The Person Identification Service as an Enabler of Electronic Healthcare Record Communication

Lukasz Piglowski

A dissertation submitted to the University of Dublin,
in partial fulfilment of the requirements for the degree of
Master of Science in Health Informatics

2007

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work, and has not been submitted as an exercise for a degree at this or any other university.

Signed:_____

Lukasz Piglowski

09th October 2007

Permission to lend and/or copy

I agree that the Trinity College Library may lend or copy this dissertation upon request.

Signed:_____

Lukasz Piglowski

Acknowledgments

I would like to thank my supervisor Damon Berry for all his invaluable advice and expertise during the preparation of this dissertation.
I am indebted to my wife for all her patience.

Summary

Allied healthcare professionals need instant access to up to date patient data from distributed medical systems. Therefore it is expected that patient identification systems will be of benefit to health care workers.

A review of international researches indicates that the person identification systems have a potential to deliver benefits to information systems in healthcare, particularly EHR systems. The use of these systems would have a major impact in four main areas:

1. Improved access to patient records
2. Reduction of medical errors
3. Increased medical workers' satisfaction and efficiency
4. Decreased fragmentation of the existing patient records among distributed systems

This study was undertaken to examine the need of patient identification systems in healthcare domain. The work carried out also explores existing types of identification systems and difficulties in their implementation in healthcare domains.

The thesis also discusses different EHR standards and their usefulness in terms of the Identity services.

The final part of the dissertation focuses on the design study. The design study uses two medical systems with patient demographic records that were implemented in Irish hospitals. It specifically looks at problems involved with quality of data. The design study also implements an abstraction layer that allows the Identity Service to be independent of underlying databases.

Table of Contents

Table of Figures

1 Introduction

The dissertation will present an examination of the potential of patient identification technology in healthcare domain. With continuing developments in patient identification, this technology will increasingly become a part of mainstream healthcare systems. The use of these systems provide for particular benefits in the healthcare domain. These systems may decrease medical errors, delayed treatments or even deaths.

The specifics of this dissertation will look at the quality of patient data and explore the types of errors existing in medical systems. This work also reveals the technological differences between medical systems that utilize patient data leading to the conclusions that the patient identification technology must integrate the disparate IT systems.

Also this dissertation will investigate the importance of the identification systems for the Irish healthcare. It is also intended to discuss the types of the unique identifiers that can be used in the patient identification systems in Ireland.
At this point it is necessary to mention that the ICT Community in healthcare in Ireland is also interested in the adoption of a unique person identifier (UPI) for use throughout the health systems. The bulk of the identified person-centred ICT solutions cannot proceed on an integrated, single health system basis, in its absence. [47]
The National Health Information Strategy[1] has defined the PPSN as the UPI in Health, to be used across both the public and private health sectors. [47]

1.1 Objectives

This study examines whether the examples of patient identification services found in international studies can be applied in Ireland. The thesis also aims to perform a design study to examine the two medical systems that utilize patient demographic records.

[1] National Health Information Strategy, Department of Health and Children, 2003

The aim of this design study is to highlight the problems with quality of data, such as duplicate records, misspellings, similar surnames etc. The patient identification services also need to interact with legacy systems. Therefore it is necessary to enable new technologies to overcome these limitations.

1.2 What is Identity

Every medical system requires an Identity that utilizes patient records - otherwise this system does not have sense of existence. The author believes that the medical systems must provide correct patient identification. Otherwise it may have dangerous impact on patient treatment. For example, the clinicians could not find a medical record for an unconscious patient and they made a wrong decision during the blood transfusion. Missing patient identification in healthcare domains might cost human lives.

The identity also exists in other industries, such as financial and government institutions. The financial institutions have developed identification systems for their customers. These systems help to identify customers and assign transactions to them. It proves to be very useful to detect credit card frauds and other suspicious financial transactions.

1.3 Introducing the Patient Identification System

The author believes that it is highly desirable to implement systems that could improve patient identification and communication between hospitals and healthcare institutions in nationwide perspective. It happens because the mobility of the population, such as changes of residence, job or tourism has significantly increased. Therefore, the medical workers need the patient identification systems which would retrieve information about past treatments.

These systems are required to collect and correlate healthcare records and support patient identification within disparate systems within a single enterprise and/or across a set of collaborating enterprises. Hospitals and clinics or even their separate departments create identification systems and assign IDs that uniquely identify patients for their internal purposes, but these IDs are meaningless outside organizations. [2]

At the beginning, the author briefly needs to explain basic terms that are involved with patient identification systems. The first term is an Identification that is defined as the process of matching a set of qualities or characteristics that uniquely identifies a person. [3] It might be also part of our daily live when we identify our family members, or friends, by their physical properties, such as voice, face or other characteristics. [4] The second term is an ID. ID is a sequence of characters that one or more systems in one ID Domain (see chapter 2) use to represent a person and bind related information. This could be numeric, alpha, and may include punctuation, etc. [5]. An example of ID is Medical Record Numbers (MRN). Healthcare professionals often use MRN as a method of identifying patients and their medical information. The numbering system including the content and format of the medical record number is usually specific to the individual organization. It means that MRN does not support access among multiple organizations and patients will be required to use MRN when dealing with different medical organizations. Health organizations have systems, which utilize MRN. For example Hospital Information Systems (HIS) provides registration, identification, and demographics services. Furthermore hospitals utilize systems named ancillary systems, which may use IDs from the registration systems. These systems would typically only know about a small population of the IDs in the ID Domain (see Chapter 2) at any point in time. Examples of these could be Laboratory Information Systems (LIS), Radiology Information Systems (RIS), scheduling systems, monitoring systems, financial systems, pharmacy systems, etc. These systems perform specific functions and use the IDs, as opposed to managing the IDs and ID information. [6]

It is instructive at this point to mention about the basic outline for the patient identification system. These systems should permit medical users to submit a search for patients' medical record using some combination of identifying parameters such as surname, name, date of birth and gender. Furthermore medical professionals also need to have the permissions to prepare patient search or request patient details from qualified staff in other location. [2]

The identity services are required to meet the changing needs of business and users. Their standard set of interfaces should allow medical applications to change identification service provider without impacting the existing applications. The common framework for person identification systems utilized in distributed medical environments is able to save costs for clinical, financial, and laboratory departments. It may also prove to reduce costs for medical insurance industry in the process of ensuring accurate care of the correct patient. For patients, it can improve quality of care across multiple providers and care organizations. [2]

We also discover some scenarios when identification service might be useful in real healthcare environment. For example: An unconscious patient were admitted to ER by paramedics but the patient's healthcare card with a unique identification number on it could not be found, but they had bills and notes in their pocket showing address and different telephone numbers. ER needs to determine whether or not the patient is facing chronic disease. The EMPI system can use available patient details and search across internal and external systems to find and link electronic records for particular patient. Other example of the use of patient identification service could be a need to search for prescription records to determine for instance; medication that patients need to take, diagnoses and undergone surgeries. MPI solutions provide patient information in seconds, that allows to make better decisions and save human lives. [1]

2 ID Domains and Master Patient Index

This section explains terms involved in the identification process. Firstly, the author needs to describe an ID Domain term. The ID Domain is a set of identifiers (IDs). For example; hospitals usually have implemented systems such as Healthcare Information System (HIS). This system can manage ID Domain what means that medical system creates IDs for patients as they entered into the system. A typical ID Domain has a domain name, which uniquely identifies it from other ID Domains. Healthcare organization can also have multiple hospitals and clinics that have implemented their own identification to make them each a separate and independent ID Domain. [5]

People can have an ID from many ID Domains and they can be assigned more than one ID, but a single ID should not be assigned to two patients. [7] Ideally there should be only one ID per person, but in production medical systems, there might be duplicated patient records. It means that a particular patient has two or more IDs in the same ID Domain. For internal consistency, the ID Domain cannot assign two persons with the same ID otherwise it is difficult to distinguish between the two entries and might cause serious errors. [5]

Multiple systems also can be located in one ID Domain and they can reference person IDs from the same ID Domain. For example, a lab system and a billing system in hospitals are part of the same ID Domain and can use the same medical record numbers to identify people. [5]

Additionally patient ID, together with its ID domain name, can create a globally unique ID then ID Domain can also be defined as Master Patient Index (MPI).

The idea of Master Patient Index is compared to a bridge that is created to enable data interoperability and information access across healthcare networks. The building blocks for this bridge are data interoperability and access. [1]

These systems correlate over multiple ID Domains but use common demographic information or traits used to help to identify a particular patient, such as, name, address, MRN, date of birth etc. [8]

Additionally Soloman *et al* distinguish two MPI systems. Apart from organizational MPI that is unique only within one healthcare organization, there is second type called wide-enterprise MPI. EMPI system brings multiple cooperating healthcare organizations together and requires interoperability among them. The EMPI provides cross-reference to the multiple specific MPIs so that a patient's information can be accessed across the enterprise based on the patient's identifier. [9]

In conclusions, we state that available EMPI technology has evolved significantly and does not require healthcare organizations to replace existing systems or databases, including older technology, which often cannot accept new identifiers, resulting in certain records and systems being left out of the e-health solution. Apart from that, EMPI systems have abilities to link records without changing or moving them. Allied professionals can utilize EMPI technology to enable the search and retrieve of all relevant patient data across systems – including newly developed environments, legacy applications and data sets of questionable completeness and quality.

2.1 Limitations, challenges, risks

The author needs to clarify PIDS system term that is used in this thesis. This term is used to describe prototype systems that might utilize multiple Mater Patient Index (MPI) systems to search, correlate and edit patient IDs. It is also important to distinguish names - PIDS systems are not MPI systems. The author believes that the PIDS systems should be integrated with MPI systems.

The Patient Identification service should have the ability to search and link patient records among distributed MPI systems. Although it is believed that PIDS systems may also store patient records, it is done only for identification purposes.

Next we must consider risks and obstacles in MPI systems. Medical systems that utilize and store patient records are changing with high frequency thus these systems need to enable interoperability and flexibility to meet requirements for constantly changing environments.

It is very likely that data integrity issues between MPI systems will be revealed when hospital departments or different organizations need to utilize the PIDS systems.

Healthcare organization can manage many dependent sites such as ER, inpatient and outpatient departments. All these departments might have different structure of medical records. MPI systems located in different locations might have separate numbering systems to index patients; they also can use different identifying elements or traits. Additionally they might have different formats of identification elements such as surname, MRN and date of birth. MPI systems also can use different matching processes to obtain record number. [11] The similar problems might be revealed when the MPI systems need to be integrated with legacy systems. These systems still work in healthcare environment and contain valuable historical, patient data. Other limitations might exist when healthcare provider still utilizes paper patient records. The paper records have no possibilities to be integrated with electronic systems thus the paper systems should be replaced by electronic systems.

It is also highly probable to have duplicate records in MPI systems. The MPI systems need to match records that have some identical identifiers, but apart from that they must identify possible matches when the records' identifying data are inconclusive. Therefore a human intervention is required to verify the data propriety. [1] The author explains that duplicate medical records error rates are significant and data quality issues should be prioritised during implementation of PIDS systems. Between January 2000 and December 2003, Initiate Systems Company wanted to estimate MPI error rate. For these purposes they analysed more than 300 master person index files at facilities across the country (United States). They estimate MPI error rate as the total number of records implicated in split records (adjusted for potential invalid linkages) divided by the total number of medical records included in the file evaluation. The results show that the current state of patient demographics data stored within systems is quite poor. A typical MPI file, the central patient system of record within a medical facility, has error and duplication rates around 8 percent and can rise higher than 20 percent. The problem with quality of data is even more significant when EMPI systems are

used in large healthcare organization. The report suggests that the number of records with errors in distributed systems is nearly 10 percent and can reach as 40 percent. For example: if single MPI system utilizes 500.0000 medical records, with an average 8 percent error rate then it indicates that approximately 40.000 records are duplicated or error-ridden. For EMPI systems that can be deployed for national wide with 3 millions medical records, even if error rate is estimated for single MPI then 500.000 people would have errors with their records. [10]

The author believes that all these problems need to be discussed before PIDS systems are implemented in hospitals.

Additionally the PIDS systems need to provide adopted security solutions for patient data safety. It needs to manage permission access to view patient data only by authorised medical workers. The permissions should be assigned to right persons who are supposed to see it at a particular point in time because wrong permissions assignment can cause problem with misdiagnoses or use confidential data to other purposes than supposed. [1] The author believes that the PIDS systems should have implemented solutions that could manage appropriate protections against misuse, detections of fraud and attempts of unauthorized access inside and outside of healthcare systems.

3 Approaches That Require a Unique Identifier

Nowadays, allied professionals such as policy experts, clinicians, administrators, and technologists are interested in developing interoperable health records that improve efficiency and quality of patient care. They started together developing a unique healthcare identifier that is used to identify patients in medical systems. Medical professionals are convinced that deploying an underlying information infrastructure for healthcare is complex process and might cost a lot of efforts but these systems might be very useful and have a lot of benefits for the healthcare world. [1]

In this section, the author discusses main approaches for Unique Identifiers for Individuals [11]. The section contains details about different types of Unique Identifiers. It also explains advantages and disadvantages of different universal identifiers in healthcare environment.

The Health Insurance Portability and Accountability Act of 1996 (HIPAA) outlines a process to achieve uniform national health data standards and health information privacy in the

United States. The Secretary of Health and Human Services (HHS) intends to publish a proposed rule on requirements for a unique health identifier for individuals. The standard is enacted with the widespread support of the industry and required by law. HHS adopts standards to support the electronic exchange of a variety of administrative and financial health care transactions [11].

They agreed that the unique identifier for individuals is an essential component of medical and administrative simplification because it will allow continuity of care, accurate record keeping and also detection of fraud. They also explain that the set of personal attributes commonly used to identify an individual such as: name, birth date, and gender are rarely captured in the same manner by each entity in different healthcare systems.

These systems assign own identifiers to individuals for use within their systems and also traverse organizational boundaries frequently.

They also believe that the unique identifiers for individuals would be useful in ordering tests, reporting results; posting results, diagnoses, procedures, and observations to charts. Additionally, allied healthcare professionals would manage some highly sensitive records in more secure manner. For example: records of mental health diagnoses or treatment, Human Immunodeficiency Virus (HIV) antibody tests, or genetic tests.

The unique identifiers for individuals would also be the critical components of administrative procedures designed to protect such information from inadvertent disclosure. [11]

3.1 Unique Identifier Approaches Based on the PPSN

The author believes that proposals of unique identifiers for individuals proposed by the HHS would be considered as Nationwide Patient Identification System in Ireland. Although the HHS prepared standards for unique identifiers in United States, there are some common features that might be used in Irish healthcare. For example: a usage of two identifiers; The Social Security Number (SSN) [12] and The Personal Public Service Number (PPSN) [13] as unique identifiers. The SSN is a 9-digit number issued to citizens in the United States.

"The SSN was meant to identify the account, not the person. However, in practice, the SSN has been adopted for numerous identification purposes outside of the Social Security system. In 1961, the Civil Service Commission adopted the SSN as an official Federal employee identifier, and in 1962 the Internal Revenue Service adopted it as the taxpayer identification

number. In 1970, financial institutions were required by law to obtain SSNs of all their customers. Although the Privacy Act of 1974 prohibited States from using the SSN without congressional authority, it allowed those already using it to continue. Then, the Tax Reform Act of 1976 authorized States to use the SSN for State and local tax authorities, welfare systems, driver's license systems, departments of motor vehicles, and for finding parents who were delinquent in child support payments."[12]

In Ireland similar functions are assigned to The Personal Public Service Number (PPSN)

PPSN *"is an identifier issued by Client Identity Services, Department of Social and Family Affairs on behalf of the Minister for Social and Family Affairs in the Republic of Ireland. The identifier is currently used for a number of public services including education, health, housing, social welfare and tax however the net is widening but this has not raised concern about functionality creep yet as occurred in other countries. The number is underpinned in legislation by the Social Welfare (Consolidation) Act, 1993 (Section 223) and a number of amendments, including data protection, has expanded its legal use as well as defining improper usage."* [13]

The author bases on information proposed by the HHS and use PPSN term instead SSN to simplify discussion about universal identifiers in this section.

This section describes benefits and limitations of unique identifier based on the PPSN that is often used as an identifier in healthcare systems. It also discusses about the examples of unique identifiers utilized in other countries.

The PPSN would be the least costly identifier to implement because many data systems already use it and it is readily available to most of the public. The PPSN is a unique personal identifier for individuals and it is unlikely to get two people with the same PPSN. It is believed that PPSN is not going to change appropriation in the future. However it is not under control of the health care industry and changes that may be made to benefit one of the many other uses of the PPSN may not be beneficial for healthcare domains.

Some people are not eligible for PPSN like children under 18 or people who receive benefits based on a spouse's earnings might be identified by the spouse's PPSN rather than their own. This could affect the accuracy of records linked using this identifier. [13]

The PPSN is not a national identity and it is designed to be used as a service enabler for the purposes of public service administration. [14]. There are no legal policies about keeping the number confidential or to limit its use. Protection of the PPSN as a health care identifier

would be unenforceable. Also healthcare providers would need to get mechanism to verify the authenticity of PPSN when it is presented as evidence of identity. [11]

In the world there are four countries such as Australia, the United Kingdom, New Zealand and The Netherlands that have implemented national patient identification numbers and they are used for administrative and medical purposes.

For example: In Australia, the number is given to each citizen at birth and to immigrants upon obtaining permanent resident status.

In the United Kingdom, the National Health Service (NHS)[2] number is used for all health related purposes. They use a new content-free number (the number reveals no information about the holder such as: sex, age, place of birth etc). However, to avoid collision problems, this numbers are correlated with sex or date of birth.

The New Zealand Health Information Service (NZHIS)[3] gives every health system user (including tourists) a unique reference number managed in the National Health Index (NHI)[4] database. This number is entirely anonymous and made up of a combination of numbers and letters totalling seven characters. [15]

These examples presents that there are heterogeneous patient identifiers in use among countries and it makes difficulties to aim interoperability between healthcare information systems. The author concludes that the unique identifier (PPSN) is used in many hospital identification systems in Ireland, although this proposal has some limitations, might be helpful to use in conjunction with other patient identifiers such as: surnames, date of birth, sex etc.

3.2 Unique Identifier Approaches not Based on the PPSN.

[2] The National Health Service (NHS), http://www.nhs.uk

[3] NZHIS - New Zealand Health Information Service, http://www.nzhis.govt.nz

[4] National Health Index (NHI), http://www.nzhis.govt.nz/documentation/dictionaries/nhi-dictionary.html

In this section, we discover the unique identifier that is based on Biometric Identifiers. This is one from many recommended solutions proposed by The American National Standards Institute's (ANSI) Healthcare Informatics Standards Board (HISB). [11]

Biometric identifiers utilizes unique physical attributes, including fingerprints, retinal pattern analysis, iris scan, voice pattern identification, and Deoxyribonucleic acid (DNA)[5] analysis. Biometric identifiers are more a class of proposals rather than an individual one, and defined as sophisticated methods of biometric identification. This proposal requires special equipment to scan or read the specific biometric attribute used for the identifier. Biometric identifiers are used by government agencies such as those concerned with law enforcement and immigration. In terms of healthcare problems, biometric identifiers provide an unique identification of patients but they are not widely used in healthcare domains. There is also currently no infrastructure to issue the identifiers or maintain them nationally. Special equipment must be provided to verify the identification of patients, and then it is required to provide training with a special curriculum for medical people. These aspects can generate extra costs to fully implement biometric technology. The patients also must be present when the identifier is issued or verified. It has been estimated that 80 percent of the times when patient records need to be accessed, the patient is not physically present; for example, when the patient telephones the provider for consultation. [11] Some biometric attributes can change due to age, injury, or disease. The biometric identifiers such as fingerprints and deoxyribonucleic acid (DNA) profiles are very sensitive data because contain important patient information. For this reason it might be difficult to prevent linkages that would be punitive or would compromise patient privacy in healthcare organizations. [11]

3.3 Approaches That Do Not Require a Unique Identifier

In this section, the author discuses the patient identification systems that base on the Master Patient Index concept and attempt to address the challenge of locating and linking medical records across organizations or enterprises, without requiring the use of a unique identifier.

---

[5] Deoxyribonucleic acid (DNA), http://en.wikipedia.org/wiki/DNA

These systems are main subject in this thesis because the author believes that these solutions might use patient identifiers such as surname, address, date of birth etc.

The author also believes that implementation of patient identification systems would not require significant changes in existing healthcare systems. These solutions are state-of-art in Irish healthcare domain and might involve new technologies that need to be tested and integrated with the PIDS systems. For these purposes, the author discusses the Person Identification Service (PIDS) specification [5] that might be very helpful to develop these systems in the Irish healthcare. *"Its goal is to provide a standard method of locating person identifiers and thus the patient's associated records across facilities and enterprises, subject to the confidentiality concerns and the right for anonymous care. This provides for a flexible set of traits that can be matched with varying quality as well as providing for correlation of person identification across multiple domains."*[16] PIDS also provides defined interfaces and a person search mechanism that use a combination of identifying parameters for each person [7]. The PIDS specification is discussed in detail in further chapters

4 Introduction to the PIDS specification

The Object Management Group, Inc. (OMG) [17] develops the PIDS specification.

It is an international organization that develops the theory and practice of object-oriented technology for distributed systems. The organization was established in 1989 and cluster over 800 cooperated companies such as: IBM, SAP AG, W3 Consortium and NASA. They work to establish common standards for application development that provide portability and interoperability object-based software for across all major software and hardware vendors. The OMG defines a standard called CORBA [17] - Common Object Request Broker Architecture. Their mission is to provide middleware program that mediates and manages the interaction between disparate applications across the heterogeneous computing platforms and networks. [17] The OMG specifies a healthcare version of CORBA that is called the Healthcare Domain Task Force (HDTF), formerly CORBAmed. [18] They are involved in producing industry healthcare Service Oriented Architecture (SOA) standards. [19]

The most recent version of PIDS specification was released in 2001 and it is formal version 1.1. The PIDS is specified using the Interface Definition Language (IDL) of CORBA. The IDL is a strongly typed declarative language that is programming language-independent. [5]

The PIDS specification is utilized in many areas such as finance, telecommunications, and healthcare. In a case of financial institutions, they exchange customers' information between their distributed systems. These systems support the identification of customers and the correlation of customer identifiers who have received services in different banks. For example customers are allowed to open bank accounts in different banks. These accounts store customer records about account history, credit card information, loan approvals and overdue mortgages. Financial institutions can share this information on demand and gain necessary records to identify and link customers from their federated nationwide or worldwide systems. [5] The author observes some analogues to healthcare because medical systems also utilize and manage patient information that is stored in Electronic Healthcare Records (EHR). [20] The patient records are distributed across a range of heterogeneous systems (MPI systems) located in different departments and institutions. These systems contain multiple patient identifiers and can share this information between hospitals, thus it is needed to develop the systems that link and correlate different patient identifiers correctly between heterogeneous domains even in the absence of a universally unique identifier. [21]

4.1 The PIDS specification – how it works

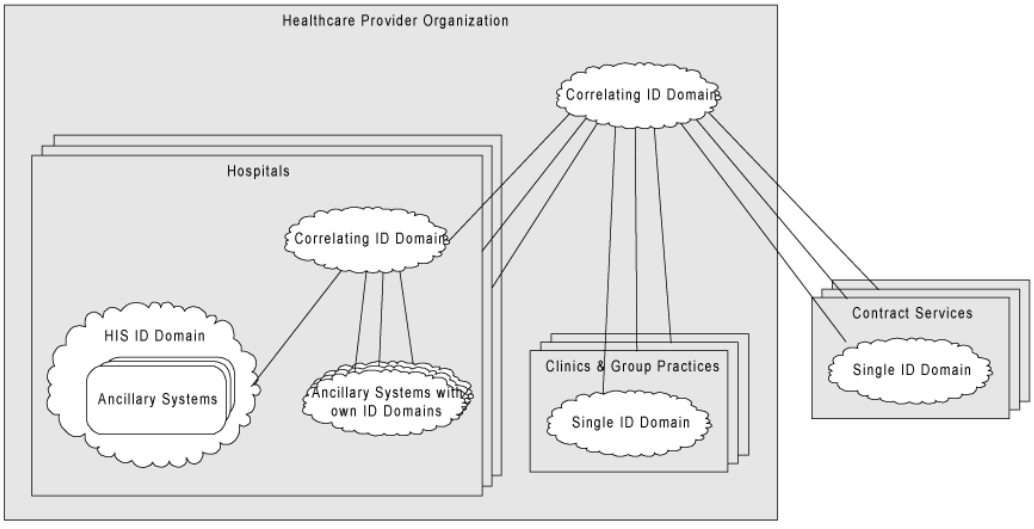


Figure 1: Domain Reference Model for PIDS, *Source: Person Identification Service specification, version 1.1*

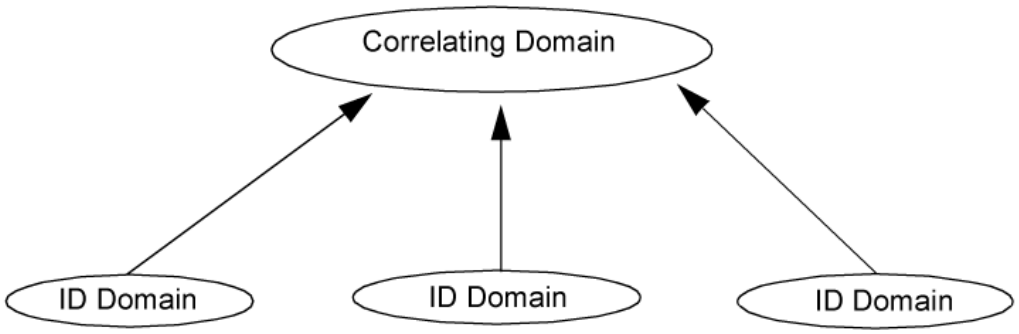


Figure 2: PIDS Identification Model Diagram, *Source: Person Identification Service specification, version 1.1*

This chapter represents a reference model (Figure 1) for healthcare as it relates to PIDS specification. This model highlights the aspects of healthcare related to person identity. The author explains that the PIDS identification model has structural elements such as the ID Domain and the Correlating Domain (Figure 2). The ID domain in the PIDS specification is defined as system or set of medical systems that maintain a unique identifier for each person identity. [5] For example, a lab system and a hospital information system can use the same medical record numbers to identify people. Additionally, a correlating domain is needed when healthcare organisations have more than one ID domain with different unique person identifiers. Thus, it is needed to have a correlating domain that is an ID domain that can cross-reference with one or more ID domains. In a correlating domain, access is allowed to patient data for all of the IDs in the participating ID domains. A correlating domain can correlate both ID domains and other correlating domains. [7]
The author states that the correlating domain term might have the same meaning as the PIDS system term that is used in this thesis.

4.2 The PIDS Interfaces

The PIDS interfaces are used for search and match patients in local and correlated domains but they are not developed to access patient information, like electronic healthcare records. The PIDS interfaces also do not specify information about storing medical data and do not cover predefined names of traits.

The author discusses briefly some interfaces that should be a part of an implementation of the PIDS system. It is also instructive at this point to mention the essential terms to understand the PIDS interfaces such as trait and profile.

A trait it is an attribute that can be used to help identify a person. Examples of a trait include name, date of birth, gender, address, etc. The traits are grouped to create a profile.

A profile is a set of traits containing the person's values for their respective traits. It means that elements of profile can be extended and customized. It also defines the appropriate meaningful compliance levels for several degrees of sophistication, ranging from small, query-only single ID Domains to large federated correlating ID Domains. [5]

4.2.1 IdentifyPerson interface

The IdentifyPerson interface is basically a query used to send traits to be matched in ID Domain or individual system. This interface provides a way to identify a person (find a potential ID) from the traits known about them. The candidate is returned after searching for persons if there are full or partial matches. This interface also includes confidence indication. This indicates how well the stored profile for that person matched the passed-in profile selector. The range of values for the confidence indicator is 0.0-1.0 with 1 meaning maximum confidence (i.e. 100%). The exact semantics of the confidence indicator is determined by the service but it is recommended to have additional guidelines specifying it. The client should be able to compare two returned candidates and determine if they are of equal confidence or determine which has a higher confidence. If the returned candidates have different confidence values, then they are returned in confidence order with the highest being returned first. The confidence value only has meaning relative to the single call it was returned from. There is no standard way to compare confidences returned from different services or from multiple calls to the same service. [5]

4.2.2 ProfileAccess interface

The ProfileAccess interface can be a query or an update used to send an ID for a specific person and to receive that person's profile. This service provides the simplest set of

functionality for any PIDS service. It returns the set of traits known about a person by the service. The interface specifies mechanism for getting profiles for more than one ID at time. It is important since performance of computer network traffic is required. The interface also consists operations how to clear, modify or add the profile of already existing IDs. [5]

4.2.3 IdentifyAccess interface

The IdentifyAccess interface contains objects (Identity) that can provide individual ID level access control via CORBA Security. The specific access control parameters might be available in some security policies applied to an Identity object with the specific object being security-unaware. These policies can be set up to prevent access on an operation/attribute basis. The service must be security-aware in order to implement security policies that vary by the parameters passed in (e.g., separate access control for each trait). The policies governing access to the interfaces are determined by administrative controls and beyond the scope of PIDS specification. [5]

4.2.4 IdMgr interface

The IdMgr interface is an interface providing the core set of functionality for managing IDs in a single ID Domain. It can allocate a unique ID to a specific profile. All the operations on IdMgr are “write” commands as opposed to “read” commands. For this reason, there may be more secure access control to prevent unwanted changes. The access control is managed by the CORBA security service and the implementation of the IdMgr service. There is only one component within an ID Domain implements this interface. If there are multiple actual systems implementing this interface, they should look like a single component. This is the interface implemented by a registration system. Other components within an ID Domain set this attribute to point to the ID manager of the ID Domain, if one exists and they have a reference to it. [5]

4.2.5 CorrelationMgr interface

The CorrelationMgr interface detects which Ids correspond to the same individual in the different sub domains and return this information on request. [8]**.** The Correlation Manager provides a standard interface to correlate and retrieve the patient IDs to enable subsequent exploration of heterogeneous systems and "collection" of scattered clinical data that resides there. [21] The correlation manager interface provides operations such as:

`load_profiles` - causes the profiles to be loaded into the correlation id domain.

`get_corresponding_ids` returns the ids in the destination id domains that correspond to the ID passed in.

`find_or_register_ids` - causes the profiles to be loaded into the correlating id domain from the specified source domains and IDs from the correlating ID domain for each profile are returned. [5]

Forslund et al also base on PIDS specification and use CorrelationMgr interface to enable different solutions to the federation problem. They describe scalable hierarchical approach. They specify a Super CorrelationMgr consisting of a coordinating, correlating PIDS EMPI server that could link all of the ID Domains with the various servers acting as proxies as well as Correlation Mgrs. Proxy is a system that performs a task for another system on their behalf. A figure presents hierarchical solutions for Super CorrelationMgr. [8]

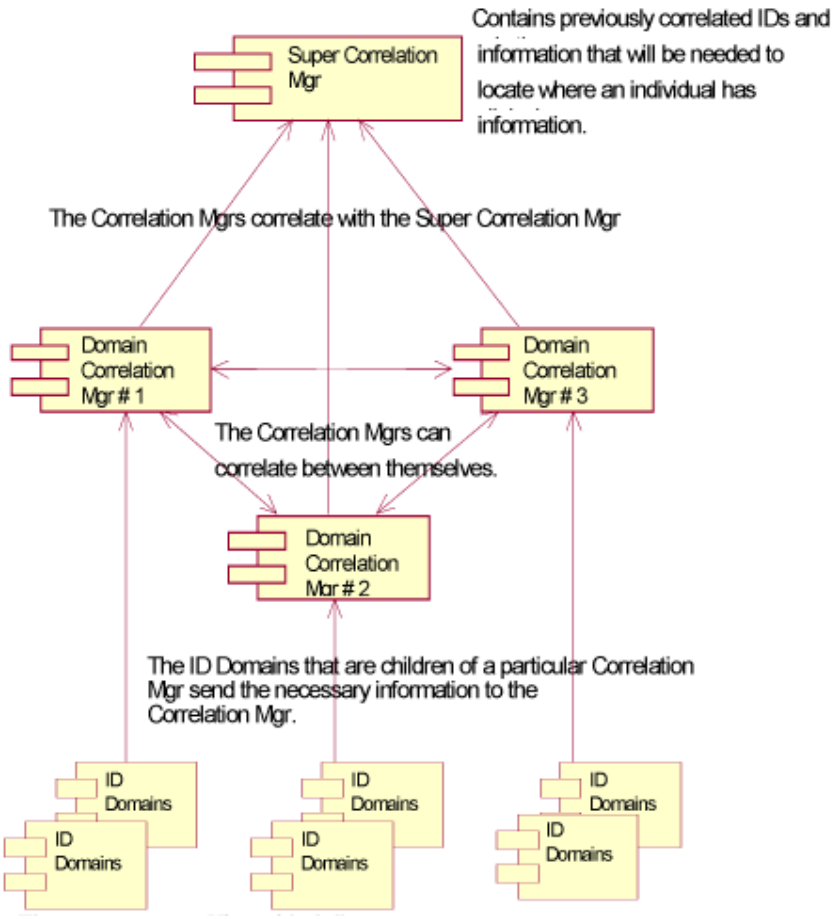


Figure 3: Hierarchical diagram showing roles of Correlation Mgrs, *Source: Federation of the Person Identification Service between Enterprises; David W. Forslund, Rebecca K. Smith, Thomas C. Culpepper.*

The Super CorrelationMgr wouldn't necessarily have to have all of the information from all of the Domain CorrelationMgrs, or be accessed at every patient admission. Most requests to the Domain CorrelationMgr by an ID Domain would be handled locally. If the patient isn't found locally or if the user needs to know what other sites have information about the patient, the request could be sent to the Super CorrelationMgr. If the Super CorrelationMgr has correlating ID's, it would return them to the Domain CorrelationMgr, which could then return them to the local site (ID Domain). If the Super CorrelationMgr doesn't have correlating ID's, it could ask the other Domain CorrelationMgrs for potential candidates and, if found, add them to the Super CorrelationMgr. If not found, the Super CorrelationMgr could register the new patient ID and inform the Domain CorrelationMgr of the new ID. [8]

4.3 The PIDS Security

The PIDS specification provides two interfaces to access information about a person, given their ID: ProfileAccess and IdentityAccess. The PIDS system relies on the underlying CORBA infrastructure and services that provide all the security mechanisms needed without exposing it in the interfaces.

The PIDS specification explains that main requirement of the PIDS system is to provide confidentiality of information that is stored about an individual.

This section explains that the CORBA Security provides robust mechanisms such as: authentication, authorization, encryption, audit trails, non-repudiation, etc. Furthermore the CORBA Security, in its default mode, allows the security concerns to be addressed without the client and server software being aware of it. It means that the security policies can be created and enforced after applications and systems have been created and installed. [5]

5 Introduction to Electronic Healthcare Record

Nowadays the patient's healthcare is no longer the sole responsibility of a single health professional, but rather is shifting to a team-based or shared-care approach. It means that under shared-care, the patient gets healthcare services from groups of health professionals representing all sectors, including primary, secondary and tertiary, all collaborating together. [23]

The author believes that development of patient identification systems should base on EHR systems. [22]

Furthermore the PIDS services should be designed to work with shareable healthcare records but it is needed to remember about the security aspects, and provide policies for authorized users. All theses issues are supported by implementation of EHR standards.

5.1 EHR Definitions

This chapter describes definitions of The Electronic Health Record (EHR). The EHR is a secure, real-time, point-of-care, patient centric information resource for clinicians. They can use EHR to get patient's information and help making decision. It also automates and streamlines the clinician's workflow, closing loops in communication and response that result in delays or gaps in care. [20]

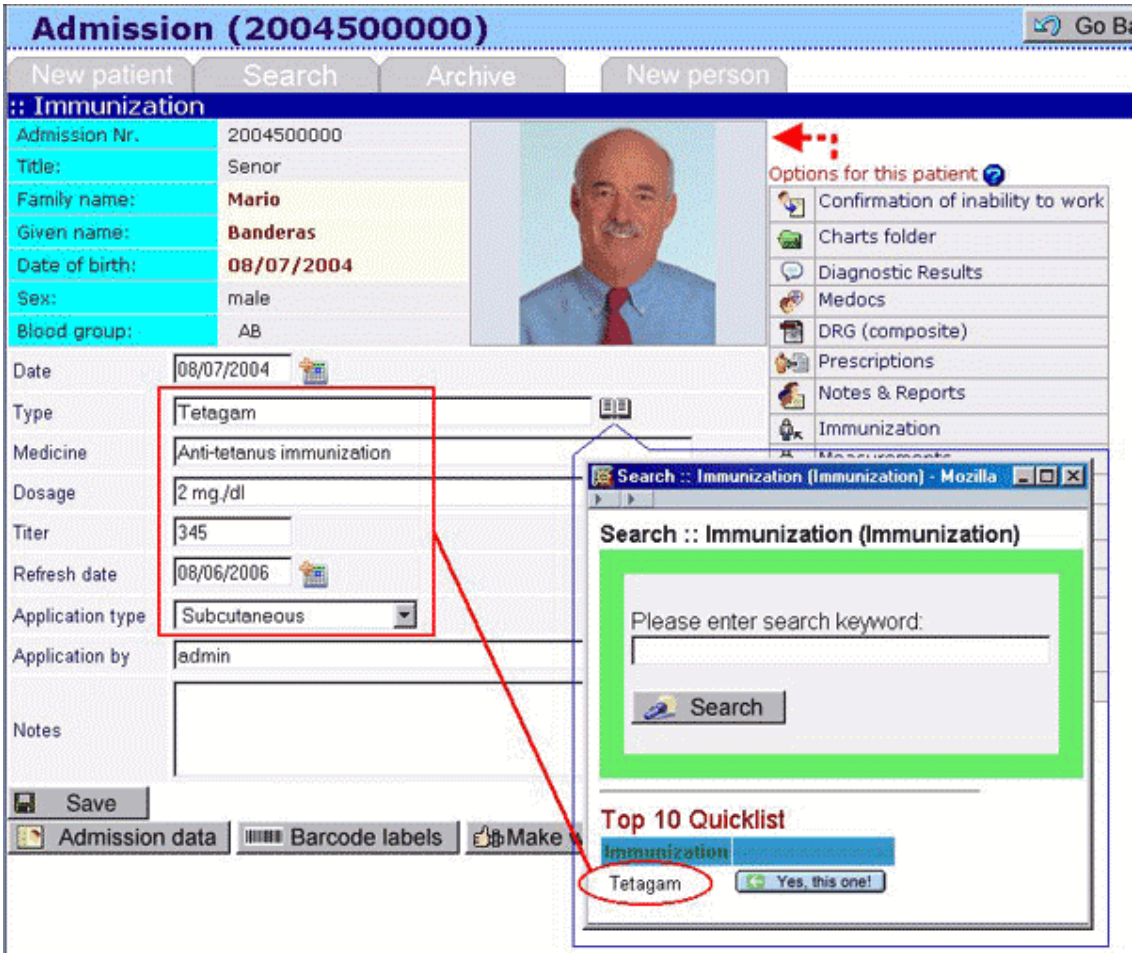


Figure 4: An example of Electronic Healthcare Record, *Source: http://en.wikipedia.org/wiki/Image:Immune_auto.jpg*

Ilias Iakovidis defines the electronic healthcare record as digitally stored health care information about an individual's lifetime with the purpose of supporting continuity of care, education and research, and ensuring confidentiality at all times. The EHR is not a goal in

itself, but a tool for supporting the continuity of care and consequently the quality, access and efficiency of health care delivery. [22]

The author believes that patient identification system should underpin sharable and interoperable electronic healthcare records. The sharable records should be independent of EHR systems, store and transmit information securely, and accessible by multiple authorised users using different applications. The difference between a shareable and non-shareable EHR is analogous to the difference between a stand-alone desktop PC and a networked PC which adds enormous benefits in terms of locating, retrieving and exchanging information using the Internet, an intranet, email, workgroup collaboration tools etc. [23]

5.2 EHR Systems

The EHR system can be defined as the set of components that form the mechanism by which patient records are created, used, stored, and retrieved. A patient record system is usually located within a health care provider setting. It includes people, data, procedures, processing and storage devices (e.g., paper and pen, hardware and software), and communication and support facilities. [24]

The EHR systems operate on EHR in order to manage and provide the information by authorized users and only to qualified users. The system can be a small group of PCs, hospital information system, or a group of hospital and primary care systems in a regional network. The EHR system should provide user friendly interface, help medical workers to retrieve the information in a fast manner, communicate easily with others, and make user's work more effective. The EHR system should also provide confidentiality at all times by meeting security requirements. The administrative systems, departmental clinical systems, or even stand-alone general practitioner's systems are not examples of EHR systems, but rather limited scope electronic medical systems or computerised medical systems. Thus, EHR supports the decentralised network of health care delivery institutions that slowly replaces hospitals as centres of care delivery. [22]

5.3 EHR Standards

Standardisation of the architecture of electronic healthcare records is essential as the records are being used to support shared care between clinicians in different disciplines and to enable mobility within and between countries of people who give and receive healthcare. [25] It is useful to understand different EHR standards because PIDS systems will utilize various EHR systems in the future.

"Unique identification of all EHR parties is clearly essential for both medico-legal and interoperability purposes. Note that it is desirable to have a "Unique Identifier" (namely, a unique number) standard for EHR and other purposes, but a "Unique Identifier standard" is not essential for unique identification. ISO/TC 215 and several other health informatics standards development organizations have or are developing client and provider identification standards that use a combination of demographic attributes for identification, without requiring a unique identification number."[24]

5.3.1 OpenEHR

The Good European Health Record (GEHR) project was a large European Union (EU)-funded project undertaken within eight European countries between 1992 and 1994. It was the first international research project to develop a specification for a "good" quality EHR based on detailed clinical, technical and medico-legal requirements. [23]

In 2002 a decision was made to merge the work of UCL and Ocean to form a new model which was called openEHR. The Foundation was formed as an independent non-profit organisation. The openEHR is not intended to be a standard in itself but rather an input to international standards such as CEN 13606, ISO/TC 215 and HL7. [23]

They promote and publish the formal specification of requirements for representing and communicating electronic health record information, based on implementation experience, and evolving over time as health care and medical knowledge develop. They also manage the sequential validation of the EHR architectures through comprehensive implementation and clinical evaluation. [26]

The openEHR foundation is dedicated to the development of an open, interoperable health computing platform, of which a major component is clinically effective and interoperable EHRs. They provide archetypes and templates of all clinical systems, and collaborate together with allied medical professionals to define the content, semantics and user interfaces of systems independently from the software. [27]

The openEHR also provides the model for Demographic Information Model (See Appendix 4). The purpose of this model is as a specification of a demographic service, either standalone, or a "wrapper" service for an existing system such as a master patient index (MPI). The model expresses attributes and relationships of demographic entities within the context of the health record or other relevant model. [63]

The openEHR also work on proper integration with terminology systems, including with: SNOMED-CT[6], LOINC[7], and ICDx[8] classifications. [27]

The author met many times with the clinical terminology during two years of the Health Informatics course and he is also interested in gaining knowledge about it. Although clinical terminology is not subject of this thesis, we gathered some brief information about the medical terminology. (See Appendix 1)

5.3.3 EN13606

The Committee European Normalisation (CEN) is the peak European standards organisation which transcends the national standards organisations of its member countries. The organisation develops CEN 13606 standard that is major revisions of previously published pre-standards[9]. In November 2001, a decision was taken by CEN to update 13606 and to adopt the openEHR archetype methodology. A Memorandum of Understanding (MOU) was signed between CEN and the openEHR Foundation to enable the Australian members of openEHR to participate in the revision project. [23] The aim of the EN13606 European Standard is to normalize the transfer of information between EHRs systems in an interoperable fashion, without specifying how to implement them. With these premises, EHRCom Task Force revised the ENV13606 pre-standard in order to draw up the EN13606

---

[6] A major ontological terminology effort by the College of American Pathologists; see http://www.snomed.org

[7] A code system for laboratory observations. See http://www.regenstrief.org/loinc/.

[8] WHO International Classification of Diseases. See http://www.who.int/classifications/icd/en/.

[9] All CEN standards are initially published as "pre-standards" (designated "ENV") for a period of three years to enable implementation experience. At the end of this period a pre-standard can either be adopted unchanged as a full de jure European standard (designated "EN"), revised to become a full standard, or discontinued. CEN is changing its naming from "pre-standard" to Technical Specification" to be consistent with ISO.

standard. [37] The EHRcom Task Force was set up to review and revise the 1999 four-part pre-standard ENV 13606 and to produce a formal standard (EN).[38] Nowadays, the standard contains five following parts:

Reference model: [38] This models the stable characteristics of the patient register and how it is organized hierarchically, defining the necessary classes in order to manage both the clinical information and that of its context.

Specification for the exchange of archetypes: [39] It includes the generic information model and the language for the representation and communication of the definitions of individual instances of archetypes. It follows the approximation drawn up by the openEHR group. It is compatible with the specification of HL7 templates.

Archetype reference and list of terms: It contains the initial set of archetypes for Europe, as well as the specification of their repository. It also defines the list of relevant terms (both regulatory and informative) necessary for other parts of this standard (for example, for certain attributes in Part 1- Reference model).

Security characteristics: These specify the measures necessary for the control of access to information, the consent and register of the communications activity.

Exchange models: Messages and interface services which allow the communication of EHR extracts and archetypes. [37]

This standard is based on the reference model that defines the set of classes and attributes. It is presented as a set of diagrams drawn using the Unified Modelling Language (UML).[10]

The reference model would be equivalent to the technical specifications of the construction blocks and the mechanisms for connecting them. [37]

This model consists of four packages such as: Extract, Demographics, Access Control and Message (see Appendix 5) which discusses the aspects of an EHR that are relevant for communication of EHR extracts between information systems. Demographics package provides a minimal data set to define the various persons, software agents, devices and organizations. [64]

[10] UML, Unified Modelling Language, http://en.wikipedia.org/wiki/Unified_Modeling_Language

The standard also uses archetype that is definition of a hierarchical combination of components of the reference model. These structures, although sufficiently stable, may be modified or substituted by others as clinical practice evolves. [37]

The ENV13606 standard explains that health records should be created, processed and managed in ways that guarantee the confidentiality of their contents and legitimate control by patients in how they are used. [39]
This standard does not specify who should have access to what and by means of which security mechanisms; these need to be determined by user communities, national guidelines and legislation. [39]
The PIDS specification also provides security that relies on the underlying CORBA infrastructure and services, which provide the security mechanisms needed to implement the patient identification system. It is useful to highlight importance of EHR standard in terms of patient identification systems. It is believed that good understanding of EHR standards might help to integrate existing EHR systems with the Identity services in the future.

5.3.4 ISO/TC 215

ISO (International Organization for Standardization)[11] is a worldwide, non government federation of national standards bodies (ISO member bodies) with a Central Secretariat in Geneva, Switzerland, that coordinates the system. The work of preparing International Standards is normally carried out through ISO technical committees. [40]
In terms of healthcare domain, they prepared ISO Technical Specification 215 (ISO/TS 215). It is also the peak international standards body for EHR and other health informatics standards. However, it is a relative newcomer to health informatics standards. An initial draft ISO EHR definition was proposed in a discussion paper written in October 2002.

[11] ISO - International Organization for Standardization, http://www.iso.org/iso/en/aboutiso/introduction/index.html, accessed in July 2007

They work on standardization in the field of Health Information and Communications Technology (ICT)[12]. The primary purpose of ISO's family of EHR standards is to maximise interoperability between electronic records. [41]

The standard ISO TC 215 has several Working Groups (WG) that are responsible for different parts of EHR development:

WG 1: Health Records and Modelling Coordination

WG 2: Messaging and communications

WG 3: Health Concept Representation

WG 4: Security

WG 5: Health Cards

WG 6: Pharmacy and Medication [42]

The WG4 Security group states that healthcare information is accessible for medical workers and patients. They explain that the accessibility of healthcare information might be provided by the Internet. The Internet is also an insecure vehicle that demands additional measures be taken to maintain the privacy and confidentiality of information. For this reason, the ISO TC 215 standard recommends to implement Public Key Infrastructure (PKI)[13] technology. [43] [44] [45]

5.3.5 HL7

HL7 (Health Level 7)[14] has in recent years become the pre-eminent US health informatics standards organisation. It was originally concerned only with messaging standards but has more recently become involved in standardisation of decision support and terminology. In late 2001, they formed an EHR SIG (Special Interest Group) which has attracted considerable interest and involvement from within the HL7 community. CEN and HL7 have signed an MOU (Memorandum of Understanding) to further co-operation between the two organisations. HL7 also develop Clinical Document Architecture (CDA). It is an XML (The

---

[12] ICT - Information and Communications Technology, http://en.wikipedia.org/wiki/Information_technology, accessed in July 2007

[13] Public Key Infrastructure (PKI), http://en.wikipedia.org/wiki/Public_key_infrastructure, accessed in August 2007

[14] Health Level 7, http://www.hl7.org/

Extensible Markup Language) [15]-based markup standard intended to specify the encoding, structure and semantics clinical documents for exchange. It is based on the HL7 Reference Information Model (RIM). The CDA is a subset of the CEN 13606 Reference Model and the CEN 13606 standard is compliant with CDA Release 2. [23]

6 Technical approaches for PIDS systems

6.1 LDAP and Relational Databases

There are various medical systems implemented within the healthcare domain. These systems often utilize databases that store medical information such as medical staff, patient demographics, medical devices, and financial data. Nowadays there are many different databases provided by many manufactures.

Before considering the types of databases, a general definition of database must be explained. A database is a structured collection of records or data that is stored in a computer. It may be discussed in terms of software and hardware issues. Database application and computers need to provide reliable and robust solutions for performance, adequate storage space and data backup and restore. [49]

This section defines two types of databases such as Directory Services [b] and Relational Database Management Systems (RDBMS) [50] that are often implemented in medical environments. Patient identity services might need to connect to different types of databases to retrieve patient IDs and also these systems would have implemented own database to store already correlated patients' IDs. Thus it is important to understand different features of Directory Service and RDBMS.

The directory service is a searchable database repository that lets authorized users and services find information related to people, computers, network devices, and applications.

[15] XML, The Extensible Markup Language, http://en.wikipedia.org/wiki/Xml, accessed August 2007

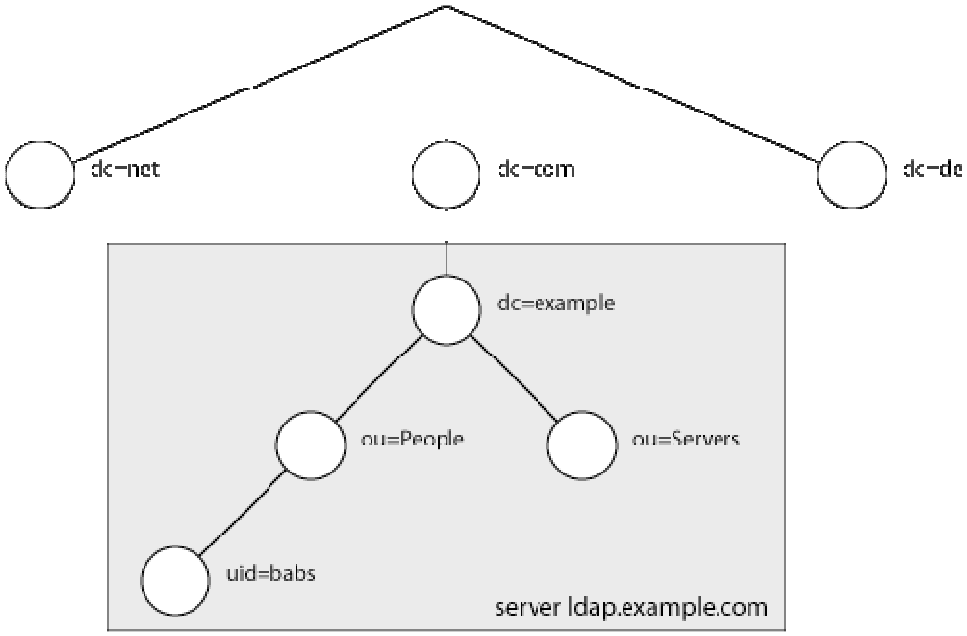


Figure 5: LDAP structure tree, Source: *http://max.berger.name/*

They are specialized databases, which are designed to provide very quick read access to the information and tuned to give faster response to high-volume lookup or search operations. It might prove distributed medical applications. [51] The Lightweight Directory Access Protocol (LDAP) [52] is an open industry standard that's gaining wide acceptance as a directory-access method. The LDAP is the lightweight version of the Directory Access Protocol and is a direct descendent of the heavyweight X.500, the most common directory-management protocol. Although they use a similar structure for data representation, LDAP and X.500 have several fundamental differences.

Nowadays many Web-based services such as email, file transfer and videoconferencing utilize the LDAP technology. [51]

The LDAP operations are based on the client–server model. Each LDAP client uses the LDAP protocol, which runs over TCP/IP, to retrieve data stored in a directory server's database. LDAP servers enable replication, for example a primary LDAP called master can send updates to a read-only replica server called slave. In the LDAP servers, data-representation format is based on XML. [51] XML is used to store the directory configuration in form of the Directory Service Markup Language (DSML)[16], and for data concerning the

[16] DSML, Directory Service Markup Language, http://en.wikipedia.org/wiki/Directory_Service_Markup_Language

connector to provide a means of representing directory structural information as an XML document. DSML describes directory entries, a directory schema or both. [53]

LDAP directories are databases arranged in hierarchical information trees representing the organizations they describe. A figure below shows an example of a three-level hierarchy. Each LDAP entry is identified by a distinguished name (DN) that declares its position in the hierarchy. [51]

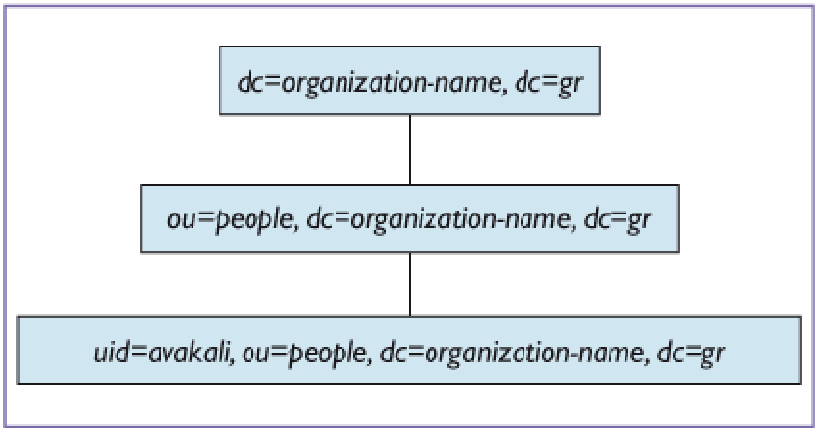


Figure 6: The example of LDAP hierarchy. A distinguished name identifies each LDAP entry and declares its position in the hierarchy. *Source: LDAP Framework,Practices, and Trends, Vassiliki Koutsonikola and Athena Vakali, Aristotle University, IEEE Internet Computing*

This section also explains main differences between LDAP database and relational databases. In relational databases, database administrators define the database schema; in LDAP, there is a fixed core schema that controls the directory hierarchy. Also, whereas LDAP objects are nested in hierarchies, relational database objects are related to each other via primary and foreign keys that connect data items.

LDAP data types and structure are flexible and extensible. In relational databases, the query processor is aware of the relationships among database objects, whereas in LDAP, the corresponding relationships are extracted during the querying process.

LDAP supports multivalued data fields what is not allowed in relational databases which states that fields with repeating values must be placed into separate tables. Finally, LDAP does not support relational database features such as views and joins.

The LDAP directories can be highly distributed, whereas relational databases are typically centralized. The directory services can be spread over a large geographical area. But from the

client side that complexity is transparent, that might be the major advantages for PIDS systems that should be designed as distributed systems. LDAP servers have built-in replication engines that can replicate some or all of their data. In relational databases the replication is also possible but it is the additional functionality. [51]

6.2 JDBC drivers

This section discusses the JDBC API (The Java Database Connectivity Application Programming Interface). The JDBC API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases. [54]
It also lets Java applications connect to and retrieve data from relational databases. It provides methods for querying and updating data in databases. The JDBC technology also provides a common base on which tools and alternate interfaces can be built. [55]
It is useful to discuss the Java technology because the study design presented in the design study presented in chapter 7 bases on Java application and the author also is inspired to use Java programming for the Identity services. We state that Java has simpler object model than other programiming languges such as C or C++, supports platform independence, what means that programs written in Java language must run similarly on any supported platforms. The main idea of Java is: one should be able to write a program once, compile it once, and deploy it anywhere. [56]
Next we must consider the JDBC Drivers tht are client-side adaptors (they are installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. [57]
There are actually four ways for a Java program to connect to a database:

6.2.1 Type 1: JDBC-ODBC Bridge

At this point it is necessary to briefly explain the Open Database Connectivity (ODBC) term. The ODBC API was developed and is being evolved by Microsoft. It provides library of ODBC functions that let ODBC-enabled applications connect to any database for which an ODBC driver is available, execute SQL statements, and retrieve results. [58]

Although the ODBC drivers are provided by all major database vendors, this technology is hard to learn because it uses programming in ODBC that is not trivial. If we compare JDBC to ODBC then JDBC is an independently-controlled standard, provides an object-oriented wrapping and redesign of ODBC API that is easier to learn. For this reason JavaSoft and Intersolv have developed a product called the JDBC-ODBC Bridge, which allows you to connect to databases for which no direct JDBC driver yet exists. [59]

The JDBC-ODBC Bridge is also called Type 1 of JDBC driver. It is a local solution, since the ODBC driver and the bridge code must be present on each user's machine. [59]

It translates all JDBC calls into ODBC calls and sends them to the ODBC driver. [60]

This solution can be used when the database and the application are on the same machine and there is no server code interposed. It is the simplest of all but depends only on the Microsoft platform. Other disadvantege is poor performance because the calls have to go through the JDBC overhead bridge to the ODBC driver. [57] Figure 8 illustrates a typical JDBC-ODBC Bridge environment.

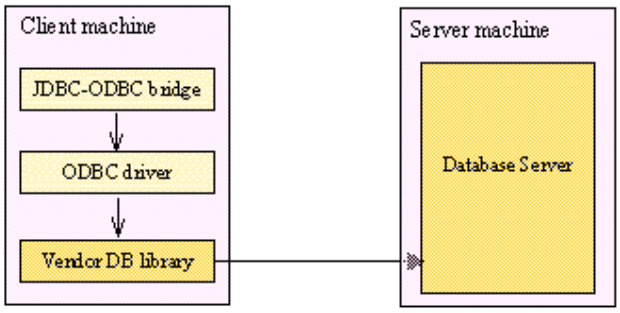


Figure 7: Type 1: JDBC-ODBC Bridge, *Source: http://www.javaworld.com/javaworld/jw-07-2000/jw-0707-jdbc.html*

6.2.2 Type 2: Native-API/partly Java driver

The JDBC driver type 2 replaces ODBC and the bridge with another local solution. [59]

This driver communicates directly with the database server; therefore it requires that some binary code be present on the client machine. [60] Although the type 2 driver provides more functionality and performance than the type 1 driver, this driver still cannot be used in internet because the vendor client library needs to be installed on the client machine. [57]

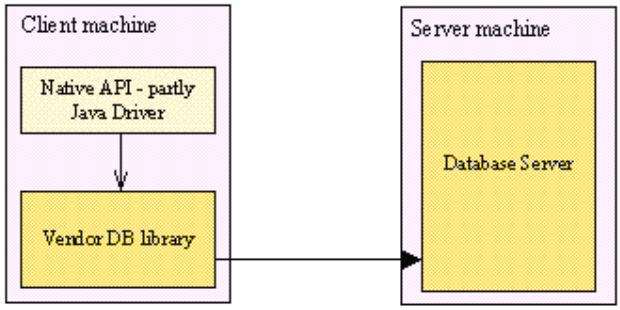


Figure 8: Type 2: Native-API/partly Java driver, *Source: http://www.javaworld.com/javaworld/jw-07-2000/jw-0707-jdbc.html*

6.2.3 Type 3: Net-protocol/all-Java driver

JDBC driver type 3 follows a three-tiered approach whereby the JDBC database requests are passed through the network to the middle-tier server. [60]

It means that an application or applet calls a middle-tier application server that in turn calls the database. The server can then connect to any of a number of databases. This method allows you to call a server from an applet on a client machine and return the results to an applet. [59] The type 3 driver is written entirely in Java and also is platform-independent as the platform-related differences are taken care by the middleware server. The client side does not have to know if the databases are changed to other vendors and there is no need for the vendor db library on the client machine. This solution can be used in the Internet and among corporate computer networks. [57]

In this case it is necessary to mention about the security aspects as it is possible to create an encrypted communication between the middleware servers and the database servers. For example: firewall that is a hardware or software device. The firewall provides services to manage, regulate and encrypt network traffic between network computers.

This solution is used in the design study (chapter 7) and we believe that the type 3 driver might be applicable in the implementations of Identity Service.

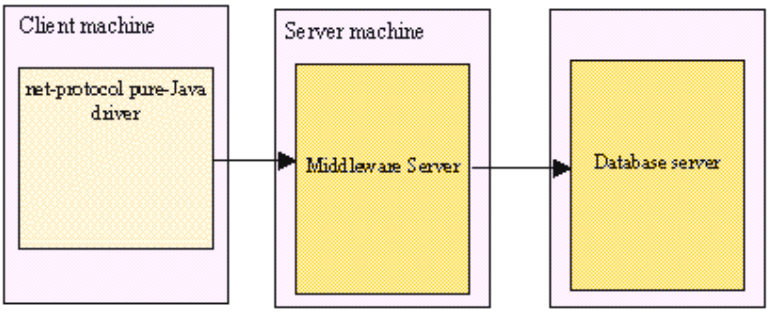


Figure 9: Type 3: Net-protocol/all-Java driver, *Source: http://www.javaworld.com/javaworld/jw-07-2000/jw-0707-jdbc.html*

6.2.4 Type 4: Native-protocol/all-Java driver

The JDBC driver type 4 converts JDBC calls into the vendor-specific database management system (DBMS) protocol so that client applications can communicate directly with the database server. This solution is completely implemented in Java to achieve platform independence and eliminate deployment administration issues. [60]

This method also can be used over a network and can then display results in a Web browser applet. [59] The JDBC driver type 4 does not require translation or middleware layers for request indirection, thus the performance is considerably improved. [57]

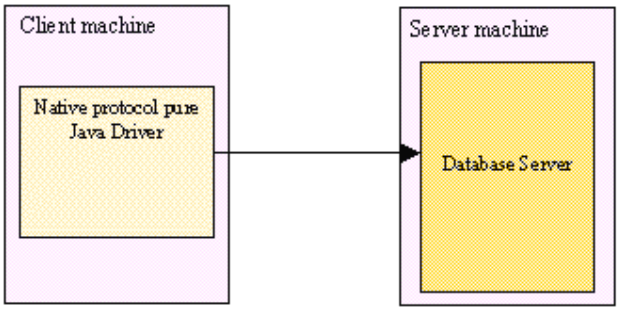


Figure 10: Type 4: Native-protocol/all-Java driver, *Source: http://www.javaworld.com/javaworld/jw-07-2000/jw-0707-jdbc.html*

7 Design Study

7.1 Introduction

This chapter discusses an implementation of a simplified patient identification service based on Java Servlet. The aim of this system is to search patient demographic records among different relational databases.
This implementation uses two databases that are implemented in Irish hospitals. It bases on the real database schemas that contain the synthetic data, re-populated by author of this thesis.

The application utilizes patient universal identifiers such as forename, surname, date of birth, telephone number. The author develops a simplified Identity Service that is inspired by PIDS specification. The author created a simple confidence indicator that is also specified in the IdentifyPerson interface (see chapter 4.2).

The author also highlights obstacles that were noticed during implementation and utilisation of this application. It is also believed that revealed issues might help to design and implement Nationwide Identity Systems in the future.

It also believed that simple searches for patient records between those two databases would help to show problems with quality of data such as: misspelling, duplicate patient records, missing data etc.

7.2 Background

In this design study, the author uses Apache Tomcat[17] server as an application server. The Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet[18] and JavaServer Pages[19] technologies. Apache Tomcat is developed in

[17] Apache Tomcat Server, http://tomcat.apache.org

[18] Java Servlets, http://java.sun.com/products/servlet/index.jsp

an open and participatory environment and it is intended to be a collaboration of the best-of-breed developers from around the world. [61]

The Tomcat server bases on Java Platform, Standard Edition[20] that provides possibilities to develop and deploy Java applications on desktops and servers. There are two principal products in the Java SE platform family: Java SE Runtime Environment (JRE)[21] and Java Development Kit (JDK).[22]
The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language. The JDK is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications.

The author develops Java Servlets to aim the simple patient identification among different patient indexes. The Servlets are part of the Java platform technology and allowing a software developer to add dynamic content to a Web server. The Java Servlets receive and respond to requests from Web clients, usually across the HyperText Transfer Protocol[23] HTTP. [62]

This simplified implementation utilizes the patient demographic records which are stored in two relational databases. Therefore it is needed to set up connectivity between the Java Servlet and two databases. The Java Servlets utilize the Java Database Connectivity (JDBC) to get physical access to databases (See chapter 6.2).
The author utilizes the Oracle database because the real database schemas were developed for the Oracle platform. However the code of Java servlet is fully independent and can be integrated with other relational databases. The author also tested the JDBC drivers for MySQL database to prove that Java Servlets can work with databases from other vendors.

---

[19] JavaServer Pages, http://java.sun.com/products/jsp/

[20] Java Platform, Standard Edition, http://java.sun.com/javase/

[21] Java SE Runtime Environment (JRE), http://java.sun.com/javase/6/docs/technotes/guides/index.html#jre-jdk

[22] Java Development Kit (JDK), http://java.sun.com/javase/6/docs/technotes/guides/index.html#jre-jdk

[23] HyperText Transfer Protocol (HTTP), ftp://ftp.isi.edu/in-notes/rfc2616.txt

7.3 The system specification

The patient identification application proposed in this thesis utilizes specific requirements as follows:

Operating system: Microsoft Windows XP Professional (http://www.microsoft.com)
Application Server/Web server: Apache Tomcat 4.1.10 (http://jakarta.apache.org/tomcat/)
Oracle Database – Rational database 9.2.0.1 (http://www.oracle.com)
Java(TM) 2 SDK, Standard Edition Version 1.4.2 (http://www.java.com)
All these applications are installed on standalone PC
Pentium IV HT, 3.0 GHz Processor, 1 GB of Random Access Memory (RAM).

7.4 Implementation phase

7.4.1 The database schema

The author uses the real database schemas that are utilized in Irish hospitals. These databases have implemented different logical schemas. It means that these databases might contain different structures of tables and columns. In this case we use the tables that store patient demographic information. The tables use columns with different names and data formats for patient identifiers such as: surname, forename, date of birth, telephone number, PPSN, MRN etc.
First database is called CLINIC and contains the table CLIENT. This table contains the following names and formats for columns.

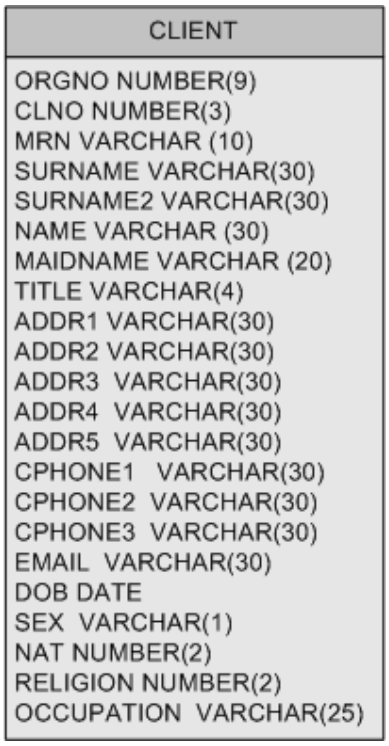


Figure 11: CLINIC database with the table CLIENT

The second database is called HOSPITAL and contains the table ACCOUNT with the following columns.

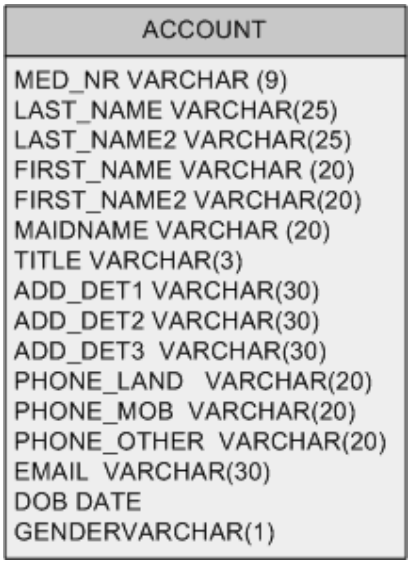


Figure 12: HOSPITAL database with the table ACCOUNT

At this point it is necessary to explain that these databases use real database schema but patient demographic data was repopulated with synthetic data.

Additionally the author develops third database. This database is called CONF and contains "dynamic" metadata for tables used above. This is intended to get the databases, tables and fields to perform a search within several databases in a transparent way.

It means that we use an abstraction layer that allows the Identity Service to be independent of underlying databases. It is instructive at this point to note that the real Identity Services might utilize patient demographic systems with different database schemas.

For this reason, the author develops database called CONF. this database contains the table SEARCH_FIELDS with the following columns.

```
# Table structure for table 'search_fields'
#

CREATE TABLE /*!32312 IF NOT EXISTS*/ `search_fields` (
  `database_name` varchar(10) NOT NULL,
  `table_name` varchar(10) NOT NULL,
  `field_name` varchar(10) NOT NULL,
  `field_type` varchar(15) NOT NULL,
  `field_req` tinyint(1) NOT NULL,
  `rel_id` int(10) NOT NULL,
  `search_group` int(10) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

REPLACE INTO `search_fields` (`database_name`, `table_name`, `field_name`, `field_type`,
`field_req`, `rel_id`, `search_group`) VALUES ('CLINIC','client','name','VARCHAR(30)',1,1,1),
       ('CLINIC','client','surname','VARCHAR(30)',1,2,1),
       ('CLINIC','client','dob','DATE',0,3,1),
       ('CLINIC','client','mphone','VARCHAR(30)',0,4,1),
       ('HOSPITAL','client','name','VARCHAR(30)',1,1,2),
       ('HOSPITAL','client','surname','VARCHAR(30)',1,2,2),
       ('HOSPITAL','client','dob','DATE',0,3,2),
       ('HOSPITAL','client','mphone','VARCHAR(30)',0,4,2);
```

The table includes columns where the SQL Statement will be executed to retrieve the data from different databases. Below are presented brief explanations for every column of the search_fields table.

DATABASE_NAME - Query to get the databases to create a connection.

TABLE_NAME - Field to get the table name from the database.

FIELD_NAME - Field to get the column name from the table.

FIELD_TYPE - Field to get the column type from the table.

FILED_REQ - Field to get the required flag of each column from the table.

REL_ID - Field to make a relationship between different columns.

SEARCH_GROUP - Field to create groups to associate columns in the search.

7.4.2 The Java Servlet configuration in Tomcat.

The Java Servlets need to be executed by Apache Tomcat server. For this reason, it is required to map a path to folder that contains the Java Servlets. Finally the end user can use any Internet browser to connect and utilize web-based application. It his cases we use path like: http://localhost:7080/pids/servlet/

The Tomcat server contains configuration XML file called web.xml.

tomcat-root-path/conf/web.xml

```
    <servlet-mapping>
        <servlet-name>dbTest</servlet-name>
        <url-pattern>/servlet/pids</url-pattern>
    </servlet-mapping>
```

The Servlets need to be compiled by the Java compiler. The commands presented below were executed in command line every time when the author made any changes in source code:

For example:

```
javac dbTest.java

javac dynamicForm.java
```

After the compilation process it possible to invoke the Java Servlet in the Internet browsers.

We develop Java Servlets such as:

ConfigurationDB.java, dbTest.java, dynamicForm.java, fieldInfo. java,formatHTML.java, utilDB.java. These Java Servlets are enclosed in appendix 3 and description for each servlet is described in Java documentation in the attached CD.

7.4.3 JDBC Configuration for Oracle database

The author configured JDBC driver for the Oracle relational database. The JDBC driver provides an access and network connection to database for the Java Servlet.

The Tomcat server contains configuration XML file called server.xml.

tomcat-root-path/conf/server.xml

This file contains all needed commands to configure JDBC drivers. The author customised the server.xml file for the specified databases:

Oracle database:

```
<Context path="/pids" docBase="pids" debug="0" privileged="true" reloadable="true"/>

       <Resource name="jdbc/clinic" auth="Container" type="javax.sql.DataSource"/>

            <ResourceParams name="jdbc/clinic">
              <parameter>
                <name>factory</name>
                <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
              </parameter>

              <!--DB username and password for dB connections  -->
              <parameter>
               <name>username</name>
               <value>username</value>
              </parameter>

              <parameter>
               <name>password</name>
               <value>password</value>
              </parameter>

              <!-- Class name for Oracle JDBC driver -->
              <parameter>
                 <name>driverClassName</name>
                 <value>oracle.jdbc.driver.OracleDriver</value>
              </parameter>

              <parameter>
                <name>url</name>
                <value>jdbc:oracle:thin:@localhost:1521:CLINIC</value>
              </parameter>
      </ResourceParams>
```

Also, it is needed to use a Java class as declaration of databases in the Java Servlet file (utilDB.java).

```
/**
 * Class to perform any database action.
 */
public class utilDB {

       // Database connection stuff
       private Connection conn = null;
       private Statement statement = null;
       private ResultSet results = null;

       private String dbHost = "localhost";
       private String dbUser="med";
       private String dbPass="hospital";
       private String strDB;

       // MySQL Connection
       //private String dbPort = "3306";
       //private String strConn;
       //private String classForName = "com.mysql.jdbc.Driver";

       // ORACLE Connection
       private String dbPort = "1512";
       private String strConn;
       private String classForName = "oracle.jdbc.driver.OracleDriver";

       // Getters/Setters dbUsername
       public String getDBuser()
       {
              return this.dbUser;
       }
       public void setDBuser(String user)
       {
              this.dbUser = user;
       }

       //Getters/Setters dbPassword
       public String getDBpass()
       {
              return this.dbPass;
       }

       public void setDBpass(String pass)
       {
              this.dbPass = pass;
```

```
        }

        // Getters/Setters DB hostname
        public String getDBhost()
        {
                return this.dbHost;
        }
        public void setDBhost(String host)
        {
                this.dbHost = host;
        }

        // Getters/Setters DB port
        public String getDBport()
        {
                return this.dbPort;
        }

        public void setDBport(String port)
        {
                this.dbPort = port;
        }

        // Getters/Setters database to connect
        public void setDB(String db)
        {
                this.strDB = db;
                // Line for MySQL
                //this.strConn = "jdbc:mysql://"+dbHost+":"+dbPort+"/"+strDB;
                // Line for ORACLE
                this.strConn = "jdbc:oracle:thin:@"+dbHost+":"+dbPort+":"+strDB;
        }
        public String getDB()
        {
                return this.strDB;
        }
```

7.4.4 Matching for patients

The author designs the patient search screen using Hypertext Markup Language (HTML) technology. The formatting of HTML pages is nested in Java Servlet file (formatHTML.java see Appendix 3) and deferent parts of page are invoked on demand. The dynamic HTML form is created to search in the system of databases, which can be from 1 to N databases and each one with its particular structure and fields. Some information can be omitted in a production system like the type of each field.

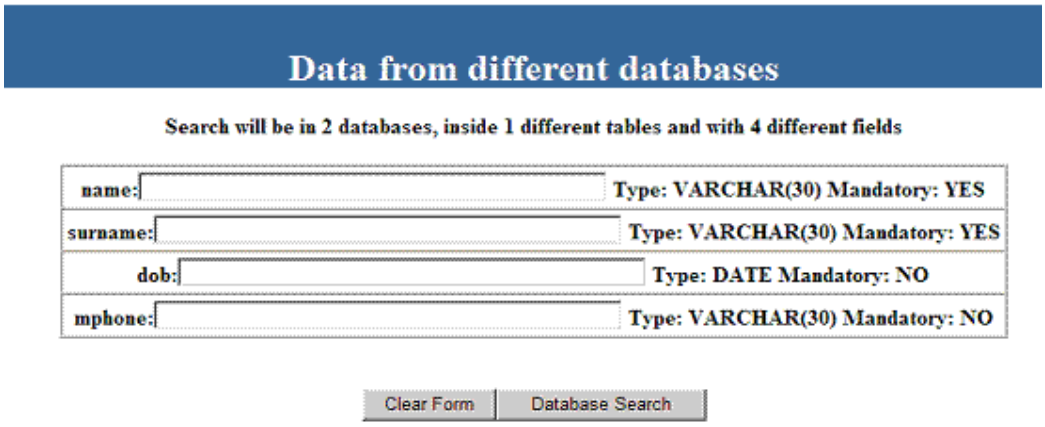


Figure 13: The page layout of the application

The Figure 14 shows a page layout of the application. There are specified four fields such as forename, surname, date of birth and mobile phone. Also, there are number of databases that are currently utilized by the search application.

The fields for forename and surname are mandatory what mean that a user needs to enter some inputs. Otherwise they get an error:

Figure 14: The error output of the application

If the users enter some valuable inputs then they get the search results:

In this case the user entered two mandatory fields:

forename=Jane and surname=Smith.

The Java Servlet prepares the query to search in two databases.

The result of the search is displayed in the table with six columns: name (forename), surname, dob (date of birth), mphone (mobile phone), matching and database. The column called database contains information about location of patient data. In this case we have two different databases named HOSPITAL and CLINIC.

The search results are sorted by the matching column.

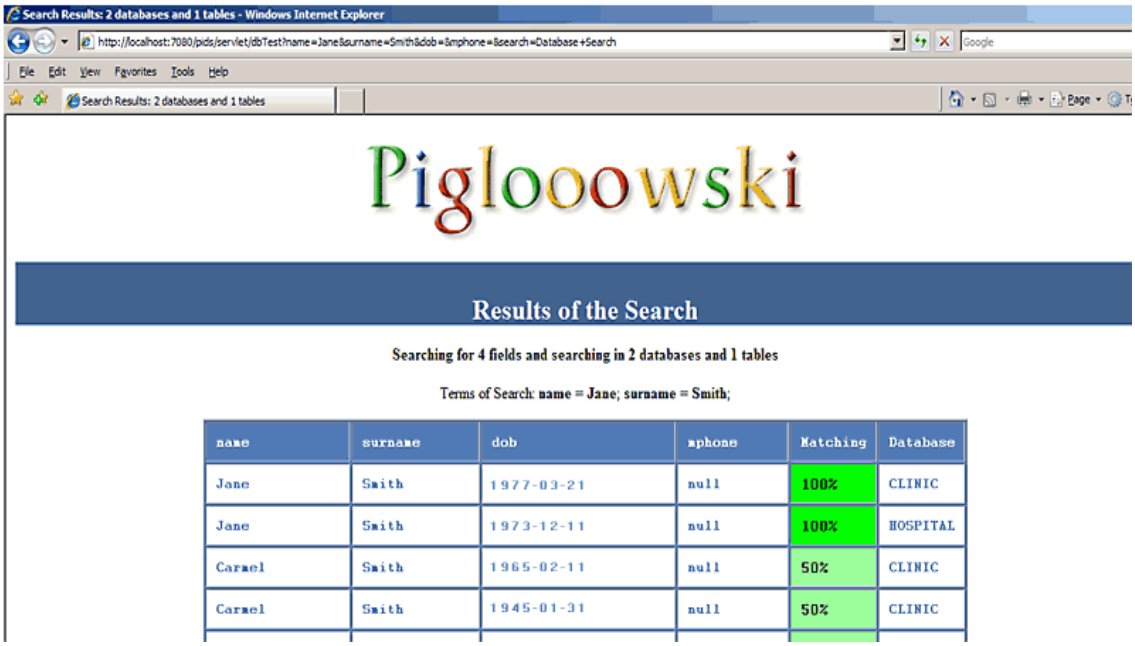


Figure 15: The results of the search with two mandatory fields.

In second case the user entered two mandatory fields and telephone number:

forename=Jane, surname=Monaghan and telephone=086 9227654

Results of the Search

Searching for 4 fields and searching in 2 databases and 1 tables

Terms of Search: **name = Jane; surname = Monaghan; mphone = 086 9227654**

name	surname	dob	mphone	Matching	Database
Jane	Monaghan	1977-03-21	0869227654	99%	CLINIC
Jane	Monaghan	1956-03-11	0869227654	99%	HOSPITAL
Jane	Monaghan	1977-11-22	null	66%	CLINIC
Jane	Monaghan	1977-11-05	null	66%	HOSPITAL
Brigid	Monaghan	1927-01-11	null	33%	CLINIC
Bernadette	Monaghan	1999-01-08	null	33%	CLINIC

Figure 16: The results of the search with two mandatory fields and telephone number.

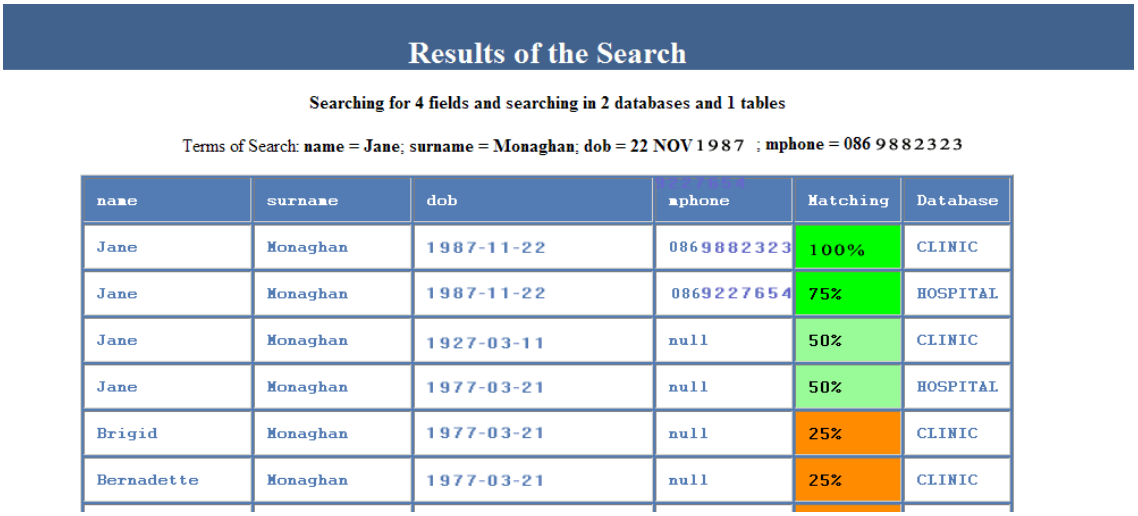
The author utilizes Structured Query Language (SQL) statements to search patient details among databases. For more details see dbTest.java file in Appendix 3

7.4.5 Confidence indicator

The simplified Identity Service has implemented the simply confidence indicator. It calculates four fields name, surname, date of birth and telephone number then it compares to the search results from two databases.

For example: the user entered two mandatory fields, date of birth and telephone number: forename=Jane, surname=Monaghan, date of birth=22-11-1987, and telephone=086 9882323

It is illustrated in Figure 18 that one patient is founded in the CLINIC database with the 100% matching. If a user enters only three fields such surname, forename and telephone number and there is an exact matching for these three forms then the search result also shows 100% of matching. This case is lustrated in Figure 17.



Results of the Search

Searching for 4 fields and searching in 2 databases and 1 tables

Terms of Search: **name = Jane; surname = Monaghan; dob = 22 NOV 1987 ; mphone = 086 9882323**

name	surname	dob	mphone	Matching	Database
Jane	Monaghan	1987-11-22	0869882323	100%	CLINIC
Jane	Monaghan	1987-11-22	0869227654	75%	HOSPITAL
Jane	Monaghan	1927-03-11	null	50%	CLINIC
Jane	Monaghan	1977-03-21	null	50%	HOSPITAL
Brigid	Monaghan	1977-03-21	null	25%	CLINIC
Bernadette	Monaghan	1977-03-21	null	25%	CLINIC

Figure 17: The results of the search with two mandatory fields date of birth and telephone number.

7.5 Evaluation results of searching patient records.

The author discovered that two databases have different columns in the tables. It means that users can enter only specified number of characters. For example the database CLINIC contains column called “SURNAME” that is used to store patient surname and is limited up to 30 characters. The database HOSPITAL contains “LAST_NAME” column and is limited to 25 characters. The similar differences exist for the forename and address columns. The similar problem exists for the date of birth formats. In this case two databases utilize the different formats. The database CLINIC contains the format like this: 1990-10-02 and second database HOSPITAL contains the format similar to this: 19/09/2003.

The author bases on the real database schemas but patient demographic data was repopulated with synthetic data to these databases. We executed numbers of the searches using multiple search criteria.

Finally, the author discovered some interesting findings. For example we founded two patients with the same surnames, first names, addressees but different ages. Each record is placed in different database. In this case, it was difficult to distinguish these two records but

the author had "next-of-kin" information about each patient. It discovered that father and son have very similar records except for their ages.

The other example shows duplicated patient records in the same database (for example CLINIC). The author discovered two patients records with different forenames but with the same surnames, addresses and dates of birth.

For example:

David Mullan , 275 Hampton Wood Avenue, 18-09-1987

Daithí Mullan, 275 Hampton Wood Avenue, 18-09-1987

It is supposed that patient was registered two times and for some reasons once time it could be used English name and other time Irish name. In this example a human intervention is needed to merge or update the patient records

We have also other examples where patients might be registered more than once time with different names or might exist in other MPI systems with different demographic details.

For example there might be duplicated records similar to these:

Christina, Smith, 1 Clareville Road, 18-09-1987

Kristina, Smith, 1 Clareville Road, 18-09-1987

Or

Bob, Crowe, 16 Brighton Square, 18-09-1987

Robert, Crowe, 16 Brighton Square, 18-09-1987

Finally, the patient demographic records also may change over time, in the case of a married female changing her maiden name to a married one.

8 Conclusions

The Identity Services can offer many improvements in healthcare domain such as: increasing quality of service for patients, saving healthcare costs and even saving human lives.

This is required by the National Health Information Strategy to implement the unique person identifier (UPI) in Irish healthcare.

The PIDS specification provides the interfaces to implement theses systems in a healthcare setting but there are also some technical areas that face some limitations. For example

integration with legacy information systems, various structures of medical systems or security concerns. Work is ongoing in these areas to evolve and overcome any deficiencies in the technology.

There are also administrative limitations that need to be standardised because the Patient Identification Systems need to update the patient records among different MPI systems but hospitals and healthcare organization have different internal policies about the access rules and privileges to patient records.

In the future the Identity Systems should form the basis for EHR standards because their main efforts aim to interoperability of the electronic patient records.

The design study provides the mechanism through which the different databases can cooperate and share data when required. This proves that there are possibilities to develop independent applications for the Identity services.

The design study also reveals the issues with quality of data, thus a human intervention is needed to make decisions if there is required to merge or update the patient records.

The national PIDS systems should be aware of changing patient records over time because of changing jobs, emigration, tourism etc.

Considering long term goals, it is clearly an ongoing iterative process. The relevant standards and with together with appropriate planning are required to implant PIDS systems in the nearest future in Ireland.

References

[1] Initiate Systems briefing, *Solving the Patient Identification Challenge in Interoperable Health Systems*, http://www.initiatesystems.com/resources/exec_summary/downloads/Documents/Patient_Identification_In_Interoperable_Health_Systems.pdf, (2007), accessed July 2007.

[2] *Service Functional Model Specification - Entity Identification Service (EIS),* Version 0.997, http://hssp-eis.wikispaces.com/space/showimage/EIS_Service_Functional_Model_for_SOA_SIG_v0997.doc, July 17 2006.

[3] Information Services Board (ISB), http://isb.wa.gov/policies/definitions.aspx#IndexI, accessed in August 2007

[4] RSA Laboratories, http://www.rsa.com/rsalabs/node.asp?id=2185, accessed in August 2007

[5] *Person Identification Service specification, version 1.1*, Object Management Group, 2001-04-04, http://www.omg.org/technology/documents/formal/person_identification_service.htm.

[6] United States Department of Health & Human Services, National Committee on Vital and Health Statistics, http://www.ncvhs.hhs.gov/app7-8.htm.

[7] *WS/PIDS: Standard Interoperable PIDS in Web Services Environments*, Eugen Vasilescu, Member, IEEE, Mihai Dorobant¸u, Member, IEEE, Sergio Govoni, Member, IEEE, Shilpa Padh, Member, IEEE, and Seong Ki Mun, IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, 2007

[8] *Federation of the Person Identification Service between Enterprises*; David W. Forslund, Rebecca K. Smith, Thomas C. Culpepper.

[9] *Analysis of unique patient identifier options*, Soloman I. Appavu, The Department of Health and Human Services, November 24, 1997

[10] Initiate Systems Company, White Paper, *A proven alternative to a unique national healthcare identifier*, http://www.initiatesystems.com/resources/exec_summary/downloads/Documents/Integrating_patient_medical_records_in_pursuit_of_the_EMR.pdf, (2003), accessed July 2007.

[11] *Health Identifier for Individuals*, A White Paper, http://ncvhs.hhs.gov/noiwp1.htm, accessed in May 2007.

[12] The Personal Public Service Number (PPSN), http://en.wikipedia.org/wiki/Personal_Public_Service_Number, accessed June 2007

[13] The Social Security Number (SSN), http://en.wikipedia.org/wiki/Social_Security_number, accessed June 2007

[14] Department of Social and Family Affairs, http://www.welfare.ie/topics/ppsn/cop.html, accessed August 2007

[15] *Unique Patient Concept: A key choice for European epidemiology*, Catherine Quantin, Olivier Cohen, Benoit Riandey, Francois-Andre Allaert,
International journal of medical informatics 76 (2007) 419–426, Journal homepage: www.intl.elsevierhealth.com/journals/ijmi.

[16] *Distributed Telemedicine Using High-Performance Computing Over the Internet*, David W. Forslund, James E. George, Eugene M. Gavrilov, Torsten A. Staab - Los Alamos National Laboratory Terry E. Weymouth - Univ. of Michigan, Srikanth Kotha - State Univ. of New York.

[17] The Object Management Group, Inc. (OMG), http://www.omg.org/, accessed June 2007.

[18] Healthcare Domain Task Force (DTF), http://healthcare.omg.org, accessed June 2007.

[19] Service Oriented Architecture (SOA), http://en.wikipedia.org/wiki/Service-oriented_architecture, accessed June 2007.

[20] *HIMSS Electronic Health Record Definitional Model Version 1.0*, HIMSS Electronic Health Record Committee, 11-6-2003.

[21] *Distributed Patient Records require Distributed Patient ID Management: Enter PIDS?* Sara Facchinetti, Damon Berry, Sebastien Pardon, Jane Grimson, Department of Computer Science - Trinity College Dublin, Ireland Control System and Electrical Engineer - Dublin Institute of Technology, Ireland.

[22] *Towards personal health record: current situation, obstacles and trends in implementation of electronic healthcare record in Europe*, Ilias Iakovidis, International Journal of Medical Informatics 52 (1998) 105–115

[23] *Health Care & Informatics Review Online-Current EHR Developments an Australian and International Perspective - Part 2*, Peter Schloeffel, 1 September 2004-HINZ 2004: Towards a Healthy Nation

[24] *HL7 EHR System Functional Model: A Major Development Towards Consensus on Electronic Health Record System Functionality,* 2004, HL7

[25] *Federated healthcare record server—the Synapses paradigm*, William Grimson, Damon Berry, Jane Grimson, Gaye Stephens, Eoghan Felton, Peter Given, Rory O'Moore, International Journal of Medical Informatics 52 (1998) 3–27

[26] Introducing openEHR, http://www.openehr.org/about_openehr/t_home_aims.htm, Accessed May 2007

[27] Introducing *open*EHR, The *open*EHR Foundation, http://svn.openehr.org/specification/TRUNK/publishing/openEHR/introducing_openEHR.pdf Accessed August 2007

[28] SNOMED Clinical Terms® User Guide, Release January 2007, January 2007 Release

[29] NHS, SNOMED Clinical Terms, http://www.connectingforhealth.nhs.uk/systemsandservices/data/snomed , accessed May 2007

[30] SNOMED Clinical Terms, http://www.snomed.org/documents/snomed_overview.pdf http://en.wikipedia.org/wiki/SNOMED_CT, accessed May 2007

[31] SNOMED Clinical Terms, Brochure, http://www.snomed.org/news/pdfs/CTbrochure0902.pdf, accessed May 2007

[32] Logical Observation Identifiers Names and Codes (LOINC), http://en.wikipedia.org/wiki/LOINC, accessed May 2007

[33] Logical Observation Identifiers Names and Codes (LOINC), http://www.regenstrief.org/medinformatics/loinc/faq/getting-started/#what-is-loinc accessed May 2007

[34] NANDA the North American Nursing Diagnosis Association, http://en.wikipedia.org/wiki/NANDA, accessed May 2007

[35] NANDA, www.healthlanguage.com/press/PR_-_NANDA.PDF

[36] International Classification of Diseases (ICD), http://www.who.int/classifications/icd/en/, accessed May 2007

[37] *Proof-of-concept Design and Development of an EN13606-based Electronic Health Care Record Service,* ADOLFO MUNOZ, ROBERTO SOMOLINOS, MARIO PASCUAL, Journal of the American Medical Informatics Association Vol.14 No.1 Jan-Feb 2007

[38] *CEN/TC251/WG1 prEN13606-1:2003 Health informatics—Electronic record communication—Part 1: Reference Model*, http://www.centc251.org, Accessed May 2007.

[39] *CEN/TC251/WG1 prEN13606-4:2003, 2006-08-01, Health informatics—Electronic record communication—Part 4: Security requirements and distribution rules*, http://www.centc251.org, Accessed May 2007.

[40] ISO - International Organization for Standardization, http://www.iso.org/iso/en/aboutiso/introduction/index.html, accessed in July 2007

[41] ISO TC 215 Standard; source: http://www.iso.org/iso/en/stdsdevelopment/tc/tclist/TechnicalCommitteeDetailPage.TechnicalCommitteeDetail?COMMID=4720, accessed in July 2007

[42] ISO/TC 215, http://en.wikipedia.org/wiki/ISO_TC_215

[43] ISO/TC 215/WG 4 Security, Health Informatics – Public Key Infrastructure - Part 1: Framework and overview, 2001-03-05

[44] ISO/TC 215/WG 4 Security, Health Informatics – Public Key Infrastructure - Part 2: Certificate profile, 2001-03-05

[45] ISO/TC 215/WG 4 Security, Health Informatics – Public Key Infrastructure - Part 3: Policy management of certification authority, 2001-03-05

[46] HL7 Standards, http://www.hl7.org/library/standards_non1.htm#Clinical%20Context%20Object%20Workgroup, accessed in August 2007

[47] A Strategic ICT Framework for the Irish Health System, 2003

[48] PKI, In the pipeline, http://www.sai-global.com/newsroom/tgs/2003-06/pipeline/pipeline.htm

[49] Database, http://en.wikipedia.org/wiki/Database

[50] *Relational database management system,* http://en.wikipedia.org/wiki/Relational_database_management_system, accessed in August 2007.

[51] *LDAP Framework,Practices, and Trends*, Vassiliki Koutsonikola and Athena Vakali, Aristotle University, IEEE Internet Computing, 09-10 2004.

[52] *Lightweight Directory Access Protocol (v3)*, M. Wahl, T. Howes, and S. Kille, IETF RFC 2251, Dec. 1997; www.ietf.org/rfc/rfc2251.

[53] *LDAP Application Development using J2EE and .NET,* Amal Barman, INDIAN INSTITUTE OF TECHNOLOGY, December 2004.

[54] JDBC, http://java.sun.com/products/jdbc/overview.html, accessed in September 2007

[55] The Java Database Connectivity (JDBC) API, http://java.sun.com/javase/6/docs/technotes/guides/jdbc/, accessed in September 2007

[56] JAVA Programming, http://en.wikipedia.org/wiki/Java_(programming_language), accessed in September 2007

[57] JDBC, http://en.wikipedia.org/wiki/JDBC_type_1_driver, accessed in September 2007

[58] ODBC-JDBC, http://www.easysoft.com/developer/interfaces/odbc/index.html, accessed in September 2007

[59] Using JDBC to create database objects http://www-128.ibm.com/developerworks/opensource/library/j-jdbc-objects/, accessed in September 2007

[60] JDBC drivers in the wild, http://www.javaworld.com/javaworld/jw-07-2000/jw-0707-jdbc.html, accessed in September 2007

[61] Apache Tomcat Server, http://tomcat.apache.org, accessed in September 2007

[62] Interface Servlet, http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/Servlet.html, accessed in September 2007

[63] The *open*EHR Reference Model, Demographic Information Model, Release 1.0.1

[64] *A Semantic Web Service-based P2P Infrastructure for the Interoperability of Medical Information systems,* IST-2103 STP ARTEMIS, SPECIFIC TARGETED RESEARCH PROJECT PRIORITY 2.3.1.11 eHealth

[65] *A GENERIC LDAP CORBA CONNECTOR*, W Radinger, M Jandl, A Szep and K M Goeschka, Vienna University of Technology, Institute of Computer Technology, IEEE Africon 2002

Appendix 1 - Clinical Terminology

The standardized terminology can provide benefits to clinicians, patients, administrators, software developers and payers. A clinical terminology can aide in providing health care providers with more easily accessible and complete information pertaining to the health care process (medical history, illnesses, treatments, laboratory results, etc.) and thereby result in improved patient outcomes. A clinical terminology can allow a health care provider to identify patients based on certain coded information in their records, and thereby facilitate follow-up and treatment. [28]

Medical professionals can use different clinical terms that mean the same thing. For example, the term heart attack, myocardial infarction, and MI may mean the same thing to a cardiologist, but, to a computer, they are all different.

This appendix contains the terms of terminologies standards used in medical domains.

SNOMED CT (Systematized Nomenclature of Medicine) CT Clinical Terms® was a joint development between the National Health Service (NHS) and the College of American Pathologists (CAP) to improve and safeguard patient care by using an agreed terminology. [29]

SNOMED CT allows a consistent way to index, store, retrieve, and aggregate clinical data across specialties and sites of care. It also helps organising the content of medical records, reducing the variability in the way data is captured, encoded and used for clinical care of patients and research. Examples of computer applications using SNOMED CT: Electronic Medical Records, Problem lists, Laboratory Reporting, Emergency Room, Charting and many others. [30]

SNOMED CT structure ensures interoperability across software applications of diseases, treatments, etiologies, clinical findings, therapies, procedures and outcomes. The breadth and depth of the terminology, as well as its computer-readable hierarchies, enable faster, reliable and consistent retrieval of robust clinical information based on flexible queries. It also crosses maps to other medical classifications already in use, such as ICD-9-CM, ICD-O3, ICD-10, Laboratory LOINC and OPCS-4. This avoids duplicate data capture, while facilitating enhanced health reporting, billing and statistical analysis. In addition, SNOMED CT is aligned with numerous key health care standards, including HL7, DICOM, ANSI and ISO. SNOMED CT contains concepts linked to clinical knowledge to enable accurate recording of data without ambiguity. The terminology's content also includes descriptions or synonyms relating to clinical concepts, each with their own unique identifier. Links, known as semantic relationships, between clinical concepts provide formal definitions. [31]

LOINC (Logical Observation Identifiers Names and Codes) is a database and universal standard for identifying laboratory observations. It was developed and is maintained by the Regenstrief Institute, Inc., an internationally-recognized non-profit medical research organization, in 1994. LOINC was created in response to the demand for an electronic database for clinical care and management and is publicly available at no cost. It is endorsed by the American Clinical Laboratory Association and the College of American Pathologist. Since its inception, the database has expanded to include not just medical and laboratory code names, but also: nursing diagnosis, nursing interventions, outcomes classification, and patient care data set. LOINC applies universal code names and identifiers to medical terminology related to the Electronic health record. The purpose is to assist in the electronic exchange and

gathering of clinical results (such as laboratory tests, clinical observations, outcomes management and research). [32]

The LOINC database provides a set of universal names and ID codes for identifying laboratory and clinical test results in the context of existing HL7, ASTM E1238, and CEN TC251 observation report messages. One of the main goals of LOINC is to facilitate the exchange and pooling of results for clinical care, outcomes management, and research. LOINC codes are intended to identify the test result or clinical observation. [33]

NANDA (formerly the North American Nursing Diagnosis Association) is a professional organization of nurses to standardize nursing terminology that was founded in 1982 and develops and refines the nomenclature, criteria, and taxonomy of nursing diagnoses. In 2002, NANDA relaunched as NANDA International in response to the broadening scope of their membership. NANDA International published Nursing Diagnosis quarterly, which became the International Journal of Nursing Terminologies and Classifications in 2002. [34]

The NANDA International (NANDA) is a non-profit, voluntary association of nurses that promotes the nursing profession through the development, refinement, classification and use of nursing language. NANDA's membership is comprised of nursing professionals working in educational, clinical, documentation, informatics and practical fields of nursing. NANDA is recognized in the United States and other countries as the pioneer in diagnostic classification of nursing. [35]

ICD (The International Statistical Classification of Diseases and Related Health Problems)

The ICD is published by the World Health Organization. It provides codes to classify diseases and a wide variety of signs, symptoms, abnormal findings, complaints, social circumstances and external causes of injury or disease.

The ICD has become the international standard diagnostic classification for all general epidemiological and many health management purposes. These include the analysis of the general health situation of population groups and monitoring of the incidence and prevalence of diseases and other health problems in relation to other variables such as the characteristics and circumstances of the individuals affected. The ICD is revised periodically and is currently in its tenth edition.

The ICD is also used world-wide for morbidity and mortality statistics, reimbursement systems and automated decision support in medicine. This system is designed to promote

international comparability in the collection, processing, classification, and presentation of these statistics. [36]

Appendix 2 Abbreviations

IT - Information Technology
ICT - Information and Communications Technology
UPI - Unique Person Identifier
ER - Emergency Room
ID – Identifier
PIDS – Person Identification Service
MRN - Medical Record Numbers
PPSN - The Personal Public Service Number
HIS - Hospital Information Systems
LIS - Laboratory Information Systems
RIS - Radiology Information Systems
MPI - Master Patient Index
EMPI - Enterprise Master Patient Index
HIPAA -Health Insurance Portability and Accountability Act of 1996
HHS - Health and Human Services
HIV – Human Immunodeficiency Virus
SSN - The Social Security Number
NHS - National Health Service
NZHIS - New Zealand Health Information Service
NHI - National Health Index
ANSI - The American National Standards Institute's
HISB - Healthcare Informatics Standards Board
DNA - Deoxyribonucleic acid
OMG - The Object Management Group, Inc.
NASA - National Aeronautics and Space Administration
IBM - International Business Machines Corporation
SAP AG - Systems, Applications and Products in Data Processing Annual General Meeting

CORBA - Common Object Request Broker Architecture

HDTF - the Healthcare Domain Task Force

IDL - Interface Definition Language

SOA - Service Oriented Architecture

EHR - Electronic Healthcare Records

PC – Personal Computer

GEHR - Good European Health Record

EU - European Union

UCL - University College London

CEN - Committee European Normalisation

MOU - Memorandum of Understanding

ISO - International Organization for Standardization

PKI - Public Key Infrastructure

HL7 - Health Level 7

SIG - Special Interest Group

CDA - Clinical Document Architecture

Language

HTTP - HyperText XML - The Extensible Markup Language

RIM - Reference Information Model

RDBMS - Relational Database Management Systems

LDAP - The Lightweight Directory Access Protocol

DSML - Directory Service Markup Language

DN - Distinguished Name

TCP/IP - Transmission Control Protocol (TCP) and the Internet Protocol (IP)

JDBC API - The Java Database Connectivity Application Programming Interface

ODBC - Open Database Connectivity

SQL - Structured Query Transfer Protocol

JRE - Java Runtime Environment

JDK - Java Development Kit

RAM - Random Access Memory

Appendix 3 – JAVA Source Code

ConfigurationDB.java

```
import java.io.PrintWriter;
import java.sql.ResultSet;
import java.util.ArrayList;

/*
 * The structure for the tables to generate the HTML Form follows this hierarchy:
 *
 * Table: SEARCH_FIELDS
 * Field: DATABASE_NAME,  VARCHAR(10)
 * Field: TABLE_NAME,  VARCHAR(10)
 * Field: FIELD_NAME, VARCHAR(10)
 * Field: FIELD_TYPE, VARCHAR(15)
 * Field: FIELD_REQ, TINYINT(1)
 * Field: REL_ID, INT(10)
 * Field: SEARCH_GROUP, INT(10)
 * Description: The fields where the SQL Statement will be executed and the data get from.
 *
 * The system will be used in this way:
 *
 * [ SEARCH_FIELDS ] Query to get the fields per table.
 *   |
 *   |----> DATABASE_NAME Query to get the databases to create the connection.
 *   |----> TABLE_NAME Field to get the tables per database.
 *   |----> FIELD_NAME Field to get the fields name per table.
 *   |----> FIELD_TYPE Field to get the fields type per table.
 *   |----> FILED_REQ Field to get the required status of each field per table.
 *   |----> REL_ID Field to make a relationship between the different fields.
 *   |----> SEARCH_GROUP Field to create groups to asociate fields in the search
 *   |______|
 */

/**
 * This Class is intended to get the databases, tables and fields to perform a search within
 * several Databases in a transparent way for the final user.
 * Could be used to generate HTML Forms for the Search form or to perform the queries in the
 * database where the search will be finally executed.
 */
public class ConfigurationDB {

	/**
	 * Method to get the item passed as parameters from the configuration table.
	 * @param PrintWriter to output data.
	 * @param String item to get from database.
	 * @return ArrayList of Strings with the items from database.
	 */
	public ArrayList getConfItem(PrintWriter out, String item)
	{
		String query = "SELECT distinct "+item+" FROM search_fields";
		ArrayList itemList = new ArrayList();
		utilDB connectionDB = new utilDB("CONF");
		ResultSet results = connectionDB.getResults(out, query);
		try {
			while (results.next())
			{
				String str = results.getString(item);
				itemList.add(str);
			}
		} catch (Exception e) {
			out.println("getItem("+item+"): Exception caught fetching
results:\n"+e.getMessage());
			out.println("<center><form><input type=\"button\" value=\"Return to
Form\" onClick=\"history.back()\"></form></center>");
		} finally {
			connectionDB.closeDB(out);
		}
		return itemList;
	}

	/**
	 * Method to get the fieldInfo items with the rel_id passed as parameter from the
configuration table.
	 * @param PrintWriter to output data.
	 * @param int to get all the fields with this rel_id.
	 * @return ArrayList of fieldInfo with the fields indexed by rel_id.
	 */
	public ArrayList getAllbyRelID(PrintWriter out, int rel_id)
	{
		String query = "SELECT * FROM search_fields WHERE rel_id = "+rel_id;
		ArrayList itemList = new ArrayList();
		utilDB connectionDB = new utilDB("CONF");
		ResultSet results = connectionDB.getResults(out, query);
```

```
                try {
                        while (results.next())
                        {
                                fieldInfo field = new fieldInfo();
                                field.fieldName = results.getString("field_name");
                                field.fieldType = results.getString("field_type");
                                field.tableName = results.getString("table_name");
                                field.databaseName = results.getString("database_name");
                                field.fieldReq = results.getInt("field_req");
                                field.fieldID = results.getInt("rel_id");
                                itemList.add(field);
                        }
                } catch (Exception e) {
                        out.println("getAllbyRelID(): Exception caught fetching
results:\n"+e.getMessage());
                        out.println("<center><form><input type=\"button\" value=\"Return to
Form\" onClick=\"history.back()\"></form></center>");
                } finally {
                        connectionDB.closeDB(out);
                }
                return itemList;
        }

        /**
         * Method to get the fieldInfo items by database_name passed as parameter from the
configuration table.
         * @param PrintWriter to output data.
         * @param String to get all the fields with this database_name.
         * @return ArrayList of fieldInfo with the fields indexed by database_name.
         */
        public ArrayList getAllbyDatabase(PrintWriter out, String database_name)
        {
                String query = "SELECT * FROM search_fields WHERE database_name =
"+database_name;
                ArrayList itemList = new ArrayList();
                utilDB connectionDB = new utilDB("CONF");
                ResultSet results = connectionDB.getResults(out, query);
                try {
                        while (results.next())
                        {
                                fieldInfo field = new fieldInfo();
                                field.fieldName = results.getString("field_name");
                                field.fieldType = results.getString("field_type");
                                field.tableName = results.getString("table_name");
                                field.databaseName = database_name;
                                field.fieldReq = results.getInt("field_req");
                                field.fieldID = results.getInt("rel_id");
                                field.searchGroup = results.getInt("search_group");
                                itemList.add(field);
                        }
                } catch (Exception e) {
                        out.println("getAllbyDatabase(): Exception caught fetching
results:\n"+e.getMessage());
                        out.println("<center><form><input type=\"button\" value=\"Return to
Form\" onClick=\"history.back()\"></form></center>");
                } finally {
                        connectionDB.closeDB(out);
                }
                return itemList;
        }

        /**
         * Method to get all the differents search_group in the configuration table.
         * @param PrintWriter to output data.
         * @return ArrayList of Integer with all the different search_group in the database.
         */
        public ArrayList getSearchGroup(PrintWriter out)
        {
                String query = "SELECT distinct search_group FROM search_fields";
                ArrayList itemList = new ArrayList();
                utilDB connectionDB = new utilDB("CONF");
                ResultSet results = connectionDB.getResults(out, query);
                try {
                        while (results.next())
                        {
                                itemList.add(new Integer(results.getInt("search_group")));
                        }
                } catch (Exception e) {
                        out.println("getSearchGroup(): Exception caught fetching
results:\n"+e.getMessage());
                        out.println("<center><form><input type=\"button\" value=\"Return to
Form\" onClick=\"history.back()\"></form></center>");
                } finally {
                        connectionDB.closeDB(out);
                }
```

```
                return itemList;
        }

        /**
         * Method to check if one field is in a table in the configuration table.
         * @param String field_name to search for.
         * @param String table_name to search in.
         * @return boolean to check if the field exists.
         */
        public boolean checkFieldInTable(PrintWriter out, String field_name, String
table_name)
        {
                boolean ret = false;
                String query = "SELECT field_name FROM search_fields WHERE table_name =
'"+table_name+"'";
                utilDB connectionDB = new utilDB("CONF");
                ResultSet results = connectionDB.getResults(out, query);
                try {
                        while (results.next())
                        {
                                if (results.getString("field_name").equals(field_name))
                                {
                                        ret = true;
                                        break;
                                }
                        }
                } catch (Exception e) {
                        out.println("checkFieldInTable(): Exception caught fetching
results:\n"+e.getMessage());
                        out.println("<center><form><input type=\"button\" value=\"Return to
Form\" onClick=\"history.back()\"></form></center>");
                } finally {
                        connectionDB.closeDB(out);
                }
                return ret;
        }

        /**
         * Method to get from Database all the fields indexed by search_group.
         * @param PrintWriter to output data.
         * @param int search_group to search in the table configuration.
         * @return ArrayList of fieldInfo with all the fields in the table indexed by
search_group.
         */
        public ArrayList getAllbySearchGroup(PrintWriter out, int search_group)
        {
                String query = "SELECT * FROM search_fields WHERE search_group =
"+search_group;
                ArrayList itemList = new ArrayList();
                utilDB connectionDB = new utilDB("CONF");
                ResultSet results = connectionDB.getResults(out, query);
                try {
                        while (results.next())
                        {
                                fieldInfo field = new fieldInfo();
                                field.fieldName = results.getString("field_name");
                                field.fieldType = results.getString("field_type");
                                field.tableName = results.getString("table_name");
                                field.databaseName = results.getString("database_name");
                                field.fieldReq = results.getInt("field_req");
                                field.fieldID = results.getInt("rel_id");
                                field.searchGroup = results.getInt("search_group");
                                itemList.add(field);
                        }
                } catch (Exception e) {
                        out.println("getAllbyDatabase(): Exception caught fetching
results:\n"+e.getMessage());
                        out.println("<center><form><input type=\"button\" value=\"Return to
Form\" onClick=\"history.back()\"></form></center>");
                } finally {
                        connectionDB.closeDB(out);
                }
                return itemList;
        }

        /**
         * Method to get from Database the databases.
         * @param PrintWriter to output data.
         * @return ArrayList of String with all the databases in the table.
         */
        public ArrayList getDatabases(PrintWriter out)
        {
                return getConfItem(out, "database_name");
        }
```

```
        /**
         * Method to get from Database the Tables of each database.
         * @param PrintWriter to output data.
         * @return ArrayList of String with all the tables in the table.
         */
        public ArrayList getTables(PrintWriter out)
        {
                return getConfItem(out, "table_name");
        }

        /**
         * Method to get from Database the Fields of each Table.
         * @param PrintWriter to output data.
         * @return ArrayList of fieldInfo with all the fields in the table.
         */
        public ArrayList getFields(PrintWriter out)
        {
                String query = "SELECT distinct rel_id,field_name,field_type,field_req FROM
search_fields";
                ArrayList fieldsList = new ArrayList();
                utilDB connectionDB = new utilDB("CONF");
                ResultSet results = connectionDB.getResults(out, query);
                try {
                        while (results.next())
                        {
                                fieldInfo field = new fieldInfo();
                                field.fieldName = results.getString("field_name");
                                field.fieldType = results.getString("field_type");
                                field.fieldReq = results.getInt("field_req");
                                field.fieldID = results.getInt("rel_id");
                                fieldsList.add(field);
                        }

                } catch (Exception e) {
                        out.println("getFields: Exception caught fetching
results:\n"+e.getMessage());
                        out.println("<center><form><input type=\"button\" value=\"Return to
Form\" onClick=\"history.back()\"></form></center>");
                } finally {
                        connectionDB.closeDB(out);
                }
                return fieldsList;
        }

        /**
         * Method constructor of the class.
         */
        public ConfigurationDB()
        {
        }
}
```

dbTest.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Class with all the logic of the Multiple Database Search.
 * To use as servlet in the Tomcat server.
```

```
 */
public class dbTest extends HttpServlet {

	/**
	 * Default Serialization.
	 */
	private static final long serialVersionUID = 1L;

	/**
	 * @see javax.servlet.GenericServlet#init(javax.servlet.ServletConfig)
	 */
	public void init(ServletConfig conf) throws ServletException
	{
		super.init(conf);
	}

	/**
	 * Inner class to maintain ordered the search results by the matching value.
	 */
	private class searchData extends Object{
		public String result;
		public int match;
		public String database;

		public searchData(String r, int m, String d)
		{
			this.result = r;
			this.match = m;
			this.database = d;
		}
	}

	/**
	 * Class which implements the Comparator interface to compare the searchData Objects.
	 * Used to sort the final list of results by the matching value.
	 */
	private class searchResultsComparator implements Comparator {
		/**
		 * Method to compare between two searchData objects, from MAX to MIN.
		 * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
		 * @param Object the first Object to compare
		 * @param Object the second Object to compare
		 * @return int -1,0,1 for greater,equal or less than respectively
		 */
		public int compare(Object o1, Object o2) {
			searchData s1 = (searchData)o1;
			searchData s2 = (searchData)o2;
			if (s1.match < s2.match) return 1;
			if (s1.match == s2.match) return 0;
			if (s1.match > s2.match) return -1;
			return 0;
		}
	}

	/**
	 * All the logic is performed inside this doGet method.
	 * @see javax.servlet.http.HttpServlet#doGet(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)
	 */
	public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
	{
		String search = req.getParameter("search");

		res.setContentType("text/html");
		PrintWriter out = res.getWriter();

		formatHTML webpage = new formatHTML();

		if (search != null) {
			ConfigurationDB confDB = new ConfigurationDB();
			ArrayList fields = confDB.getFields(out);

			if (checkParameters(fields, req))
			{
				ArrayList databases = confDB.getDatabases(out);
				ArrayList tables = confDB.getTables(out);
				webpage.headerHTML(out,"Search Results: "+databases.size()+"
databases and "+tables.size()+" tables");
				webpage.logoHTML(out, "/pids/images/Piglowski.gif",479,110);
				webpage.titleHTML(out,"Results of the Search");
				webpage.bodyHTML(out,"<b><center>Searching for
"+fields.size()+" fields and searching in "+databases.size()+" databases and
"+tables.size()+" tables </center></b><br>");
				webpage.bodyHTML(out, "<center>Terms of Search:");
```

```
                                for (int i=0; i<fields.size(); i++)
                                {
                                        fieldInfo f = (fieldInfo)fields.get(i);
                                        if ((req.getParameter(f.fieldName) != null) &&
(req.getParameter(f.fieldName).length() > 0))
                                                webpage.bodyHTML(out, "<b>"+f.fieldName+" =
"+req.getParameter(f.fieldName)+"</b>; ");
                                }
                                webpage.bodyHTML(out, "</center>");
                                // Get search groups...
                                ArrayList searchGroups = confDB.getSearchGroup(out);
                                // Hashmap to save the query indexed by database
                                HashMap map_queries = new HashMap();
                                // Iterate over the search groups and get all the fields per
group
                                for (int i=0; i<searchGroups.size(); i++)
                                {
                                        Integer search_group = (Integer)searchGroups.get(i);
                                        ArrayList toSearch = confDB.getAllbySearchGroup(out,
search_group.intValue());
                                        // Iterate over the fields by search_group and create
the query to the database
                                        String query = "SELECT ";
                                        String table = "";
                                        String database = "";
                                        for (int j=0; j<toSearch.size(); j++)
                                        {
                                                fieldInfo field = (fieldInfo)toSearch.get(j);
                                                query = query + field.fieldName;
                                                // Insert comma in the query avoiding the last
one
                                                if (j < toSearch.size()-1) query = query + ",";
                                                table = field.tableName;
                                                database = field.databaseName;
                                        }
                                        query = query + " FROM " + table + " WHERE ";
                                        // Iterate the search fields to finish the query
                                        for (int j=0; j<fields.size(); j++)
                                        {
                                                fieldInfo field = (fieldInfo)fields.get(j);
                                                String param =
req.getParameter(field.fieldName);
                                                if (confDB.checkFieldInTable(out,
field.fieldName, table))
                                                {
                                                        if (param.length() != 0)
                                                        {
                                                                // Insert or in the query
avoiding the last one or if the parameter is empty
                                                                if (j>0) query = query + " or ";
                                                                query = query + field.fieldName +
" = '" + param + "' ";
                                                        }
                                                }
                                        }
                                        map_queries.put(database, query);
                                        // DEBUG webpage.bodyHTML(out,query+"<br>");
                                }
                                // Create an ArrayList where the results will be stored ordered
by the matching value.
                                ArrayList SearchResults = new ArrayList();
                                // Calculate the increase per match field in function the not
null parameters
                                int n=0;
                                for (int i=0; i<fields.size(); i++)
                                {
                                        fieldInfo f = (fieldInfo)fields.get(i);
                                        if ((req.getParameter(f.fieldName) != null) &&
(req.getParameter(f.fieldName).length() > 0)) n++;
                                }
                                int match_inc = 100/n;
                                int match = 0;
                                String res_str;
                                // Iterate over databases and perform the queries.
                                for (int i=0; i<databases.size(); i++)
                                {
                                        String base = (String)databases.get(i);
                                        String query = (String)map_queries.get(base);
                                        // DEBUG webpage.bodyHTML(out,"Consulta para
"+base+"<br>SQL Query: "+query+"<br>");
                                        utilDB connDB = new utilDB(base);
                                        ResultSet results = connDB.getResults(out, query);
                                        try {
                                                if (results.first())
                                                {
```

```
                                                do {
                                                        match = 0;
                                                        res_str = "";
                                                        for (int j=0; j<fields.size();
j++)
                                                        {
                                                                fieldInfo field =
(fieldInfo)fields.get(j);
                                                                String param =
req.getParameter(field.fieldName);
                                                                String result =
results.getString(field.fieldName);
                                                                res_str = res_str + "<td
style=\" font-size: 10pt; font-family: Courier; color: #547cb4; background-color:
white\"><b>"+result+"</b></td>";
                                                                if ((param != null) &&
(param.length() > 0))
                                                                {
                                                                        if ((result !=
null) && (result.length() > 0))
                                                                        {
                                                                                if
(result.equals(param)) match = match + match_inc;
                                                                        }
                                                                }
                                                        }
                                                        // res_str, base, match
                                                        searchData searchD = new
searchData(res_str,match,base);
                                                        SearchResults.add(searchD);

                                                } while(results.next());
                                                //webpage.bodyHTML(out, "</tr>");
                                                // Close the table
                                                //webpage.bodyHTML(out,
"</table></center><br/></h5>");
                                        }
                                        //else webpage.bodyHTML(out,"<center><h2>NO
RESULTS</h2></center><br>");
                                        } catch (Exception e) {
                                        out.println("Exception caught fetching results
in Database "+base+":\n"+e.getMessage());
                                } finally { connDB.closeDB(out); }
                        }
                        // If there are results print the table
                        if (SearchResults.size() > 0)
                        {
                                // Order list of results by the matching value
                                Collections.sort(SearchResults, new
searchResultsComparator());
                                // Header for the results
                                webpage.bodyHTML(out, "<h5> <br/><center><table
border=\"1\" cellpadding=\"10\"  style=\" font-size: 10pt; font-family: Courier; color:
white; background-color:#547cb4 ; line-height: 134%; text-align: justify;  margin-top:6;
margin-bottom:6; margin-left:20; margin-right:20\">");
                                webpage.bodyHTML(out, "<tr>");
                                for (int j=0; j<fields.size(); j++)
                                {
                                        fieldInfo field = (fieldInfo)fields.get(j);
                                        webpage.bodyHTML(out,
"<td><b>"+field.fieldName+"</b></td>");
                                }
                                webpage.bodyHTML(out, "<td><b>Matching</b></td>");
                                webpage.bodyHTML(out, "<td><b>Database</b></td>");
                                for (int k=0; k<SearchResults.size(); k++)
                                {
                                        searchData s = (searchData)SearchResults.get(k);
                                        webpage.bodyHTML(out, "<tr>");
                                        webpage.bodyHTML(out, s.result);
                                        webpage.bodyHTML(out, "<td style=\" font-size:
12pt; font-family: Courier; color: "+"black"+"; background-
color:"+getMatchColor(s.match)+"\"><b>"+s.match+"%"+"</b></td>");
                                        webpage.bodyHTML(out, "<td style=\" font-size:
10pt; font-family: Courier; color: #547cb4; background-color:
white\"><b>"+s.database+"</b></td>");
                                        webpage.bodyHTML(out, "</tr>");
                                }
                                webpage.bodyHTML(out, "</table></center><br/></h5>");
                        } else webpage.bodyHTML(out,"<center><h2>NO
RESULTS</h2></center><br>");
                } else
                {
                        webpage.headerHTML(out,"Parameters Error!!");
                        webpage.logoHTML(out, "/pids/images/Oops.gif",328,110);
                        webpage.titleHTML(out,"Bad parameters!");
```

```
                                webpage.bodyHTML(out,"<center>Some mandatory parameters are
missing, check the values</center>");
                        }
                } else {
                        webpage.headerHTML(out,"Bad Request!!");
                        webpage.logoHTML(out, "/pids/images/Oops.gif",328,110);
                        webpage.titleHTML(out,"Bad Request!");
                        webpage.bodyHTML(out,"<center>Are you kidding?</center>");
                }
                webpage.bodyHTML(out,"<br><br><center><form><input type=\"button\"
value=\"Return to Form\" onClick=\"history.back()\"></form></center>");
                // Close HTML Document
                webpage.footerHTML(out);
        }

        /**
         * @see javax.servlet.http.HttpServlet#doPost(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)
         */
        public void doPost(HttpServletRequest req, HttpServletResponse res)throws
ServletException, IOException
        {
                doGet(req, res);
        }

        /**
         * Method to get the color of the output depending of the matching value.
         * @param int Matching value.
         * @return String with the color in hex format to use with HTML.
         */
        public String getMatchColor(int match)
        {
                String color = "";
                if ((match <= 100) &&  (match >= 75)) color = "#00FF00"; // Green
                if ((match < 75) && (match >= 50)) color = "#98FB98"; // Less green
                if ((match < 50) && (match >= 25)) color = "#FF8C00"; // Orange
                if ((match < 25) && (match >= 0)) color = "#FF0000"; // Red

                return color;
        }

        /**
         * Method to check parameters and ensure that the mandatory ones are used.
         * @param ArrayList of fieldInfo with the fields of the search.
         * @param HttpServletRequest with the request to the servlet.
         * @return boolean to check that the parameters are fine.
         */
        public boolean checkParameters(ArrayList fields, HttpServletRequest req)
        {
                boolean ret = true;
                for (int i=0; i<fields.size(); i++)
                {
                        fieldInfo field = (fieldInfo)fields.get(i);
                        if (field.fieldReq == 1)
                        {
                                if (req.getParameter(field.fieldName).length() == 0)
                                {
                                        ret =  false;
                                        break;
                                }
                        }
                }
                return ret;
        }
}
```

dynamicForm.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
```

```
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * This class analyse some tables in the database and create a HTML Form to search in our
system
 * of databases, which can be from 1 to N databases and each one with its particular
structure
 * and fields. Some information can be omited in a production system like the type of each
field
 * and some input error recovery could be added further. Used as servlet.
 *
 */
public class dynamicForm extends HttpServlet {

	/**
	 * Default Serialization.
	 */
	private static final long serialVersionUID = 1L;

	/**
	 * @see javax.servlet.GenericServlet#init(javax.servlet.ServletConfig)
	 */
	public void init(ServletConfig conf)
	throws ServletException {
		super.init(conf);
	}

	/**
	 * @see javax.servlet.http.HttpServlet#doGet(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)
	 */
	public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
	{
		formatHTML webpage = new formatHTML();

		res.setContentType("text/html");
		PrintWriter out = res.getWriter();

		webpage.headerHTML(out,"Test Report with a number of databases");

		webpage.logoHTML(out, "/pids/images/Piglowski.gif",479,110);
		webpage.titleHTML(out,"Data from different databases");

		ConfigurationDB confDB = new ConfigurationDB();

		ArrayList databases = confDB.getDatabases(out);
		ArrayList tables = confDB.getTables(out);
		ArrayList fields = confDB.getFields(out);

		if ((databases != null) && (databases.size() > 0))
		{
			webpage.bodyHTML(out, "<center><b> Search will be in
"+databases.size()+" databases, ");
			webpage.bodyHTML(out, "inside "+tables.size()+" different tables ");
			webpage.bodyHTML(out, "and with "+fields.size()+" different
fields</b></center><br>");

			webpage.bodyHTML(out, "<p><CENTER><TABLE BORDER=\"1\" CELLPADDING=\"3\"
CELLSPACING=\"0\">");
			webpage.bodyHTML(out, "<FORM METHOD=\"GET\"
ACTION=\"http://"+req.getRemoteHost()+":"+req.getServerPort()+"/pids/servlet/dbTest\"
onSubmit=\"return validate()\">");

			for (int a=0; a<fields.size(); a++)
			{
				fieldInfo field = (fieldInfo)fields.get(a);
				webpage.bodyHTML(out, "<tr><th>");
				webpage.bodyHTML(out, field.fieldName+":<INPUT
NAME=\""+field.fieldName+"\" size=\"50\">");
				String tmp = "";
				if (field.fieldReq == 1) tmp = "YES";
				else tmp = "NO";
				webpage.bodyHTML(out, "Type: "+field.fieldType+" Mandatory:
"+tmp);
				webpage.bodyHTML(out, "</tr></th>");
			}
			webpage.bodyHTML(out, "</TABLE>");
			webpage.bodyHTML(out, "<br><br><INPUT TYPE=\"reset\"  VALUE=\"Clear
Form\"><INPUT TYPE=\"submit\" NAME=\"search\" VALUE=\"Database Search\">");
```

```
                        webpage.bodyHTML(out, "</FORM></CENTER>");
                }
                webpage.footerHTML(out);
        }
}
```

fieldInfo.java

```
/**
 * Class to use as structure to manage data coming from the database.
 */
public class fieldInfo extends Object
{
        public String fieldName;
        public String fieldType;
        public int fieldReq;
        public int fieldID;
        public String tableName;
        public String databaseName;
        public int searchGroup;

        /**
         * Method constructor.
         */
        public fieldInfo()
        {
                super();
        }
}
```

formatHTML.java

```
import java.io.PrintWriter;

/**
 * Class which extends the Object class to manage with the HTML format.
 */
public class formatHTML extends Object
{
        /**
         * Method constructor.
         * @param PrintWriter to output data.
         * @param String with the string to print in the HTML file as body.
         */
        public void bodyHTML(PrintWriter out, String body)
        {
                out.println(body);
        }

        /**
         * Method to create the header in the HTML.
         * @param PrintWriter to output data.
         * @param String with the string to print in the HTML file as title.
         */
        public void headerHTML(PrintWriter out, String title)
        {
                out.println("<html>");
                out.println("<head>");
                out.println("<title>"+title+"</title>");
                out.println("<style>");
                out.println("h3    { font-size: 12pt; font-family: Arial; color: white;
background-color:#FFFFFF ; line-height: 134%; ");
                out.println("text-align: justify;  margin-top:6; margin-bottom:6; margin-
left:6; margin-right:6}");
                out.println("h5    { color: #42628e; text-align: left; text-indent:0px;
margin-top:0; margin-right: 10; margin-left: 10;");
                out.println("margin-bottom:0 ;background-color: white; font-size:12pt; font-
family: Courier}");
                out.println("h4    { font-size: 14pt; font-family: Arial; color: white;
```

```
background-color:#42628e ; line-height: 134%; ");
                out.println("text-align: center;  margin-top:6; margin-bottom:6; margin-
left:6; margin-right:6}");
                out.println("p     { text-align: justify; margin-top:6; margin-bottom:6;
margin-left:10; margin-right:10}");
                out.println("h1    { color: white; text-align: center; background-color:
#42628e; font-size: 20pt; }");
                out.println("h2    { color: white; text-align: center; background-color:
#42628e; font-size: 20pt; font-family: Courier}");
                out.println("A.sub { text-decoration:underline; }");
                out.println("A     { text-decoration:none; }");
                out.println("</style>");
                out.println("</head>");
                out.println("<body bgColor=\"#FFFFFF\"  text=\"#000000\" link=\"#004466\"
vlink=\"#888888\" >");
        }

        /**
         * Method to insert an image logo in the HTML.
         * @param PrintWriter to output data.
         * @param String with the path to the image in the server.
         * @param int with the width of the image.
         * @param int with the high of the image.
         */
        public void logoHTML(PrintWriter out,String logo, int w, int h)
        {
                out.println("<!------------------------------------------------------>");
                out.println("<!-- Logo                                              -->");
                out.println("<!------------------------------------------------------>");
                out.println("<center><img alt=\""+logo+"\" src=\""+logo+"\" width=\""+w+"\"
height=\""+h+"\"  border=\"0\"/></center>");
        }

        /**
         * Method to insert a message as title of a section in the HTML.
         * @param PrintWriter to output data.
         * @param String with message as oversized text.
         */
        public void titleHTML(PrintWriter out, String msg)
        {
                out.println("<!------------------------------------------------------>");
                out.println("<!-- Title Message                                     -->");
                out.println("<!------------------------------------------------------>");
                out.println("<h1><br/>"+msg+"<br></h1>");
        }

        /**
         * Method to create the botom of a page in the HTML.
         * @param PrintWriter to output data.
         */
        public void footerHTML(PrintWriter out)
        {
                out.println("<html>");
                out.println("<br/>");
                out.println("<br/>");
                out.println("<!------------------------------------------------------>");
                out.println("<!--Botom line                                         -->");
                out.println("<!------------------------------------------------------>");
                out.println("<div id= \"nifty3 \">");
                out.println("<table width= \"100% \" bgcolor= \"#42628e \">");
                out.println("<tr><td wifth= \"20% \"> </td><td width= \"60% \">");
                out.println("</td><td wifth= \"20% \"> </td></tr>");
                out.println("</table>");
                out.println("</div>");
                out.println("<!--End main table-->");
                out.println("</td>");
                out.println("</tr>");
                out.println("</table>");
                out.println("</body>");
                out.println("</html>");
        }

        /**
         * Method constructor.
         */
        public formatHTML()
        {
                super();
        }
}
```

utilDB.java

```
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;


/**
 * Class to perform any database action.
 */
public class utilDB {

	// Database connection stuff
	private Connection conn = null;
	private Statement statement = null;
	private ResultSet results = null;

	private String dbHost = "localhost";
	private String dbUser="med";
	private String dbPass="hospital";
	private String strDB;

	// MySQL Connection
	private String dbPort = "3306";
	private String strConn;
	private String classForName = "com.mysql.jdbc.Driver";

	// ORACLE Connection
	//private String dbPort = "1512";
	//private String strConn;
	//private String classForName = "oracle.jdbc.driver.OracleDriver";

	// Getters/Setters dbUsername
	public String getDBuser()
	{
		return this.dbUser;
	}
	public void setDBuser(String user)
	{
		this.dbUser = user;
	}

	//Getters/Setters dbPassword
	public String getDBpass()
	{
		return this.dbPass;
	}

	public void setDBpass(String pass)
	{
		this.dbPass = pass;
	}

	// Getters/Setters DB hostname
	public String getDBhost()
	{
		return this.dbHost;
	}
	public void setDBhost(String host)
	{
		this.dbHost = host;
	}

	// Getters/Setters DB port
	public String getDBport()
	{
		return this.dbPort;
	}

	public void setDBport(String port)
	{
		this.dbPort = port;
	}

	// Getters/Setters database to connect
	public void setDB(String db)
```

```
        {
                this.strDB = db;
                // Line for MySQL
                this.strConn = "jdbc:mysql://"+dbHost+":"+dbPort+"/"+strDB;
                // Line for ORACLE
                //this.strConn = "jdbc:oracle:thin:@"+dbHost+":"+dbPort+":"+strDB;
        }
        public String getDB()
        {
                return this.strDB;
        }

        /**
         * Method to close the connection manually.
         * @param PrintWriter to output data.
         */
        public void closeDB(PrintWriter out)
        {
                try {
                        if (this.conn!=null) this.conn.close();
                } catch (Exception e) {
                        out.println("closeDB: Exception caught closing Database
connection:\n"+e.getMessage());
                        out.println("<center><form><input type=\"button\" value=\"Return to
Form\" onClick=\"history.back()\"></form></center>");
                }
        }
        /**
         * Method to get from Database the databases to search in.
         */
        public ResultSet getResults(PrintWriter out, String query)
        {
                try {
                        Class.forName(this.classForName);
                        this.conn = DriverManager.getConnection (strConn,dbUser,dbPass);
                        this.statement=conn.createStatement();
                        results=statement.executeQuery(query);

                } catch (Exception e) {
                        out.println("getResults: Exception caught during Database
transaction:\n"+e.getMessage());
                        out.println("<center><form><input type=\"button\" value=\"Return to
Form\" onClick=\"history.back()\"></form></center>");
                        closeDB(out);
                }
                return this.results;
        }

        /**
         * Method constructor.
         */
        public utilDB(String sDB)
        {
                setDB(sDB);
        }
}
```

Appendix 4 - Reference Model demographic Package

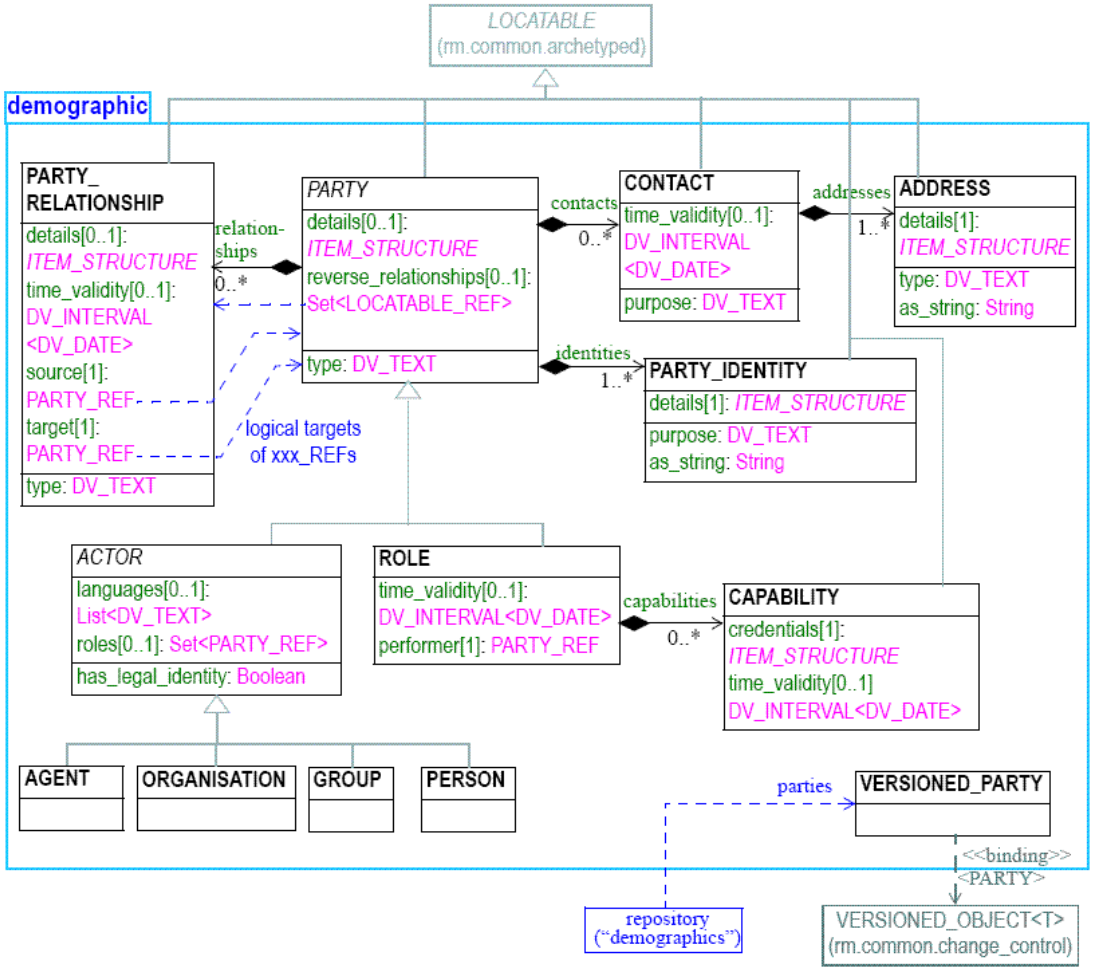


Figure 18: Reference Model demographic Package, *Source: The openEHR Reference Model, Demographic Information Model, Release 1.0.1*

Appendix 5 - Reference Model, EHR_Extract model diagram

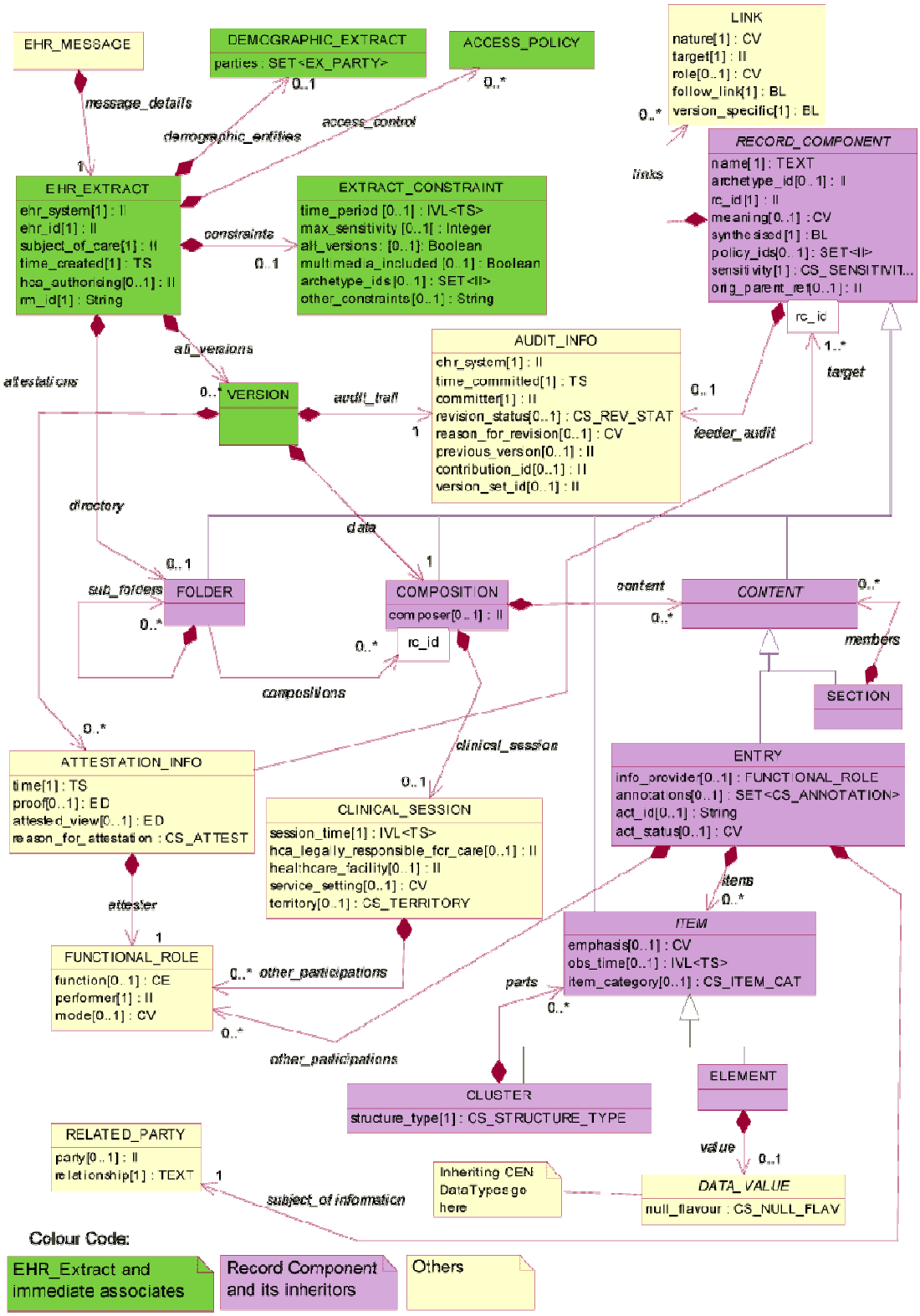


Figure 19: Reference Model, EHR_Extract model diagram, *Source: CEN/TC251/WG1 prEN13606-1:2003 Health informatics, Electronic record communication, Part 1: Reference Model.*