Master's Thesis
within the study program "Applied Computer Sciences"

Concept for the integration of pseudonymization services in a translational research database

Lars Reimann

at the department for Medical Informatics,
Universitätsmedizin Göttingen

Bachelor's and Master's Thesis
Center for Computational Sciences
Georg-August-University Göttingen

19. Mai 2010
Georg-August-University Göttingen
Zentrum für Informatik

Goldschmidtstraße 7
37077 Göttingen
Germany

Tel. +49 (5 51) 39-1 44 14
Fax +49 (5 51) 39-1 44 15
Email office@informatik.uni-goettingen.de
www www.informatik.uni-goettingen.de
Hereby, I declare that I created this thesis myself and that I have not used other sources besides the given references and aids.

Göttingen, 19. Mai 2010
Acknowledgments

First, I thank Prof. Dr. Ulrich Sax for his time and frequent support for this thesis. He greatly increased my understanding of numerous involved aspects.

Furthermore, I thank M. Sc. Krister Helbing for co-supervising this thesis.

Additionally, I thank B. Sc. Steffen Zeiss who provided an application which made further data processing possible.

Last, but not least, I thank my parents, Karl-Heinz Reimann and Gudrun Reimann for their unconditional support and encouragement to pursue my interests and M.A. Catharina Keßler for her editing help.
Abstract

The secondary use of quality assured study-data originating from clinical trials is an important step for research networks to generate new hypotheses and conduct feasibility studies.

The i2b2 software project, which is based on an open source software license, provides graphical query and analysis tools for medical data records. These tools are based on a relational database whose underlying scheme is optimized to handle large amounts of data.

Using pseudonymization within the i2b2 framework acting as a research database is an important goal to satisfy privacy protection rules and to enable additional features, for example the ability to inform patients about new medical findings if they are generated during the secondary use of study-data. Pseudonymization and de-pseudonymization of patient records can be performed using the software product Pseudonymization-Service provided by the TMF e.V.

First, this thesis summarizes the basic concepts for privacy protection within medical research networks. Continuing, it provides background information for the components of the involved software products, focusing on the i2b2 Hive. Further, it analyzes and presents possible use cases where all software components interact. To support pseudonymization on the link between an import interface handling study database exports and the i2b2 Hive, an implementation which supports the integration of strongly encrypted patient identifiers into the i2b2 database has been created. Alternatives and possible optimizations to this approach are being discussed.

This thesis concludes with a short summary, a general discussion and an outlook for possible future work.

Results show that it is generally possible to integrate pseudonymization into the i2b2 Hive. For exporting pseudonyms, additional database interactions are required.
# Table of Contents

1. **Introduction** 6  
   1.1 Problem definition ................................................................. 8

2. **Basic concepts of privacy protection** 10  
   2.1 Privacy protection situation in Germany .................................................. 10  
       2.1.1 Generic data protection concepts of the TMF .................................. 12  
   2.2 Privacy protection situation in the U.S. .................................................. 14  
   2.3 International privacy protection: Similarities / Differences ...................... 16

3. **i2b2 – background information** 17  
   3.1 Informatics for Integrating Biology and the Bedside ................................ 17  
   3.2 The i2b2 software package ..................................................................... 17  
       3.2.1 Eclipse Rich Client Platform ......................................................... 20  
       3.2.2 Oracle Database ........................................................................... 21  
       3.2.3 Apache Tomcat ............................................................................. 22  
       3.2.4 Gridsphere Portal Framework ....................................................... 22  
       3.2.5 JBoss Application Server ............................................................... 23  
       3.2.6 Spring Web Framework .................................................................. 24  
       3.2.7 Apache Axis2 Service Framework ................................................. 24  
       3.2.8 Apache Ant .................................................................................. 24  
       3.2.9 Sun Java ..................................................................................... 26  
   3.3 Applicational area .................................................................................. 27  
       3.3.1 Main features ................................................................................ 28

4. **Conceptual model** 29  
   4.1 PID, PID-Generator and Patient-List ....................................................... 29  
   4.2 Interfaces of Pseudonymization-Service ............................................... 30  
       4.2.1 PSD-Service ................................................................................ 31  
       4.2.2 SDB-Service ................................................................................ 32  
       4.2.3 FDB-Service ................................................................................ 38  
   4.3 Interfaces of i2b2 software .................................................................... 39  
       4.3.1 Database scheme ......................................................................... 39  
       4.3.2 Communication between cells ....................................................... 40  
       4.3.3 i2b2 messages ............................................................................. 42  
   4.4 Use cases / interface scenarios ............................................................... 44  
       4.4.1 PSD-Service between import interface and i2b2 Hive ..................... 45  
       4.4.2 PSD-Service between multiple SDBs and import interface ............... 48  
       4.4.3 PSD-Service between SDB and multiple provider-specific FDBs ...... 50
1. Introduction

As an umbrella organization for networked medical research, the Telematics Platform for Medical Research Networks (TMF) e.V. aims at creating universal solutions for the optimization of essential working processes [1]. Amongst its goals which are relevant for this thesis are: "Solution of questions spanning networked medical research, e.g. on the collection, processing and exchange of research data" and "Development and extension of efficient IT infrastructures and their implementation in cross-institutional networked structures" [1].

According to [2], the ongoing TMF project "IT Strategie" (German for strategy) surveys IT-infrastructures, -tools and -services which have been initiated, realized and provided by the TMF for networked research in Germany within the last years. Furthermore, it evaluates the applicability of freely available solutions to the German research environment. The project consists of several sub-items and is lead by Prof. Dr. Hans-Ulrich Prokosch of the University Hospital Erlangen.

One of this sub-items' goal is to elicit the IT components which have been developed by networked research projects in the U.S.. Especially, the software components created by Informatics for Integrating Biology and the Bedside (i2b2), a National Institute of Health (NIH)-founded National Center for Biomedical Computing (NCBC) [3], is being evaluated. The i2b2 Center is led by co-directors Isaac Kohane, M.D., Ph.D and John Glaser, Ph.D.

One of the goals of i2b2 is to broadly provide clinical investigators with the software tools necessary to collect and manage project-related clinical research data in the genomics age as a cohesive entity [4]. This main i2b2 software product is called the i2b2 Hive which is freely available under the i2b2 Software License [5], grants many rights and is comparable to an open source license.

Translational research in the medical context means that the results of the actual research being done on human subjects should benefit them more quickly. This could be expressed in any number of applicational areas. An example of the general idea of translational research would be that participants take part in a clinical trial which examines a rare form of disease. This disease has no viable treatment yet, and suitable medicament is still being searched for or is still under development. The goal would be to feed back the relevant information of the findings of the clinical trials to the participants more quickly. This goal concerns patient, research networks as well as pharmaceutical companies as involved parties. Additionally, [6] states that translational research is also being seen as a key missing
component for pharmaceutical companies to increase their return on investment. However, to broaden translational research in Germany, several privacy protection issues arise.

Existing TMF privacy management applications are the Patient Identifier (PID)-Generator [7] and the Pseudonymization-Service (PSD) [8]. The PID-Generator has been used in German research networks for some time now [9]. Since the end of 2009 the PSD is also available at the TMF e.V. [8].

The free software i2b2 was assessed by the working group TMF IT strategy because researchers lack an exploratory research tool which allows them to work on their data [10] which often originates from various sources and databases. As a prerequisite, the TMF working group has identified tasks before the software i2b2 can be considered 'ready to production' especially in the area of patient data privacy protection.

As part of the TMF-project "IT-strategy", "Transfer of the analyzing tool i2b2 to the German environment", work package 1 (WP1) has identified main problem areas which are intended to be overcome by visiting the i2b2 working group in Boston, MA in 2009 [11].

Several meetings in mid-2009 have lead to a memorandum of understanding, a collaboration between the i2b2 National center for Biomedical Computing, Brigham and the Women's Hospital and Partners Healthcare in Boston (M.D., Ph.D. I. Kohane), the Medical Informatics Department of the University of Erlangen-Nürnberg (Prof. Dr. H.-U. Prokosch) and the Department of Information Technology at the Göttingen University Hospital (Prof. Dr. U. Sax). This collaboration intends to render the i2b2 product generally useful by integrating new applications or other refinements which are generated by the previously named German parties. In return, membership in the i2b2 Academic User Group is granted and the i2b2 software is (generally) made freely available to the public. An intended long product lifetime enables entities which install and use the i2b2 software to have a tool available which can, in an advanced way, query all available existing data sources and manage the data.

As first steps, an installation routine has been set up at the University of Erlangen which facilitates the setup of of the i2b2 package for new sites. Adding a user interface for this routine is part of a current work package, together with the creation of other administrative tools for setting up i2b2 data marts and extracting hierarchical meta-data from routine clinical systems into an i2b2 conforming ontology. Additionally, the "Pathifier" platform provides a time line-based visualization based on clinical data elements.

Work Package 2 (WP2) with the title "Identity and Privacy Services" consists of two sub packages:

- WP2.1 Integration Concept for TMF PID-Generator
- WP2.2 Integration Concept for TMF Pseudonymization Service
These two packages are subject to current research and the integration concept of pseudonymization together with the aim to actually integrate pseudonymization into i2b2 are the subject of this Master's Thesis.

This would result in an enhanced i2b2 product and introduce new use cases which would be better suited for the German privacy laws. For example, the need to let patients benefit from contributing their patient data to clinical research by letting them take part in the results. However, patient data sets are not allowed to contain personal information which could lead to an identification of individuals. However, patient identification within those records is only allowed in an anonymized- or pseudonymized way. The anonymized way can, by its nature, not support the re-identification of patients and thus, the concept of pseudonymization is preferred. Additionally, the different data sets have to be stored at a different physical location [10]. For the concept of data privacy protection in this specific case the preliminary work of P. Debold [12] and the work of the authors of [13] (among others) are presented in Chapter 2.

1.1 Problem definition

According to the authors of [11] there are three key scenarios where i2b2 can be adopted.

a) the usage in disease-driven research networks  
b) the deployment as a front-end for data warehousing within hospitals  
c) the utilization of i2b2 for exploratory research of large single data sets

With the option of i2b2 to hold data from various sources and transfer information between networks which possibly contains patient identification data, it is in conflict with applicable German law. An adoption must therefore carefully consider local legal and security requirements.

On the one side there is for example the "Healthcare Team" [14] which can be taken as a synonym for entities which provide medical services for a patient and thus generate a lot of data. This is called the primary use of data. In conjunction with this data-generating side resides the electronic health record (EHR) which stores all examination data in a centralized system. The online connectivity of the EHR with a centralized system is planned to be designed in Germany by mid-2011 and it ought to replace large portions of paper-based, print-out and other costly methods [15].

On the other hand there are research institutes which not only want to access the previously recorded patient data for research purposes as a secondary use to develop new medical treatments or compare findings with historical records, but to notify the patient of new results which could be beneficial to him / her. German law strictly forbids that identification data occurs within the secondary use of data. Additionally, data has to be stored in different physical locations and/or under different organizational control.
At the interface between the two sides in this simplified scenario (in a real world application many more organizational units and additional components are present) anonymization and pseudonymization tools are to be implemented in the form of the PID-Generator and the Pseudonymization-Service (PSD).

A target group orientated graphical user interface (GUI) such as i2b2 has to integrate such methods as a requisite to become widely adopted in Germany. It is therefore intended to analyze possible scenarios how this could be accomplished with a strong focus on protecting patient privacy and retaining his / her own control over his / her medical data. Furthermore, key questions of the thesis are:

- What are the mandatory key elements of a current translation medicine infrastructure?
- Which elements are needed to connect patient care with research supporting components?
- Which modifications are needed to existing i2b2 Hive components to integrate them in the TMF IT-Strategy?

This thesis does not intent to create new concepts for privacy protection. It aims at implementing already available concepts within the components of the analyzed software components.
2. Basic concepts of privacy protection

This chapter outlines important parts of the basic concepts of privacy protection. They act as a basis to understand the later concepts. Many privacy protection concepts have been enacted as law in several countries.

Privacy designates a lawful sociopolitical norm which protects individual-related data or data transferable to individual-related data from misuse while handling or storing such data. The major aspect is to protect the personal rights of an individual [16].

Individual-related data hereby denotes particulars of a natural person which can be directly used to identify an individual.

The character of data transferable to individual-related data is that details of a natural person only identify a person if they are used in relation with data from other (often freely available) data.

2.1 Privacy protection situation in Germany

The most important concept in Germany is the informational self-determination. It describes the right of a person to basically decide when or if her / his personal information is being used.

Because of the historical development of the federal states in Germany, the federal law (Bundesdatenschutzgesetz (BDSG) [17]) regulates privacy protection together with the laws of the particular federal state. In general, the laws of the federal states are very similar and are comparable among each other. There exist only minor differences and it will not be distinguished between the laws of specific federal states. For example, institutions which are located in Lower Saxony are bound to the Niedersächsisches Datenschutzgesetz (NDSG) [18]. The paragraph 25 is of special importance as it describes the requirements for handling individual-related data within research projects.

The basic right of informational self-determination (Basic Law for the Federal Republic of Germany (GG) [19], Article 1, 2) may conflict with the basic right 'freedom of (scientific) research' (GG, Article 5). Figure 1 shows a generic overview of this problem. Generally, this conflict is overcome by a justification to a person why her / his data needs to be used. The person in question needs to sign an agreement and is informed in what manner the data is being used. Additionally, an application for a research project must give the appropriate reasons for processing and / or storing personal data. The application is then carefully verified, for example by the German Research Foundation (DFG). However,
the whole process is much more detailed. A good overview of the complete work-flow can be found in [20].

Also, it is Good Scientific Practice to use personal data only on the basis of necessity and then preferably in a pseudonymized or anonymized way.

Anonymization is the process of removing any information from a data set which can be used to identify an individual. Data is considered anonymous if it cannot be linked to a specific or determinable natural person [21]. The most important attributes for example are full name, address, date of birth, social security number. However, complete anonymization is often very difficult to achieve. It may be possible for an attacker to cross-link obtained non-identifying data with publicly available data like telephone books, school registers or information available on the Internet. Often, such an attacker uses electronic data processing equipment to perform such a task. The more computing power and storage space such an attacker might posses, the easier it may be to analyze large data sets and find information to re-identify an individual.

The BDSG is therefore based on the the principal of commensurability thus is also takes the technical development into account. A possible translation of paragraph 3, (6) could be:

"Anonymization changes individual-related data in such a manner, that particulars about personal or material circumstances cannot, or can only with a disproportional effort of time, expenses or manpower be linked to a specific or determinable natural person."
Methods used for anonymization are for example automated removal of protectable data fields, generalization, introduction of errors or the use of mathematical one way methods for obtaining a code (like one-way hash functions). Nevertheless, the complete anonymization of data required for medical research is difficult because it often contains detailed analyses of illnesses.

**Pseudonymization** could be described as a laxer form for anonymization as it is possible, in some predefined cases, to make assumptions about the individual to whom the pseudonym belongs.

"Pseudonymization is the replacement of names and other identifying criteria with a code for the reason to eliminate or severely exacerbate the possibility to identify the person in question. The code is the so-called pseudonym." [21]

A real world example would be an Internet user who does not want to register on an Internet platform with his / her real name and thinks of an alias which does not allow a third person to make conclusions about the users’ original name. However, in time he might submit additional information which may allow a third party to dissolve the pseudonym.

Within medical research networks, the process is quite similar. However, a pseudonym for a patient is generated using a cryptographic function. A trusted party who is already authorized to work with person-related data can generate an alphanumerical code for a specific person. A third party or an attacker cannot gain the secret from which the pseudonym was being derived. This assumes that the attacker cannot gain physical access to the secret and that the process of generating such pseudonyms is only used by a trusted party.

An additional approach has theoretically evaluated how a patient could become more involved in controlling his / her own data [13]. The concept centers around a hull security scheme where the patient can grant or revoke access to individuals or organizations. A security token grants access to a specific hull¹ and can be shared for the purpose of backup for example among relatives. It also analyzes the case of patients who are unable to communicate or are deceased. In such cases previously authorized backup tokens can be used to access or revoke patient data.

### 2.1.1 Generic data protection concepts of the TMF

The bundling of knowledge and expertise into so-called competence networks has become a common practice in Germany to overcome barriers of federal states. A competence network concentrates efforts of mostly regional / supra-regional acting companies. Matters concerning all parties of a competence network are often discussed within an umbrella

---

¹ The hull security scheme also reminds of the general concept of hierarchical protection domains (also protection rings) in computer security [22], [23].
organization. The bundling of networks also generates a stronger international reception which is essential for commercial success [24]. Competence Networks and their umbrella networks are often allowed to apply for funding at offices of the Bundesministerium (German for Federal Ministry), for example the Federal Ministry of Education and Research (BMBF) [25].

The medically-oriented competence networks focus on cooperation between fundamental researches and hospitals which research on the same diseases. Thereby, they generate synergies by sharing resources [26].

In 2006 the TMF has released a book on generic data protection for medical research networks, written by C.-M. Reng and P. Debold [26]. Since then their concepts have become widely accepted and adopted, because they match for a large number of practical scenarios. They also address the usage of the TMF tools, Patient List, PID-Generator and Pseudonymization-Service. Since both generic concepts and their similarities have already been mentioned in [27], this thesis will only summarize the concepts and methods which are prerequisites for the next chapters.

Since the i2b2 Hive is intended as a research database, the generic concept B (pseudonymization of research data within scientific orientated research networks) applies.

A core element of patient data management is the separation of basic patient data which directly identifies a person (IDAT) and medical data (MDAT), which has been generated by examinations and/or clinical trials. This splitting has several advantages. First, it is clear that identifying data (IDAT) is not allowed to be exported to third parties for research purposes. By grouping IDAT under a domain the important nature of this type of data is emphasized. Second, it allows the separate processing of IDAT and MDAT which is needed by the following concepts. Further, it reduces the amount of data which eventually has to be transferred (securely) over a network for processing as IDAT mainly consists of character data (for example UTF-8) and MDAT may contain additional examination-related data like binary images, laboratory results, etc..

The method of separating the informational power also plays an important role. The general idea is to also physically split data sets and only allow authorized parties to (manually) link data sets together. A popular example are separate databases, for example a database holding only data of type I and a second database, physically separated and possibly under different organizational control, holding only data of type II. An (internal or external) attacker has only access to one type of data and a possible misuse is minimized. Studies predict that within corporations employees are the main source for illegal extraction of data with 20,3% of the given ways for data theft [28].

Already in operation within networks is the so-called 'Patient-List', which acts as a mapping between IDAT and a patient identifier string. A PID also has multiple functions and therefore advantages. The PID represents the first stage of pseudonymization as it is generated from basic patient data which should be derived from data already electronically
available from insurance cards [26]. An organizational unit which is allowed to handle patient data can generate a PID from IDAT using the PID-Generator. A unique person should hereby translate to a unique identifier for the purpose of quality assurance. This process is not always deterministic due to similar names, dates of birth, etc. of patients. A manual screening process warns about such collisions to minimize the occurrence of synonyms and homonyms. A large enough target set is supported for the possible mapping of many patients (30-Bit = $2^{30}$).

If a subject takes part in several clinical trials, possibly in different locations, an identical PID assures that medical data of the patient is comparable while protecting the patients' privacy on a basic level. The patient list can be used network-wide or locally within an institution.

A second level of pseudonymization can be used when exporting data to a third party for research purposes [26]. The PID is transformed using additional cryptographic methods to a pseudonym of the 2nd order (PSN). Identical PIDs hereby also translate to the same PSNs. The PSN is generated via a cryptographic key which is stored on a smart-card. Securing this secret is manageable when, for example, only a single person is authorized to enable the generation of pseudonyms and the smart-card is stored in a secured physical location. De-pseudonymization is only possible by using this inverse cryptographic function which is a hard problem for an attacker as it would require a large amount of computing and / or financial power. It therefore complies with the concept of commensurability mentioned earlier. A possible scenario for using a stronger pseudonym is for example the long term storage of medical data for historical reasons but also other scenarios are possible. Additional use cases will be addressed in Chapter 4.4. Details of the Pseudonymization-Service and its interfaces will be presented Chapter 4.2.

Work on privacy protection is an ongoing process and the generic concepts are continuously being revised by the TMF. With the help of feedback from research networks which have implemented generic data protection concepts a revised modular approach is being developed, as many networks do need such a concept because they are using a mixture of parts present in both concepts [29]. For example, for the integration of biobanks containing tissue samples and / or genetic information about a patient, an additional concept has been presented in [30].

### 2.2 Privacy protection situation in the U.S.

In the U.S. protecting patient privacy also plays a significant role. The important parts will be presented in short in this chapter. In 1996 the 'Health Insurance Portability and Accountability Act' (HIPAA) was enacted, mainly protecting employees from loosing health insurance coverage when loosing jobs and reducing bureaucracy by introducing electronic health care standards.
HIPAA contains a 'privacy rule' which specifies which data is considered individuals’ health information by defining the term protected health information (PHI). A list of 18 identifiers was stipulated as PHI. Among them are for example names, addresses, dates (a full list can be found under [31]). The privacy rule was issued by the Department of Health and Human Services (HHS) to fulfill HIPAA requirements. 'Covered entities' in this case are institutions which generate and handle patient information. Covered entities have to fulfill certain regulations like disclosing stored information to an individual in a limited amount of time. The Office for Civil Rights (OCR) has responsibility for implementing and enforcing the Privacy Rule and can assign money penalties in case of an infraction [32].

A security role within HIPAA basically addresses the same issues concerning patient privacy protection by defining three key elements:

- Administrative Safeguards
- Physical Safeguards
- Technical Safeguards

The i2b2 software is consistent with the HIPAA privacy rule by making only the limited data set available to most of the Hive. The limited data set consists for example of the following (potentially identifying) information:

- admission, discharge, and service dates;
- dates of birth and, if applicable, death;
- age (including age 90 or over)
- five-digit zip code or any other geographic subdivision, such as state, county, city, precinct and their equivalent geocodes (except street address).

A more recent statute is the Genetic Information Nondiscrimination Act (GINA) of 2008. It mainly exists to prohibit the improper use of genetic information in health insurance and employment and bars employers from using individuals’ genetic information when making hiring, firing, job placement, or promotion decisions [33]. This act takes into account, that genetic information is becoming more easily available after the success of the Human Genome Project (HGP) and laboratory tests often include (at least parts) of genetic testing.

Further, the Patient Safety and Quality Improvement Act of 2005 creates Patient Safety Organizations (PSOs) to collect, aggregate, and analyze confidential information reported by health care providers [34]. In principle, this and the above regulations should also alleviate fear or hesitations when patients should give their personal information in the context of patient care. In a worst case scenario a patient does not utilize patient care because she / he fears that individual-related data is not safely handled.

Additionally, the FDA has released guidelines on good clinical practice and the conduct of clinical trials [35]. They are not mandatory but they contain methods which are generally
useful throughout the clinical sector. Some of them also address privacy protection but they mainly contain organizational guidelines protecting the integrity of operational sequences, for example the usefulness of a clinical trial.

### 2.3 International privacy protection: Similarities / Differences

The Data Protection Directive (95/46/EC) of the European Union (EU) [36] was approved on 1994-10-24. Since then, it has been an important step to legally bind member states of the EU. However, an EU Directive requires member states to achieve a particular result without dictating the means of achieving that result [37]. Therefore, member states have a certain liberty when implementing a directive within national laws.

It also contains special rules for privacy protection concerning scientific research. Mainly, Article 6 contains the statements concerning scientific research:

- Member States shall provide that personal data must be:
  - (b) collected for specified, explicit and legitimate purposes and not further processed in a way incompatible with those purposes. Further processing of data for historical, statistical or scientific purposes shall not be considered as incompatible provided that Member States provide appropriate safeguards.

However, the article has a broader scope than the German BDSG. It includes the usage of individual-related data for statistical purposes.

Additional directives which address privacy protection are:

- 97/66/EC - protection of privacy in the telecommunications sector
- 2002/58/EC: privacy and electronic communications

Generally, U.S. and European privacy protection regulations are comparable. The U.S. however, defines more precise regulations like the PHI, whereas in Europe, especially Germany, regulations often leave more room for interpretation. Definitions of specific regulations are often left for the courts to decide, analyzing specific cases which may act as a future reference.

In the U.S. strong efforts are made to protect patient privacy in electronic records by automatically removing identifying information using natural language processing (NLP). For example, i2b2 also includes and is working on integrating a dedicated NLP module [38].
3. i2b2 – background information

So far, two workshops have been held, addressing the i2b2 software package installation and setup as well as presentation of features and plans for implementing an i2b2 infrastructure in Germany. The 1\textsuperscript{st} workshop was held in Erlangen at the University Hospital Erlangen, Medizinisches Zentrum für Informations- und Kommunikationstechnik [39] in December 2009. The 2\textsuperscript{nd} workshop took place at the offices of TMF e.V. in Berlin in February 2010. As a follow up action, a mailing list and an online forum have been initiated. They are accessible on the websites of the TMF upon request.

Concerning pseudonymization, slides have been prepared for both workshops by the author of this thesis with the support of K. Helbing, U. Sax and S. Zeiss. The slides mainly contain details about the Pseudonymization-Service and preliminary ideas how the PSD could be integrated into i2b2.

3.1 Informatics for Integrating Biology and the Bedside

Informatics for Integrating Biology and the Bedside itself is a National Center for Biomedical Computing. It is part of several NCBCs which are distributed all over the U.S. [40]. I2b2 in general has a comprehensive agenda which is depicted together with core tasks on its website [3]. This thesis centers around the software developed by the i2b2 Center spanning several 'cores':

"The i2b2 Center is developing a scalable computational framework to address the bottleneck limiting the translation of genomic findings and hypotheses in model systems relevant to human health. New computational paradigms (Core 1) and methodologies (Cores 2) are being developed and tested in several diseases (airways disease, hypertension, type 2 diabetes mellitus, Huntington's Disease, rheumatoid arthritis, and major depressive disorder) (Core 3 Driving Biological Projects)."[3]

3.2 The i2b2 software package

The i2b2 software package basically consists of three layers. The most basic layer consists of a relational database for storing various types of information needed for the operation of i2b2. The relational database management systems (RDBMS) which can be used for this layer are the Oracle Database and the Microsoft SQL Server. This layer is called the i2b2
Data Mart. It holds several storage repositories required for the operation of the i2b2 Hive. For example, data used for the operation of specific cells and project-related data.

Theoretically, other RDBMS could also be used for i2b2 data storage as discussed at the 2nd i2b2 workshop. The possible RDBMS discussed namely were MySQL² and PostgreSQL³ because they were seen as more cost-effective by the participants. MySQL is open source under the GNU General Public License (GPL)⁴ and offers different levels of enterprise support. PostgreSQL is mainly known as a scientific database which is being used in many universities. It is mainly known for its high performance [41] especially under Solaris⁵, but Solaris is optimized to operate databases in general.

The second core layer is called the i2b2 Hive. It consists of a collection of web-services running on the Apache Tomcat servlet container [42] and the Java Platform Enterprise Edition (EE) Application Server JBoss. The core 'cells' perform different tasks with the underlying data from the clinical research chart (CRC) which relates to all kinds of clinical data.

![Figure 2: i2b2 Hive [43]](image)

Figure 2 shows the i2b2 Hive [43] with its dark-blue core cells. Light-blue cells are core-associated cells and cells with white background are optional and/or long-term aims with work still in progress. However, some core-associated cells like 'Natural Language Processing' are also still being researched upon.

---
³ [http://www.postgresql.org](http://www.postgresql.org)
⁴ [http://www.gnu.org/licenses/](http://www.gnu.org/licenses/)
The third layer consists of client programs which send requests to the i2b2 Hive and query cells for different types of data, the presentational layer. Up to this date, there are two client suites available. The i2b2 Workbench, a User Interface (UI) based on the Eclipse Rich Client Platform (RCP) [44], and the i2b2 Web Client using Ajax programming techniques [45]. Using the three-tier architecture described below, additional client programs are also an option. A possible future scenario might be the querying of patient-related data by mobile clients.

Figure 3 shows a typical, so-called three-Tier architecture where the business logic is located in a separate tier (middle). Data is queried and displayed on the clients (left tier). A RDBMS is the foundation of this architecture, storing basic data in its relations (tables). This model follows established model for Enterprise Eclipse RCP [46] with a presentation tier on the client (Eclipse based), an application server containing business logic and a database tier for storing the underlying data the business tier accesses.

Figure 4 depicts the same architecture now applied to the i2b2 components. So far, as clients for the presentation layer, the i2b2 Workbench and the Ajax Web-Client have been implemented. The business logic completely resides within the web-services of the i2b2 Hive. Clients exchange data with the i2b2 Hive via Representational State Transfer (REST) or the Simple Object Access Protocol (SOAP)\(^6\). SQL queries and replies to and from the database layer, which may contain an Oracle or Microsoft SQL database server, are carried out by the Java Database Connectivity (JDBC) Application Programming Interface (API).

\(^6\) For SOAP/REST see Chapter 4.3.2
3.2.1 Eclipse Rich Client Platform

The i2b2 Workbench, the primary application which is used to access the i2b2 Hive, manage and display search results, is implemented using the Eclipse Rich Client Platform. The Eclipse RCP is a byproduct from the Eclipse Integrated Development Environment (IDE). It is used to develop general purpose applications and is available for multiple platforms (e.g. Microsoft Windows, GNU/Linux and Mac OS). Instead of navigating through and displaying source code the RCP uses connections to web-services to display the desired data. In this context, the i2b2 Workbench displays several views which query and display data from different cells within the i2b2 Hive:

- Navigate Terms: A tree based view which allows to browse predefined classifications (also called: Ontology View).
- Find Terms: Provides a full text search among the Navigation Terms.
- Query Tool: Navigation Terms can be dragged and dropped here. Using logical AND / OR / EXCLUDE operators, the result set can be constrained.
- Previous Queries: Automatically stores issued queries with a meaningful name and time-stamp.
- Workplace: Relevant Queries can be dropped here to store them for later use or to share them among other group users.
- Time-line View: Automatically displays the resulting patient population, showing documented facts in a time-line based view.
- Export Data: Exports the results set for further processing or analysis.
- Import Data: Provides the possibility for higher-level users to import data using text files and manually mapping data to the corresponding table columns used in the underlying database.
Informatics for Integrating Biology and the Bedside also provides developers with the source code [47] and the necessary documentation [48] to integrate the latest version of the i2b2 Workbench into a standard Eclipse release. Using this approach, developers are also encouraged to develop custom plug-ins which add new functionality to the clients. For example improved visualization or analysis of patient-related data.

Database access is moved to the i2b2 Hive which minimizes the number of simultaneous database connections from clients compared to a client / server architecture. Instead of opening a connection for every user, the Hive is able to reuse already opened connections. This approach also shortens the time a client has to wait for resulting data as connections to a database are already open and stay in this state. This also enables the design goal of the CRC to be able to quickly handle databases which contain "hundreds of millions of rows" [49]. The keyword for this feature is called connection pooling which is generally found in enterprise-level applications.

### 3.2.2 Oracle Database

The Oracle Database (Oracle) is the database of choice at both the Universität Göttingen (UMG) and the Universität Erlangen. Therefore, a brief outline of its capabilities and usage within i2b2 is given here.

Besides the Structured Query Language (SQL) Core Standard SQL:2003 [50], the Oracle Database 10g (which is the recommended version for i2b2 v1.3) supports a large number of features. Among them are for example stored procedures and functions and the proprietary extension language 'Procedural Language/Structured Query Language' (PL/SQL) which can be directly saved within the database. PL/SQL can perform multiple computational tasks within the database including loops, conditional statements and exceptions.

Data inside the database is stored in the form of tablespaces. Different tablespaces (e.g. SYSTEM, USERS) map to different storage locations on the disk and may need different access privileges. For example, the i2b2 Hive uses the tablespace USERS per default.

Data imported into a database for use in i2b2 during a development process may require to provide the necessary undo statements to remove or rollback the data in case of a failed data import. For example, it was not possible to empty a test database again after the i2b2demodata (v1.3) had been imported. Due to the large data set, a manual removal also failed. To circumvent this problem during future import processes, database transactions could possibly be used as they allow a complete rollback of transactions on a database level.

The Oracle Database Express Edition (XE)\(^7\) was used during the work with the i2b2 software as it was immediately available. Oracle grants free use of this database as it only

supports a limited amount of hardware. For example, it used only one Central processing unit (CPU), a maximum of 1 gigabyte of computer memory and a maximal data-file size of 4 gigabyte but it has no restriction in its features. A licensed version of Oracle does not have these constraints. The Oracle database is the only closed-source component within the i2b2 software complex.

### 3.2.3 Apache Tomcat

Apache Tomcat [42] in general is used to provide access to web applications using servlets and Java Server Pages (JSP). Servlets are the Java equivalent for the well-known Common Gateway Interface (CGI) server technology which can be used to programmatically respond to Hypertext Transfer Protocol (HTTP) requests. They process the request and formulate a situation specific response. JSPs are sitting on top of servlets and contain a mixture of markup and scriptlets (blocks of Java code). Both concepts act together and are used to dynamically generate web content or other (non-visual) messages on the basis of the Hypertext Markup Language (HTML) or the Extensible Markup Language (XML).

In the i2b2 software v1.3 Apache Tomcat is used to provide authentication via the Project Management (PM) cell which is implemented as a web application, and access to the Gridsphere Portal Framework. The version of Apache Tomcat used in i2b2 v1.3 is 5.5.26.

### 3.2.4 Gridsphere Portal Framework

The Gridsphere Portal Framework was developed at the Albert Einstein Institute [51]. It was designed to enable developers to manage grid resources and visualize applications in so-called GridPortlets. It further supported personalization of content, single sign on and content aggregation from different sources [52]. It also aimed at unifying parts of organizational entities which share a common interest in specific grid resources under the concept of Virtual Organizations (VO). Members of a specific VO may contain people or research groups from different real world organizations like universities or corporations. As they need the same access to a grid resource (e.g. weather forecast simulations, genome analysis) and share common concerns they are grouped together to a (multinational) VO.

However, the original website of Gridsphere Project⁸ is disconnected for an unknown period of time. Discussions on the developer- and user mailing-lists for Gridsphere have also discontinued since 2006. No announcement of the Gridsphere team could be discovered as to why the project has been discontinued. Presumably to an end of funding project members have taken up other positions and work on grid projects continues for example within the German D-Grid initiative⁹. This may also be the reason why newer versions of the i2b2 software have removed Gridsphere from their software package and use alternate methods to manage user-, group- and other resources (see Chapter 6).

---

⁸ [http://www.gridsphere.org](http://www.gridsphere.org)
In the i2b2 software framework (v1.3), Gridsphere is used for managing users, groups and assign roles to them. It is also used to configure the Uniform Resource Locators (URL) of i2b2 Hive cells. Cells and their parameters have to be entered at specific locations. The exact process is described in the corresponding installation guide [53].

The Gridsphere configuration is error-prone as also stated in [27]. Also, efforts have been made by S. Mate at the Universitätsklinikum Erlangen to automate the configuration for the purpose of evaluating the i2b2 software. However, a custom configuration cannot easily be automated as it would contain (possibly multiple) custom projects and a user / group configuration which reflects the desired use cases of i2b2. For example, it has yet to be determined if an existing authentication scheme within a hospital or research institute can be mapped to an equivalent within the i2b2 software, or if additional accounts have to be generated. If the number of users is manageable during an introduction phase of the i2b2 software, the usage of dedicated accounts would be time-saving.

Gridsphere also has problems with changing network interfaces inside a virtual machine running on Ubuntu 9.10. During the work with the i2b2 Hive it became necessary to exchange VMs with S. Zeiss for development purposes. However, once configured and copied to another host operating system, Gridsphere stopped showing the correct core cells. It apparently registers cells for a specific network interface. If parts of this interface change (e.g. Internet Protocol-, or hardware addresses), Gridsphere becomes confused and stops displaying Hive cells and does not allow to reconfigure these interfaces. But as Gridsphere is completely replaced within the next version of i2b2 (v1.4), no efforts have been made to alleviate this problem.

3.2.5 JBoss Application Server

The JBoss Application Server acts as a middleware in the i2b2 software. Middleware actually corresponds to the 2nd (middle) tier of the three-tier architecture described above. It contains the business logic within deployed web-services which are contained in Java Archives (e.g. Axis Archives, .aar). Multiple web-services running on JBoss together form the functionality of the i2b2 Hive. The term 'application server' paraphrases the allocation of resources needed to offer Java enterprise-level services via a network. In the case of the i2b2 software the services do not offer the display of a dynamically created websites. It rather provides the result set of patients which has been previously queried via XML messages.

JBoss provides additional enterprise-level functions, for example clustering, load balancing and failover management which are out of scope for this document. The JBoss

---

10 http://www.ubuntu.com
version used within the i2b2 Hive (v1.3) is 4.2.2. As of now, the latest stable of JBoss is 5.1.0 dated 2009-05-23.

3.2.6 Spring Web Framework

The Spring Web Framework basically fulfills the same or sometimes extended functions of the Enterprise Java Beans (EJBs). EJBs are the programmatic models a developer can use to implement the business logic. They have been introduced to the Java EE architecture to provide reusable functions and methods which are commonly needed to implement applications. A 'Enterprise Bean' is another modern code word for managed objects in Java. So-called 'Containers' try to abstract the methodology further by handling common tasks like object creation, the invoking of constructors and configuring object (for example via XML-based configuration files). They therefore support the paradigm of creating reusable code.

The Spring Web Framework is considered to be more lightweight in certain areas than Enterprise Java Beans. It makes strong use of the method of 'Inversion of Control'. Instead of a client program incorporating the knowledge about dependencies on the server side into its code, the coupling of how classes and functions act together is moved to the framework on the server side. This is done by generally using abstract interfaces which inherit only from common objects. If internal dependencies are altered, for example due to code re-factoring or introduction of new functionality, changes need only be made within the framework on the server side. The client program does not need to be changed, except when entirely new public interfaces are introduced. Exposing only a limited number of functions to the client, the client-side code has a differentiated way of communicating with the server.

Both, EJBs and the Spring Web Framework are used within the i2b2 core server. Spring also provides integration with the Apache Axis web services framework.

3.2.7 Apache Axis2 Service Framework

Apache Axis2 is the second major branch of Apache Axis whose revised architecture supports both SOAP and REST style connections to web-services. These are the two methods a client can use to communicate with an i2b2 cell and the cells among each other. Among other features of Apache Axis2 for deploying and managing web-services is the support for the Web Services Description Language (WSDL) which is used to provide a machine-readable and human-readable format to provide web-services over networks. It automatically exposes the available services on a target machine including network addresses and ports. Within the enterprise-level application i2b2, WSDL acts together with an additional file (services.xml) which specifies the receiver (class) of a message and XML-schema(s) to verify the correctness of the XML-files.

http://www.jboss.org/jbossas
3.2.8 Apache Ant

Apache Ant is a Java library and command line tool whose mission is to drive processes described in build-files as targets and extension points dependent upon each other [54].

This general description lets the reader assume that Apache Ant (from now on called Ant) can be used to perform a large variety of tasks. This is in fact the case. Ant is primarily in use to build applications. It assists in the tasks which arise when the source code of an application is to be translated into its object code, byte code or other binary code. Although Ant is mainly used to build applications based on Java, it is not limited to this particular programming language.

Ants is used throughout the discussed software components. For example, all cells of the i2b2 Hive use Ant tasks to compile the i2b2 source code and also perform additional tasks, like inserting data into databases via script execution. Ant is also used in the distribution of the Pseudonymization-Service.

The usage of Apache Ant, was also suggested for the application i2b2Dataimport which is currently being developed by S. Zeiss at the Universitätsmedizin Göttingen. Exemplary Ant scripts have been provided for the purpose of further simplifying the data import process into i2b2. The desired goal would be an application which queries project-specific data (e.g. project name, project domain, database user name, etc.) and uses these variables to perform an automated data import.

Build-files are XML-based and thus can be edited with a normal text editor or an XML-editor which may have additional functionality when working with markup languages.

Another standpoint for using Ant is that today's development of applications take place under heterogeneous circumstances. Development is performed on different platforms, under different Operating Systems (OS) and with the help of various integrated development environments. A current approach during development cycles, is often to export or distribute a snapshot of an application by exporting the current version out of the IDE. The source contents of the application are often packed into an archive and then distributed via the Internet. However, if this package arrives at the target for enhancement, auditing or bug-tracking, certain problems can arise. For example, the settings, on the source machine where the primary development takes place, are often incompatible with the target machine. Although Java IDEs bring their own Java Runtime Environment (JRE), this is not mandatory. Additionally, it is almost certain that (at least some) environment variables are incompatible with the target machine.

Libraries which are included on the source machine to provide additional functionality are normally imported from their standard position within the file system (after they have been manually installed or installed by using a package system). If snapshots are distributed in the mentioned manner, it could occur that important libraries are omitted in the package. Using Ant, libraries normally reside in the /lib folder in the root directory of a project.
Using this method, they can be included from this position in the file system. For a developer it is easy to grasp which libraries the project depends on and all necessary files are included in a source distribution.

Another example is the Ant `basedir` which describes the base directory from which all path calculations are done [55]. An analysis of the Ant environment of the i2b2 CRC-cell (v.1.3) has shown that this cell does not invoke the `basedir` correctly, so that it is not possible to build the CRC project outside the directory where the source files reside in. The problem has been reported via the i2b2 bug-tracker. However, this also shows the complexity of a correct build environment within a large project.

An IDE certainly brings all the prerequisites to build and distribute a software package in the correct way. But because there are often no easy guidelines a developer can follow, there are many states an interim version can be in during development.

Therefore, a solution to the problem can be to integrate Ant at the beginning of a development process and develop an application around a fully functional Ant build environment. Ant is supported by most IDEs and additionally usable via the command line.

### 3.2.9 Sun Java

The object-oriented programming language Java has been developed by Sun Microsystems since the 1990s. In 2010 a survey [56] determined that 70% of 53 participating computer science institutes in Germany teach Java as an introduction to computer programming.

The specialty of Java is the early use of a Virtual Machine (VM) which executes compiled Java Byte-Code. This approach made Java a favorite language for cross-platform programming as the VM was available for numerous platforms. Arguments that Java executes applications more slowly than other, native programming languages have been overcome by the general availability of more computational power. Just-in-time (JIT) compilers have further diminished this difference [57].

A Java Development Kit (JDK) provides developers with the necessary tools to design applications (for example a Java compiler). The Java Runtime Environment is needed to execute compiled byte-code on a target platform. A JDK includes the JRE. The JRE is needed in i2b2 to execute Apache Tomcat, the JBoss Application Server and the i2b2 Workbench. The main Java version used for the i2b2 Hive v1.3 was version 5.0 (also known as Java v1.5). The newer version 6 of Java was incompatible and resulted in errors within the i2b2 core cells.
3.3 Applicational area

The i2b2 software can be used in a variety of scenarios:

i. Near or as addition to a data warehouse

ii. Point of aggregation of multiple clinical trials

iii. Project-based processing and analysis of single data sets.

With the property of being completely open source and the fact that several large institutions within the U.S. are already using a production version of the i2b2 software, it makes the application of i2b2 a theoretical alternative to a data warehouse. In the U.S., an infrastructure already exists which for example handles the i2b2 software setup and installation. This infrastructure is not yet present in Germany. Due to the open source license is is possible to analyze the inner workings of the i2b2 server and possibly add missing parts or database tables. Storing data within the predefined i2b2 database tables would make it possible to access the database with additional tools. For example for the use of storing additional patient-related data which is needed in a particular hospital or research network. However, due to the complexity of the i2b2 software a dedicated programming personnel would be required who can also react to software upgrades and software changes.

For the second scenario (ii) a Java-based import tool is currently being developed at the Universität Göttingen. The goal of this tool is to provide the means to automatically import CDISC-ODM-based study data into the i2b2 Hive for secondary use. Details for this import mechanism are available at the UMG. However, the specific import method of parsing and reshaping available structured study data to be compatible with the i2b2 Hive database schema is also possible for study data of a different format. The reuse of study-data is of great possible benefit for researchers as it has often undergone a quality management process. Once the data has been imported into the i2b2 Hive, researchers can perform a visual (pre-) processing of the available data and formulate research questions with the help of the ontology browser and other aids integrated in i2b2 Workbench. They can also assess the number of potential patient data sets available for research. Nowadays, these first steps are often performed by simpler means, for example with the help of spreadsheet processing programs or by working on semi-structured data sets which have been exported from other sources. The i2b2 software framework provides a much more intuitive way of drafting research questions. It could also be used to heighten the quality of research proposals.

The third scenario (iii) whose scope is smaller than the others could still be beneficial to project-based research questions. If the import interfaces that facilitate the data usage in

---

12 http://www.cdisc.org/odm
the direction of i2b2 have been created and discussed, the effort for using i2b2 within a single project would be considered low. If an underlying database is available, the i2b2 software could be setup in an isolated installation. Automated example scripts for installing the i2b2 software core have been created by S. Mate within the scope of [27]. If visualization methods which are needed for a specific project are not (yet) available, they could simply be developed and incorporated into the i2b2 Workbench as a new visualization plug-in. Such efforts are also supported by the i2b2 developer team as it increases the usability and usefulness of the i2b2 software for other parties as well.

3.3.1 Main features

Once the i2b2 Workbench has been properly installed [58] and configured by supplying the correct domain and URL parameters within the i2b2workbench.properties file, it acts as a main starting-point for displaying patient result sets and analyzing them.

Additionally, the i2b2 Workbench provides 17 different methods of visualization which have already been presented in detail in [27].
4. Conceptual model

This chapter will present the necessary technical details of the involved components to identify possible starting points and interfaces which can possibly be used to integrate pseudonymization. Additionally, it will append information on how the specific services operate to facilitate future usage of the associated software pieces. It will specify the most practical scenarios as well as theoretical integration within the i2b2 Hive.

4.1 PID, PID-Generator and Patient-List

The PID-Generator is used to manage patient-related identifying information in conjunction with a patient list.

Patients who participate in a study / clinical trial have to sign a consent form first, which allows the involved institutions to acquire personal and medical data. Early during the clinical trial process, the patients' identifying information is replaced with an alphanumerical sequence by the PID-Generator. The resulting PID represents a pseudonym of the first order. As the patient has to be identifiable at some point within the complete work-flow, a separate module is responsible for managing this process. This module securely stores a list of patients together with their identifying information and their patient identifier within a database [59]. This Patient-List is used from other components within the research network as a service component to requests PIDs for patients who participate in clinical trials or identify an individual. For the remainder of a study, a participant only uses his or her issued PID.

For follow-up studies, the same PID can be requested at the service module to enable a mapping between these studies. In case that a different patient identifier would be issued, medical data would not map to the same patient and correlations between the results would not be possible. This can decrease the chance of successful answers to research questions.

An actual PID contains 6+2 alphanumerical characters. Two characters are used as a check-sum and for error correction. Literals which can be confused with numbers are removed from the alphabet [27]. Although the PID-Generator uses a hard to invert pseudorandom number generator, it is stored relatively near to the identifying information. If the underlying patient-list would be compromised patients could be easily identified by a third party. Therefore, the PID is only the first instance of pseudonymization. Secondary usage of study-data within research databases requires a second step of pseudonymization which is realized by the PSD-Service and its components.
The work-flow of the mentioned components has been created and managed by the CIOffice\textsuperscript{13} of the Universitätsmedizin Göttingen for use within research networks. Additional components are present within this concept and the PID-Generator, the Patient-List and the Pseudonymization-Service are only parts of it. Further information is available upon request.

Terminology used later with the context of CDISC-ODM denominate patients as Subjects and the patient identifier as SubjectID. This directly correlates between the terms used within the CDISC-standard. This thesis assumes that study-data available in the ODM format already uses the PID-Generator and that the SubjectID holds a valid patient identifier.

\subsection*{4.2 Interfaces of Pseudonymization-Service}

The Pseudonymization-Service consists of three major components. The core segment where the pseudonymization is done, is the actual PSD-Service. It is encompassed by two neighboring services which represent ingress and egress services in the sense of strengthening or creating the cryptographic pseudonym.

On the one side resides the Study-Database-Service (SDB). Typically, for the research database in question, medical data is generated from clinical trials. Therefore, in an optimal case this data is already held within a study database. The SDB-Service accepts medical data, a patient identifier and a transfer method. The transfer method reflects the action which should be performed at the target location.

After pseudonymization, the target location is typically a research database (FDB) where medical data is stored alongside a strong pseudonym. This would be the optimal situation according to privacy protection and for a possible long-term storage. The FDB-Service is also capable of triggering a message where the direction of pseudonymization is reversed. The need of such messages arises if new results have been generated within a research project which are of potential use for the participant of the original study. Additionally, the patient has a right to be informed about such results.

For example, a high risk for a patient for a genetically predisposed illness has been newly identified by analyzing large amounts of gene data. If such a need arises, an ethical committee can decide to de-pseudonymize a specific data set. The original identifying information of the patient is then looked up and the medical findings are transferred to the appropriate institution if the patient is still available. Nevertheless, such messages are considered to occur in an infrequent manner only.

\textsuperscript{13} http://www.mi.med.uni-goettingen.de/forschung/cioffice/cioffice.htm
4.2.1 PSD-Service

The TMF Pseudonymization-Service resides as an intermediary component in the logical middle between the SDB-Service and the FDB-Service. Since it cannot be directly interfaced with from an external entity, this sub-chapter presents some of the supported methods. They act as a basis for later chapters and introduce the underlying functionality.

The **PSD-Service** is designed to work in conjunction with a smart-card. The smart-card houses the secret for the cryptographic methods which are used to transform a PID into a PSN. The access to the secret on the smart-card is itself protected by a personal identification number (PIN). Additionally it is advised, that the physical access to the primary and the secondary, identical (backup) smart-card is restricted to a limited number of personnel. For example, card access is only granted to a technician who is already working with person-related data within an institution and whose task it is to supervise pseudonymization requests from other parties. Additionally, a second person with a higher position within the institutional hierarchy, for example a head of department, would require access to the primary and the secondary card in case of exceptional circumstances. PINs should not be stored near the smart-cards in case of theft. Ideally, the 6-digit PIN is only issued once to the personnel and stored in a different physical location afterward. PIN codes could be securely stored within a software vault under the departments head's oversight. A real vault could also be used for storing both the second smart-card and the PIN. These security measures should be sufficient to protect pseudonyms within a research network. If a nation-wide patient-list would ever exists (which is doubtful because it would violate the current understanding of data protection law), additional security measures would have to be developed to further protect the identity of patients. Within the computer science community, this would normally include theoretical scenarios for situations, where one is forced to reveal secret tokens. However, not every real-world scenario can be perfectly secured against an attacker and this topic is considered out of scope for this document.

The Pseudonymization-Service supports the anonymization of patient identifiers. This is important when a data transfer to a research database is only used in one way. In the case of an anonymization request, the PSD transforms a PID to random string by using the following method [60]:

i. Generation of a 32-bit random value using the Java class SecureRandom [61] and a predefined seed value.

ii. Using the SHA-256 cryptographic hash function on the in (i) generated value.

iii. Usage of the first 24 byte of the SHA-256 [62] encrypted hash value and Base64 encoding of that value results in a 32-Byte random ID.

The Java class SecureRandom complies with Federal Information Processing Standard (FIPS 140-2 [63]) for pseudorandom number generators (PRNGs). However, the initial
seed value is currently based on a fixed integer number with a fixed increase for each iteration. It would be more secure if this seed value would be derived from a real random input. For example, a specific user action at a random point within a time interval (e.g. moving a computer mouse or issuing commands which result in hard-drive activity) would result in a stronger value for an initialization vector (IV).

When a pseudonymization of a PID is requested by a message from the study database the following workflow is used:

i. Using of a Java RSA/ECB/PKCS1Padding cipher instance

ii. Passing the patient ID and the private-key stored on the corresponding smart-card to the cipher instance from (i) for encryption.

iii. Base64 encoding of the resulting string-value from (ii) and returning the encrypted PSN to the caller.

This workflow is concrete, but the cipher instance in (i) deserves a short explanation. RSA is the abbreviation for a popular algorithm for public key cryptography named by the initial letters of the inventors surnames (Rivest, Shamir and Adleman) published in 1978. It can be used for encryption and signing messages. ECB describes the method of operation of the encryption algorithm. In Electronic Codebook mode, a message is encrypted block-wise with an adjustable block length. The resulting encrypted string is then padded using the PKCS1 method. Without padding the RSA algorithm may be vulnerable to attacks [64].

Concerning import interfaces, the PSD-Service itself plays only a minor role as it has no direct interfaces to work with. It is designed to operate in conjunction with the SDB-Service and the FDB-Service only. Both services connect to the PSD-Service during the initialization phase. The IP address of the PSD-Service must be specified in special configuration files and firewall rules have to be set to allow necessary connections to the PSD-Service ports. After the initialization, the PSD-Service communicates with the neighboring services only over a secured transport way.

The Pseudonymization-Service has additional features like creating smart-cards, a front-end for managing user data, a PGP secured email system and reissuing new pseudonyms in case of smart-card loss or damage. The interested reader is referred to the PSD documentation [8],[60],[65],[66] because those properties are out of scope for this document.

4.2.2 SDB-Service

The SDB-Service connects the study database with one side of the Pseudonymization-Service. The study database contains data from clinical trials, which has often undergone quality management. Due to operating costs, study databases are often only available for a limited amount of time. The goal is to use the SDB-Service to transfer this data into a research database for additional research projects and possible long-term storage. However,
if data leaves the study database, it is only allowed to do so in an anonymized or preferably, in a pseudonymized way. Thus, the SDB-Service acts as the connection point for implementing a custom adaptation for transferring data into a research database. Figure 5 depicts the interfaces on the SDB-side, as well as the side connecting to the Pseudonymization-Service.

The SDB-Service possesses two types of input interfaces and one output interface which connects to the PSD-Service. The first input mechanism operates on a file-system directory by using a polling mechanism. An automated, sequential testing of a customizable directory on a target machine is able to detect new files which have been inserted into that specific directory. The time interval which is used for this polling can be customized, to reflect the output load of the study database. Normally, the interval is set to a short time to ensure a quick processing of newly available data.

Input files which originate from an export of a study database record set have to comply to a specific XML-format which is ensured by an XML-schema. An SDB-Service compliant XML-file basically consists of two important parts. A header- and a body-section. The header-section contains the patient identifier which is to be pseudonymized and some transport information which is needed for processing the message at the PSD-Service. Important transport information is the actiontype which would map to the type "store" when data is to be transferred from a study database to a research database. Additional action types are also possible. For example if a patient requests a removal of his / her medical data from databases which store his / her information in a pseudonymized way, a "delete" request would be formulated. If this request arrives at the receiving service, the specific implementation at the FDB-side can commit steps to delete information in the database stored under that specific pseudonym. A full list of actiontypes is available within the context of the PSD-Service user manual.

---

14 [http://www.tmf-ev.de/site/services/psd/psd.xsd](http://www.tmf-ev.de/site/services/psd/psd.xsd)
A user name which queries the action is also required within the header of the message. This user must have been previously registered by an administrator of the Pseudonymization-Service on the services' user interface. If the user does not have a valid account the request of message processing is denied and an error message is generated. Error messages are outputted in the event log of the service and an error-directory on the machine file-system. Thus, messages which generate a program error are not lost and can eventually be re-queued after the error has been corrected. This approach is also helpful when processing a large number of files. As each patient-record translates to one file within the file-system, a potential large number of error messages may have a common cause which can then be easily identified and corrected.

The second main part of the XML-file is the medical record section (MDAT). The body section can hold any type of medical data which is to be stored within the research database. Preferably this data is already structured like CDISC-ODM, but also proprietary data can be encoded within this section of the XML-file. A description of the CDISC-ODM and other formats is available in [67]. For example, it could be possible to include image data obtained from medical examinations. Even if the target database already contains a pseudonymized patient record, relational tables could be linked via the pseudonym and binary image data could be inserted in an additional table or data field large enough to handle medical images. The SDB-Service supports a maximum of 150 megabyte (MB) of data contained within the body section.

Listing 1 shows an example of the XML-format used for the Pseudonymization-Service. Text in square brackets acts as a placeholder and has been omitted for readability. The ODM-tag normally contains schema definitions and additional descriptions to indicate the type of the ODM-format in question. All data beneath the ODM-tag (Listing 1:16) contains the medical study data. The index-element (Listing 1:12) contains an actual patient identifier. Details for the PID and the PID-Generator can be found Chapter 4.1.
Internal Java functions then encrypt the MDAT [66] because its contents are not important for the remaining work-flow. Only the correct decryption at the target service (in this case the FDB-Service) is necessary to insert the MDAT into the research database. The research database may also implement additional security measures for the medical data or it is already encrypted within the context of the study database. However, the transport layer is secured by the cryptographic protocol 'Secure Socket Layer' (SSL). An RSA public-/private-key infrastructure is used to ensure that data is not altered or viewable during the transport to another source. A man-in-the-middle attack with forged certificates is theoretically possible, but given the circumstance that such medical services often operate within virtual private networks (VPNs) of the involved institutions, the risk is minimal if the original certificates are well protected from unauthorized access.

The web-service interface supports the sending of a properly constructed XML-file to a target URL where the service is running on. On success, an acknowledgment is returned to the caller indicating a positive message deposit to the invoked web-service.

A typical Java-based approach for working with the web-service will be given here in short to further explain the web-service. They shall not act as a programming example but rather as a reference to a basic programming principle when working with semi-structured data like XML and web-services. Short listings of source code will substantiate the procedures. These specifications generally also apply to the approach which is used for the program "i2b2Dataimport" currently being developed by S. Zeiss as a student internship at the UMG. This background information is needed to understand the concepts presented in this and later chapters.

Listing 1: Example of a PSD-compliant message format

```xml
<?xml version="1.0" encoding="UTF-8"?>
<psd:MDAT xmlns:psd="http://www.tmf-ev.de/site/services/psd/psd.xsd">
  <header>
    <action>
      <actiontype>store</actiontype>
      <from>SDB</from>
      <to>FDB</to>
      <destination>FDB</destination>
      <returnpath>SDB</returnpath>
      <userID>admin</userID>
    </action>
    <index>Z5R93DEW</index>
    <ack/>
  </header>
  <body>
    <ODM [...]>
      [study data]
    </ODM>
  </body>
</psd:MDAT>
```
The first step in implementing a web-service client which is using the XML-format for message exchange is to derive Java classes from the XML-schema the message uses. There are many XML schema languages available and also usable in Java. The involved services use the original World Wide Web Consortium (W3C) XML schema language\(^\text{15}\), also known as XML Schema Document (XSD). Basically, it allows developers to define where in an XML-document specific tags and attributes are allowed and of which data-type these components have to be. For example, a 'simple type' component may be a string, integer or date. Overall, there are 19 data-types. Complex types allow additional (sub-)elements and carry attributes. Many additional logical constraints, like defining a specific sequence in which tags have to appear, may exactly define the layout of the associated XML-file further.

Analyzing this schema representation, the Java Architecture for XML Binding (JAXB) can derive basic Java classes from the XSD-file. These classes contain the same information as the schema and also supply get- and set-methods for elements and attributes. The Java classes can be instantiated via the class \texttt{ObjectFactory} and thus represent Java-Objects when a XML-file which validates against the schema. The process of transforming an XML-document into Java-Objects within the computer memory is called unmarshaling. A complete overview of the JAXB architecture can be found under \cite{68}. Listing 2 shows the Java class representation of the header format which has been derived from the PSD-schema file. It contains variable definitions (line 5-8), get- and set-methods (line 10-13) and annotations which are indicated via an @-sign. Here, annotations are used to specify that elements are bound to the XML representation (line 1) and reflect if they have to occur in a specific order.

\begin{lstlisting}[language=Java]
@XmlElement(required = true)
protected TAction action;
@XmlElement(required = true)
protected String index;

public TAction getAction() { return action; }
public void setAction(TAction value) { this.action = value; }
public String getIndex() { return index; }
public void setIndex(String value) { this.index = value; }
\end{lstlisting}

\textit{Listing 2: JAXB derived class representing the header of a PSD-compliant XML representation for the SDB-webservice}

In addition to the Java representation of \texttt{THeader}, there are the classes \texttt{TAction} and \texttt{TBody}. Together, they allow to form a complete representation of a PSD message within the context of Java Objects. It is possible to form such a message from various data

\footnote{\url{http://www.w3.org/2001/XMLSchema}}
sources, for example an XML-file, a comma-separated values (CSV) file or a direct database connection.

The next step is to deliver this message to the web-service client implementation so that it can be processed and sent over the network to the corresponding web-service. The SDB-Service uses the Java API for XML Web Services (JAX-WS) for both communication endpoints. At an endpoint, the complexity of the underlying implementation is hidden as it only supports its publicly announced interfaces. Thus, client and web-service are able to use completely different programming languages. The SDB-webservice and web-client implementation are both using JAX-WS for this purpose. The Endpoints exchange messages via SOAP.

As proxy interface which mediates between Java Objects and the SOAP interface the SDB-webservice currently uses an outdated version of Codehaus XFire\textsuperscript{16}. This does not pose a problem for the current sample SDB-webservice sample application but it may be necessary for future production versions to upgrade this dependency to a current version. The direct successor of XFire is called Apache CXF which is available under \[69\].

To connect SDB-Service Client to the corresponding web-service the parameter \texttt{SDB\_CLIENT\_URL} has to be manually entered within the source file of the class \texttt{SDBServiceClient}. After that any custom implementation can create a new instance of the web-service client and pass a message of the format \texttt{TMDAT} (which contains a complete PSD-XML-message with header and body) to \texttt{handleLocalAction}. This parent method is also responsible for handling messages which arrive via the file-system. It processes the message and is the next step to send it to the PSD for pseudonymization.

```java
protected TMDAT interfaceTest(TMDAT message) {
    SDBServiceClient wsc = new SDBServiceClient();
    SDBLocalHandlerPT port = wsc.getSDBServiceHttpPort();
    TMDAT responseTMDAT = port.handleLocalAction(message);
    return responseTMDAT;
}
```

\textit{Listing 3: Sending a message to the SDB-webservice and returning the response}

\textsuperscript{16} http://xfire.codehaus.org
4.2.3 FDB-Service

The FDB-Service which resides on the opposite side from a logical point of view has similar interfaces as the SDB-Service (cp. Figure 6). The difference is that the main interfaces of the service act as an outgoing interface when pseudonymized data is to be stored in a research database or other FDB-related database updates are initiated. It also has file- and directory-based output capabilities, as well as a Java-based web-service interface. The incoming interface however, from the research database to the FDB-Service, has only one defined action. The only request which can come from a FDB is a request to notify a commission about a new medical finding for a patient (alertFinding). All other messages either store, delete, request or update information in the research database.

A decrypted XML-message travels further down the typical path from a Spring producer to a Spring consumer. A Spring producer originally receives the message from the PSD-Service over the external interface. The FDB-Service internal consumer acknowledges the correct reception and processes the message further. If the message is set to be delivered to the file-system, its content is decrypted and the stored by invoking classes telling the operating system to store the file.

If the file-based output method is not used (by negating the FDB-config parameter fdb.writeMessageToDisk=true), the message is passed to a handler which can process the message further. The application design of the Pseudonymization-Service determined that from this point on, the incoming message must be treated by the specific database application which is in use at the target location. This approach is considerate, as the target environment may be very heterogeneous in terms of database type and operating system.

The necessary classes of the handler are implemented as an interface (FDBSpecificAction). A custom application then has to implement these Java
interfaces and define further processing actions. Predefined methods for storing, deleting or anonymizing patient data records are available. A likely scenario would be the implementation of a direct database interface with the help of a corresponding database driver. Specific action types should translate to SQL statements and perform the desired action. For example, a delete request could first translate to a lookup, if a patient with its corresponding PSN and medical data is actually present in the database. If that is not the case, the caller is notified about the request failure. On a successful lookup, the corresponding data is removed. A scenario where an incoming message triggers a regular database update (in terms of updating or adding medical data for a patient record) is also imaginable.

If a pseudonymized XML-message arrives at the FDB-Services, it has the contents of the index-element (see Listing 1, line 12) transformed to a strong cryptographic pseudonym.

```
1 <index>TrjSUaELB5pfgBEZZyuzWZHbKFmzdRQyS4GL4Y3TT2KD25a0a/fa194+LnJWJCOYT+Un0vYSwLoLwy13xNV/wGFDan045swBs68/Ss+79WFou9ctoUSNxiR5D2uJkgoAxT2MsjQhYo0j0BiZmwsryFxOie/1BncOoeFZCPs/z4=
</index>
```

Listing 4: Pseudonym (PSN) showing an increased entropy in comparison to a PID

All the three addressed services, the PSD-Service with its neighboring SDB-Service and FDB-Service, make use of Apache Tomcat whose function has already been discussed earlier. An overview of these components will be shown later on the context of possible target machines.

### 4.3 Interfaces of i2b2 software

The i2b2 software needs to be connected to different data sources. The current method is to accept the custom database schema, transform the data source to be compatible with the target scheme and finally insert data into tables.

However, the i2b2 project announces that custom plug-ins can be created by other parties which interact with the message flow of the i2b2 Hive cells. Such a cell can be used to perform various tasks on the data objects. To evaluate the potential interfaces for pseudonymization and possible future approaches, the i2b2 message format is analyzed. This step is a prerequisite to decide whether the i2b2 message format is compatible with the PSD- and other services.
4.3.1 Database scheme

Although the database is not a software interface in a common sense, it must be mentioned because of its importance for the general initialization of the i2b2 Hive. The database centers around one table used as connection hub for tables providing additional information. This star-schema is a common practice when building large databases. The i2b2 developers have closely followed this practice and the accepted terminology as their central table is called OBSERVATION_FACT. It contains necessary codes which can occur during a medical examination. The post fix "_fact" indicates its central status. Other tables depending on this central table are indicated by the post fix "_dimension". This concept is also used by data warehouses which apply filters by limiting the output to a specific dimension.

The additional "_dimension" relations (tables) are using values stored within attributes (columns) as foreign key of the master table. Using this scheme, necessary information can be queried when needed using relational joins. The i2b2 developers also refer to the performance advantage of having a star-schema with a limited amount of tables [70]. Working with tables with large row numbers they state that joining tables with large indexes has a significant performance advantage over joining multiple tables with smaller indexes. A second advantage is that new data simply results in new rows added to the different tables as the general table layout does not change (considerably) over time. The third advantage they identified is the identification of orphaned concept data. If a join is performed over the fact, concept and provider table, data which is not included in the join must be data with an invalid concept code.

There are some system database schemas which are required by the i2b2 core cells. Among them are the i2b2hive, i2b2metadata and i2b2workdata. These schemas have to be created and filled during the setup process of the i2b2 Hive. Especially the i2b2metadata schema deserves special attention as it contains information on how the clinical data is organized. Contents of this table are used to display the hierarchical data structure of the Ontology cell. Project related data can be stored in an additional custom schema which needs to be announced to the Hive afterward.

In [27] the underlying i2b2 database schema is described in detail. This includes working with the i2b2 table structure on data import from a data warehouse and using stored procedures to comply with the custom table format. Additionally, it contains a list which clarifies and describes all i2b2 tables used in v1.3 of the i2b2 data mart. The reader is referred to this document for further information.

4.3.2 Communication between cells

The different cells of the Hive communicate over the Simple Object Access Protocol or the Representational State Transfer. These transfer methods have been mentioned before in this document within the context of web-services. They are vital for those services and
the i2b2 cells. For example, the i2b2 Workbench client uses REST or SOAP to communicate with the Ontology management cell to query available meta-data. The Ontology cell itself again uses REST or SOAP to access the Project management cell to authenticate the previous request.

Later, also the content which is transferred over these transfer methods is important. An analysis of a possible integration concept for the PSD-Service has to include an evaluation of the i2b2 Hive messages and their underlying concepts. These concepts may also be important for the future work with the i2b2 software package.

REST has first been described by R.T. Fielding in his doctoral thesis which is available online [71]. The following citation describes the architectural style of Representational State Transfer:

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use." [72]

The essence of REST is, that a client resides in a specific state, for example a starting or intermediate state. By accessing a service URL and fetching the data the client transfers to another state. The data he received may contain references to other resources and the client may chose to query data at additional URLs as well. Using this way he traverses the network. While the client is not in the state of traversing to another resource he is resting in the previous state. REST also follows the concept of loose coupling as the client application does not know how the response is generated or which programming language is used. Public interfaces of cells are eventually exposed via WSDL or Universal Description, Discovery and Integration (UDDI) [70]. REST cannot only be used over HTTP but also other protocols. However, usage over the standard Internet protocol is most common.

In contrast to Representational State Transfer the Simple Object Access Protocol is a complex standardized protocol. It superseded the method of XML-remote procedure call (RPC) where the body of an HTTP-POST request was comprised of an XML-body [73]. A typical SOAP-message is written in XML and comprised of a header and a body. An envelope directly contains both items. The structure is depicted in Figure 7 [74]. The SOAP messaging framework¹⁷ defines additional terminologies for the message exchange. For example, messages contain a sender and receiver of a message. Intermediaries are able to forward message, for example for the purpose of load balancing.

¹⁷ [http://www.w3.org/TR/soap12-part1/]
In most cases, a client or server application framework is capable of automatically handling and processing SOAP-messages at communication endpoints. An implementation does not have to handle those messages directly and the contents of the body are normally mapped to specific high-level methods or functions which offer or request transmitted data. This methodology is commonly used to reduce the complexity of web-services because not every developer has to implement functions for SOAP-message handling.

The default transfer method for i2b2 cells and workbench has been configured to use the REST transfer method. However, both concepts are supported.

### 4.3.3 i2b2 messages

Higher-level i2b2 messages which rely on the described mechanisms have been created by the i2b2 developers. The overall structure of these messages is fixed because it is defined in associated schema files. Sent and received messages are validated against schemas to ensure correct formatting. The validation process can consume CPU-time, especially if the number of messages is high. However, the validation process ensures that no malformed message can be accepted by i2b2 components.

A basic i2b2 message is either a request message or a response message as an answer to a previous request message. For example, a request message can originate from the i2b2 Workplace as a result of a user interaction. Request messages can also be exchanged between cells when one cell needs to query another cell first to fulfill a request. For example, if data is queried from the workbench to the i2b2 Hive to display a patient set that complies to specific concept codes a request to the Project Management cell is made first to authenticate this request. In general, this is done by sending a request containing the triplet 'domain, user, password' to the PM cell. If the PM cell confirms that a user has access rights to the data in question, the original request is processed further.

Each i2b2 XML request message consists of message header, a request header and a message body. A message header contains meta information about the message. For example, software version numbers, origin and destination and time of the message in
question. The message header is validated by the schema i2b2.xsd. A request header contains a wait time value. This value can be modified at the sending application indicating the amount of time a client is willing to wait for a result. This enables the i2b2 Workbench to cancel requests which take too much time, especially on large data sets. Also, it enables the Workbench to display a predicted remaining time which is normally fed back from the CRC cell when requesting patient sets or patient data. The prediction algorithm is accurate in most cases\(^\text{18}\).

A message body contains cell-specific XML which is again validated be the specific cell the request was sent to. Some more important requests and responses have been grouped into objects. Some of them which are candidates to be used for pseudonymization integration are presented later. Figure 8 displays the generic structure of i2b2 messages.

![Diagram of i2b2 message construction including schema files](image)

An i2b2 response message basically contains the same information as a request message. The original request header can be echoed back to a client and is optional. A response header mostly contains status information about the current query. For example it indicates the completion of a query, status of a pending query or an error message if the processing can not be completed. The message body of a response message contains the actual data again in a semi-structured from with an associated schema.

Mentionable is, that drag and drop actions within the workbench user interface between different views are also validated by a schema. The user interface lets one only drop valid patient sets or direct patient IDs. This shows how the work-flow complex interactions can be controlled via XML.

\(^\text{18}\) While working with the i2b2 software ca. 90% of the remaining time was calculated satisfactory. The remaining time became displayed more inaccurately with queries taking longer than 1 minute (\text{v.1.3}).
Of particular interest are i2b2 high-level messages because they combine different data types under a common vocabulary. They also reflect the intermixture of web-service and medical knowledge which was incorporated into the i2b2 software.

A vocabulary data object (VDO) contains every data relating to what the Ontology service can process. This includes paths to concepts, synonyms of concepts and visual attributes for displaying hierarchy information.

A patient set message (PSM) holds information about the patient set which was queried using concepts from the Ontology cell. It is primarily used to formulate queries and responses for patient sets and storing and managing them afterward. Additionally a response can also contain the cardinality of the result set.

A patient data object (PDO) contains any information related to a patient like clinical observations, demographics and provider data [49] which is stored in the database.

These objects all share the same architecture of how they are created. The web-service in question (cell) generates a request handler for all use cases. The request handler creates a data access object (DAO) which encapsulates the database access. After reception data from the database the DAO creates one of the object described above. This object is then returned through the request handler to the originating web-service and finally transferred over the network.

**4.4 Use cases / interface scenarios**

This chapter presents the use cases which have been deemed most important in the context of study and research databases working together with an import interface and the i2b2 Hive.

First, the pictographs used within the following graphics have to be explained further to establish a point of reference for the following use cases.

An *import interface* can be seen as a generic interface which is used to prepare and convert any kind of data to a compatible target format for insertion into the underlying i2b2 database. This step has to be done for all concepts at some point within the import process. Currently, this can be done by executing SQL-insert statements which insert data directly into the i2b2 database or by an admin using the CRC Uploader from the i2b2 Workbench on a delimited text file [75].

A *sending service* is an abstraction of any type of service which transfers necessary data over a web-service or similar interface to a processing entity which resides at another point within a network. In some cases this represents the SDB-Service. Therefore, the same color scheme has been chosen. Following this pattern, a *receiving service* accepts data from another network source. The received data may have processes in the steps
before to comply with a target data format. In some cases this translates to an FDB-Service otherwise it performs a similar function.

The **PSD-Service** can be used to pseudonymize or anonymize patient identifier, either with a cryptographic key or a one-way function. Therefore, it contains a 'lock' symbol. Otherwise, it relays (medical) data untouched to the corresponding service specified within a data message as introduced in Chapter 4.2.1.

The **i2b2 Hive** is depicted by the typical comb- or cell layout. It represents the i2b2 database or the parts responsible for the data import process.

### 4.4.1 PSD-Service between import interface and i2b2 Hive

Figure 9 shows two basic concepts which can occur during an import process. The import interface has transformed data into a compatible i2b2 format. Data has been provided by other databases or sources. This process is similar to the first two tasks of an extract, transform, and load (ETL) process. This three-step process is needed to make large data sets from different sources compatible with a target database and finally insert it.

A sending service accepts data in the type of format provided by the import interface and is responsible for adding a compatible PSD-Service header. Further, the PSD-Service performs the requested operation of pseudonymization or anonymization. In the next step, a receiving service needs to handle the anonymization string or the pseudonym and apply the correct formatting which was previously generated by the output interface. It replaces the patient identifier with the pseudonym. After successful completion of this task the data is finally inserted into the research database which is represented by the i2b2 Hive in this case. Although an automated process is desired, some steps may include a manual action carried out by a staff member.

When a finding is generated for a pseudonymized patient data set a data export from the i2b2 domain could be transferred almost directly to an FDB-Service because it resides very near to this service. Only the data format has to conform to a PSD-Service message.
The second approach depicted on the bottom of Figure 9 represents a strict in-house concept. Here, the research database is located within the same context of an organizational entity and a pseudonym of the first order (PID) has been generated. The PID is consistently being used for all study data which arrived at the import interface. A sending service transfers over a network to a receiving service which finally performs the import process. The receiving service is near to the database or for example managed by a database admin with the necessary access rights within an institution.

In this case a weaker cryptographic pseudonym is used and data from the research database would not be allowed to leave the narrow confinements of this scenario. Additionally, if the patient list would become compromised, a research database containing only pseudonyms of the first order, would become a point of weakness. On the other hand, if the i2b2 Hive contains pseudonym of the second order it could be made available to a much greater number of users without major security concerns.

The scenario where the PSD-Service sits between an import interface and the i2b2 Hive is the most concrete and most practical accordingly. In conjunction with K. Helbing from CIOffice at the UMG the further analysis of this use case has been decided. Example data sets have been provided to represent a study database export. The data sets consisted of CDISC-ODM in version 1.2 and 1.3 containing medical data. These sets are also the input to the tool i2b2Dataimport currently being developed at the UMG by S. Zeiss. This tool performs the following tasks:
• Parsing and unmarshaling of CDISC-ODM.
• Generating an ontology for the i2b2 Hive to the level which can be archived within the context of CDISC-ODM.
• Generation of SQL scripts for the i2b2 ontology and other necessary i2b2 tables (conceptDimension, patientDimension and observationFact).

The PSD-Service has to take the output data of the importer and perform pseudonymization of the described data set.

The i2b2 Hive, specifically the i2b2 database structure, supports encrypted identifier from different source systems [76]. This is realized by multiple steps. First, a unique integer number is assigned to each patient which is inserted into the data repository. The main location for patient identifier is the relational table PATIENT_DIMENSION where the identifier (patient_num) is a primary key. This table contains additional rows for patient related data like age in years, postal code, race and text description. Some information may be required to assess research questions based on the age of a patient. However, if the age and an upload date of the data is given, the birth date of a patient could be extrapolated. A birth date (among others) is considered identifying information and must only be used according to data protection rules. Further, it is not likely that race or religion are of use to formulate research questions. The fact that they exist in the i2b2 Hive shows the potential data warehouse application but those data fields are not necessary in terms of pseudonymization. A patient identifier and an import date of the data set are the only data fields used within this context.

The second requirement is the relation PATIENT_MAPPING. It allows that encrypted patient identifier can be entered alongside with a source system identifier string indicating from which database the identifier originated. The attribute patient_num is used to enable a mapping between a patient number and an encrypted identifier (EID). The database scheme allows an EID to be as large as 200 characters and a source system description up to 50 characters. This is sufficient for the PSD-Service as a pseudonym is just under 200 characters and a source system identifier usually is short (e.g. KN_AHF or KFO).

The previously presented patient data objects also support patient mappings inside an XML-tag called 'pid_set'. Listing 5 shows an example representation of this section within a PDO. The PSN has be replaced by a placeholder for readability. The source-attribute of the tag patient_id (line 3) indicates that this patient is represented by the integer number 1 within the i2b2 Hive. Different source systems which are using different patient lists and therefore different PIDs and PSNs are given in line 4-5. Thus, this approach is not only suitable for integrating pseudonymization within the context of one competence network using one patient list, but combining several different pseudonyms within the same research database.
This PDO representation is specifically supported for upload requests to the i2b2 Hive via the CRC [76]. A response message should also contain information of patient mapping. However, they may not be displayed in the i2b2 rich client because it works primarily with integer patient numbers. Anyhow, this is not a major disadvantage because a PSN is only needed when data is to be exported to initiate new finding. It would be sufficient if patient data containing a PSN could be exported through the i2b2 Workbench. Alternatively, a direct database export or an FDB-Service could be used (see Chapter 5 for further information).

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;pid_set&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>&lt;pid&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>&lt;patient_id source=&quot;hive&quot;&gt;1&lt;/patient_id&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>&lt;patient_map_id source=&quot;KFO&quot; status=&quot;A&quot; *&gt;[PSN]&lt;/patient_map_id&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>&lt;patient_map_id source=&quot;AHF&quot; status=&quot;A&quot; *&gt;[PSN]&lt;/patient_map_id&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>&lt;/pid&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>&lt;/pid_set&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Listing 5: Example PDO code snippet showing a PID set containing patient mappings to source systems

### 4.4.2 PSD-Service between multiple SDBs and import interface

Figure 10 again shows the data flowing from left to right on the x-axis. A secure patient list contains the mapping between identifying data and PIDs. Registration for clinical trials is conducted by either registering a new PID via the PID-Generator or reusing an existing identifier by querying the patient list via the matching algorithm. For the remainder of the clinical trial only the patient identifier is used which detached from the original identifying data of a participant. Results are stored within study databases. Different trials can hereby translate to distinct databases. These databases have in common, that stored PIDs are unique among the storage repository because the patient list has been used consequentially.

On export of single or multiple databases, the PSD-Service is used to transform patient identifier to PSN and additionally transport medical data along the way to the receiving service. When the data has arrived at the receiving service it can either be transferred to an i2b2 instance via the import interface (upward direction). The second option is to transfer of the pseudonymized study-data to another custom research database (downward direction).
In this case, an ODM export from the study database would consist of ODM-compliant XML files. The SubjectID would contain a PID. A sending service would have to add a PSD-Service required header containing the necessary information. For example action-type, PID, PSD-authorized user and direction of transfer. Some of this information is contained within the database export, others (for example PSD-Service user name) are not and would have to be added at this point.

A desired export format would be based on data sets for single patients. For example, a semi-structured file containing PID and MDAT. The sending service would then have to ensure that the patient ID only exists in the PSD-Service compliant header so that the PSD-Service can strengthen the pseudonym. The PID is not allowed to occur on the output side of the PSD-Service together with a PSN. Otherwise, the PSN could be cross-referenced with the PID. If that would be the case of the output format, additional steps would have to be taken to ensure this fact. For example by introducing a placeholder for all PID occurrences which does not occur naturally within a study database export.

An implementation of this concept would transfer PSD-compliant messages to the SDB-Service on export. Using ODM, one data set per-patient export would be optimal as this would translate to one message which could be pseudonymized utilizing the PSD-Service. The contents of the SubjectID would have to be removed from the data set and placed in the message header only. If an import interface relies on the content of the ODM SubjectID, the pseudonym would have to be inserted at this position after pseudonymization. Alternatively, there is a development version of an XSL
Transformations script available, which transforms an ODM input file to the PSD-compliant header format [77].

If a per-patient message export is not possible and a file contains information of more than one patient, an incremental pseudonymization would have to be performed. The PSD-Service would replace one SubjectID at a time. After finishing one message, the complete data set arrives at the FDB-Service. The same message would have to be transferred back to an SDB-Service and the next SubjectID (containing a PID) would have to be processed. If all PIDs within a data set have been completely transformed to PSNs, the task would be complete. The data set could then be processed further using an ODM parser or other tools and inserted into an i2b2 Hive or a custom research database.

If a finding is generated within this scenario, it would need only a wrapper class which transforms the data export from the i2b2 domain to a PSD-compliant finding message. As the import interface is located in close proximity (supposably on the same machine) to a FDB-Service the import tool could be adapted to support this kind of output or a wrapper tool can be created accordingly.

4.4.3 PSD-Service between SDB and multiple provider-specific FDBs

The third use case addresses the likely scenario that different organizational units are maintaining provider specific research databases. These institutions may well be connected within a research network. Therefore, a network-wide patient list is used for maintaining a study database. Data being exported out of a study database to provider-specific research databases has to cross the Pseudonymization-Service for pseudonymization or possibly anonymization. The required SDB-Service and FDB-Service which are connected to the SDB and FDB respectively on both sides of the PSD-Service have been omitted in Figure 11 for a better overview.

In case of pseudonymization, a provider-specific FDB would then contain strong pseudonyms of the second order. Different institutions can then export pseudonymized data containing a PSN from their databases. Again, sending- and receiving services transport this data securely over a network. The import interface in this case receives data which is already pseudonymized or anonymized.

Combining multiple data sets into the i2b2 Hive has the benefit that more information is available to formulate research questions or perform other tasks on patient sets. However, if a provider chooses to set up a local i2b2 software installation, project specific data sets could be exchanged easily, as the data is already present in a scheme which is i2b2-compatible. If a research network can agree on a common ontology data exchange would become even more fluid because only project data would have to be exchanged. The concept codes would already be present in the ontology tables. An effort to generate interchangeable concept codes based on the International Statistical Classification of Diseases and Related Health Problems, 10th Revision (ICD-10) and the German
modification of International Classification of Procedures in Medicine (OPS) is currently ongoing at the Universitätsmedizin Göttingen\(^\text{19}\).

The difference between the second scenario and this use case is that a provider-specific research database already contains a pseudonym of the second order, because the PSD-Service is used early within the data flow. The previously described altering of export sets would have to be handled by an SDB-Service and an FDB-Service respectively.

When data sets are exported from provider-specific locations, sending and receiving services would securely transfer the data to a target machine where an import program can be executed. In case of CDISC-ODM compliant data, an import interface would additionally have to create insert statements for patient mappings and assign consecutive numbers to every patient in the set.

If a finding would be generated within the i2b2 domain, it is likely that a data export would not take the complete reverse path. An additional interface would have to be designed allowing a reverse path between an export occurring within the i2b2 domain and an FDB-Service which normally provides pseudonymized data for the provider-specific FDBs.

All use cases have in common, that further exports out of the i2b2 Hive to external research projects would require further anonymization or pseudonymization of the data set, as for example proposed in [78]. An identifier called PSN2 would replace a PSN for

\(^{19}\) Personal communication with U. Sax and S. Zeiss in April 2010.
additional security or the identifiers are anonymized. A PSN2 would be created by using the PSD-Service with a different cryptographic key on an additional smart-card. It is likely, that parts of the implementation used for the above concepts can be reused for this process if the need arises.

4.5 Analysis of interface scenarios

This chapter will address additional technical details which have to be considered for all presented use cases. Different institutions have heterogeneous environments with different operating systems (OS) and machine types. Additionally, there are three types of use cases for computers in general within the context of a typical medical informatics division:

i. Office use

ii. Development / software-testing

iii. Server / mainframe use.

The office use of computers is normally needed by employees and reserved for day-to-day business. These machines can normally not be used for long-term development procedures or software-testing. If those tasks are part of analysis and research processes, it would be recommended to have dedicated hardware reserved for these purposes. This has several advantages. First, project work is done in place and using the work space of an institution. Therefore, hardware can be time-shared by project participants. Second, software projects do not have to be migrated to a different environment on project completion. A software project is from the beginning integrated into the confines of the department and may also be accessible over the internal network. Further, issued hardware which has been lent to project participant remain within the department and the risk of lost / misplaced hardware is minimized. Finally, copies of software components remain physically on those machines enabling to put them into a quick ready-to-use state should the need arise.

The use of server components is normally being reserved for production versions of software elements and other solutions (e.g. a storage server). The UMG for example uses VMware ESX\(^{20}\) products (among others) within their data center to host different applications\(^{21}\). Therefore, a production version of the i2b2 Hive installation could be migrated there.

Accessing software elements on virtual machines and enhancing them on the running VM is also possible. However, some projects, like the i2b2 software package, may require access to additional virtual machine features like rebooting, unrestricted (or bridged) network access, snapshot and hardware management. Such access rights are not easily manageable and normally only accessible for server administrators due to security


\(^{21}\) Personal communication with U. Sax.
concerns. For the i2b2 project a dedicated virtual machine providing the above mentioned features has been proven useful both for development and (small scale) production use. However, possible license costs have to be considered. Ideally, campus licenses are available for the product of choice.

The extensible markup language is used throughout the different parts of the use cases. The different services of the Pseudonymization-Service process XML-messages only. The import interface transforms XML-messages. The i2b2 Hive does not use an XML database because it is designed to handle large data sets on a relational basis but the i2b2 rich client and the i2b2 Hive make strong use of XML-messages and validation. The basis of the study database export used within this thesis was also in CDISC-ODM. Where database exports are not in XML format, it is often necessary and advisable to convert custom data formats to an XML-based one. Using this approach, meta-data about data sets can be processed more easily. The availability of meta-data is often crucial for a long-term usage of data sets. If a traditional spread sheet usage (or a similar simple data format) is still being used, it should be possible to migrate software tools to handle semi-structured data and display it accordingly. A conversion from a modern XML-based format back to a custom spread sheet structure could be a time-consuming step and should be avoided.
4.5.1 Technical feasibility

To clarify the various hardware and software requirements of the presented components, Table 1 shows an overview of the different software components and their hardware and software requirements.

<table>
<thead>
<tr>
<th>entity type</th>
<th>possible machine type</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Oracle) Database (i2b2 Hive component)</td>
<td>real (a RDBMS within a VM is possible but unlikely to be used due to performance impairment of the VM layer)</td>
<td>MS Windows, Linux, Solaris, Itanium, AIX</td>
</tr>
<tr>
<td>i2b2-Hive</td>
<td>VM (recommended: 2 CPU, 2-4 GB RAM)</td>
<td>Linux</td>
</tr>
<tr>
<td>i2b2-Hive (high load)</td>
<td>real, cells in different bridged VMs or cells on different real hosts (very high load, distributed environment)</td>
<td>Linux</td>
</tr>
<tr>
<td>PSD-Service</td>
<td>real (possibly Oracle VirtualBox) with smart-card terminal</td>
<td>Linux, MS Windows</td>
</tr>
<tr>
<td>SDB-Service</td>
<td>VM, real (example implementation based on Java webservices)</td>
<td>Linux, MS Windows</td>
</tr>
<tr>
<td>FDB-Service</td>
<td>VM, real (additionally see SDB-Service)</td>
<td>Linux, MS Windows</td>
</tr>
<tr>
<td>sending-/receiving-service</td>
<td>near to appropriate database (same machine or VM)</td>
<td>various</td>
</tr>
<tr>
<td>Import-Interface</td>
<td>Java compatible PC</td>
<td>various</td>
</tr>
<tr>
<td>Workbench (Eclipse RCP)</td>
<td>Client-PC</td>
<td>(MS Windows, Apple Mac)</td>
</tr>
</tbody>
</table>

Table 1: Software entities with possible machine type an operating system

In addition to the hardware or virtual machine requirements it is likely that many machines have to be protected by a firewall for security reasons. This can sometimes present a problem because web-services and other entities may require access to special ports to communicate with an outside network. Any software which requires special access rights to special ports should clearly indicate these in the software documentation. If possible, examples should be given for the most popular firewall types. For example, the i2b2 Workbench requires access to the port 7070 per default to authenticate Workbench users. These ports can be changed according to the target environment. However, any changes to a default configuration should also be documented, for example in an internal wiki.

The underlying network between the software components is also of interest. In most cases all components of the use cases are used within the context of an internal local area network (LAN). In this case no security concerns arise as all data travels internally and can most certainly not be read by a third party. If data sets containing medical contents are transferred between multiple external institutions, they are connected via a VPN in an
optimal case. Such a data transfer could almost be seen as an internal transfer because data between the networks is highly encrypted. When a VPN is not present, for example between a study database export transfer (containing a PID and medical data) and an SDB-Service additional security has to be considered (for example a secure copy or SSL mechanism). Some form of encryption should always be used for data transfers. If that is not possible the means for a secure transfer have to be created first. An Email transfer of data containing pseudonyms of the first order should be avoided. If there is no practical alternative available, a public- / private key encryption is the minimum security requirement.

The design of SDB-Services and FDB-Service is necessary to make interface connections to both sides possible. Implementations for those interfaces are considered to be of at least medium complexity. A reference implementation which interfaces with a popular relational database and supports multiple message types would eligible.

Interfacing the PSD-Service with the tool i2b2Dataimport is feasible but requires manual user action at some points. Hand operation is required when processing CDISC-ODM files, starting the pseudonymization and on database import. A possible optimization is presented in Chapter 5.4.

4.5.2 Ease of use

Although a manual intervention is required to integrate pseudonymization into the i2b2 Hive it is not a downside at the moment. Because some components are still in development and version numbers change on a regular basis, it is positive user experience to be able to see what actually happens within the different components. This helps to better understand the complexity of the involved components. Also, the presented scenarios are not applied by a single person only. At interconnection points the involved parties have to agree on interfaces which suit the needs of the current scope of application.

For a use of patient identifier within the contexts of the same institution, it does not conflict with privacy protection rules to use a pseudonym of the first order (PID). However, a possible export of research databases which leaves the confines of an in-house use, would have to strengthen the used pseudonyms at least once, possibly twice (based on the nature of the export) or perform anonymization. An institution would have more options which can be directly performed if pseudonymization is already integrated into the research database. If data is to be stored for a longer time period, using pseudonyms of the second order is the method which complies well with accepted privacy protection rules.

The method presented in Chapter 4.4.1 to support strong PSNs within the i2b2 Hive could be adapted at any time to support the integration of PIDs. The implementation presented in Chapter 5 could be easily modified to support PIDs for in-house use within the i2b2 database.
5. Implementation

The first use case working in conjunction with the i2b2Dataimport application and presented in the previous chapter has been chosen to prove the general feasibility to integrate pseudonymization into the i2b2 Hive. The import interface is realized by a Java application which provides import statements for the i2b2 Hive on a project basis. A project basis is for example a complete export of a study database via CDISC-ODM v1.2 or v1.3. The import interface provides SQL import statements based on files which correspond to a single patient and multiple observations for a patient. It also provides a structure for an i2b2 Hive ontology and concept codes which were not used for this implementation. As mentioned before, this interface program was provided externally. The implementation of this thesis accepts the necessary output of the import interface and provides pseudonymization for it. As the data flows mainly in the direction of the i2b2 Hive to act as a research database, the action-type 'store' is supported. The implementation is also prepared to support future action-types and is easily expandable.

A complete implementation based on multiple web-services and automated wrapper interfaces was out of scope for this thesis. Custom output interfaces of study databases were not analyzed in detail because they are only accessible by the corresponding members of competence networks. Thus, the input data for the import interface was based on XML data files. However, an improved concept which considers these interfaces and suggests optimizations will be presented in Chapter 5.4.

The implementation is called psd-i2b2 and is designed to perform two independent tasks. These tasks are independent because pseudonymization in general must involve at least three different machines. It would be possible to locate one of the neighboring services of the PSD-Service (the SDB-Service or the FDB-Service) on the same machine as the Pseudonymization-Service. However, for security reasons this is not recommended. The PSD-Service uses internal messages and database components which could be compromised more easily by a local user. Additionally, there are reasons for restricting physical access to the PSD-Service because of the secret token stored on Smart-cards which have been mentioned earlier.

The minimum of three involved machines was also verified by an earlier concept implementation which was not feasible. This weaker concept intended to pseudonymize statements present on one machine only. The design was to issue a message, for example an entry in the patient register or the observation table to the PSD-Service over the SDB-Service. On arrival at the FDB-Service the completely pseudonymized message ought to be returned to the machine which originally triggered the pseudonymization and still contained material which was yet to be pseudonymized. The original message which
contains the patient identifier would have to be deleted in a secure manner before the pseudonymized message would arrive again from the FDB-Service. However, it could not be assured that a portion of the original message and the pseudonymized result were not present on the same machine at the same time. If this happened, the data of both messages could be cross-referenced by their file name, arrival- / departure time or other criteria. A secure overwrite, the usage of randomized file names and a long wait time for the completion of the pseudonymized result coupled with a randomized file order would have been required. These facts were considered too complex for the simple benefit of having the resulting data on the same machine as the output of the study database. Therefore, this concept was not practical and had been discarded for the mentioned reasons. The final concept now strictly enforces the use of different machines.

The implementation supports one modus operandi for each side of the PSD-Service. The first main functionality is called the SDB-mode. This mode analyzed the amount of data outputted by the import interface and excepts parameters required for the operation. It then prepares XML messages compliant with the PSD-Service. These messages can then be sent to an SDB-Service on a target machine for pseudonymization.

The second mode operates on the FDB-side of the involved services and is called FDB-mode accordingly. It processes pseudonymized messages and prepares them for insertion into the i2b2 Hive. The concept follows the project-based approach of the import interface and thus can be used to handle multiple target databases.

The scope of the implementation can be summarized as as set of wrapper methods which transform data to input and output rules of the involved interfaces. Additionally, it supports multiple forms of error handling and specifically checks if messages are well formed. It can also be easily modified or expanded, for example if the underlying i2b2 database scheme changes or additional action-types have to be supported.

### 5.1 Pseudonymization between import interface and i2b2

The implementation uses the Python programming language [79] as a basis. This language is similar to the Java language and supports both the object-oriented and the procedural programming approach. The Java language is considered to be somewhat overloaded with functionality in certain aspects by the author. Coding in the Python language often results in shorter code which can be more easily comprehended. For example, the JAXB framework for parsing XML files supports a vast variety of functions. A reader would have to familiarize him- / herself with large parts of the framework to understand the basic concepts. For Python there are multiple frameworks for XML processing available and most of them are considered more lightweight and do not lack functionality compared to their Java equivalents. This perfectly suits the scope of this thesis as there are already complex frameworks involved in the i2b2 software package. Additionally, it supports the
desired ease of use of the implementation concept. Python does not use square brackets to indicate a block level and no trailing semicolon. A block level is indicated by indentation only. This helps to improve code readability. Further, the language also supports multiple operating systems like Java and is already available on most of them.

5.1.1 SDB-mode

In SDB-mode, the implementation supports entering of the required parameters for a PSD-compliant message via the command line. Following the message layout presented in Chapter 4.2 the implementation supports entering of the following parameters:

- action=actiontype
- user=<PSD-authorized user>
- dir=<directory holding the output of i2b2Dataimport>.

Other parameters which depend on the input mentioned above are automatically set to static values by the application. For example, the text contents of the tag <from> and <to> which indicate the message direction are always set to 'SDB' and 'FDB' respectively. Also, the tag contents of <destination> and <returnpath> are set in the same manner for the SDB-mode. The setting of two tags containing both the same content and having the same meaning may seem redundant in this case. But there are cases where messages originate from the PSD-Service and the tag contents serve a different purpose. For example, if a getContext message requests patient data from the target database for the delivery via email, the message originates from the PSD-Service and the returnpath-tag contains the user name which issued the request.

The user variable must match a user who has been created in advance via the web front-end of the Pseudonymization-Service. It is recommended that users of the PSD-Service are kept at a minimum for security reasons. As there are no passwords involved in issuing a message to the PSD-Service, an unfriendly user could try to guess user names and send large messages which would consume processing time and/or result in a denial of service.

The directory containing the output files of i2b2Dataimport is controlled via the -dir switch of the implementation. Per default, the program assumes that a directory with the name 'output' exists in the current path.

Additional parameters are not needed for an initial pseudonymization but they could be added in future versions using the same approach which was used for this implementation. The application separates a command line parameter into the actual parameter name (which is fixed) and any value entered after the equals sign. This value could for example map to an IP address in later versions which maps to the address of a target SDB-Service.
Checking access rights

The next step in the program control-flow is to check the access rights of the directory which has been previously entered via the command line. The tool i2b2Dataimport creates four directories which correspond to the names of the i2b2 Hive target tables. Only two of these directories are important for this implementation, namely the folders 'patientDimension' and 'observationFact'. They contain the import statements which contain patient identifiers. A directory 'psd-output' is created during the execution of the application. It contains the messages which are PSD-compliant and have been prepared to be sent over the SDB-Service to the PSD-Service. Table 2 shows all involved folders and their relevance for the import process. A green color indicates that these directories are needed for the final import process on the target machine running the research database.

<table>
<thead>
<tr>
<th>directory name</th>
<th>relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>conceptDimension</td>
<td>i2b2Dataimport specific</td>
</tr>
<tr>
<td>observationFact</td>
<td>relevant; contains patient observations</td>
</tr>
<tr>
<td>ontology</td>
<td>i2b2Dataimport specific</td>
</tr>
<tr>
<td>patientDimension</td>
<td>relevant; contains patient related data</td>
</tr>
<tr>
<td>psd-output</td>
<td>created by psd-i2b2; contains prepared messages for the PSD-Service (in this case: SDB-Service)</td>
</tr>
</tbody>
</table>

Table 2: Directory names of import interface and psd-i2b2 and their relevance

The red color denotes that the implementation needs read access to these directories. If the psd-output folder does not exist, it is created by the application. On the other hand, if the directory already exists from previous runs it is checked for write access and not created again. All directories have in common that they are checked for their existence. If one directory is not found the program exits with an appropriate error message written to the log-file.

Working on input files

When all access rights are tested positive, the implementation continues to work on input files located in both relevant directories. If there are no files present psd-i2b2 exists with an appropriate log message. Continuing the control-flow files from 'patientDimension' are handled first. The implementation creates a list of all text files (with the extension .txt) and list entries are processes in a sequential manner.

A memory copy of the file contents is created and from this point on all further operations are using this memory copy. However, this means that the machine running the program must have enough RAM to store this copy. Additional checks as to whether the amount of available memory is sufficient are not performed.
File names are extracted during this process and kept until the intermediate processes finish successfully and a new output file can be created. For each input file an according output file is written later in the process. I2b2Dataimport uses the following naming scheme for output files:

```
<table name> <source network> <sequential numbering string>.txt
```

Before an output file is finally written the following intermediate steps are performed:

- The PID is checked for a correct formatting
- The PID is replaced with a special unique identifier string
- A PSD-compliant message is created.

The export of data from the study database contained patient identifiers which were not actually created by a PID-Generator as the export was a test data set. Therefore, it became necessary to simulate real PIDs because the PSD-Service only accepts identifiers of the correct format. Since it is practical to detect a possible ill-formatting of a PID early during the process, a check for correctness was introduced. If one of the input files of the implementation fails to provide a correct PID it exits with an error message. Thus, the implementation only operates on valid patient identifiers.

A complete PSD-compliant message must have any possible occurrences of patient identifiers removed from the payload data. Payload data in this case are SQL-statements needed for the import process. This data is not allowed to contain PIDs because payload data is not inspected by the PSD-Service. The only place where a PID is allowed to occur is the header section of a message. If a patient identifier occurred within the payload and a message was sent over the SDB-Service to the PSD-Service and existed at the FDB-Service, it would contain a pseudonym of the second order (a PSN) within the header section and a pseudonym of the first order (a PID) in the body-section. This is not allowed because it would enable the cross-referencing between PID and PSN. The resulting pseudonymization would be invalidated. Following this reasoning a PID should only occur once in the input data, specifically at the place where the value of the attribute `PATIENT_NUM` is located within an SQL-statement.

The unique identifier which is used as a placeholder for a sequential patient number is realized as a string value. This value should be chosen carefully so that the probability that it occurred within the input data is minimized. This also includes possible free text observations which may be inserted into the i2b2 Hive at the appropriate place (`observation_blob`). This value could be changed to contain a one-way hash value of sufficient length to further reduce the probability of collisions.
Files within the directory 'observationFact' are handled next but with a call to a different
detection function for the place of the patient identifier. When all files have been
processed the SDB-mode ends by showing the number of files which have been processed.

Creating of PSD-compliant XML messages

To create messages which are compliant with the PSD-Service and accepted by the SDB-
Service the lxml Python library [80] has been used. It provides the necessary features which
construct the needed XML structure.

The used concept is presented here in short as it represents a core step in the
pseudonymization process. The first task is to create a hierarchical representation of an
XML document within the computer memory. This is done using the etree module of
the lxml library. It provides the methods to create XML tag elements and work with XML
namespaces. XML namespaces are used to eliminate naming conflicts of tags in documents
which use names from multiple sources. The namespace of a PSD-message is shown in
line 1 of Listing 6. The listing shows an exemplary creation of several important elements
used in a PSD-message. The basis of all tags is a root element which is created in line 3.
The PSD-namespace has to be specified for all tags. Therefore, every element including the
root element, concatenates the namespace with the actual element name. The index
element which is a direct child node of the header element must contain the patient
identifier. These steps are performed in line 4-6. The body element is designed to hold
medical information in a semi-structured form. For this reason, a sub-element with the
name i2b2sql is introduced which holds this information in the form of UTF-8 encoded
SQL-statements. The described steps are performed at the end of Listing 6 in line 7-8.

```
1 XML_NAMESPACE = "http://www.tmf-ev.de/site/services/psd/psd.xsd"
2 XML = "{%s}" % XML_NAMESPACE
3 root = etree.Element(XML + "MDAT", nmap=NSMAP)
4 header = etree.SubElement(root, XML + "header")
5 index = etree.SubElement(header, XML + "index")
6 index.text = pid
7 body = etree.SubElement(root, XML + "body")
8 sql = etree.SubElement(body, "i2b2sql")
9 sql.text = unicode(data, 'utf-8')
```

**Listing 6: Example of constructing a PSD-compliant message with the lxml library**

Non ASCII characters which can occur in the input data are encoded in the UTF-8
character set. Using this method, special characters like German umlauts are retained
during transfer over the services. This does not only apply to characters of Western
character sets but theoretically also to other foreign character sets. If no encoding process
was used, special charters would be lost when an output file is written. The FDB-mode of
the implementation is later used to restore the encoded characters to their original form.
After the message construction the in-memory tree is serialized to a text representation and the XML declaration is added.

Validation

The serialized object is handed over to an XML-Schema validator which is included in the lxml library. This represents an additional part of quality assurance for the input data of the SDB-Service. This guarantees that the receiving service does not have to deal with invalid messages. If the validation fails for some reasons, an appropriate error message is written to the log-file and the application exits. The validation is performed using the official XML-Schema which is also available on the Internet (see footnote 14). The schema-file is included in the implementation for offline-use.

During the development process a small discrepancy has been detected in the involved XML-Schema. PSD-XML files use a namespace prefix called \texttt{psd} (see Listing 1, line 2). However, if this prefix is used according to the XML specification \cite{81} the prefix would also have to be used for all child elements of the root element. If this was implemented the document would not validate against the PSD schema. There are two possible solutions for this issue. First, the namespace prefix can be omitted because it is not absolutely necessary. This resulted in a valid PSD-compliant message which validated against the existing PSD schema. The second option would be to modify the underlying schema file to enforce namespace prefixes for all elements within a PSD XML file. However, this would include modifying the PSD-Service itself because it did not accept messages which included proper namespace prefixes. The first solution was used for the present implementation.

Identification of PID within source files

The implementation has to properly identify the place of the patient identifier within the source SQL-statements. This cannot be done by using a string search function because it would not know for which string to search. Also, the PID is not always at the same numerical position within an SQL-statements because the name of project could change resulting in a shorter or longer character count.

A solution to this problem uses the method of regular expressions. With the help of regular expressions the exact pre- or post-conditions which occur around the patient identifier can be defined. For example, the patient identifier always occurs in the values-section of an SQL-statement. As it is a string values it is encompassed by to colons (\texttt{'}). Further, it is always the first value according to the database scheme. Finally, the PID consists of exactly 8 alphanumerical characters. These pieces of information can all be used to construct an appropriate regular expression. Terminating the identification function, leading and trailing colons are removed from the identified PID.
This process will continue to work as long as the underlying database scheme is not changed. The SQL-statements of the observationFact table use a slightly different regular expression to identify a PID as it occurs on the second position within the SQL values-section.

### 5.1.2 FDB-mode

The FDB-mode of the implementation is designed to operate on files which are outputted by the FDB-Service. Analogous to the SDB-mode, it depends on a directory parameter which indicates the location of data files which are about to be inputted into the research database. The directory parameter defaults to location where the outgoing messages arrive (in this case: `/opt/fdbservice/FDB_IN` on the target machine running a Linux OS). Checks for access rights and writability also apply to this directory.

Arriving pseudonymized messages are automatically renamed by the FDB-Service. A trailing timestamp is added which consists of a date portion, a time portion and additional numbers to distinguish messages which arrive at the same point in time. However, this represents an area of improvement of the FDB-Service. Due to the renaming the output files of the FDB-Service have changing file name extensions on each message. This means that a receiving service or an implementation like the one presented in this thesis cannot process files based on their extension. This however, is a standard procedure to exclude other file types from being included in the processing which may reside in the above mentioned directory. A possible solution could be to insert the timestamp between the original file name an the file extension. Using this approach, files contents can be distinguished more easily with the help of their extension as the PSD-Service is able to handle files with a different extension than XML. The file name extension would be retained.

Additional command line switches specify a project name and a source network name. The project name is needed to differentiate i2b2 Hive target databases which are normally created on a project-specific basis (besides the Hive core tables). The final messages then contain this name to insert data into the correct database. The source network name is required to indicate which research or competence network originally created the patient identifier. This way, a PID can be tracked back to the corresponding issuer. Theoretically, the i2b2 Hive can also store pseudonyms which originated from multiple networks running multiple PID-Generators each. This may be a topic for future discussions, nevertheless this implementation is already prepared to supports this.

### Processing of final patient table entries

The main task of the FDB-mode of the implementation is the processing of output messages from the FDB-Service and transforming them so that they can be directly used as SQL import statements for the i2b2 Hive. The first task within the program logic is to
open and work on all pseudonymized messages which are targeted at the patientDimension table. These files contain the primary data which is needed first during the process to also prepare all messages containing medical observations. If the study data is not contradictory, every medical observation should be mappable to a patient entry.

Each message addressing the patientDimension target table contains a pseudonym of the second order and the previously mentioned unique identifier in the body-section. The first step is to parse the XML-message and gain access to its semi-structured contents. This is done by using the lxml library again but now in the opposite operating mode. The library unmarshalls the XML contents into a tree and accessor methods are used to gain access to the tag contents. In this case the PSN and the original SQL-statement are of importance.

If both contents are valid, the next core step of the implementation is performed. This phase needs the support of a Python dictionary data structure. A Python dictionary is able to store an unordered set of key: value pairs. Different methods provide access to those pairs and allow to walk over all dictionary entries. The dictionary is used in the following manner: A key in this case translates to a PSN. If a new PSN is identified within a message, the PSN is stored as a new key entry. If the PSN is not already in the set, a so-called 'patient counter' is incremented which indicates the overall number of patients described by the source files. The number of patients processes is stored as the value within the dictionary. Using this approach, the dictionary contains key: value pairs which can be used as a lookup table later. The patientDimension messages which are finally written to the disk now contain an arbitrary, but sequential integer number as patient identifier (PATIENT_NUM attribute).

**Processing of final observation table entries**

On completion of the processing of all entries related to the patientDimension table, the observationFact table entries are processed next. The output files of the FDB-Service again contain a PSN in the header-section of a message and the corresponding SQL-statement in the body-section. The steps of unmarshaling and checking the PSN for validity are repeated analogous to the processing of the patientDimension messages. But now, the dictionary contains all necessary key: value pairs to lookup the corresponding PSN to an integer number. Only when patient numbers are represented by integer numbers they are compatible with the target database scheme. After the lookup process all unique identifiers are replaced with the actual patient number. This process is only possible if the original source file outputted by the import interface creates exactly one file containing all the medical observations for the patient in question. Otherwise the implementation would have to operate on a per-insert-statement basis (which would also be possible). But the import interface created the files in a manner optimal for post-processing. Finally, the completed import statements are again written to the disk named with an appropriate file.
The proper UTF-8 encoding is used to maintain special characters which may occur in the input data.

**Creation of pseudonymized i2b2 import statements**

The actual creation of pseudonymized SQL import statements is achieved by walking though the complete dictionary mentioned above. In the final state of the dictionary it contains most of the information needed to fill the PATIENT_MAPPINGS table. Each entry in the dictionary translates to two import statements. The first entry is used to create a mapping between the patient mappings table and the patientDimension. It contains the same integer patient number as encrypted patient identifier (EID) and patient number with the attribute containing source network set to 'HIVE'. The second entry contains the PSN as encrypted identifier and the corresponding integer as patient number. Additionally, the specified source network identifier is set, the active state of the pseudonym is declared and date entries are added which indicate the time of import and also enable to potentially update an identifier. Table 3 clarifies the resulting structure. Different PSNs correspond to different unique pseudonyms. Complete PSNs have been omitted in Table 3 for readability.

The status string 'A' indicates that the pseudonym is actively being used, meaning that a research network is storing IDAT at the moment in conjunction with the patient list and the PID-Generator.

<table>
<thead>
<tr>
<th>PATIENT_IDE</th>
<th>PATIENT_IDE _SOURCE</th>
<th>PATIENT_NUM</th>
<th>PATIENT_IDE _STATUS</th>
<th>UPDATE _DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HIVE</td>
<td>1</td>
<td>A</td>
<td>2010-05-19 8:00</td>
</tr>
<tr>
<td>[PSN-A]</td>
<td>KFO</td>
<td>1</td>
<td>A</td>
<td>2010-05-19 8:00</td>
</tr>
<tr>
<td>2</td>
<td>HIVE</td>
<td>2</td>
<td>A</td>
<td>2010-05-19 8:01</td>
</tr>
<tr>
<td>[PSN-B]</td>
<td>KFO</td>
<td>2</td>
<td>A</td>
<td>2010-05-19 8:01</td>
</tr>
</tbody>
</table>

*Table 3: Final PATIENT_MAPPINGS table used to integrate pseudonyms*

Status strings which indicate that the pseudonym is inactive ('I') and cannot be linked to real patients any longer are imaginable. This status could also be used if patient identifiers have been anonymized by the PSD-Service. Exchanged messages over the PSD-Service do not change its structure when anonymization is used. The index-element of a message would simply hold a random string instead of a PSN.

Another possible value is 'D' if the original patient identifier has been deleted at the source system. In this case, the pseudonym is still valid and can be de-pseudonymized to its original PID-form using the PSD-Service. But the PID would not be associated with identifying data anymore. In this case a de-pseudonymization attempt would not be made.

The status 'M' for 'merged' for a patient EID could be used to indicate that the identifier did not originate from a valid instance of the PID-Generator but that it has been
transferred from another research database (while respecting patient privacy protection) that uses a different method of generating patient identifiers. In any case, an implementation has to apply the correct value according to the information provided by the original (study-) data provider. The status 'A' of the implementation at hand has been used for the purpose of demonstration. The correct value for test-pseudonyms would have been the status 'I' for invalid.

In contrast to the per-patient file approach of the observationFact and patientDimension tables, the patient mapping table uses a single file to store the final import statements. It is located in a sub-directory of the specified FDB-Service directory called 'psn_patientMapping'. This final step concludes the operation of the FDB-mode of the implementation.

### 5.1.3 Top level module psd-i2b2

The top level module psd-i2b2 contains the main-function which is responsible for executing the implementation. It is the only occurrence of the main-function throughout the source package. All other modules are imported when the need arises. When the application is first executed, presumably without any command line parameters, a function which prints the correct syntax is loaded. This function also displays usage information if any of the necessary command line parameters are incomplete or specified in a wrong way.

Above all else, a logger instance is created which is responsible to add additional information which is generated during the run-time of the program to the log-file. The log-level of the logger instance can always be modified to generate additional debug output. The standard log-level 'INFO' has been used which appends all information to the log-file which is classified INFO and above. This includes program errors for example. The logger object can be passed as an object to any parameter list of a function call within the top level module enabling the function to access the program-wide logging mechanism.

Further, all functions comply to the accepted documentation standard of Python modules. They include a short description of their purpose as first declaration within the source code. Additional comments have been place to further explain single statements or steps within the implementation. This renders the application easily expandable and easily debugable should the need arise in the future.

The package structure of the implementation is similar to the Java programming language. The main module is a file with the extension '.py' indicating that the source code is written in the Python language. All other modules are located in a sub-directory below the main module called 'psd_i2b2'. This layout is used for compatibility reasons with other applications. If a module of the package psd-i2b2 needs to import internal methods from the same package, it can directly do so because all source files are located within the same directory below the main module.
Figure 12 shows a flowchart of the implementation psd-i2b2. The figure shows the two main branches including the involved function calls.

**Figure 12: Flowchart depicting the main parts of the psd-i2b2 implementation**
5.2 Concept for export on finding

The rich client i2b2 Workbench allows to export patient sets which have been generated by searching for concept codes and other criteria. This export is primarily based on the arbitrary integer number which was assigned to each patient earlier by the implementation. When a new finding is generated during further research upon the resulting patient set it becomes a necessity to inform a patient about the specifics of this finding. The medical finding could include results for medication, a new form of therapy or the information about a genetic predisposition.

The direction of the data flow now becomes reversed and the receiving service which is normally located next to the FDB-Service becomes the sending service for the new finding message. In case of a finding message the data travels from the FDB-Service but its endpoint is the PSD-Service in this case. It must have been correctly configured in advance to receive and process finding messages. This includes contact addresses of an ethical committee which receives the message and (electronically) informs the attending physician. The information contains an 'ArztKey' which authorizes a valid user of the Pseudonymization-Service front-end, together with the original patient identifier, to query the actual contents of the finding message.

From the technical point of view, database querying and message wrapping is to be performed to enable finding message for the i2b2 Hive. The first step is to determine the possible set of input data formats for a service implementation. The most basic case would be a text file containing patient numbers. Since the i2b2 Workbench will possibly act as an intermediate point for acquiring medical data for researchers, the input data set could also consist of proprietary files based on spreadsheet calculation. The XML format would again be the preferred exchange format but extracting i2b2 Hive patient numbers from other content is also possible. Data formats can be institution-specific.

If the input data set has been determined the next task for an implementation is to query the i2b2 Hive project database and map the patient number to an encrypted EID with the help of the patient_mapping table. This can be accomplished by connecting to the underlying database and issuing an SQL JOIN request. At the least, this request has to link the input data with the patient mappings and the observationFact table containing the medical data. The resulting table rows would each translate to a single finding message for the FDB-Service.

An XML-message would have to be constructed in a similar way to the construction used in the presented implementation. The final XML-message would then contain the encrypted patient identifier in the form of a PSN and medical data in the body-section. Additional information like user name, action-type and message direction would also have to be provided according to the PSD specification.
The fact that the sending service of a finding process requires a direct database connection to create finding messages is not considered a downside in this case. As a result, it actually introduces an additional security level as i2b2 Workbench users have in most cases no direct access to the i2b2 project database. Users would have to carry their finding request message to a person who is responsible for generating such messages. Additionally, this person can exercise oversight over all finding messages and ensure that medical data is properly protected during the process.

Further, the circumstance that the pseudonym of the second order is not displayed directly within the context of the i2b2 Workbench or i2b2 Web-Client is not a disadvantage either. Researchers are not required to directly work with pseudonyms. But they must be properly informed of the designated work-flow: For example, that a finding is possible and whom to contact in such cases.

Should the need arise to provide external researchers with complete data sets already processed by the PSD-Service, either the use of a PSN2 or anonymization of the data set are possible as proposed in [12],[78]. A PSN2 is the original PSN encrypted again with a different secret by the Pseudonymization-Service. The method is determined by justifying the desired method during the application for research [12].

5.3 Alternative approach via PDOs

Users who hold the role 'admin' or 'manager' within the i2b2 Workbench are allowed to use the file import wizard to upload new data to the i2b2 Hive. This process depends on a CSV input file whose exact layout (position of data heading, if available and type of delimiters) can be chosen by the user. The import wizard then processes the input file based on the specified information. However, it does this by constructing a valid XML representation of a Patient Data Object (PDO) containing the source data from the CSV-file. The data is then sent to the clinical research chart cell where it is handled by SQL-statements and procedures which insert the data into the corresponding database tables. This has the advantage that the business logic included in that cell post-processes the data object and handles the different designated use cases [76]. This way, the end-user does not have to perform certain steps manually and the amount of manual interaction is reduced. For example if a database grows continuously and new patients are added in chunks, a custom pseudonymization implementation would have to include logic for determining the current patient count. Further, the business logic includes certain checks for the input data set. For example, checks which is destined for the column 'update_date'. If the patient data set of a PDO is more up to date than the information stored within the patient_mapping table, the affected rows are automatically updated.

A custom implementation for an import wizard could be designed to operate on more than just CSV-files. Especially when the export of a study database is already present in a
semi-structured format like XML, a step which transforms the data from XML to a CSV-based format and back again into an XML representation of a patient data object would be dispensable. An application could directly create a PDO which is then sent to the clinical research chart for insertion into the i2b2 Hive (given the necessary authentication against the project management cell). A validation of such an object would be a mandatory part to ensure its correctness. XML-Schema files for such a task are included in source package of the i2b2 server release. Such an import process is currently possible for observations, patient mappings and event mappings. Event mappings are the equivalent of patient mappings when an event contains an encrypted identifier from a non-i2b2 source. The table ENCOUNTER_MAPPING is then used in the same way as patient mappings to integrate encrypted event identifier into the i2b2 software package.

Practically, such an application could also be integrated as a cell into the i2b2 Hive and use the project management cell for authentication. A web-service would then have to accept CDISC-ODM on the incoming side and output a proper PDO. The important contents for pseudonymization have been presented in Listing 5 and an example of a complete patient data object is available in the CRC documentation [76]. However, whether all ODM contents can be mapped into a patient data object would be a topic for future discussion.

5.4 Future optimizations

There is additional scope for future optimizations for the proposed concepts and especially the involved software components. However, some projects like the i2b2 software project are continuously improved by external developers. Also, the inner structure of provider-specific study- and third party research databases is generally beyond reach for custom improvements. Nevertheless, older databases which do no support a semi-structured data interchange format should be upgraded over time.

The segments which can be optimized generally reside at interconnection points where data is transformed from one data format to another. A straightforward enhancement would combine parts of the i2b2 import interface and the pseudonymization routine on the side of the study database. These components could be merged so that one application prepares messages according to the format which is needed by the PSD-Service. It would be up to the developer to decide which implementation should be expanded. The general concepts could be integrated into both programs. If the need arises, the interfaces could be enriched to support additional PSD-message types. For example, delete or update messages originating at a study database could be processed and trigger an equal response within the i2b2 Hive database. This would further support a research database which is kept up to date. When a research database state is final, for example when the original study database is discontinued and data resides only within a research database, the project database should be marked accordingly and maybe converted to a read-only operating mode.
An additional improvement would be to add general database connectivity (e.g. JDBC) to the FDB-side of import interfaces. As it is assumed that the i2b2 Hive database does not change large portions of its table structure an implementation could process incoming pseudonymized messages and perform the desired database action directly. Necessary parameters like project names and database access credentials would have to be queried in advance. The i2b2 business logic which is already in place in the CRC cell could serve as a first reference and starting point. At the moment, the import has to be manually triggered but can, for example, be automated with Apache Ant (see Chapter 3.2.8).

At the moment it is difficult to connect to SDB- / FDB-Services via web-services to realize an asynchronous communication of the involved components. The reason for this is that specific reference implementations (SDB- / FDB-side) do not exist yet. Such reference implementations would have to support the following properties to further increase the practicality of the presented use cases:

- Support a standardized input set (for example, a commonly used study database or interchange format like CDISC-ODM / Health Level 7 (HL7)).
- Support all message types of the PSD-Service to enable asynchronous communication between study- and research databases and research databases and the PSD-Service.
- Provide the logic for a typical direct connection to a research database (for example, an Oracle-, MySQL- or XML-based database).
- When large databases are used, a reference implementation should optimize the access speed by creating or updating necessary database indexes.
- Support logging or record database transactions on the FDB-side to enable rollbacks and undo operations (sometimes this feature is already integrated into RDBMS).
- A dense documentation to facilitate the use of additional instances of research databases which integrate pseudonymization at different locations.

The implementation concept used within this thesis applied the PSD-Service relatively late during the process. When several institutions set up their own research databases (see Chapter 4.4.3) which may be based on a different software than the i2b2 framework, a reference implementation would further expedite this specific use case. A provider-specific research database would already contain a PSN when data is to be exported to additional research databases like the i2b2 Hive. This process could also allow to evaluate potential useful software packages more easily. A PSN would already be present within a structured export from a provider-specific research database. This would reduce the work effort needed to load data into the target database. An import interface would still be responsible for creating the necessary entries in the patient_mapping table in case the i2b2 Hive is being used.
Further, test data could be replaced with pseudonymized data up to a degree as the security of patient identifiers is much higher. If exports to external researchers are made, it is also easier to perform the transformation of PSN to PSN2. For example, an external target research institution would simply operate its own FDB-Service. The PSD-Service holding the secret which is responsible for transforming PSN to PSN2 would then be located within the control of the first institution. The external target network would then process incoming messages or a complete data set according to their own needs and environment.
6. Summary and discussion

This thesis has analyzed different aspects of the integration of pseudonymization services into research databases, specifically the i2b2 software package. The basic concepts of protecting patient privacy in Germany acted as a foundation for this effort. A short excursion covered the current situation in the United States and some European countries.

The underlying frameworks and methodologies used by the two main software components which have been analyzed in this thesis were presented in detail. Their individual importance has been made clear to the reader, especially in the area of web-services, databases and building applications from their source.

The nature of the interfaces of the Pseudonymization-Service and its components Study-Database-Service and Research-Database-Service has been shown and their classification within the greater context of network-wide Patient-Lists has been pointed out.

Use cases which integrate the previously mentioned components in different scenarios of study- and research databases have been created, depicted and described. They have been supplemented by technical details which are important if an i2b2 Hive installation is being considered at other locations.

The specific use case of having a semi-structured study database export and wanting to strengthen contained patient identifiers of the first order (PIDs) with the help of the PSD-Service into pseudonyms of the second order (PSNs) has been implemented. The resulting data set consists of a statement necessary to fully integrate pseudonyms into an unaltered i2b2 database scheme. However, manual interaction is still required at some parts of the import process. Therefore, an alternative to this approach and possible optimizations for the involved components have been proposed to facilitate future use of pseudonymization.

All of the involved components have been set up on private computers running on derivatives of the Linux operating system. To be able to work with them and to provide a realistic development environment, a minimum of three machines was needed to support the described use case. A script collection from S. Mate at Universitätsklinikum Erlangen has greatly accelerated the first installation procedures of the i2b2 Hive. The Linux shell scripts have been slightly modified to suit the local needs.

Further, the author participated in workshops in Erlangen and Berlin where specifics of the i2b2 software package have been discussed and early concepts of this thesis and the involved Pseudonymization-Service have been presented.

Within the context of the original problem definition and the topic of this thesis, the central questions have not only been answered but supported by a proof of concept
implementation. Methods used within this proof of concept could also be adapted to third party implementation as it has been released via the Common Development and Distribution License (CDDL) which allows free code usage in other applications.

The integration of PSNs into a research database is a new improvement that can be adopted by other databases. This pseudonym is not stored in the primary patient data table but in an extra patient mapping table which represents an additional security step. The fact that PSNs are not allowed to be a primary key within a target database is a precondition of the PSD-Service. Patients are only identified by a newly assigned integer number. This helps that the pseudonyms are not displayed in an unnecessary fashion.

The fact that the patient mapping table is already prepared to store encrypted patient identifiers shows that the i2b2 Hive database scheme was carefully designed by U.S. developers and patient privacy protection was kept in mind. However, additional information on actively used patient mappings within U.S. i2b2 installations is somewhat sparse. The assumption is, that encrypted patient identifiers are currently not used on a large scale basis. References to this topic could not be found and details of patient mappings were not discussed on the mailing-list of the i2b2 academic user group since the author has been a member of this list.

The development of additional database scripts which work directly on the i2b2 Hive for importing and exporting patient data including pseudonyms could be beneficial for future use of the i2b2 Hive as a research database or medical data warehouse.

In general, the i2b2 Hive and the presented approach are adequately usable to integrate pseudonymization into this specific research database. However, the currently necessary manual processing steps could present a small obstacle for an end-user. The user would have to be familiar with several technologies and operating systems. An approach which uses graphical user interfaces and automates certain processes would have been better suited to fit the needs of an end-user. It has been considered to implement a graphical UI but it could not be developed within the scope of this thesis. Additional interoperability issues would have risen when using graphical interfaces and different operating systems.

An alternative to an i2b2 Hive installation as a research database would have to support pseudonymization in an equal or similar manner: the storage of pseudonyms within the underlying database. Pseudonyms should not be stored as primary key within a core table for security reasons. Alternatives should either support pseudonyms out of the box or by simple additions to the underlying storage method. The i2b2 star-schema for storing relations is probably the fastest and most expandable as additional tables can simply be added and joined with existing attributes when necessary.

In terms of usability the 1.3 version of the i2b2 Workbench had some drawbacks. Some user actions froze the software and it was reported unresponsive by the operating system. However, the underlying data was not affected and previously stored queries were still
available to continue the work at the last point. When the network was slow or not available, the Workbench also reported that it is not responding any more. Further tests have shown that the network timeout value is very high for connecting to the project management cell. During this time the Workbench does not accept any user input. The time-out value can be decreased but nevertheless, the application is reporting a wrong state to the Microsoft Windows Operating System during that time. For example, if the wrong domain was chosen accidentally the action could not be canceled for a lengthy period of time. The analyses functions of resulting data sets are of simple graphical design but can hold lots of parameters. If more sophisticated graphical representations are needed, the resulting data could be exported or a Java visualization plug-in could be integrated into the Workbench at any time due to the open source approach of i2b2.

Additional use cases may be derived from the ones presented in this thesis. It is likely that most of the presented components are reused for those concepts. However, the exact process of a new medical finding message has not been analyzed in detail in this thesis due to time constraints. Especially, the work-flow of messages between a PSD-Service, an ethical committee and the operator of the Patient-List storing patient identifying information could be analyzed further. Look-up processes which allow the mapping of PIDs to IDAT could be included in such an analysis. The structure of an ethical committee would also be of interest. It can either approve or disprove of medical finding messages to be sent to the corresponding medical doctor. Surely, the individual case must be evaluated but a set of guidelines for example decisions could be created. Data protection officers and doctors with the necessary fundamental medical knowledge could be part of such a committee.

There are additional security considerations when the i2b2 Hive is productively being used. Research networks are often connected via a VPN, but the communication between i2b2 Hive cells is not encrypted at the moment. The cells make use of the standard hypertext transfer protocol to transfer data. Due to the distributed nature of the i2b2 Hive, cells can be set up at different physical locations. This mainly supports the performance aspect of the cells but also creates security problems. The different cell must exchange data over the underlying network. In a typical switched environment it is easy to capture all of the transferred data from any computer within the same network. Requests from the client applications and responses from the i2b2 Hive cells containing medical data can thus be recorded by a third party. Within these messages, passwords to access the PM cell and thus access the i2b2 Hive are stored in clear text form in each message. An attacker could steal access credentials and create his own authenticated client application. When data is transferred over longer distances its contents are eventually being protected by a VPN, but messages are still unencrypted during the complete transit. This fact makes use of the i2b2 Hive over a standard internet connection impossible due to privacy protection.
A possible solution could be the introduction of certificates to secure the underlying transport layer. If each cell is enhanced with a certificate, it can only exchange information with an authenticated cell. These certificates could be issued locally for testing purposes by using self-signed ones. A client application would also have to integrate certificates to secure the communication between the tiers. However, this approach can also enable additional possibilities. A researcher could be given a time-limited certificate to install within its client application. Then the appropriate Hive cells are also set up with a corresponding certificate. During the time the certificate is valid, a researcher would have project-based access to the research database over a strongly encrypted communication channel. When the certificate expires, it is automatically invalidated by the i2b2 Hive and access is no longer possible. The Java Enterprise Edition application framework fully supports securing web-applications in this manner [82].

Currently the presented use cases lack a practical concept for user management. There are many places within the involved software components where user names and passwords have to be specified. A preliminary list where user names and / or passwords have to be specified consists of the following entries: computer access (desktop and file access), i2b2 project management cell, PSD-Service, study database access (project and admin access), smart-card PIN numbers, PGP key pass-phrases (for PSD email encryption), certificate key passwords and web-service portal access. A generic concept for managing authentication for the complete work-flow was out of scope for this thesis. But it should be considered that it is not trivial to manage access rights securely. A weak point in the authentication chain are often user:password combinations which have been written down or made otherwise accessible to unauthorized entities. A starting point could be the expansion of the i2b2 project management cell to include credentials for additional components.

Besides that a working PGP infrastructure for secure business email communication should always be considered. The necessity for a receiver of a PSD-message to possess a valid public email encryption key could further support an encrypted email infrastructure within research networks. A today's minimum email security feature should be the signing of outgoing messages so that a receiver is able to verify the sender and ensure that the email has not been altered during transit. In addition, it signals the receiver that sending an encrypted email to the original sender is possible by using a public key. Private key servers could be used to not expose business email addresses to senders of unwanted bulk email.

The i2b2 Hive supports the integration of encrypted patient identifier which originate from different sources. Thus, identifiers which originate from different PID-Generators are possible within the i2b2 Hive. If it cannot be guaranteed that two networks issue different PIDs for a person with the same identifying information while using different instances of the PID-Generator, merging identifiers from multiple sources into the patient mappings table should be avoided. However, if the identifiers are completely different, for example
because different initialization vectors and random number generators are used to generate new patient identifiers, even the same patient would map to completely different PSNs when using the PSD-Service. A patient would then have multiple entries within a project based research database. The patient could be identified as a duplicate only by closely examining the medical data. If data of the same patient would accumulate in one research database the chance of re-identification would rise with each additional entry. This is also the case if additional data for the same patient is subsequently added to the database because the same patient identifier translates to the same pseudonym [21]. However, multiple observations for one patient also increase the chance of new research results. It must be carefully considered whether such approaches do not violate data protection laws. Individual cases should be discussed with the appropriate data protection officers.

### 6.1 Future work

The i2b2 Hive version used within this thesis is generally version 1.3. During the design period of this thesis, the new production version 1.4 was released by the i2b2 developer team. It includes several improvements mainly in the area of usability and stability. The discontinued 'Gridsphere Portal Framework' was completely removed. This component was replaced with a custom software to administrate the project management cell, registering the i2b2 Hive cells in general and other tasks which have been previously accomplished by Gridsphere. This replacement removed an error-prone step in the installation of the i2b2 Hive. Gridsphere had also particular problems with changing IP addresses when the i2b2 Hive was installed to a virtual machine. For example, if the outer network environment changed and new IP addresses were automatically assigned, the registered cells had to be reconfigured to the new addresses. However, Gridsphere became confused with changing addresses and a reconfiguration was often not possible. The new application also removes some complexity and runs on the same JBoss Application Server as the Ontology and Data Repository cells [83]. An updated i2b2 virtual machine image supports IP address auto-configuration so that it can be connected with a client application easily for demonstration purposes.

The general database layout of the i2b2 Hive has not been changed so that encrypted patient identifiers are still supported. The import of patient mapping tables can function in the same way as in version 1.3. However, there may be changes for ontologies and concepts codes. The import process should be evaluated for the new version. Possible changes should be integrated into the i2b2Dataimport so that an automated wrapping of CDISC-ODM is still possible. At some point it should be decided if the desired import way is via a direct database connection or in conjunction with the i2b2 business logic located in the CRC cell. Of course, also both methods could be used. This thesis has supported the first approach but the possibility of an i2b2 Hive plug-in or web-service which accepts output
messages of the FDB-Service, transforms them to a PDO and sends them to the CRC cell could be evaluated.

Work on the installation scripts created at Universitätsklinikum Erlangen also continues for the 1.4 version of the i2b2 Hive\textsuperscript{22}. Up to today the script has been enhanced by a graphical, console-based user interface which queries necessary parameters and performs additional tasks. Ideally, the development for this tool continues for future i2b2 software releases.

Most recently, the release candidate (RC) 1 of the 1.5 branch of the i2b2 query and analysis tool has been published within the internal area of the i2b2 website which is accessible after registration\textsuperscript{23}. Release candidates can provide a valuable insight for future changes of software. Analyzing release candidates can enable better planning necessary alterations to involved steps like installation, import and pseudonymization. Nonetheless, an update or a new install of a production version should only be performed when a stable version is available. Otherwise, installations could be damaged or rendered defective. Using virtual machines, backups of production versions should be made due to the complexity of the update process.

A website which lists current development efforts of the i2b2 team can also act as a starting point for future work\textsuperscript{24}. Among the current endeavors are, for example:

- The Workbench ability to find specific values in labs or gene expression.
- File Repository Cell - A core cell that manages files using the i2b2 Workbench, allowing large file storage (images and microarray results) and referencing by the clinical research chart.
- Image Annotation Cell - Infrastructure to allow image viewing and annotation in the clinical research chart.

Further, the uploading of data to the clinical research chart by using the Workbench is being improved. With the current effort to integrate image storage into the i2b2 Hive, pseudonymization of images and image meta-data could be an interesting future topic. Also, the integration of data originating from German bio-material databases into the i2b2 Hive could be evaluated.

If large data sets are being worked on in the future, it is possible that some software components (except the externally developed i2b2 Hive) presented in the use cases of this thesis may need constant improvement. The need for revised versions is normal for a software product which has been so far only used within small work-groups. Revised versions would have no need for additional functions in the beginning but in terms of refactoring the source code, adding additional documentation and optimizing for speed and

\textsuperscript{22} http://wwwcip.informatik.uni-erlangen.de/~sisemate/
\textsuperscript{23} https://www.i2b2.org/RC/Download.php
\textsuperscript{24} https://www.i2b2.org/software/inprogress.html
usability. The goal of a broad research database is only possible when large data sets can be pseudonymized or anonymized in a sufficient amount of time. This amount of time should be smaller compared to the time needed to actually work with the research database and generate new research results. Specifically web-services and web-clients can be a good spin-off when they are well designed. If it can be determined that a service is vital for long-term use cases, dedicated software developers could be considered.

The Pseudonymization-Service could be evaluated to support also encrypted finding attachments. At the moment, it already supports encryption text messages for a medical finding. However, the medical contents are attached to the email in an unencrypted form. A medical doctor who receives this encrypted message must already have a security token to decrypt the message. The file attachment could be encrypted using the same key. If a new electronic infrastructure is established in Germany, medical doctors are required to authenticate themselves against a patient so that electronic contents become available to them. It could be evaluated if this authentication mechanism is compatible for encrypting finding messages and their medical file attachments.
# A. Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Java) EE</td>
<td>(Java) Enterprise Edition</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BMBF</td>
<td>Bundesministerium für Forschung und Bildung</td>
</tr>
<tr>
<td>CDDL</td>
<td>Common Development and Distribution License</td>
</tr>
<tr>
<td>CDISC-ODM</td>
<td>Clinical Data Interchange Consortium-Operational Data Model</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CRC</td>
<td>Clinical Research Chart (within the context of the i2b2 Hive)</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma-Separated Values</td>
</tr>
<tr>
<td>DAO</td>
<td>Data Access Object</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>EID</td>
<td>Encrypted Identifier (within the context of the i2b2 Hive)</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise Java Bean</td>
</tr>
<tr>
<td>ETL</td>
<td>Extract, Transform, and Load</td>
</tr>
<tr>
<td>EU</td>
<td>European Union</td>
</tr>
<tr>
<td>FDA</td>
<td>(U.S.) Food and Drug Administration</td>
</tr>
<tr>
<td>FDB</td>
<td>Forschungsdatenbank [german for research database]</td>
</tr>
<tr>
<td>GG</td>
<td>Grundgesetz [German for Basic Law for the Federal Republic of Germany]</td>
</tr>
<tr>
<td>GINA</td>
<td>Genetic Information Nondiscrimination Act</td>
</tr>
<tr>
<td>GPL</td>
<td>GNU General Public License</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HGP</td>
<td>Human Genome Project</td>
</tr>
<tr>
<td>HHS</td>
<td>Department of Health and Human Services</td>
</tr>
<tr>
<td>HIPAA</td>
<td>Health Insurance Portability and Accountability Act</td>
</tr>
<tr>
<td>HL7</td>
<td>Health Level 7</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>i2b2</td>
<td>Informatics for Integrating Biology and the Bedside</td>
</tr>
<tr>
<td>ICD-10</td>
<td>International Statistical Classification of Diseases and Related Health Problems, 10th Revision</td>
</tr>
<tr>
<td>ID</td>
<td>Identifier</td>
</tr>
<tr>
<td>IDAT</td>
<td>Identifying Data</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>IV</td>
<td>Initialization Vector</td>
</tr>
<tr>
<td>JAX-WS</td>
<td>Java API for XML Web Services</td>
</tr>
<tr>
<td>JAXB</td>
<td>Java Architecture for XML Binding</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JIT</td>
<td>Just-In-Time</td>
</tr>
<tr>
<td>JRE</td>
<td>Java Runtime Environment</td>
</tr>
<tr>
<td>JSP</td>
<td>Java Server Pages</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MB</td>
<td>Megabyte</td>
</tr>
<tr>
<td>MDAT</td>
<td>Medical Data</td>
</tr>
<tr>
<td>NCBC</td>
<td>National Center for Biomedical Computing</td>
</tr>
<tr>
<td>NDSG</td>
<td>Niedersächsisches Datenschutzgesetz</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>OPS</td>
<td>Operationen- und Prozedurenschlüssel [German for International Classification of Procedures in Medicine]</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PDO</td>
<td>Patient Data Object (within the context of the i2b2 Hive)</td>
</tr>
<tr>
<td>PHI</td>
<td>Protected Health Information</td>
</tr>
<tr>
<td>PID</td>
<td>Patient Identifier</td>
</tr>
<tr>
<td>PIN</td>
<td>Personal Identification Number</td>
</tr>
<tr>
<td>PRNG</td>
<td>Pseudorandom Number Generator</td>
</tr>
<tr>
<td>PSD</td>
<td>Pseudonymisierungsdienst [German for Pseudonymization-Service]</td>
</tr>
<tr>
<td>PSM</td>
<td>Patient Set Message (within the context of the i2b2 Hive)</td>
</tr>
<tr>
<td>PSN</td>
<td>Pseudonym</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RC</td>
<td>Release Candidate</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SDB</td>
<td>Study-Database</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>TMF</td>
<td>Telematikplattform für Medizinische Forschungsnetze (e.V.) [German for Telematics Platform for Medical Research Networks]</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>UTF-8</td>
<td>8-bit Unicode Transformation Format</td>
</tr>
<tr>
<td>VDO</td>
<td>Vocabulary Data Object (within the context of the i2b2 Hive)</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WDSL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Document</td>
</tr>
</tbody>
</table>
### B. List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Overview of Basic Laws relevant for scientific research (cp. [20])</td>
</tr>
<tr>
<td>Figure 2</td>
<td>i2b2 Hive [43]</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Typical three-Tier architecture with encapsulated database access</td>
</tr>
<tr>
<td>Figure 4</td>
<td>three-Tier architecture of i2b2</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Layout of the study database service of the PSD</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Layout of the research database service of the PSD</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Layout of a SOAP-message [74]</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Basic i2b2 message construction including schema files</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Import concept with PSD-Service (top) and transfer through (bottom) approach</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Data flow scenario placing the PSD-Service between study database domain and research database domain</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Use case addressing provider-specific research databases and pseudonymization</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Flowchart depicting the main parts of the psd-i2b2 implementation</td>
</tr>
</tbody>
</table>
C. List of Code-listings

<table>
<thead>
<tr>
<th>Listing</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Listing 1</td>
<td>Example of a PSD-compliant message format</td>
</tr>
<tr>
<td>Listing 2</td>
<td>JAXB derived class representing the header of a PSD-compliant XML representation for the SDB-webservice</td>
</tr>
<tr>
<td>Listing 3</td>
<td>Sending a message to the SDB-webservice and returning the response</td>
</tr>
<tr>
<td>Listing 4</td>
<td>Pseudonym (PSN) showing an increased entropy in comparison to a PID</td>
</tr>
<tr>
<td>Listing 5</td>
<td>Example PDO code snippet showing a PID set containing patient mappings to source systems</td>
</tr>
<tr>
<td>Listing 6</td>
<td>Example of constructing a PSD-compliant message with the lxml library</td>
</tr>
</tbody>
</table>
### D. List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Software entities with possible machine type an operating system</td>
</tr>
<tr>
<td>Table 2</td>
<td>Directory names of import interface and psd-i2b2 and their relevance</td>
</tr>
<tr>
<td>Table 3</td>
<td>Final PATIENT_MAPPING table used to integrate pseudonyms</td>
</tr>
</tbody>
</table>
E. Appendix

E.1 Source code of implementation: psd-i2b2

Module: psd-i2b2.py

```python
#!/usr/bin/env python

# handles command line parameters
import getopt
# provides system access
import sys
# package which provides PSN functionality
from psd_i2b2 import check_dir_access
from psd_i2b2 import handle_files

# Author: Lars Reimann
# Date: 2010-05-19
# Released under CDDL (http://www.sun.com/cddl/cddl.html)

# The prepared .xml files for the PSD-Service arrive in ./output/psd-output.
# Copy these to the SDB-Service output directory to init pseudonymization.
# Also copy statements with do not have to be pseudonymized (e.g. ontology
# and conceptDimension) to the target machine.
# Use the FDB-mode on the target machine to finalize the process.
# Pseudonymized sql statements will be created under the FDB-IN dir.

def usage():
    """
    Print usage information
    """
    print "
    psd-i2b2 is used to work on i2b2 compatible sql statements created by
    i2b2Dataimport.jar. It pseudonymizes statements over the PSD-Service
    for PATIENT_DIMENSION and OBSERVATION_FACT. The FDB-Side creates
    PATIENT_MAPPINGS statements used to finally integrate pseudonymization
    into the i2b2 Hive.
    The program can be used in two modes ( SDB / FDB ). Therefore,
    pseudonymization is a two-step process.
    Be sure to use one machine for each Service (SDB/FDB). Otherwise, PID
    and PSN could be present on the same machine which would violate
    patient privacy.
    
    Usage:
    
    Mode 1 (SDB-side):
    -s, --sdb               Use SDB-mode (mandatory for SDB-mode)
    
    Mode 2 (FDB-side):
    -f, --fdb               Use FDB-mode (mandatory for FDB-mode)
```
-i, --in=<dir>     Input directory of FDB-Service
(defaults to /opt/fdbservice/FDB_IN)
-p, --project=<proj>     Name of the project target tables
-n, --network=<net>     Name of the network running an instance
General switches:
-h, --help     Display this help message

def main(argv):
   
   try:
   
      "sdb", "action" 
      "user", "dir", "fdb", "in", "project", "network", "help")
      action = 'store'
      user = 'superuser'
      sdb = None
      # out = '/opt/sdbservice/SDB_OUT'
      fdb = None
      fdb_in = '/opt/fdbservice/FDB_IN'
      dir = './output'
      
      # This names the target project name table
      project = 'i2b2demodata'
      
      # Assignes a network name to patient mappings
      # (e.g. where did the PID come from)
      network = "AHF"
   
   except getopt.GetoptError:
      print "Error while parsing command line options"
      usage()
      sys.exit(1)

   for opt, arg in opts:
      if opt in ('-h', '--help'):
         usage()
         sys.exit(1)
      #sdb mode
      elif opt in ('-s', '--sdb'):
         sdb = True
         logger.info("Using SDB-MODE.")
      elif opt in ('-a', '--action'):
         # default action is to generate store messages
         action = arg.lstrip("")
         logger.info("Using STORE mode for PSD-messages")
      elif opt in ('-u', '--user'):
         user = arg.lstrip("")
         logger.info("Using PSD-user: " + user)
      elif opt in ('-d', '--dir'):
         dir = arg.lstrip("")
         logger.info("Using i2b2Dataimport output directory: " + dir)
         
      # fdb mode
      elif opt in ('-f', '--fdb'):
         fdb = True
         logger.info("Using FDB-MODE.")

   return None

if __name__ == '__main__':
   main(sys.argv[1:])
elif opt in ("-i", "--in"):
    fdb_in = arg.lstrip('=')
    logger.info("Using FDB-directory: " + fdb_in)
elif opt in ("-p", "--project"):
    project = arg.lstrip('=')
    logger.info("Using Project name (target tables): " + project)
elif opt in ("-n", "--network"):
    network = arg.lstrip('=')
    logger.info("Using Network name: " + network)

source = "".join(args)

# use sdb-mode and check for parameters
if ( sdb == True and action != None and user != None and dir != None
     and fdb == None ):
    print "Checking access rights..."
    temp = check_dir_access.check_sdb(dir)
    if ( temp == True ):
        print "Access rights OK"
    else:
        print "You have not the neccessary access. See the log."
        logger.error( temp )
        sys.exit(2)

# work on input files
    temp = handle_files.read_sdb(dir, action, user, logger)
    if ( temp == True ):
        print "Input files correctly parsed"
    else:
        print "An error occured while processing input files. Check the log!"

# use fdb-mode and check for parameters
elif ( fdb == True and fdb_in != None and project != None and
       network != None):
    print "Using FDB-Mode!"
    temp = check_dir_access.check_fdb(fdb_in)
    if ( temp == True ):
        print "Access rights OK"
    else:
        print "You have not the neccessary access. See the log."
        logger.error( temp )
        sys.exit(2)

# work on input files
    temp = handle_files.read_fdb(fdb_in, project, network, logger)
    if ( temp == True ):
        print "Input files correctly parsed"
    else:
        print "An error occured while processing input files. Check the log!"

elif ( sdb == True and fdb == True ):
    print "SDB-Mode and FDB-Mode cannot be used at the same time"
    print "Check your command line arguments"
else:
    print "Mandatory parameters have not been set!"
    usage()
    sys.exit(1)

return 0

if __name__ == '__main__':
### check_dir_access.py

```python
#!/usr/bin/env python

# Author: Lars Reimann
# Date: 2010-05-19
# Released under CDDL (http://www.sun.com/cddl/cddl.html)

def check_fdb(directory):
    """
    Checks if FDB_IN directory used by the FDB-Service are existent, readable, writeable
    """

    if ( not os.access(directory, os.F_OK) ): return "Path not found: " + directory
    elif ( not os.access(directory, os.R_OK) ): return "Path not readable: " + directory
    elif ( not os.access(directory, os.W_OK) ): return "Path not writeable: " + directory
    else:
        # maybe check for empty directories
        temp = check_create_dir(directory + '/psn_patientDimension')
        if ( temp != True ): return temp
        temp = check_create_dir(directory + '/psn_observationFact')
        if ( temp != True ): return temp
        temp = check_create_dir(directory + '/psn_patientMapping')
        if ( temp != True ): return temp

# end def
```

---

```python
***
Passes the argument list to the main() function
***

# use logging
import logging
import logging.handlers

# create a logger
LOG_FILENAME = './log.psd-i2b2'

logger = logging.getLogger("psd-i2b2.py")
logger.setLevel(logging.INFO)

handler = logging.FileHandler(LOG_FILENAME)

# create a formatter
formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s: %(message)s")

handler.setFormatter(formatter)

logger.addHandler(handler)

main(sys.argv[1:])
sys.exit(0)
```

---
def check_create_dir(dir):
    
    # check for a psd output directory
    output = dir
    if ( not os.access( output, os.F_OK ) ):  
        # path not found, creating it 
        try: 
            os.makedirs(output) 
        except: 
            return output + " could no be created!" 
        # create OK 
        return True 
    else: 
        # path exists, check for writeability 
        if ( not os.access (output, os.W_OK) ): 
            return "Path not writeable: " + output 
        else: 
            # all ok 
            return True 
    return True 

def check_sdb(directory):
    
    if ( not os.access(directory, os.F_OK) ): 
        return "Path not found: " + directory 
    elif ( not os.access(directory, os.R_OK) ): 
        return "Path not readable: " + directory 
    elif ( not os.access(directory, os.W_OK) ): 
        return "Path not writeable: " + directory 
    else: 
        # check for subdirs created by i2b2Dataimport 
        patientDimension = directory + '/patientDimension' 
        if ( not os.access( patientDimension, os.F_OK ) ): 
            return "Path not found: " + patientDimension 
        elif ( not os.access( patientDimension, os.R_OK ) ): 
            return "Path not readable: " + patientDimension 
        elif ( not os.access( patientDimension, os.W_OK ) ): 
            return "Path not writeable: " + patientDimension 
        else: 
            observationFact = directory + '/observationFact' 
            if ( not os.access( observationFact, os.F_OK ) ): 
                return "Path not found: " + observationFact 
            elif ( not os.access( observationFact, os.R_OK ) ):
return "Path not readable: " + observationFact

eelif ( not os.access( observationFact, os.W_OK ) ):
    return "Path not writeable: " + observationFact

else:
    # check for a psd output directory
    psdoutput = directory + '/psd-output'
    if ( not os.access( psdoutput, os.F_OK ) ):
        # path not found, creating it
        try:
            os.makedirs(psdoutput)
        except:
            return psdoutput + " could no be created!"
        # create OK
        return True
    else:
        # path exists, check for writeability
        if ( not os.access (psdoutput, os.W_OK) ):
            return "Path not writeable: " + psdoutput
        else:
            # all ok
            return True

Module: handle_files.py

#!/usr/env python

# work with input file patterns
import glob

# identify pid in input data
import pid

# sys access
import sys

# psd xml
import psd_xml

# psn functions
import psn

# patient mappings
import patient_mapping

# --------------------------------------------------------------------------
# Author: Lars Reimann
# Date: 2010-05-19
# Released under CDDL (http://www.sun.com/cddl/cddl.html)
# --------------------------------------------------------------------------

def read_fdb(directory, project, network, logger):
    
    """Function responsible for creating pseudonymized statements
    """
    
dict_psn = {}
dict_pat_mappings = ""

    # sadly, no pattern can be used due to trailing timestamp
    matches = glob.glob ( directory + '/patientDimension*' )

    if matches:
        # process patientDimension files
        temp = process_patientDimension( matches, directory, project, dict_psn, logger )
if ( temp == True ):
    # process observationFact files
    matches2 = glob.glob( directory + '/observationFact*' )
    if matches2:
        # handle files
        temp = process_observationFact( matches2, directory, dict_psn, logger )
        # for each dictionary entry, create two insert statements
        for psn, pat_num in dict_psn.iteritems():
            # create patient mapping entries for hive and sourcesystem
            result = patient_mapping.create( project, str(pat_num), str(pat_num), "HIVE" )
            sql_pat_mappings += result
            # psn with sourcesystem
            result = patient_mapping.create( project, psn, str(pat_num), network )
            sql_pat_mappings += result
            # write patient mappings file
            sql_pat_filename = directory + '/psn_patientMapping/patientMapping.sql'
            # needed to finally insert into table
            postfix = "COMMIT;
"
            sql_pat_mappings += postfix
            # finally, write the sql file
            with open( sql_pat_filename, 'w') as newfp:
                newfp.write(sql_pat_mappings)

        return True
    else:
        return False
else:
    logger.error( "No files in directory at: %s", directory)
    return False

def process_observationFact(matches, directory, dict_psn, logger):
    """Processes and observationFact and creates output which corresponds to the patients_nums in patientDimension"""
    matches.sort()
    number_files = 0
    for match in matches:
        number_files += 1
        with open(match, 'r') as fp:
            # full file name = fp.name
            # split into name and ext
            splitfilename = fp.name.rsplit('.', 1)
            # print splitfileame
            if ( splitfilename is not fp.name ):
                # file name
                name = splitfilename[0]
                # only the filename
                name2 = name.rsplit('/', 1)
# strip trailing .xml. [0] contains desired result
name3 = name2[1].rsplit('.', 1)
print name2[1]

# parse the file and get psn and sql statement
# passing a file object should be faster in this case
values = psd_xml.parse_doc(fp, logger)
if values is not None:
    # print values
    psn_id, sql = values
    # print psn_id
    temp = psn.check_psn(psn_id)

    # the psn seems valid
    if ( temp == True ):
        # lookup psn
        try:
            number = dict_psn[psn_id]
        except KeyError:
            logger.error( "A PSN in the file %s has not been found in
the patient lookup table. Make sure that your patients correspond to your
observationFacts!", fp.name )
            return False

        # result is the original sql with PATIENT_NUM replaced
        result = psn.replace_psn( sql, str(number) )

        if (result != None):
            # file names
            newfilename = directory + '/psn_observationFact/' +
            name3[0] + '.sql'
            # open file for writing and auto close
            with open( newfilename, 'w') as newfp:
                newfp.write( result.encode('utf-8') )
        else:
            logger.error("Replacing identifier with seq-nr failed in
%s", file.name)
            return False
    else:
        logger.error( temp )
        return False

# number of files processed
print "Number of files: %s" %number_files
return True

def process_patientDimension(matches, directory, project, dict_psn, logger):
    """
    Processes and pseudonymizes patientDimension statements.
    """
    matches.sort()
    number_files = 0
    # counter for number of patients
    # assures a real patient count
    # start counting at 1
    counter = 1

    for match in matches:
        # start counting at 1
        number_files += 1
        # with ... as auto closes file pointers
        with open(match, 'r') as fp:
# full file name = fp.name
# split into name and ext
splitfilename = fp.name.rsplit('.', 1)
# print splitfilename
if ( splitfilename is not fp.name ):
    # file name
    name = splitfilename[0]
    # only the filename
    name2 = name.rsplit('/', 1)
    # strip trailing .xml. [0] contains desired result
    name3 = name2[1].rsplit('.', 1)
    print name2[1]

# parse the file and get psn and sql statement
# passing a file object should be faster in this case
values = psd_xml.parse_doc(fp, logger)
if values is not None:
    # print values
    psn_id, sql = values
    temp = psn.check_psn(psn_id)
    # the psn seems valid
    if ( temp == True ):
        # add psn to dictionary if it is not already in it
        if ( psn_id not in dict_psn ):
            dict_psn[psn_id] = counter
            # increment patient counter
            counter += 1
        # result is the original sql with PATIENT_NUM replaced
        # do not simply number all patient but lookup them in the
        # dict
        result = psn.replace_psn( sql, str(dict_psn.get(psn_id)) )
    name3[0] + '.sql'
    # open file for writing and auto close
    with open( newfilename, 'w') as newfp:
        newfp.write(result)
    # print dictionary for debug
    #for item in dict_psn.keys():
    #    print "dict_psn[", item, "] = ", dict_psn[ item ]

def read_sdb(directory, action, user, logger):
    """
    Component for the SDB-mode of the program.
    Processes input files created by i2b2Dataimport.
    """
    # pattern of the output files used by i2b2Dataimport
    patientDimension = directory + '/patientDimension/*.txt'
    observationFact = directory + '/observationFact/*.txt'
    matches = glob.glob( patientDimension )
232 if matches:
233 # process patientDimension
234 temp = process_sdb(matches, directory, action, user, logger, "patientDimension")
235
236 # return an error up the hierarchy
237 if ( temp == False ):
238 return False
239
240 # process observation Fact
241 matches = glob.glob(observationFact)
242 if matches:
243 temp = process_sdb(matches, directory, action, user, logger, "observationFact")
244
245 # return an error up the hierarchy
246 if ( temp == False ):
247 return False
248
249 # if both matches succeed
250 return True
251
252 else:
253 # no files in observationFact
254 logger.error( "No files in observationFact at: %s", observationFact)
255 return False
256
257
258 else:
259 # no files in patientDimension
260 else:
261 logger.error( "No files in patientDimension at: %s", patientDimension)
262 return False
263
264 def process_sdb(matches, directory, action, user, logger, table):
265
266 # Identifies PID within i2b2Dataimport files and prepares them for
267 # the PSD-Service. It finally writes them to an output directory.
268 
269 matches.sort()
270 number_files = 0
271
272 for match in matches:
273 number_files += 1
274 # print match
275 with open(match, 'r') as fp:
276 # read entire file into memory
277 file = fp.read()
278
279 # full file name = fp.name
280 # split into name and ext
281 splitfilename = fp.name.rsplit('.', 1)
282 if ( splitfilename is not fp.name ):
283 # file name
284 name = splitfilename[0]
285 network = name.split('- ', 2)
286 # extract competence network name
287 # print network[1]
288 # file name
289 # only the filename
name2 = name.rsplit('/', 1)
print name2[1]
# file extension
extension = splitfilename[1]
# print extension

if (table == "patientDimension"):
  temp = pid.identify_pid_patientDimension(file)
elif (table == "observationFact"):
  temp = pid.identify_pid_observationFact(file)
else:
  # should not happen
  return False

if (temp):
  # PID
  result = pid.check_pid(temp)
  # PID is correct
  if (result != False):
    # replace PID with unique identifier and write according xml file
    end_result = pid.replace_pid(file, result)
    # create xml and validate
    final_result = psd_xml.create_doc(end_result, result, action, user, logger)

    # file names
    newfilename = directory + '/psd-output/' + name2[1] + '.xml'
    # open file for writing and auto close
    with open( newfilename, 'w') as newfp:
      newfp.write(final_result)

    logger.info( "PID: %s in file %s", result, newfilename)
  # PID is ill-formatted
  else:
    logger.error( "PID is ill-formatted in %s", fp)
    return False

# no PID in file
else:
  logger.error( "PID not found in file %s. Skipping the file!", fp)
  fp.close()

print "Number of files: %s" %number_files
return True

Module: patient_mapping.py

#!/usr/bin/env python
import string
# time functions
from time import gmtime, strftime

# Author: Lars Reimann
# Date: 2010-05-19
# Released under CDDL (http://www.sun.com/cddl/cddl.html)
def create(project, patient_id, patient_number, network):
  
  Creates an insert statement for the i2b2 patient_mappings table
prefix = "Insert into 
project_database = project
schema="(PATIENT_IDE, PATIENT_IDE_SOURCE, PATIENT_NUM, PATIENT_IDE_STATUS, UPDATE_DATE, DOWNLOAD_DATE, IMPORT_DATE, SOURCESYSTEM_CD, UPLOAD_ID)"
values ="values (" + patient_ide + "," + patient_ide_source + "," + patient_num + "," + patient_status + "," + now + "," + "null" + "," + now + "," + "null" + ");"
result = prefix + project_database + table + schema + values + "\n"
return result

Module: pid.py

# regular expressions
import re
# string methods
import string

# ------------------------------------------------------------------
# Author: Lars Reimann
# Date: 2010-05-19
# Released under CDDL (http://www.sun.com/cddl/cddl.html)
# ------------------------------------------------------------------
def identify_pid_patientDimension(datafile):
  """
  Operates on SQL syntax used by i2b2Dataimport. Works as long as the place
  of PATIENT_NUM is not changed (e.g. database scheme). It occurred first in
  the sql values section
  """
  # search for location of PID
  m = re.search( "\([0-9A-Z]{8}"", datafile)
  # if first regex is successful
  if ( m ):
    # if regex does find the first string
    if ( m.group(0) != None ):
      # find the word PID
      n = re.search( "\w+", m.group(0) )
      # return the PID
      return n.group(0)
    else:
      return False
  else:
    return False
def identify_pid_observationFact(datafile):
    """
    Operates on SQL syntax used by i2b2Dataimport. Works as long as the place
    of PATIENT_NUM is not changed (e.g. database scheme). It occurs 2nd in the
    sql values section.
    """
    # search for location of PID
    m = re.search( "\(\'\w*\',\'\w*\"", datafile)
    # if first regex is successful
    if ( m):
        # if regex does find the first string
        if ( m.group(0) != None):
            # find the word PID
            n = re.search( "\'\w{8}"", m.group(0) )
            if ( n):
                # return the PID stripped of colon chars
                return n.group(0).strip("")
            else:
                return False
        else:
            return False
    else:
        return False

def check_pid(pid):
    """
    Checks PID for validity. It is not allowed to contain certain characters.
    And only uppercase chars are allowed.
    """
    if ( pid.find('B') == -1 ):
        if (pid.find('I') == -1 ):
            if (pid.find('O') == -1 ):
                if (pid.find('S') == -1 ):
                    # for ensurance, convert to uppercase
                    copy = pid.upper()
                    return copy
                else:
                    return False
            else:
                return False
        else:
            return False
    else:
        return False

def replace_pid(file, pid):
    """
    Replaces PID with a unique identifier in file. This is because the
    original
    PID is not allowed to occur within the pseudonymized file. Otherwise
    PID and PSN could be cross-referenced.
    """
    uidentifier = "UNIQUE_IDENTIFIER666"
    # replaces all occurences of PID
    # PID should also be unique in that data set.
    return file.replace( pid, uidentifier)
Module: psn.py

```python
#!/usr/bin/env python

# base64 handling
import base64
# strings
import string

# --------------------------------------------------------------------------
# Author: Lars Reimann
# Date: 2010-05-19
# Released under CDDL (http://www.sun.com/cddl/cddl.html)
# --------------------------------------------------------------------------

def check_psn(psn):
    
    Checks if a PSN is valid
    
    try:
        # raises a TypeError is a psn contains is s were incorrectly padded or
        # non-alphabet characters were present in the string
        base64.decodestring(psn)
        # for the moment, a PSN is fixed to a length of 172
        if ( len(psn) == 172 ):
            return True
        else:
            return "PSN is of wrong length. Please check the source file."
    except TypeError:
        return "PSN is ill-formatted. Please check the source file."


def replace_psn(sql, sequential_number):
    
    Replaces a unique identifier with a seq-nr for use in i2b2.
    Every patient gets assiged an integer nuber.
    
    uidentifier = "UNIQUE_IDENTIFIER666"
    return sql.replace( uidentifier, sequential_number )
```

Module: psd_xml.py

```python
#!/usr/bin/env python

# import xml library and functions
from lxml import etree

# --------------------------------------------------------------------------
# Author: Lars Reimann
# Date: 2010-05-19
# Released under CDDL (http://www.sun.com/cddl/cddl.html)
# --------------------------------------------------------------------------

def parse_doc(data, logger):
    
    Parses a PSD-compliant message. As is comes directly from the PSD,
    we assume that its format is correct.
    
    tree = etree.parse(data)
    if ( tree != None ):
        root = tree.getroot()
```
# find the index element containing the PSN
index_element = root.find("./index")

if ( index_element != None ):
    # print index_element.text
    # find the original SQL
    sql_element = root.find("./i2b2sql")
    if ( sql_element != None ):
        # check for unicode here
        return index_element.text, sql_element.text
    else:
        return None
else:
    return None
else:
    return None

def create_doc(data, pid, psd_action, user, logger):
    
    Constructs a PSD-compliant message from the input file.
    It is prepared for expansion for additional message types.
    
    # the PSD namespace
    XHTML_NAMESPACE = "http://www.tmf-ev.de/site/services/psd/psd.xsd"
    XHTML = "{%s}" % XHTML_NAMESPACE
    # NSMAP = {'psd' : XHTML_NAMESPACE} # the default namespace (psd prefix)
    NSMAP = {None : XHTML_NAMESPACE} # the default namespace (psd prefix)

    # the root element
    root = etree.Element(XHTML + "MDAT", nsmap=NSMAP)

    header = etree.SubElement(root, XHTML + "header")
    action = etree.SubElement(header, XHTML + "action")
    actiontype = etree.SubElement(action, XHTML + "actiontype")
    actiontype.text = psd_action
    h_from = etree.SubElement(action, XHTML + "from")
    h_from.text = "SDB"
    h_to = etree.SubElement(action, XHTML + "to")
    h_to.text = "FDB"
    destination = etree.SubElement(action, XHTML + "destination")
    destination.text = "FDB"
    returnpath = etree.SubElement(action, XHTML + "returnpath")
    returnpath.text = "SDB"
    userID = etree.SubElement(action, XHTML + "userID")
    userID.text = user

    index = etree.SubElement(header, XHTML + "index")
    # PID here
    index.text = pid
    ack = etree.SubElement(header, XHTML + "ack")
    ack.text = "true"

    body = etree.SubElement(root, XHTML + "body")
    # body text is not allowed
    sql = etree.SubElement(body, "i2b2sql")
    # Original sql statement is inserted here
    sql.text = unicode(data, 'utf-8')

    serialized = etree.tostring(root, xml_declaration=True, encoding='UTF-8', pretty_print=True)

    # open psd.xsd for validation
    xmlschema_doc = etree.parse("./psd_i2b2/psd.xsd")
xmlschema = etree.XMLSchema(xmlschema_doc)

# validate the above document
if ( xmlschema.validate(root) ):
    # return the checked doc
    return serialized
# otherwise document is not valid
else:
    logger.error( xmlschema.assertValid(root) )
    return False
E.2 PSD-Service XML-Schema (psd.xsd)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:psdat="http://www.tmf-ev.de/site/services/psd/psd.xsd"
    targetNamespace="http://www.tmf-ev.de/site/services/psd/psd.xsd"
    elementFormDefault="qualified" attributeFormDefault="unqualified">

<!-- Unencrypted part containing final action and PID/PSN -->
<xsd:complexType name="T_Action">
    <xsd:sequence>
        <!-- Action, e.g. "Store in FDB" -->
        <xsd:element name="actiontype" type="xsd:string"/>
        <!-- source -->
        <xsd:element name="from" type="xsd:string"/>
        <!-- destination -->
        <xsd:element name="to" type="xsd:string"/>
        <!-- destination service -->
        <xsd:element name="destination" type="xsd:anyURI"/>
        <!-- returnpath service -->
        <xsd:element name="returnpath" type="xsd:anyURI"/>
        <!-- user information -->
        <xsd:element name="userID" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="T_Header">
    <xsd:sequence>
        <!-- Action, e.g. "Store in FDB" -->
        <xsd:element name="action" type="psdat:T_Action"/>
        <!-- index can be PID or PSN -->
        <xsd:element name="index" type="xsd:string"/>
        <!-- boolean retval -->
        <xsd:element name="ack" type="xsd:boolean"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="T_Body">
    <xsd:sequence>
        <xsd:any maxOccurs="unbounded" minOccurs="0" processContents="lax"/>
    </xsd:sequence>
</xsd:complexType>

<!-- Complete MDAT -->
<xsd:complexType name="T_MDAT">
    <xsd:sequence>
        <xsd:element name="header" type="psdat:T_Header"/>
        <xsd:element name="body" type="psdat:T_Body"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="MDAT" type="psdat:T_MDAT"/>
</xsd:schema>
```
F. References


[70] S. Murphy, "Informatics for Integrating Biology and the Bedside (i2b2) - Workbench software release of June 1, 2007," Informatics for Integrating Biology and the Bedside, MGH, Jun. 2007.


[76] "i2b2 Clinical Research Chart (CRC) - Design Document v1.0," Informatics for Integrating Biology and the Bedside, MGH.


