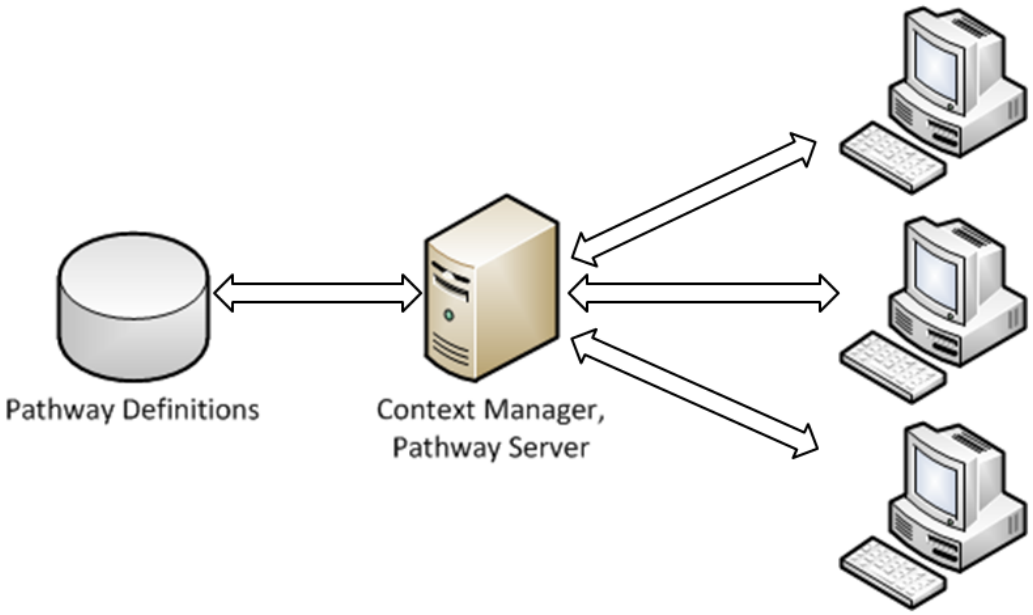

Figure 7.3: Pathway server

Figure 7.3 illustrates this concept. The context manager and pathway server has access to a database defining clinical pathways. Such clinical pathways are defined in a way that every single step has a clear successor which could be notified once an action is completed by one party (physician, nurse, etc.). Clients connected to such a server would then get notified of upcoming actions and examinations. This could speed up health care processes, since physicians already can have a look at the patients data while the patient herself is still on her way.

Having such a system integrated into an openEHR environment would even offer the opportunity to make use of the defined archetypes, since openEHR allows to model workflows by the use of the ACTION and INSTRUCTION archetype. A system controlling the workflows for existing electronic health records could improve clinical pathway definitions and make them more abstract since they are defined in archetypes.

End

CHAPTER 8

Results and Conclusion

In the area of hospital information systems, usually many different approaches for storing electronic health records are present. Whereas most of them are built upon a fixed and not flexible relational database schema, some are dealing with archetype based records and standards. Although the implementation overhead for archetype based systems is much higher than for systems based on relational database schemes, it is worth designing and building such systems. Not only are they better to maintain and to extend but moreover are they possible future-proof systems.

It is not only unavoidable to switch to such archetype based standards to ensure maintainability and future-proof approaches but also to enhance the interoperability between existing standards, since archetype based standards better support both, semantic and structural interoperability. Hence, data to be exchanged does not lose information. From the point of a user interface view such standards are hard to implement, since there is no fixed user interface and therefore, complex engines and frameworks have to be build in order to ensure appropriate applications and usability.

It has been shown that the usage of the archetype based standard openEHR is possible in this prototype implementation. The back-end responsible for storing and retrieving medical data is completely build upon this standard. An archetype enabled front-end was build to allow the rendering and visualization of such data.

As a proof-of-concept it has been shown that the integration of a patient synchronized application within such an existing environment is possible and brings benefits with it. It supports the clinicians in their all day work by minimizing the effort of constantly selecting patients within applications and it minimizes the errors made due to a diagnosis stored in a wrong patient's record.

8.1 Conclusion

The motivation for this work was to extend a given integration architecture running in the back-end and make use of the provided services. Furthermore, the implementation of a future-proof archetype based front-end application based upon this back-end was being focused.

From the back-end side of view, the usage of the existing integration architecture took away a lot of implementation work and allowed to focus on different parts, thus leading to a more sophisticated front-end architecture especially in the areas of the patient synchronization application architecture.

The front-end as being developed upon existing frameworks, extending and refining them proofed that the implementation effort of archetype based user interface generation is much higher than for standard software, but totally worth it. After having introduced a good and capable framework in background, it has been shown that the extension of this framework to new graphical user interface elements can be done rather easily.

Also the patient synchronized application approach was proven to be a valid and good working implementation. By using a well established framework for providing interfaces to different platforms and technologies, the effort of connecting to the existing patient synchronized application context is kept at to a minimum.

All in all it has to be considered that this work has been carried out as a proof-of-concept, thus not addressing every single technical topic. Some parts have been implemented as dummy implementations to show the whole picture and prove that the architectural design is working. The future work section (see Chapter 7) outlines many possible extensions to the existing back- and front-end which could lead to a much more reliable and maintainable allover architecture.

Bibliography

[1] Bernd Blobel. Comparing EHR Models: openEHR, HL7 V3 Specs, EN/ISO 13606, CCR. Technical report, eHealth Competence Center, University of Rgensburg Medical Center, Franz-Josef-Strauß-Allee 11, D-93053 Regensburg, Germany.

[2] B. G. M. E. Blobel, K. Engel, and P. Pharow. Semantic Interoperability: HL7 Version 3 Compared to Advanced ArchitectureStandards. In *Methods Inf Med*, pages 343–353, Regensburg, Germany, April 2006.

[3] D. Kalra. Electronic Health Record Standards. In *IMIA Yearbook of Medical Informatics*, pages 136–144. IMIA and Schattauer GmbH, 2006.

[4] Oliver Johannes Bott. "The" Electronic Health Record: Standardization and Implementation. In *2nd OpenECG Workshop*, pages 57–60, Berlin, Germany, 2004.

[5] A. Begoyan. AN OVERVIEW OF INTEROPERABILITY STANDARDS FOR ELECTRONIC HEALTH RECORDS. *Integrated Design and Process Technology*, June 2007.

[6] Health Level Seven International. Clinical Document Architecture. http://www.hl7.org, last accessed: July 2011.

[7] European Committee for standardization. Health informatics - electronic health record communication - Part 1: Referencemodel, February 2007.

[8] P. Schloeffel, S. Heard, D. Kalra, D. Lloyd, and T. Beale. OpenEHR - Introducing openEHR, 2006.

[9] Asuman Dogac, Gokce B. Laleci, Thomas Aden, and Marco Eichelberg. Enhancing IHE XDS for Federated Clinical Affinity Domain Support. *IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE*, 11(2):213–221, March 2007.

[10] International Organization for Standardization. Health informatics - Electronic health record - Definition, scope and context. http://www.iso.org, last accessed: July 2011.

[11] Peter Schloeffel, Thomas Beale, George Hayworth, and Heather Heard, Samand Leslie. The relationship between CEN 13606 , HL7 , and openEHR. 7.

[12] T. Beale and S. Heard. OpenEHR - Architecture Overview, 2008.

[13] S. Garde, P. Knaup, E.J.S. Hovenga, and S. Heard. Towards Semantic Interoperability for Electronic Health Records. *Methods Inf Med*, 46:332–343, 3 2007.

[14] M.L. Simoons, N. van der Putten, D. Wood, E. Boersma, and J.P. Bassand. The cardiology information system: the need for data standards for integration of systems for patient care, registries and guidelines for clinical practice. *European Heart Journal*, 23(15):1148–1152, 2002.

[15] Keith J. Dreyer. *PACS: A guide to the digital revolution*. Springer, 2006.

[16] H. K. Huang. *PACS and Imaging Informatics: Basic Principles and Applications*. John Wiley & Sons, 2010.

[17] Mattias Forss, Erik Sundvall, and Mikael Nyström. openEHR related Knowledge Tool Project by Linköping University.

[18] Seref Arikan, Dr. Tony Shannon, and Professor David Ingram. Opereffa - openEHR REFerence Framework and Application. http://opereffa.chime.ucl.ac.uk, last accessed 12.06.2011, 6 2009.

[19] Koray Atalag et al. GastrOS - openEHR based Endoscopy Application. http://gastros.codeplex.com, last accessed: 12.06.2011, released December 2010.

[20] Atalog K, Yang HY, and Warren J. On the maintainability of openEHR based health information systems - anevaluation study in endoscopy. In *18th Annual Health Informatics Conference*, pages 1–5, Melbourne, Australia, 2010.

[21] Atalag K and Yang HY. From openEHR Domain Models to Advanced User Interfaces: a Case Study inEndoscopy. *Wellington*, 2010.

[22] Peter Haas. *Medizinische Informationssysteme Und Elektronische Krankenakten*. Springer, 2005.

[23] EH Shortliffe. The evolution of health-care records in the era of the internet. *Medinfo*,

9:8–14, 1998.

[24] Ragnar Nordberg. EHR in the Perspective of Security, Integrity and Ethics. *Medical and Care Compunetics*, 3:291–298, 2006.

[25] Murat Gök. Introducing an openEHR-Based Electronic Health Record System in a Hospital. Master's thesis, Georg-August-Universität Göttingen - Abteilung für Medizinische Informatik- Universitätsmedizin Göttingen, 37083 Göttingen, Deutschland, May 2008.

[26] Peter Schloeffel. Current EHR Developments: An Australian And International Perspective. *Health Care and Informatics Review Online*, 8(3), September 2004.

[27] Lori Rosmus. Electronic Health Records: Benefits and Drawbacks. *University of Alberta Health Sciences Journal*, 2(1):38–40, May 2005.

[28] Nir Menachemi and Taleah H Collum. Benefits and drawbacks of electronic health record systems. *Risk Management and Healthcare Policy*, 4:47–55, May 2011.

[29] K. A. Kuhn and D. A. Giuse. From Hospital Information Systems to Health Information Systems. *Method Inform Med*, 40:275–87, 4 2001.

[30] Fitzpatrick NK and Shah S and Walker N and Nourmand S and Tyrer PJ and BarnesTR and Higgitt A and Hemingway H. The determinants and effect of shared care on patient outcomes and psychiatricadmissions - an inner city primary care cohort study. *Soc Psychiatry Psychiatr Epidemiol*, 39(2):154–163, 2004.

[31] Wang SJ, Middleton B, Prosser LA, Bardon CG, Spurr CD, CarchidiPJ, Kittler AF, Goldszer RC, Fairchild DG, Sussman AJ, KupermanGJ, and Bates DW. A cost-benefit analysis of electronic medical records in primary care. *Am J Med*, 114(5):397–403, 2003.

[32] Harris MF, Priddin D, Ruscoe W, Infante FA, and O'Toole BI. Quality of care provided by general practitioners using or not using division-baseddiabetes registers. *Med J Aust*, 177(5):250–252, 2002.

[33] Zurita L and Nohr C. Patient opinion: EHR assessment from the users perspective. *Stud Health Technol Inform*, 107(2):1333–1336, 2004.

[34] Westin AF. Public attitudes toward electronic health records. *Privacy and American Business*, 12(2):1–6, 2005.

[35] Health Level Seven International. HL7 - About. http://www.hl7.org, last accessed: July

2011.

[36] Robert H Dolin, Liora Alschuler, Calvin Beebe, Sandra Lee Biron, Paul Vand Boyer, Daniel Essin, Elliot Kimber, Tom Lincoln, and John E Mattison. The hl7 clinical document architecture. *Journal of the American Medical Informatics Association*, 8(6):552–569, 2001.

[37] Health Level Seven International. HL7 Standards. http://www.hl7.org, last accessed: July 2011.

[38] George W Beeler. HL7 Version 3 - An object-oriented methodology for collaborative standardsdevelopment. *International Journal of Medical Informatics*, 48(1):151–161, February 1998.

[39] S. Gaion, S. Mininel, F. Vatta, and W. Ukovich. Design of a domain model for clinical engineering within the HL7 ReferenceInformation Model. *Health Care Management (WHCM), 2010 IEEE Workshop*, pages 1–6, Feb 2010.

[40] Barry Smith and Werner Ceusters. HL7 RIM: An Incoherent Standard. *Studies in Health Technology and Informatics*, 124:133–138, August 2006.

[41] Jeffrey M Ferranti, R Clayton Musser, Kensaku Kawamoto, and Ed Hammond. The Clinical Document Architecture and the Continuity of Care Record: ACritical Analysis. *Journal of the American Medical Informatics Association*, 13(3):245–252, May / Jun 2006.

[42] Robert H Dolin, Liora Alschuler, Sandy Boyer, Calvin Beebe andFred M Behlen, Paul V Biron, and Amnon Shabo. HL7 Clinical Document Architecture, Release 2. *Journal of the American Medical Informatics Association*, 13(1):30–39, Jan / Feb 2006.

[43] S.K. Andersen et al., editor. *IHE based Interoperability - Benefits and Challanges*. Organizing Committee of MIE 2008, ISO Press, 2008.

[44] IHE International Inc. Integrating the Health Care Enterprise. http://www.ihe.net, last accessed: July 2011.

[45] Christopher D Carr and Stephen M Moore. IHE: a model for driving adoption of standards. *Computerized Medical Imaging and Graphics*, 27(2):137–146, March 2003.

[46] IHE International Inc. IHE Profile Design Principles and Conventions. http://wiki.ihe.net/index.php?title=IHE_Profile_Design_Principles_and_Conventions,

last accessed: July 2011.

[47] Thomas Beale. Archetypes: Constraint-based Domain Models for Future-proof InformationSystems, 2002.

[48] T. Beale, S. Heard, D. Kalra, and D. Lloyd. OpenEHR - Support Information Model, 2008.

[49] T. Beale, S. Heard, D. Kalra, and D. Lloyd. OpenEHR - Data Types Information Model, 2007.

[50] T. Beale, S. Heard, D. Kalra, and D. Lloyd. OpenEHR - Data Structures Information Model, 2007.

[51] T. Beale, S. Heard, D. Kalra, and D. Lloyd. OpenEHR - Common Information Model, 2007.

[52] T. Beale, S. Heard, D. Kalra, and D. Lloyd. OpenEHR - EHR Information Model, 2008.

[53] T. Beale. OpenEHR - Integration Information Model, 2006.

[54] T. Beale. OpenEHR - Archetype Object Model, 2008.

[55] T. Beale and S. Heard. OpenEHR - Archetype Definition Language 2.0, 2007.

[56] T. Beale and S. Heard. OpenEHR - The openEHR Archetype System, 2007.

[57] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1996.

[58] Juval Löwy. *Programming WCF Services*. O'Reilly Media Inc., second edition, November 2008.

[59] Adam Freeman and Allan Jones. *Programming .NET Security*. O'Reilly and Associates, Inc, first edition edition, June 2003.

[60] Thomas Claudius Huber. *Windows Presentation Foundation - Das umfassende Handbuch*, volume 1. Galileo Computing, 2008.

[61] Microsoft MSDN. Windows Presentation Foundation, 2011.

[62] Klaus Bayrhammer. Design and Implementation of an Integration Framework for an ElectronicHealth Record Based Hospital Information System. Master's thesis, Vienna University of Technology - Faculty of Informatics, May 2011.

[63] J.K. Zhang and W. Xu. Web Service-based Healthcare Information System (WSHIS): A Case Study forSystem Interoperability Concern in Healthcare Field. In *Biomedical and*

Pharmaceutical Engineering, 2006. ICBPE 2006. InternationalConference, pages 588–594, Dec 2006.

[64] IHE International Inc. IHE IT Infrastructure (ITI) Volume 1 (ITI TF-1) IntegrationProfiles, August 10, 2010.

[65] Antonio Soriano, Daniel Ruiz, Juan M.García, and Carlos A. Montejoand David Gil. A study of standards involved in telemedicine systems.

[66] Yutaka Ando, Masami Mukai, Takumi Tanikawa, Takashi Hongo, Shoji andNakashima, Yasuaki Hayashi, Hiroyuki Sonoda, Shoei Takada, Noriomi Suzuki, Yoshiaki Hayatsu, and Masayoshi Seki. The First Japanese Implementation of IHE-ITI EUA/PSA and the Impactof Visual Integration. In Klaus A Kuhn, James R Warren, and Tze-Yun Leong, editors, *Medinfo 2007: Proceedings of the 12th World Congress on Health (Medical)Informatics; Building Sustainable Health Systems*, pages 2647–2651, Amsterdam, 2007.

[67] CCOW Technical Committee. HL7 Context Management 'CCOW' Standard: Best Practices and CommonMistakes, May 2006.

[68] IHE International Inc. IHE IT Infrastructure Technical Framework Volume 2a (ITITF-2a) Transactions Part A - Sections 3.1 - 3.28, August 10, 2010.

[69] Sistine Ann Barretto. *Designing Guideline-based Workflow-integrated Electronic Health Records*. PhD thesis, Department of Information Technology (Research), University of South Australia, June 2005.

[70] IHE International Inc. IHE IT Infrastructure (ITI) Volume 2x (ITI TF-2x) Volume2 Appendices, August 10, 2010.

[71] Coffey RJ, Richards JS, Remmert CS, LeRoy SS, and Schoville RR andBaldwin PJ. An introduction to critical paths. *Qual Manag Health Care*, 1:45–54, Fall 1992.

[72] Lars-Eric Olsson, Jón Karlsson, and Inger Ekman. The integrated care pathway reduced the number of hospital days by half:a prospective comparative study of patients with acute hip fracture. *Journal of Orthopaedic Surgery and Research*, 1(3), September 2006.