Technische Universität München

Fakultät für Informatik

Bachelor's Thesis in Informatics

Mitigation of privacy issues in ontology-based knowledge bases

Sebastian Vogl

Technische Universität München

Department of Computer Science

Bachelor's Thesis in Informatics

Mitigation of privacy issues in ontology-based knowledge bases

Verringerung von Privatheitsproblemen in ontologie-basierten Wissensbasen

Author	Sebastian Vogl
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Dr. Holger Kinkelin; Dr. Heiko Niedermayer; Marcel von Maltitz, M.Sc.
Date	June 10, 2015

Informatik VIII
Chair for Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, June 10, 2015

Signature

Zusammenfassung

Das Ziel dieser Arbeit ist es, die Privatsphären Problematik einer auf Ontologien basierenden Wissensbasis, im Rahmen eines intelligenten Gebäudeumfeldes, zu analysieren und mindern. Deshalb werden zuerst die Datenschutzanforderungen der Benutzer und der zur Verfügung stehenden Dienste analysiert, vorhandene Konflikte abgeleitet und versucht diese zu lösen, wobei die Privatsphäre der Benutzer des Systems im Vordergrund steht. Ausserdem wird ein Entwurf, einer Privatsphäre respektierenden Umgebung, basierend auf einer bereits bestehen Ontologie, erstellt, um eine praktische Analyse durchführen zu können. Schließlich werden die Ergebnisse zusammengefasst und verifiziert, wie weit Privatsphäre durch den Systementwurf durchgesetzt werden kann, welche der Konflikte gelöst werden konnten und ob die Implementierung die Analyse Ergebnisse umsetzt.

Abstract

The intention of this thesis is to analyse and mitigate privacy problems of a ontology driven knowledge base in the context of a smart building environment. Therefore we will first analyse privacy requirements of the users and provided services, then derive conflicts and try to solve them while preserving the privacy of the system's user. Furthermore we will develop a privacy preserving design based on an already existing ontology-driven knowledge base and implement the design, to analyse the on-hand experience. Finally we conclude and compare on how much privacy we gained by the new system design, which conflicts could be solved and evaluate the on-hand implementation, whether the design conclusions can be implemented correctly.

Contents

1 Introduction 1
1.1 Motivation 1
1.2 Intend 2
1.3 Outline 2

2 Background 3
2.1 Comparison: Knowledge base versus database 3
2.2 Ontology 4
2.2.1 Definition 4
2.2.2 Interaction 5
2.3 Related work 5

3 Analysis 9
3.1 Exemplary system specification 9
3.2 Use cases 12
3.3 User-Privacy 13
3.4 Privacy versus Functionality 14

4 Privacy Approach 17
4.1 Initial situation 17
4.2 Privacy problem mitigation 18
4.2.1 Information granularity 20
4.2.2 Minimum/ Optimum analysis and user preferences 20
4.2.3 Input filtering 21
4.2.4 Output filtering 22
4.2.5 Security Control 23
4.2.6 Logging and Accounting 24

5 Implementation 25
5.1 Architecture overview 25
5.2 Setting 27
5.3 Documentation 28
5.3.1 Variable documentation 28

5.3.2 Program procedure 30
5.4 Example 32

6 Evaluation 35
6.1 Privacy evaluation 35
6.1.1 Privacy questions 35
6.1.2 Bits of information 37
6.2 System design evaluation 40

7 Conclusion and Outlook 43

Bibliography 45

List of Figures

2.1 Visual ontology example. Classes are black, Instances red, Object-properties blue and Data-properties green 6

3.1 Overview of the exemplary system. 10

4.1 Abstract visualisation of the problematic system. 18
4.2 Theoretical mitigation of the problematic abstract system. 19
4.3 Overview of the system’s "Minimum-/ Optimum-Analysis". 21
4.4 Visualisation of the input section. 22
4.5 Visualisation of the output section’s person search service. 23

5.1 Overview of the plug-in design. 26
5.2 Schematic flow graph of the Security Control. 33

6.1 Comparison of the granularity steps. 38
6.2 Comparison of the example sequence. 41

List of Tables

5.1 Privacy preferences of the system's users. 27
5.2 Encoding the user id. 28

Chapter 1

Introduction

The first chapter will clarify, where the point of interest lies, what we will analyse, what we expect to achieve and how this thesis is structured.

1.1 Motivation

The usage of digital connected buildings and services is increasing, whether it are simple devices like domestic home stereo speaker and smart TVs or complex business online-payment and company room authorisation systems. A lot of different data has to be stored, managed and distributed through this systems. The logical consequence is to create a central service that takes this task on. A service that only distributes information to users or other services that are supposed to receive exactly this data, not more and not less. A working system, that respects privacy of the users, is able to block attacks from outside and within itself and is although user-friendly. A lot of services provided as user-friendly and secure, for example the eBay [1] database and Apple's iCloud [2], had a data breach in the last year. [1] Such information leaks are obviously harmful when a lot of personal information, like payment details, are available for other people than oneself to use them without limitations. But not only such direct data leaks can be harmful for a user of software, even information that seem harmless on first glance can leak indirect information about a person. For example, its online behaviour. As a representative of existing analyser software, we take "Ad-ID" [3]. Ad-ID provides a platform for advertising companies to help them track the user's interaction with different medial behaviour. So companies can analyse this data and tailor their advertisement depending on the user's behaviour. That implicates that they are able to receive harmful information from the collection of not so harmful information.
The same principle of different external companies for one user can be found in a smaller version of a smart building inside a single company. There we also have a

[1] http://www.ebay.com
[2] https://www.icloud.com
[3] http://www.ad-id.org/about

service providing information to one or more users, where direct, as well as indirect, information could leak. As we want to achieve a privacy friendly system, we have to address this topic.

1.2 Intend

With this thesis we will evaluate whether it is from the privacy point of view useful or wise to use ontologies as a information distribution system, which problems will be faced and how they are to be solved.
We will focus on how to integrate user privacy aspects in an ontology-based knowledge base. The question is how to implement data privacy goals into the system and in which aspects an ontology will help or hinder the integration. We will evaluate whether there are specific features of an ontology and how to use or avoid them. We will determine which security mechanisms will be needed to reach our predefined privacy goals, where we can implement our defined rules and which data is really needed in our knowledge base. In the end we will design and implement an example for an existing ontology-based knowledge base to verify whether the theoretical approach is practical realizable.

1.3 Outline

Before we start our analysis we will clarify some thematics, that will be needed later in the thesis and we introduce related work, which is similar to the topic of this thesis, in chapter 2. In the beginning of the thesis's chapter 3 we will relocate the general thematic of privacy in ontology driven knowledge bases to a more handy use case of a "smart building" with a location determination system. To work on concrete user privacy aspects we will define data privacy goals and determine conflicts with the function of the designed knowledge base system. In the following chapter 4 we will try to implement the knowledge of chapter 3 in a possible solution or at least a mitigation of the problem. This mitigation will be used as a basis for a plug-in concept design and implementation for an existing ontology driven knowledge base in chapter 5. Finally in chapter 6 we will retrospectively evaluate the previous chapters and add a short outlook for possible future theses.

Chapter 2

Background

In chapter 2 we will discuss the difference between knowledge bases and databases, summarize background knowledge and refer to related work.

2.1 Comparison: Knowledge base versus database

First of all, we have to figure out what a knowledge base is and how it differs from a database. Generally both, a knowledge base and a database, are great tools to gather, store and distribute a lot of different information. In an abstract "black box" view both systems have an input and an output. So they can store provided information and retrieve stored information. The difference lies in the information processing mechanism. For example, Wolfram Alpha by Wolfram research [1] defines their knowledge base as following: "Powering Wolfram|Alpha and the Wolfram Language, the continuously growing Wolfram knowledge base is by far the world's largest and broadest repository of computable knowledge." [2] So a knowledge base is a "[...]repository of computable knowledge[...], further it is knowledge understandable for machines. Contrariwise a database can be seen as a system managing "raw" data and optimising queries on this data. [3] A widely known representative for the use of databases would be Google Inc.[2]. For a better understanding of this topic, we first discuss, what knowledge is. Knowledge consists of information which consists of data. Data is fundamental. It just describes a fact which can be observed and converted into a savable form. Information is the connection and summary of data. Knowledge is the connection of all together. A connection of information and previously obtained information can be concluded to knowledge. [4] [5]
A knowledge base may resemble a database, but is not necessarily one, meaning it uses data to reason about it and gains knowledge from the data and information received. Obviously you can store the raw data itself in a knowledge base and you can create optimised or specific queries for databases, to switch the role of both system, but that is

[1]https://www.wolfram.com
[2]https://www.google.com

not the reason they were designed for.
So where are the privacy issues. As shown before databases and knowledge bases are actually not the same. They both store information, but the usage of information differs. That means, we may have to modify the database optimised security algorithm for knowledge bases in general, as data itself and knowledge from this data is likewise relevant.

2.2 Ontology

In this section we want to specify ontologies in general, the ontology language used and address important background knowledge which will be necessary for this thesis. We will not explain ontologies in detail as this thesis is not about ontologies itself, but the privacy on working with ontologies. For more information about ontologies itself, please refer to the given bibliography.

2.2.1 Definition

There are a lot of different definitions of ontologies in the Artificial-Intelligence literature. [6] We will use the definition by Tom Gruber.
The term "Ontology" origins from the philosophy, concerned with studying existence and the nature of existence. The technical term "Ontology" was first used by Artificial-Intelligence researcher to describe computable knowledge with self-regulated reasoning. An exemplary usage of reasoning on a ontology-based knowledge base can be found in the previously mentioned Wolfram|Alpha. [2]
As for the technical term it describes an artefact, by a set of representational primitives, with the purpose to model a domain of knowledge or discourse. This representative primitives include information about their meaning and constraints on their logically consistent application. They consist in general of classes, attributes and relations between them. In other words an ontology can be viewed as a cluster. It is a higher abstraction level of hierarchical and relationship data models, with the intend to model or summarise knowledge about individuals and their relation to other individuals, as well as their attributes together and for themselves.
To get an more abstract view of the theoretical data structures of ontologies we use ontology languages to model knowledge for individuals with their attributes and relationships. As we use this more "semantic" level of models instead of a logical level, we can differentiate an ontology approach even more from a database language which is more on a physical level. An ontology language, that will be used within this thesis, will be explained in the next subsection. [7]

2.2.2 Interaction

For interaction with our ontology we will use the Web Ontology Language (OWL). OWL is a specification of the World Wide Web Consortium (W3C) to create, publicise and share ontologies. An ontology in OWL consists of "Classes", "Individuals", "Data-properties" and "Object-properties". Classes have names and can inherit from each other in a hierarchical order, where the top class is "Thing" and every other class is a successor of at least this class or more. An Individual has a name and is an instance of one or more classes. We can link Data-properties and Individuals by axioms or link Individuals with each other by Object-properties. Data-properties are typically named in verb form to create a more "readable" context. They connect Objects with Individuals. Objects can be primitive data types or defined data types within XML, RDF and OWL definitions. Object-properties connect two Individuals with each other. They also have names and the same properties in range and name as data properties. [8]
For visualisation we take a similar example like the one used by Noy McGuinness's in "A Guide to Creating Your First Ontology". [6] Please use figure 2.1 for reference. At top level we have the class "Thing" from which "Wine" and "Winery" inherit. An instance of wine is "Château Lafite Rothschild Pauillac" and an instance of winery is "Château Lafite Rothschild". The Object-property of "Château Lafite Rothschild" is "produces" and for "Château Lafite Rothschild Pauillac" "isMaker", to link those two instances. The Data-property of "Château Lafite Rothschild" is "isLocatedIn" with its object "France".
We will now shortly discuss the Manchester Syntax as we will need it for querying the knowledge base in the implementation chapter later of the thesis. The Manchester syntax is another language to define or structure ontologies. It was created to be more "readable" and not contain any logic identifier. For example, we have a building containing rooms and one person called "PersonA". If we want to get the room with the person inside, we use the syntax: "Room THAT contains VALUE PersonA". If we use the previous defined ontology definitions we can assign "Room" as a class, "PersonA" as a instance of the class "Person" and "contains" as a Object-property. "THAT" is a operator which defines that a class description follows for class "Room". Therefore we want an instance of the class "Room" that has the following description. "VALUE" identifies, that we want the instance "PersonA" to be connected with Object-property "contains". [9] [8]

2.3 Related work

Privacy in general is a very widespread problematic with a lot of related work to it. Therefore we will only address a small part of available references.
In "Hippocratic databases" [10] by Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant and Yirong Xu they describe a very similar privacy problem as the one addressed in this thesis. But in stead of a knowledge base, it aims at a database. On a fundamental level the general privacy problem of knowledge bases and databases is

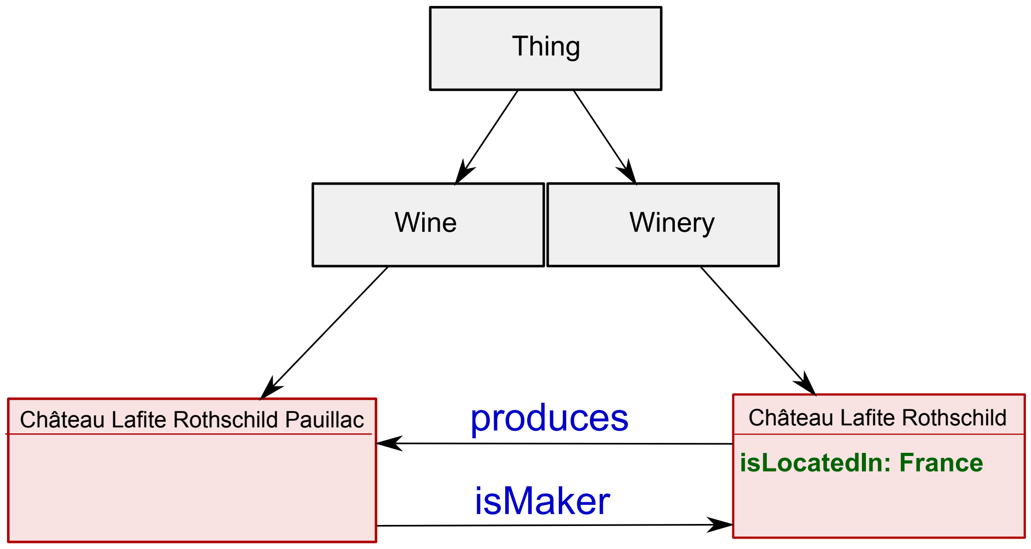

Figure 2.1: Visual ontology example. Classes are black, Instances red, Object-properties blue and Data-properties green

similar. In both cases we demand that the privacy of the user's data, stored inside the database or knowledge base, is respected. But as shown in section 2.1 the data does not have to be the same but the concept to preserve privacy may be similar.
An other concept to protect privacy of a person within a group of persons is "K-Anonymity". It was introduced in 2002 by Latanya Sweeney to be able to create a data model for public release, where specific data is not linkable to one person of a group of different persons. She created a compromise between higher privacy and loss of the data's accuracy. "K" is a parameter in form of a natural number. For k-Anonymity we have sensible information and a identifier for a person. We get k-Anonymity if the identifier of a person can be differentiated from at least k-1 other persons, thus we can create groups of people within the group of all people. A greater k represents higher anonymity within the released data. [11]
Based on "K-Anonymity" is "L-Diversity". As k-Anonymity has flaws, for example you can link a person and sensible data if you got background knowledge of that person, L-Diversity tries to intervene this flaws. For L-Diversity we start with a K-anonymised group of persons and information. We have L-Diversity if an equivalence class has at least L "well-represented" values for a sensible information. When every equivalence class of the table has L-Diversity, our whole group has L-Diversity. [12]
Finally based on "K-Anonymity" and "L-Diversity" is "T-Closeness". As L-Diversity may be difficult to achieve, Ninghui Li, Tiancheng Li and Suresh Venkatasubramanian defined a new model: "An equivalence class is said to have t-Closeness if the distance between the distribution of a sensitive attribute in this class and the distribution of the attribute in the whole table is no more than a threshold t. A table is said to have t-Closeness if all equivalence classes have t-closeness." [13]

At last an interesting concept of privacy-preserving aggregation of time series, which can be handsome in the context of energy aggregation within a building. The problem stated here is, that we have several participants sending data to one data aggregator, for example to create consumption statistics for energy billing. They try to create a system that the aggregator doesn't learn about the data of one participant, but only the data from every one. [14]

Chapter 3

Analysis

In chapter 3 we will specify an exemplary system and the user's privacy needs to summarize the privacy problem of our knowledge base.

3.1 Exemplary system specification

The mitigation of privacy issues in knowledge bases is in general a very abstract thematic. Therefore we will relocate the general problematic on a more handy use case of a "Smart Building" with an ontology-based knowledge base. As it is more demonstrative to work with an actual system, we will define specific input and output services for our knowledge base.
The system can be subdivided in three parts: input-section, internal-section and output-section. Each section is connected by plug-ins. The input-section consists of three input types, which will, for example, help to locate a person's exact position in a building:

- static knowledge, e.g. by a room map of a building
- configured knowledge, e.g. by a calender of room occupation
- virtual live knowledge, e.g. by a location detection module

The input service's data will be managed by the ontology driven knowledge base and from there handed over to the output-section, if requested. The output-section consists of the following three components:

- A public section, which everybody can openly access.
- A semi-private API where other systems can request information.
- An emergency application, which gives us the possibility to override some privacy rules.

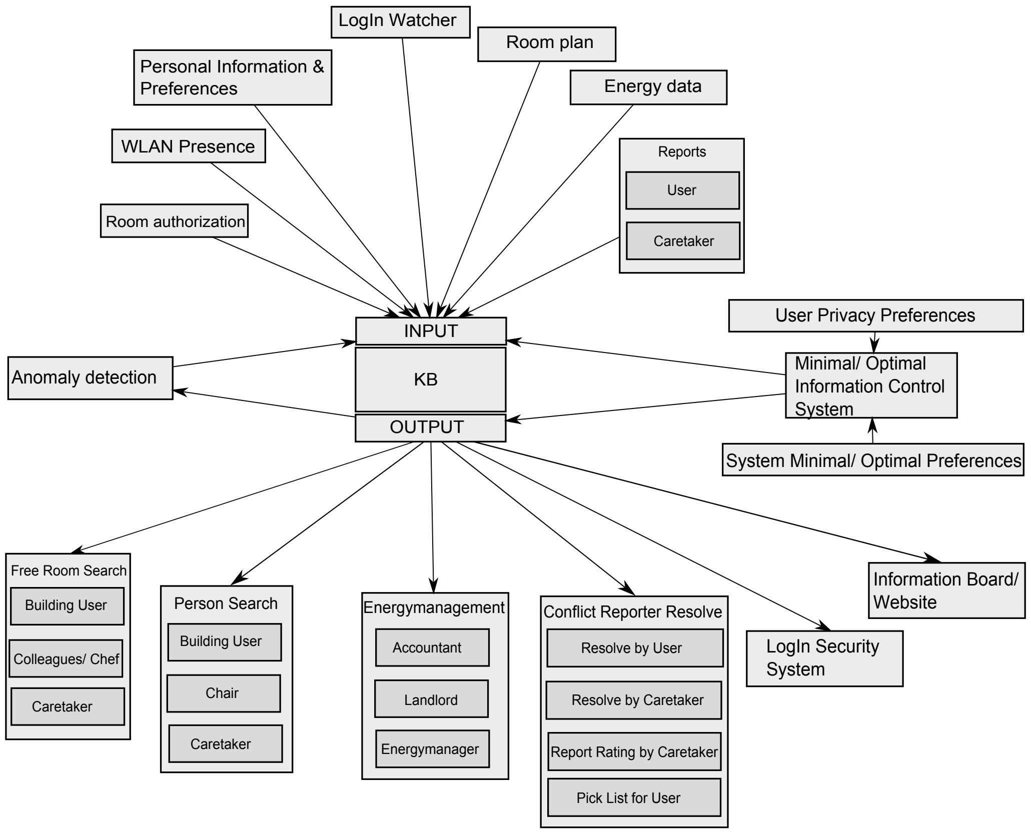

Figure 3.1: Overview of the exemplary system.

As each section is defined beforehand, we are able to design only the plug-ins variable. Therefore privacy rules will only be possible to apply in the input-section's and output-section's plug-ins. Additional we should note here that the only way to insert, respectively request, information to and from the knowledge base will be through these plug-ins. To have some concrete information to reflect on, we introduce the following exemplary system: Figure 3.1 shows an overview of the system. As input we add:

- A room authorizing system, so we know if someone uses a room which need authorization to access.
- A wlan presence detection module for live position tracking.
- A log in watcher, so we know who logs in on which device with her or his user account.
- A room plan, so the system can match position data and rooms.
- A report system, so user can report malfunctions or problems.
- Personal information and preferences on security of the users.
- Energy data, also energy consumption data by the users.

For the output section we define six services, which have access to the knowledge base. Some services need different roles as not every information, a service can request, is to be accessed by the person using the service. For example, a building has private rooms for their manager, so these rooms shouldn't be accessible by normal guests. The six services are:

- A free room search service, where you can search free rooms depending on your access level for restricted rooms. In this case there are special rooms for departments within a building, which can only be accessed by the department's employees and the caretaker, who can access all rooms in the building. We differentiate between "Normal User", "Colleagues/ Chefs" with higher access level and "Caretaker" with the highest access level. A higher access level implies more restricted rooms are accessible.

- A person search service, which allows us to locate a registered person within the building. The search output is restricted to a certain accuracy depending on the preferences of the person searched for. The preferences are divided in two cases "Normal User" and "Colleagues/ Chefs". So that it is possible for colleagues to search their co-worker with an exact position, but an inexact for non colleagues. The caretaker has a exceptional position. He can look up the exact location of every registered person in the building without explicit permission in case of emergency, but only in case of emergency.

- An energy management service accessible by an "Accountant" to link a persons and its energy consumption, a "Landlord" to create the energy bill and an "Energymanager" to control general usage, for example, to turn off lights in an unused room.

- A conflict report resolve system, which displays reports in a pick-list for the system's user or caretaker to solve them. The caretaker can rate reports depending on their trustworthiness and reliability of the reports.

- A log-in security system to monitor unauthorized log-ins. For instance, a person is not inside the building, but logs-in a computer that is inside a room of the building.

- A public information board, to display information for guests of the facility.

In addition to the dedicated input and output plug-ins we add a "Anomaly Detection Plug-in". This plug-in retrieves or updates information from the knowledge base. It is a part of the report system. For example, if the plug-in recognizes an higher energy consumption in the kitchen of the building, even if there should be no person, it can send an report, for instance of an open fridge door. With the example system in hand we can determine problems with a working system in the next section.

3.2 Use cases

The current smart building's knowledge base is unrestricted, so that every output application can request information from the knowledge base without any limitations. This unrestricted access model provides great functionality related to the output-section as long as there are only docile applications. The problem we are facing are non-docile applications that try to access information not dedicated for them or in other words at least try to access data in some way that seems dubious and not needed for the applications function. Furthermore everyone can restrict their data input freely by his own discretion. Information needed by output services may not be available as the input was restricted. Next to this direct problems we may face indirect problems. Even if there is some kind of restriction by definite rules plus input and output are synchronised that there is not too much or less information, a malicious service may be able to get its information by indirect questioning.
Following will be shown some exemplary behaviours of the current system:

Positive behaviour

- Bob allowed exact tracking of his position. Therefore his colleagues can find him faster and the fire department knows his position in case of emergency.
- The login-security-system detects a login for Bob's user account in building A, but Bob seems to be in building B. The system can now warn Bob or deny login for Bob's account.
- The anomaly detection system discovers a sudden higher energy consumption in the kitchen. It sends a report to the nearest Person to look after the problem.
- Bob searches a free room to work with Charlie. Therefore he uses the free room search service, which states "Room 2".
- The energy management system detects switched on light in room 1, but nobody is inside the room. The energy management system turns off the light.

Negative behaviour

- Alice is very safety-conscious and doesn't share any information with the system. As Bob have to hand over an important report to Alice, he uses the person search function. But as there are no information about Alice, he don't know where she is.
- In the building a fire broke out. As the location system doesn't know Alice's position, Alice can't be found by the fire department.

- Bob doesn't care about his information data in the system and allows everybody to access his information. Meaning even an thief can check his home address and salary.
- A thief searches an unoccupied room to steal from. Therefore he uses the free room search service, which states "Laboratory 1".
- Eve's chef checks his employee's location every 10 seconds to determine working and not working people.

To mitigate such problems we have to define proper user privacy aspects and try to find solutions on how to implement them for our use case "Smart Building" as defined in section 3.1. A proper definition of user privacy aspects will be given within the next section 3.3.

3.3 User-Privacy

After we have defined use cases and problems in the previous section, we have to think about statements which have to be handled, when talking about user privacy. As there is no concrete definition of statements for data privacy, we will use definitions from general known security systems. Such a summary of privacy aspects, every system interacting with users should apply, was written by Dr. J. Bizer and Jaap-Henk Hoepman. Additionally Dr. J. Bizer published in the paper "Sieben goldene Regeln des Datenschutzes" [15] seven common basic rules for data privacy. Together with the "Privacy Design Strategies" by Jaap-Henk Hoepman [16] the following points can be followed:

Legality
Data must be legally collected. The user has to know about the data collection.

Agreement and choice
A user has to agree to save his data in the knowledge base and has the choice over how much data of him is stored.

Appropriation and collection limitation
The collected data has to have a purpose and should only be used for this purpose.

Necessity and data minimisation
The collected data must be used, not needed data shouldn't be collected.

Transparency
The person has to know which data is collected and what it is used for. She or he has to be notified if anything changes in the amount of data collection.

Information security
The collected data has to be secured and must not be available to bystanders. By the general definition of data security, it can be differentiated in:

- Confidentiality
  The data stored in the knowledge base should only be given to the intended recipient. Confidentiality of entities is also referred to as anonymity
- Integrity
  The accuracy and consistency of data should be maintained that there is no possibility to modify the data unauthorized.
- Availability
  The data has to be available when it is needed.

Privacy compliance
The security and privacy preserving mechanisms of the system should be available and demonstrated when asked

Accuracy and quality
The stored data in the system has to be up to date, correct and should be periodically checked.

Control and accountability
The data retrieval and data issuing must be logged.

We will summarise these points in questions which should be answerable with our system design in chapter 4 and 5. We will later use them for our system evaluation in chapter 6:

- Who (persons) can see the user's data? Who should see the user's data?
- Which data is collected? Which data is stored and how long?
- Is there a fixed dimension of data collection and storing?
- Are the user able to decide what data about them is stored?
- Who (system components) can access the user's data?
- Can you look up who saw which data? How long is the record stored?
- In which interval is the user's data checked and updated?

3.4 Privacy versus Functionality

We will now take a look at the problems we have to face when we try to apply privacy to a already functional system. Privacy often comes along with security when thinking about the usability of a working system. To achieve privacy we have to cut down important information. To achieve security we have to implement mechanisms to stop attackers from intervening with system procedure which, for example, can slow down

the system's response time. The study "Security and Usability: Analysis and Evaluation" of Kainda, Flechais and Roscoe exactly points out that statement, but as security is not so important for this thesis we will focus on privacy. [17] Starting with our largest conflict, the conflict of user privacy and a working system:
First let us remember one of the negative behaviours in section 3.2: "Alice is very safety-conscious and doesn't share any information with the system. As Bob have to hand over an important report to Alice, he uses the person search function. But as there are no information about Alice, he don't know where she is." The problem stated here may not be that bad and one can find other solutions for the problem, but you can see that the user's privacy preference conflicts with the optimal response of the search service. Now we take a look at a more intense scenario of section 3.2: "In the building a fire broke out. As the location system doesn't know Alice's position, Alice can't be found as fast by the fire department." In this case we see, that there is the same problem that Alice's preference intervenes with our system, but rather than in the previous case, Alice's life can be in danger now. Even if Alice's preference is to let nobody know her exact position we have to "protect her from herself" to say it out blunt. So we should determine which services really need specific information and which information can be left out of the information input flow.
Let's remember another behaviour of our current system: "A thief searches an unoccupied room to steal from. Therefore he uses the free room search service, which states "Laboratory 1"." The current system does not differentiate between employees and non employee, which states the problem in our scenario. Furthermore we can't determine the thief, because we have no protocol of the last search queries. So it could be an employee or an outstanding person. Even if it could differentiate between employees and non employees, there is no difference between employee's departments. So even specific rooms for departments may be accessible to other departments which may creates problems with the departments room plan.
The next problem is how to scale a information. In other words how to scale the granularity of information to create less privacy conflicting information. For example: What is less privacy invasive than "The person is in building A, Room 8 on the left corner near the door."? One may say "The person is somewhere in the universe.", but that's a quite extreme scale as it is even the most inaccurate. So we have to find an acceptable raster.
Finally there are two points which should be noted here. The first one is that data received from the knowledge base may depend on other data or implicates other information directly. For example, the electricity bill for a department is always paid by the same person. We may follow out of this information that there is only one person in that apartment. To mitigate this problem we may receive payment by department summarized and not by person. The other problem that should be stated here is that we may need conflict resolver plug-ins, when two or more input services produce conflicting data. For example, an employee is at two places at the same time. The problem with such

resolver plug-ins is, that they can receive a lot of information from the knowledge base and generate own knowledge out of that data. We have to create an environment for this plug-ins, that we can trust them not to reveal information what are not supposed to be revealed. For example, our location conflict resolver plug-in is based on behavioural analysis. It knows a specific behaviour of a person to solve location conflicts by overriding the conflicting location by the "usual" location. Now a attacker could initiate a conflicting location position to receive the behavioural location without rising any suspicion for the attack detection system.
This two cases will not be directly evaluated as they would go beyond the scope of this thesis.
With all the information above we will now try to mitigate the privacy problems of our system. As seen before it won't be possible to completely solve the problem, because privacy, security and the work flow of a system are conflicting generally.

Chapter 4

Privacy Approach

In this chapter we will evolve a theoretical approach to mitigate the problems stated in chapter 3. But before that there will be a short introduction in an existing ontology-based knowledge base, which will be our basis for the development.

4.1 Initial situation

As shown in figure 4.1 our system has three parts:

- An input section with its corresponding input plug-ins. The input plug-ins are the connection between the knowledge base and the sensor data which is to be stored in the information management system.
- A "Broker" containing the ontologies. These ontologies are the reasoning intelligence, responsible for storing, managing and issuing information. The broker is connected to the outside only by the input and output plug-ins.
- An output section with its output plug-in. The output plug-in is the connection between services receiving data from the knowledge base. It can receive requests by services which are asking for information, query the knowledge base for them and distribute the data correctly to the querying service.

The common information flow is from top to bottom, but there can be a back-flow if information are depending on other information already stored in the knowledge base. As an example for normal behaviour we have location sensors installed in a building to provide a person search service. This sensors identify, for example, Alice leaving room A and entering room B. This information will be send to the input plug-ins, converted to a understandable format for the knowledge base and stored inside the knowledge base afterwards. If her colleague Bob wants to know where Alice is, he can use the person search service's interface which will contact the output plug-in connected to the broker. When Bob searches Alice with the person search service, he sends a request to the output plug-in, which converts the query to an ontology understandable format

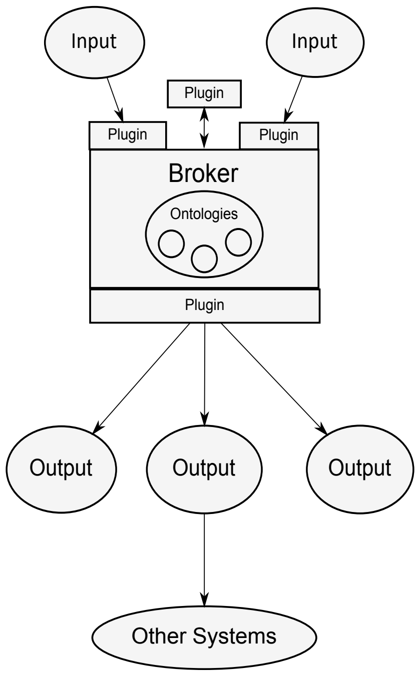

Figure 4.1: Abstract visualisation of the problematic system.

and forwards it to the broker. The broker will reason on the query and return the most exact answer. In this case it would be: "Near the door in room B." That such a system is unwanted from the privacy point of view was shown in chapter 3. To specify the problem in a short sentence: Everybody can freely, without restrictions access the most exact information about persons and systems, stored in the knowledge base. Therefore we will try to mitigate the problem on this example system, with our conclusions from chapter 3, in the next section.

4.2 Privacy problem mitigation

As a basis to mitigate our problems stated in chapter 3 we will use figure 4.1 from the previous section. To visualise what each segment will do, exemplary input and output is introduced: We have an output service to locate a person in the building, which later called "B". This service can access two kinds of input information: A person's location and personal information of people. For example, user Bob asks the system "Where is Alice?". This system will answer "In room 3" as it got this information from the location input beforehand.
In figure 4.2 you can see an overview of the system with security features, which will be discussed in the following:

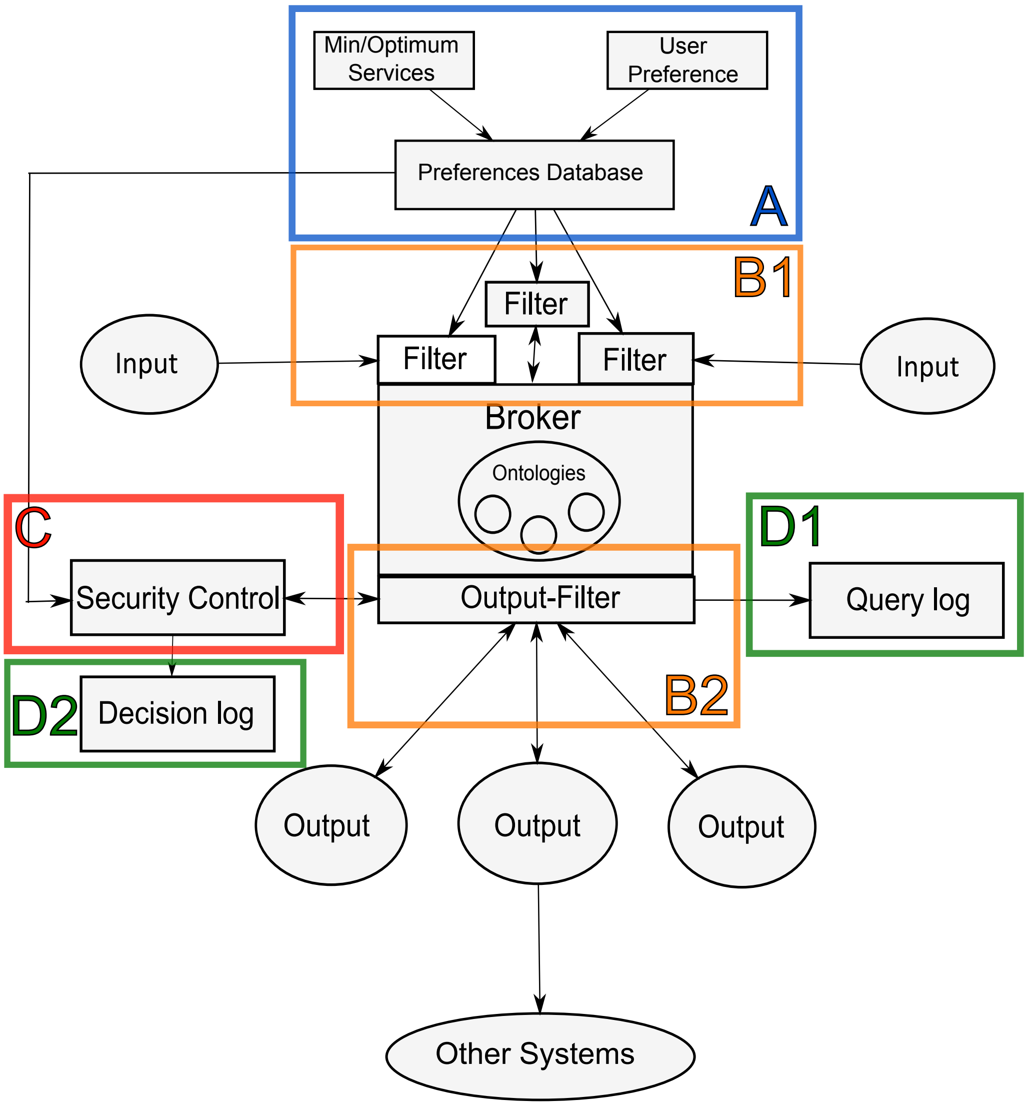

Figure 4.2: Theoretical mitigation of the problematic abstract system.

4.2.1 Information granularity

At first we have to define information granularity. That means we have exact information data and think about what the most suitable and imprecise grade is, that we can assign to a information type. Afterwards we have to subdivide the gap between precise and imprecise and rank the subdivisions according to how privacy invasive this information is. Thus we created our informations granularity steps. This schematic has to be introduced for each input type we have.
In our example we have a person's location and personal information. The person's location is ranked as following: building < floor < corridor < chair < room < exact. Where "Building" is the most inaccurate and "exact" the most accurate answer possible. "Floor", "Corridor" and "Chair" are ranked on how much information we gain about the exact position. For personal information we have: guest/employee status < chair member < id (devices MAC address) < name < residential address < income. In this case "Name" is lower ranked as "Address" as more than one people having the same name in a public accessible building is more likely, than people having the same name and living at the same residential address.

4.2.2 Minimum/ Optimum analysis and user preferences

The first segment is the preference segment (Segment A in figure 4.2). For this segment we first have to analyse the minimum and optimum information granularity of the output services. The services we talk about are depending on information about user of the system and have had a information granularity analysis, which was discussed in the previous subsection 4.2.1. In the minimum/ optimum analysis we check what are the information needed by the service for minimum function requirements and what are the services for optimal functioning. Minimal functioning is stated by a red bar, optimal by a blue one. As we can see in figure 4.3 we have our person's location and personal information, each with its granularity defined before. In addition to that we can see three categories of the person search service B (B0, B1, B2). Each category equals a group of persons, where 0 is the general or across groups service (e.g. employees in a company), 1 is for group internal (e.g. a single department of a company) and 2 is for exceptional positions (e.g. the caretaker or fire department of a company). For this differentiation we add a blue bar to signalise, that this service, respectively group of persons, has special requirements depending on the situation. For a good example of the difference between them, we take a look at the personal information input. We can see here that the minimal information required are the "Guest/Employee Status". So it can be determined whether the person found by any input service can be linked to an existing employee account and its privacy preferences or a generic default account. The optimal information available here is the name of a person, meaning there is no need of the residential address or the income for our current person search service. As for the person's location input, minimal and optimal information retrieval is the same, as

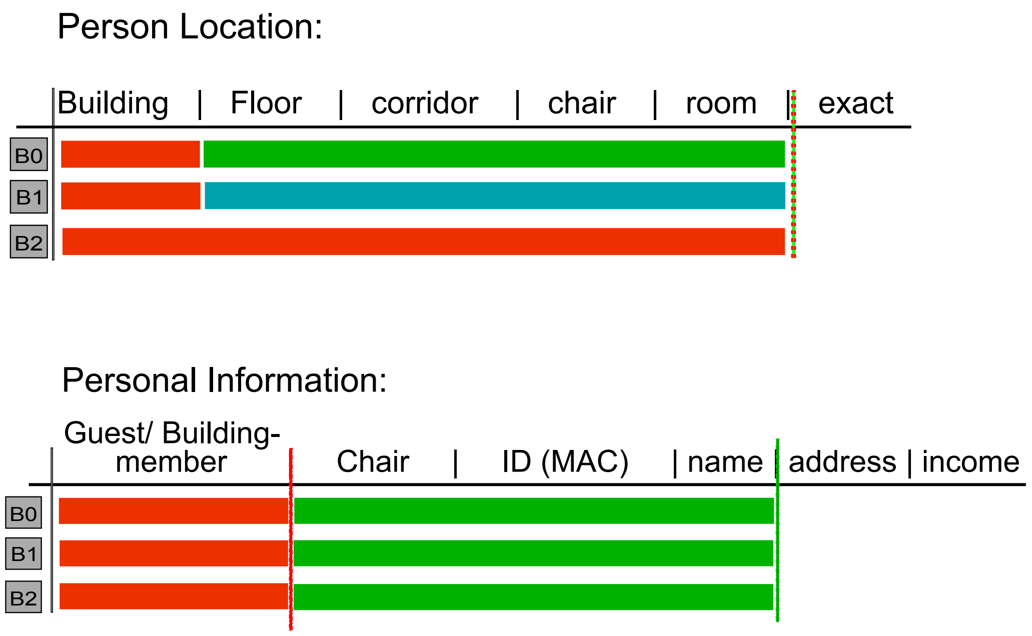
Person Location:
Building | Floor | corridor | chair | room | exact
B0
B1
B2
Personal Information:
Guest/ Building-member
Chair | ID (MAC) | name | address | income
B0
B1
B2

Figure 4.3: Overview of the system's "Minimum-/ Optimum-Analysis".

all information are needed to pin point a person. In this case we defined that an exact position is not needed, because a room positional tracking is enough to find a person. The information retrieved by the minimal/ optimal analysis will restrict the possibilities, users can choose in their preference settings. They won't be allowed to give less information than the minimum or more information than the optimum. For further reference please take a look at the Bachelor thesis "User controlled privacy settings for Smart Environments" of Besjan Saidi, as he was confronted with a very similar problem. [18] Both of this information are stored in a preference database influencing the decision of the input filter-plug-ins in the next section. In case of a change in the service's usage or addition of an other service to the system, we have to reconsider our information granularity and minimum/ optimum analysis for this specific service.

4.2.3 Input filtering

The input filter plug-ins (Segment B1 in figure 4.2) extend the old plug-ins, so they still connect sensors and knowledge base, but filter unnecessary information. They receive minimum/ optimum information and user preferences from the preferences database and use this information to make their decision about data granularity. Please refer to figure 4.4.
We can have three different situations:
First the input is above the optimum of granularity. In this case the input plug-in will set the granularity back to the optimum. For example, our person location tool sends metre accurate positional data. As the preference database and user's preference determined

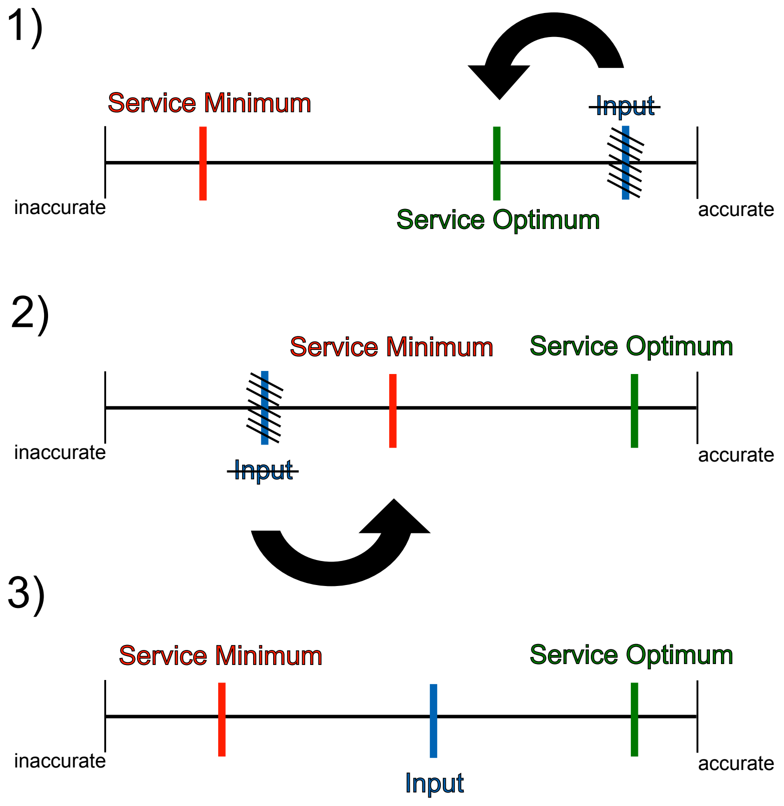

Figure 4.4: Visualisation of the input section.

only room accuracy, the filter will send a room accuracy information to the knowledge base.
Second, we have an input that is below the minimum. This one is more difficult to solve. In case of the user's preference, it will be set to a higher level meeting the minimum. If the sensory data is not sufficient enough the system has to send a warning to the building's administrator for problem resolving.
Third, the input is between minimum and optimum. In this case the system will stay with the inputs granularity and won't change it.

4.2.4 Output filtering

The output filter (Segment B2 in figure 4.2) is similar to the input filter, yet different. Both, filter unwanted information and enforce a higher granularity value if necessary. But, to the contrary of the output section, the input filter can decide on its own and is only depending on the preference database. The output filter only enforces a decision of the "Security Control", that is mentioned in the next segment. The output filter only receives queries from users and forwards it to the knowledge base. It then sends information of the user and query to the Security Control and enforces granularity to the

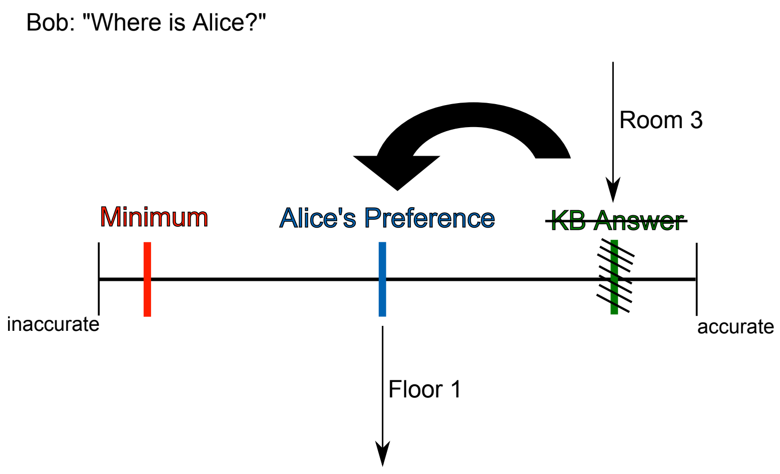

Figure 4.5: Visualisation of the output section's person search service.

answer or completely denies an answer, depending on the Security Controls decision. In figure 4.5 we can see an exemplary sequence of the output filter. The filter receives the query "Where is Alice?" and receives the information "room 3" from the knowledge base. In this case the Security Control decided based on Alice's preference the more inaccurate answer granularity of "floor". So the output filter changes the answer "room 3" to "floor 1" as room 3 is located in floor 1.

4.2.5 Security Control

The Security Control (Segment C in figure 4.2) is the core of the output section. All queries and answers has to be rated by the Security Control, but it is not able to respond to queries and enforce answer granularity. It is only connected to the output plug-ins and the preferences database. The Security Control receives information about the querying user, the query itself and the service which is used, as well as information about the user's and service's preferences from the preference database. To make a decision the Security Control consists of two instances.
The first instance is stateless and the decision only depends on the given information of the preference database. For example, if Alice doesn't allow to be tracked by Bob, his query "Where is Alice?" will fail, because of the first instance's decision. If Alice would have allowed it, a second instance has to decide.
The second instance is stateful, meaning it depends on former queries of the user. A very simple example is, if Bob asks the knowledge base ever couple seconds "Where is Alice?" the answer would be denied as we don't want people monitoring others. There a more than one way to design this stateful control system. For example, one could use another ontology system, storing information which were given to the user before. So we can evaluate any differences in the information received from the actual knowledge

base and the smaller user based "knowledge ontology". But as of the complexity and time-consumption to implement such a decision concept, we take an implementation of a token based concept. In our token based concept an user will have an amount of tokens that can be spend. Each query will cost a specific amount of tokens. If the user reaches zero on her or his token account, the query will be rejected. The token account restocks over time and the user can query our knowledge base again. The amount of tokens to be spend is increased by the privacy invasiveness of the query itself and the diversity of the previous queries. If the queried person is a colleague or the querying user is a chef or the caretaker of the building, the amount to spend will decrease again. As this calculation is depending on the concrete service, an detailed explanation and example of the calculation will be given in chapter 5.

4.2.6 Logging and Accounting

The last two segments (Segments C1 & C2 in figure 4.2) are the logging function of the system. This section will track all decisions of the Security Control and the queries, respectively answers, given back to the user, whereas every user has his own account. This will help to identify peoples malicious behaviour. Therefore we store a time stamp, the querying user, the query itself, the answer given by the knowledge base and the decision reason. For the output plug-in we will have two separated parts containing "no access" queries and "access" queries. The Security Control log stores the reason of its decision additionally. Out of security reasons the system itself is only allowed to append information into the log, there will be no function to alter or delete previously added information from the log. The only way to edit its content will be by the administrator of the system.

Chapter 5

Implementation

In this section an actual implementation of the theoretical approach from chapter 4 will be shown and each section will be addressed with commentary. In the end of this chapter there will be a short example of the working system.

5.1 Architecture overview

In this section the conceptual design of the components and its structural arrangement will be shown. For a graphical overview of the plug-in and name reference please refer to figure 5.1. There will be seen three different colours. Thereby green stands for actual Java-classes, black are external files and orange is inside the ontology-driven knowledge base.
The design structure will be shown exemplary for one service and is based on the approach shown in figure 4.2. The service used is the "person search function". It receives information of the three sources "Device LogIn", "Wlan Presence" and "Personal Information". In general every input and output gets its own plug-in as shown in figure 5.1. Every input plug-in actually receives two types of data. The first type is the raw data send to the plug-in (e.g. by sensors), the second type is the preferences setting. The preference settings will be pushed to the plug-in by the preferences database, which itself is fed with the user preferences and the minimum/ optimum setting of section **??**. With this two information the input plug-in decides whether an information is allowed to proceed in the knowledge base or not. In this implementation we want to lie the focus more on the output section and an overall working system design, so the input will already be filtered correctly.
As there may be conflicts by the information itself, the "Location Conflict Resolver" will fix it, strictly based on a predefined policy. In this case "Device LogIn" and "Wlan Presence" could conflict if the two sources provide different location information. Therefore "Device LogIn" has a higher priority than "Wlan Presence". "Personal Information" has the same priority as "Device LogIn" as there should be no conflicts.
The output part of the system will be an "Separated-Decision-Enforcement-Point" design,

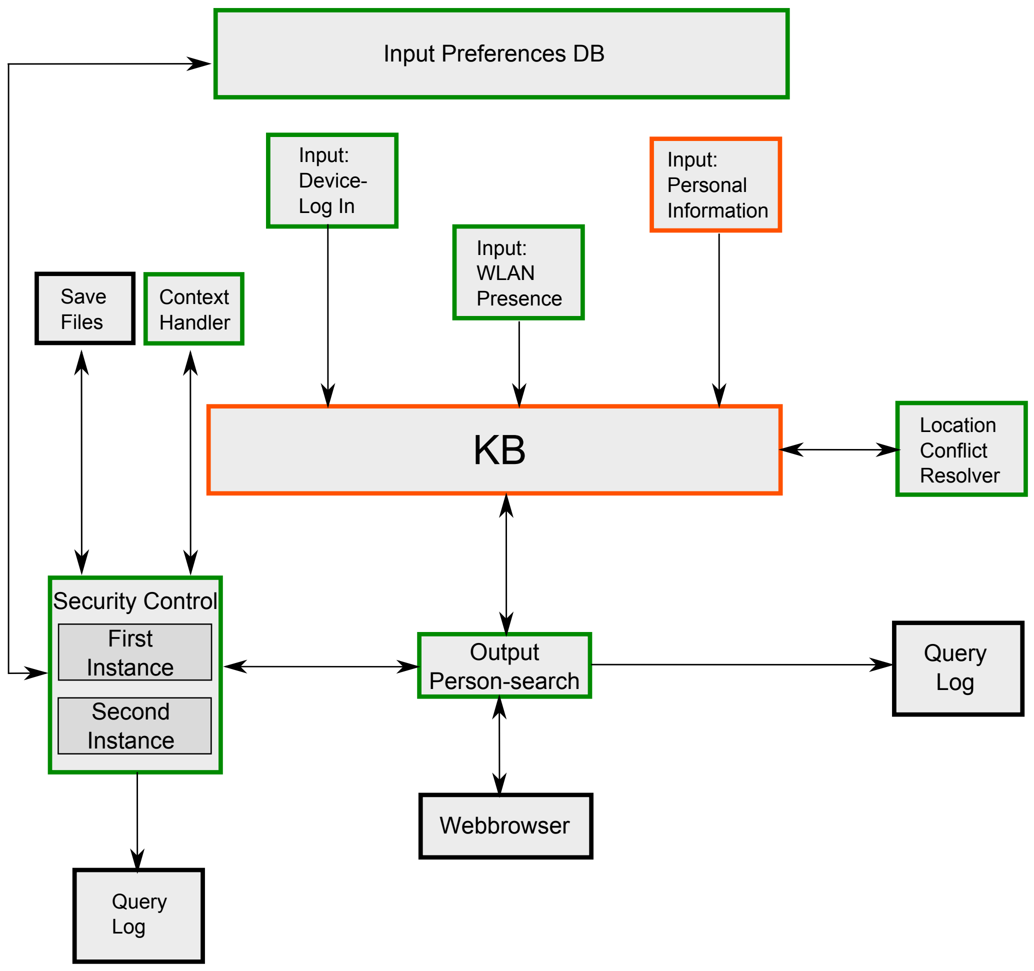

Figure 5.1: Overview of the plug-in design.

meaning the decision process and the enforcement process will be strictly separated from each other. The output plug-in itself can't decide on whether a search query is allowed or not. It just receives the query and passes the decision process to the "Security Control". The output plug-in sends a triple containing "service id" (in our case: person search), user id (the user who requested the data) and the query (in this case the user id of the person searched for) itself. Afterwards the privacy control sends a number back which sets the granularity of the information given to the user. The Security Control itself will be detailed discussed in section 5.3.
The last part of the conceptual design is the logging section. The log files are external text files containing one line for each query. The system will only add information and not alter or delete any previous log entries.

5.2 Setting

For a better understanding of the functions in the program code we will now define an exemplary scenario of the system's usage.
We will have five people stored in our system. That are Alice, Bob, Charlie, Eve and Dave. Alice and Bob are colleagues. Alice is Bob's chef and they work in "Department A", also called "Department 1". Charlie works in Department B ("Department 2") inside the same building, floor and corridor. Eve is the caretaker of the building and Dave is a guest of the facility. We define four user groups: public, employee, colleagues and private. Our scale of privacy invasive location information from less to more privacy invasive will be set as following:
no access (-1) < building (0) < floor (1) < corridor (2) < chair (3) < exact/room (4)
The number in brackets is for the system to generate granularity based on the exact location of a person given by the knowledge base. The granularity creation is introduced later with the "Security Control". Each user's preferences for a specific group of persons will be shown in table 5.1. As Dave is a guest, he didn't choose any preferences by himself, so default "guest preferences" were used. Here we have the problem, that we are actually not able to track guests correctly, because they may don't even have a traceable device with them. Thus we only know, that they are somewhere in the building.

	public	**employees**	**colleagues**	**private**
Alice	-1	-1	0	4
Bob	0	1	4	4
Charlie	0	2	4	4
Dave	-1	0	0	0
Eve	4	4	4	4

Table 5.1: Privacy preferences of the system's users.

The user id for each person inside the system will be encoded to a number as shown in table 5.2. For example: Alice is no guest of the facility so the last digit is 0. She works in "Department 1", therefore the second digit from the right is 1. The digit in front is a ongoing count. Alice received number 1. Consequential Alice's id is "110". We should note here, that this kind of id encoding is very privacy invasive as the id relinquishes a lot of information itself. But for this demonstration it helps us to distinguish people from each other. As for the numerical id itself, it was chosen to simplify the coding process, because actual numbers are better to administer inside the back-end of our system.

	ongoing number	**department**	**is guest**		**user id**
Alice	1	1	0	=>	110
Bob	2	1	0	=>	210
Charlie	1	2	0	=>	120
Dave	1	0	1	=>	101
Eve	1	0	0	=>	100

Table 5.2: Encoding the user id.

5.3 Documentation

We will now discuss the program's variables, functions and general procedures. For better visual clearance we use brackets ([...]) for variables. Methods are *emphasized.*

5.3.1 Variable documentation

Before we take a look at the program procedure we define the variables used in the program. This section is to be seen as a reference for the program documentation in the following sections. First we have to define variables, which are not directly depending on the query itself:

[idUser]
: This value is the user id, provided to every person. It is necessary to query the knowledge base and depends on the schema defined in section 5.2.

[idRequest]
: This value is a user id, provided to every person and depending on the schema defined in section 5.2. This is the user id of the person, who is searched.

[serviceID]
: This value defines the id of the Service that is used. For example, the person search service has id number 8. The ids can be freely assigned as long as they are unique to each service.

[timeToRecover]
: This value determines in seconds how long it takes to recover spend tokens.

[maxValue]
: This value determines the maximum token count for each account.

[recoverValue]
: This value determines how many tokens are recovered for each *timeToRecover* step.

[granularityAdd]
: This value determines how much effect the granularity value will have. (The effect of this value will be shown later.)

[diversityAdd]
: This value determines how many points should be added, if the user tracked different persons. (The effect of this value will be shown later.)

[maxGranularity]
: This value sets the maximum of the granularity value. Meaning the digit assigned to the most privacy invasive answer. In our case this are the scale values assigned in section 5.2.

[saveFile]
: This value determines the name of the querying user's save file.

[currentTimestamp]
: This value saves the current timestamp for decision. The timestamp is encoded in Unix Epoch.

[userGranularityPreference]
: This value is the user's preferred granularity value according to the preference database.

The following variables are depending on the user and are external stored in each user's save file named in [saveFile]-variable. It will contain following values:

[value]
: This value is the current "token account" for the querying user.

[timestamp]
: This value is the timestamp of the last allowed knowledge base access of the user.

[queryCounter]
: This value is the counter for all queries the user made since the save file initialisation.

[userRequestCounter] (for every user of the system)
This value is the counter of queries for one specific user.

5.3.2 Program procedure

We will now discuss the program's procedure of a user's person search query.
After a coldstart of the system, we first have to initialize the user's save files with [value], [timestamp], [queryCounter] and [userRequestCounter] for each user. Afterwards every user has to log in its account. As a user of the person search function, we can now send a search request with the web browser, containing our own id and the id of the person we are searching for.

Output plug-in—The plug-in "Output Person Search" receives the tupel containing [userID] and [requestID]. Where [userID] is the id of the querying user and [requestID] is the id of the queried user. Whether the user gains access and in which granularity the access is granted, is determined by the "Security Control". Therefore the *getAccess* method of the Security Control is called, as this is the only accessible method from outside.

Security Control—As the Security Control is more complex, please refer to figure 5.2. We will now run-through the mentioned sequence diagram step by step.
The *getAccess* method first checks if the searching person is the caretaker and whether there is an emergency or not. If both are true, there is instant access. For example, in case there is a fire in the building the fire department can use the caretaker's account to gain access to the exact people's location. Then it is checked if the person wants to use the self-search function, for example if somebody got lost and doesn't know where she or he is. If true this will also gain instant "exact" access. If not the user's request counter is raised by one and the *firstInstance* is called.
The *firstInstance* is stateless and queries the preference database based on the [userID] and [requestID], whether the requested person wants to grant access to her or his location or not. If not, we log the decision and grant no access. If yes the *secondInstance* is called.
The *secondInstance* is stateful, meaning its decision depends on previous queries. Therefore a system similar to a token system was implemented. Depending on the previous questioning and diversity of questioning you have to pay tokens from a token account. If the user's token account reaches zero the user won't be able to receive answers from the knowledge base. The token account will slowly refill over the time. Therefore we first have to define the following values:

- [timeToRecover]
- [maxValue]
- [recoverValue]
- [granularityAdd]
- [diversityAdd]
- [maxGranularity]
- [saveFile]
- [currentTimestamp]
- [userGranularityPreference]

Afterwards we load the following values from the user's save file, defined in [saveFile]:

- [value]
- [timestamp]
- [queryCounter]
- [userRequestCounter] (for each user)

Then we reset the token account ([value]) of the querying user depending on the [timeToRecover] value. For each [timeToRecover] step since the last query we add an amount of tokens to the user's token account, depending on [recoverValue], but not more than [maxValue].
Then the actual decision is done. If the token account has more than zero tokens access is granted if not the access is forbidden. When the access is forbidden, the [userRequestCount] is increased, all previously loaded values are saved again, no access is granted and the decision is logged.
The log contains the following information:

- The current time.
- The value of [idUser].
- The value of [idRequest].
- The value of [idService].
- The value of the decisions granularity.
- The decision itself.
- An explanation for the decision.

If the token account has enough tokens, access is granted and we have to calculate the new [value]. Therefore we initialise a new variable called [valueSub] which determines the count of tokens we subtract from the token account. First we add the value [granularityAdd] which is depending on the granularity of the answer given to the user. As we want more privacy invasive queries to be more "token-expensive", we divide [userGranularityPreference] through [maxGranularity], multiply the result with [granularityAdd] and round to get an integer. This is our new [valueSub]. Afterwards we subtract one token each from [valueSub] if the querying person is a colleague or the chef of a user as they should query them more often than other persons. The same goes for the caretaker, who has to act with a lot of different people. Thereafter we increase [valueSub], to block people from checking locations of all people in the building, as employees shouldn't work that often with employees of other departments. This addition is depending on the [diversityAdd], [userDiversity] and [maxUser] in the same schematic as the addition for granularity. Where [maxUser] is the current count of registered users in the system. After that calculation we check whether [valueSub] is greater zero to not accidentally add tokens to the account as well as prevent a zero addition. If [valueSub] is greater than zero we subtract it from the current token account. The account is not set to zero, as we don't want user from counting their tokens. For example, a user has only one token left and makes a very invasive query as they know the will be set to zero. Afterwards the old timestamp is overwritten with the new one and the [userRequestCount] is increased. Then all values are saved, the decision is set to "access" and the granularity, based on the user's preference, is returned. Additionally the decision is logged in the same schematic as in the "no access" part before. After that we jump back to the output-plug-in.

Output plug-in—Back at the output-plug-in we receive the granularity value from the "Security Control". In the case that the value is "-1", we refuse access to the data for the user. Should the value be greater or even than "0", we query the knowledge base with "Room THAT contains VALUE PersonX", where "PersonX" is the name of the queried person. For more information to the OWL-language please refer to subsection 2.3. In our case "Room" is the most exact location. After we received the answer, we check the given granularity and apply a new granularity to our exact answer, if necessary. In this case the output plug-in knows the room structure itself, but we also could query the knowledge base for the room's location. To link the correct dependencies of room, corridor and floor. The logging of the answer given to the user will be similar to the logging in the "Security Control", except we don't have an decision explanation and we store accepted and not accepted access in two different files. Afterwards we can return the adjusted answer to the user and wait for the next query.

5.4 Example

We will now take a look at an example query procedure. Our querying user is Alice. We initialise [maxValue] with "12", [diversityAdd] with "4" and granularityAdd with "4". The preferences and personal relationship is as described in section 5.2.
The schema is as following:

Alice ⟶ "Person searched for" ("Granularity preference") : "Plugin's Answer"

Followed by the calculation of the new [value] and [diversityAdd].
Alice queries starting with Bob, then herself, Charlie, Bob, Eve, Bob and in the end herself again. For the calculation process please also refer to figure 5.2.

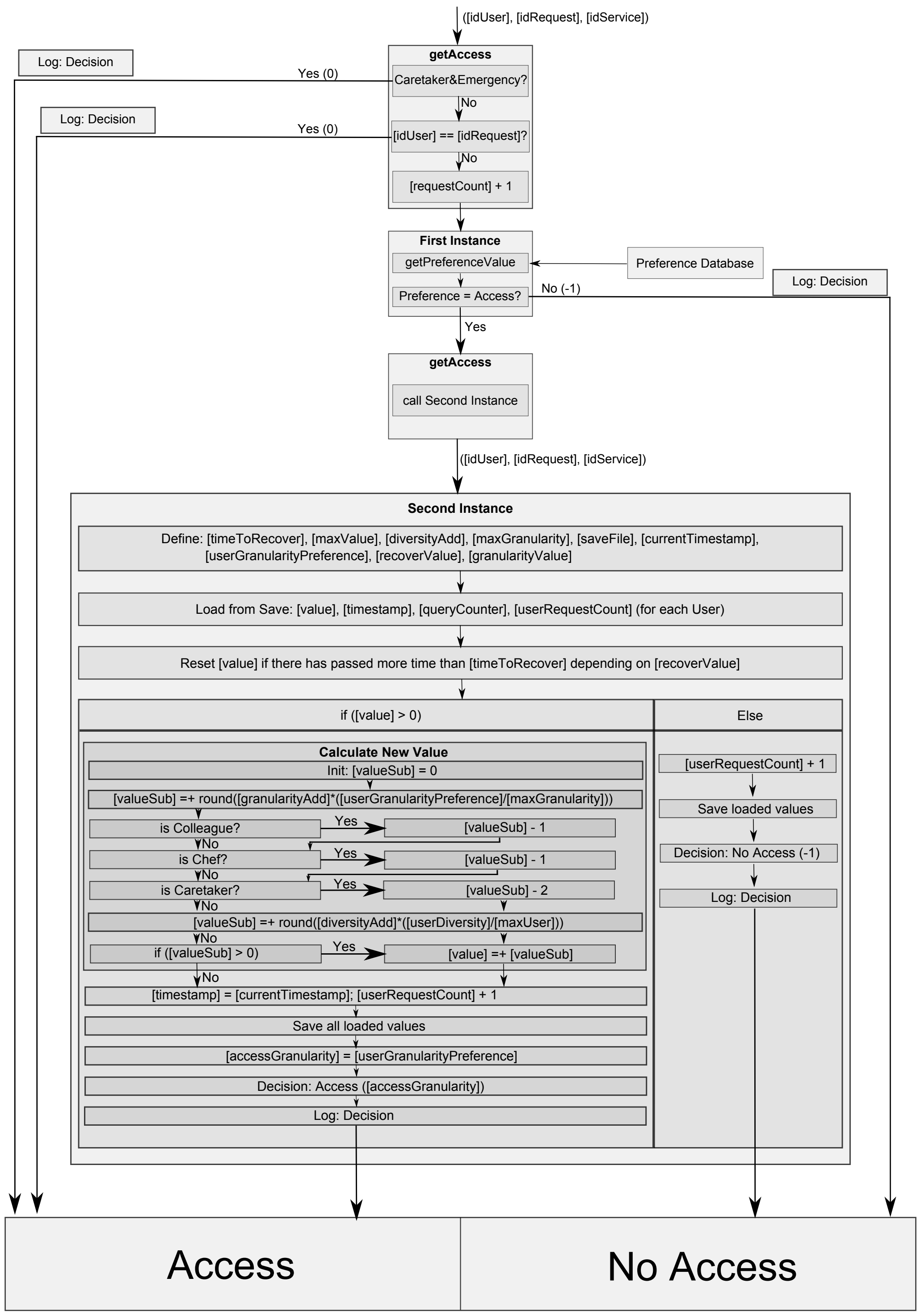
([idUser], [idRequest], [idService])
getAccess
Caretaker&Emergency?
Yes (0)
Log: Decision
No
[idUser] == [idRequest]?
Yes (0)
Log: Decision
No
[requestCount] + 1
First Instance
getPreferenceValue
Preference Database
Preference = Access?
No (-1)
Log: Decision
Yes
getAccess
call Second Instance
([idUser], [idRequest], [idService])
Second Instance
Define: [timeToRecover], [maxValue], [diversityAdd], [maxGranularity], [saveFile], [currentTimestamp], [userGranularityPreference], [recoverValue], [granularityValue]
Load from Save: [value], [timestamp], [queryCounter], [userRequestCount] (for each User)
Reset [value] if there has passed more time than [timeToRecover] depending on [recoverValue]
if ([value] > 0)
Else
Calculate New Value
Init: [valueSub] = 0
[valueSub] =+ round([granularityAdd]*([userGranularityPreference]/[maxGranularity]))
is Colleague?
Yes
[valueSub] - 1
No
is Chef?
Yes
[valueSub] - 1
No
is Caretaker?
Yes
[valueSub] - 2
No
[valueSub] =+ round([diversityAdd]*([userDiversity]/[maxUser]))
No
if ([valueSub] > 0)
Yes
[value] =+ [valueSub]
No
[timestamp] = [currentTimestamp]; [userRequestCount] + 1
Save all loaded values
[accessGranularity] = [userGranularityPreference]
Decision: Access ([accessGranularity])
Log: Decision
[userRequestCount] + 1
Save loaded values
Decision: No Access (-1)
Log: Decision
Access
No Access

Figure 5.2: Schematic flow graph of the Security Control.

Alice ⟶ Bob (4) : "Room 1"
$value = 12 - (4 * \frac{4}{4} - 1 - 1) = 10$
$diversityCount = 1$

Alice ⟶ Alice (4) : "Room 2"
$value = 10$
$diversityCount = 1$

Alice ⟶ Charlie (2) : "Corridor 1"
$value = 10 - ((4 * \frac{2}{4}) + (4 * \frac{1}{5})) \approx 7$
$diversityCount = 2$

Alice ⟶ Bob (4) : "Room 1"
$value = 10 - ((4 * \frac{4}{4}) - 1 - 1 + (4 * \frac{2}{5})) \approx 3$
$diversityCount = 2$

Alice ⟶ Eve (4) : "Room 3"
$value = 10 - ((4 * \frac{4}{4}) + (4 * \frac{2}{5})) \approx -3$
$diversityCount = 3$

Alice ⟶ Charlie (2) : "No Access"
$value = -3$
$diversityCount = 3$

Alice ⟶ Alice (4) : "Room 2"
$value = -3$
$diversityCount = 3$

Chapter 6

Evaluation

In this chapter we will evaluate our system design and implementation. We also take an outlook on what can and should be further researched and wasn't dealt with in this thesis. In the end is a conclusion based on the evaluation.

6.1 Privacy evaluation

At first we will evaluate the privacy aspect of our system design and implementation based on the previously in section 3.3 listed privacy questions. After that we take a look at another possible, more mathematical, evaluation of information retrieved from our system.

6.1.1 Privacy questions

In section 3.3 we defined questions that should be answerable with our system design. The case of the system's logging feature is a bit special, therefore we skip this part of the system in this section and outline it in section 6.2. We will now discuss the answers for these questions with our system design:

- Who (persons) can see the user's data? Who should see the user's data?
  "Who should see the users data?" can easily be answered with "The people the user wants to see their data". This problem is ministered by our "User Preference Database" where we store exactly the decision of the system's users and answer depending on them. The only downstroke, where we limit the user's decision, is the problem of the service's minimum information requirements. But we can't solve this problem without breaking the system's functionality.

- Which data is collected? Which data is stored and how long?
  The data collected depends on the data needed by the underlying "Output Services". Through our "Minimum-/ Optimum-Analysis" we can say that we only store needed information and skip too much or unnecessary data. For the question which data is stored, we have to split in two cases. First, data that is stored

permanently in a file ("save-data"), and second data that is only available while the system is running ("live-data"). As for the save-data, we store only information about user's queries and their interaction with the system (e.g. the overall query count). But that information can be overwritten optionally, by a new initialisation of the user's save-files. For the live-data, we store information as long they are up-to-date. It should be added here, that we have no influence on what the services record before they hand them over to the input plug-ins and after they receive data from the output plug-ins. For example, we may have a database that is feeding our knowledge base with personal information data. So we can't verify what is stored inside that database and if it goes along with our privacy settings.

- Is there a fixed dimension of data collection and storing?
  The dimension of data storage is determined by the information granularity. As described in section 4.2.3 we only store data which meets the minimum requirement of the underlying services or the level of information allowed by the user. Actually we delete information, that are higher than our optimum analysis of the services, even if the user allowed us to store more data than we need.

- Are the user able to decide what data about them is stored?
  The users can decide in the input section which data about them is stored on a previously defined scale. But they are restricted by the "Minimnum-/ Optimum-Analysis" as mentioned before.

- Who (system components) can access the user's data?
  That depends on the implementation. On the input section, only the plug-ins can access the data they are responsible for. In our implementation, every output plug-in in the output section can access basically all information inside the knowledge base. As our implementation has separate points of decision and enforcement, it's actually the function of the output plug-ins to restrict the access for information to a service. In our implementation we first ask the "Security Control" for permission and then query the knowledge base for the answer. So the plug-in only has access to the information if the "Security Control" granted access.

- Can you look up who saw which data? How long is the record stored?
  As we created log-files for our system design which store every action with the knowledge base. We can not only see who saw which data, but as well: "Was the user's access restricted or the query response altered? And why?" and "How many times did he search for a specific user, how many times in general?". This logs should only be accessible by the system's administrator and are saved until they are deleted or manually overwritten.

- In which interval is the user's data checked and updated?
  The user's data store interval depends on the input itself. In general they are updated if there is a change or the input-plug-in sets a fixed interval. For example,

the data is updated when a user leaves room A and enters room B. The user can check his own data at any time, without consequences for his account statistics.

6.1.2 Bits of information

In his article "A Primer on Information Theory and Privacy", Peter Eckersley introduced a way to link the ability of identifying a person within others to the mathematical entropy, measured in bits. [19] The general entropy is the expected value of information we can receive from an event, sample or character, thus characterizes the uncertainty of our information. [20] We will use the principle for our information retrieval measurement. Our formula to calculate the reduction of entropy measured in bits is as following:

$$\Delta S = -\log_2(Pr(X = x)) \tag{6.1}$$

Pr(X=x) is the probability that a fact is true. In our location case it is the probability that a person is in a specific room. When we learn a new fact, for example, the floor of the person's location, this fact reduces the entropy of the person's location. [19]

6.1.2.1 Granularity

We will now evaluate our granularity steps with a similar method as the one previously introduced. We will use the reduction of entropy (ΔS) as comparison. So we can determine on how much one step would reduce the entropy of the person's location (S). At this point should be said, that the entropy of each step is strongly depending on the structure of a building and has to be done for each building separately. We assume that the likelihood of a person in a specific room is evenly distributed. This likelihood depends, for example, on the building's structure, room usage and amount of rooms of each granularity step. The information we receive are only momentary and we assume that in the measured time, there is no change of the people in a room. In our case we used a granularity schema of a building of "no access < building < floor < corridor < exact/room". Where "no access" obviously has an entropy approaching zero as we don't even know if the person is inside the building or not. Our measurement reference unit is "Room" as it is the smallest unit. Our exemplary building is structured even, meaning every corridor and floor has the same number of rooms. The building has three floors and each floor has 15 corridors, containing 25 rooms each. That makes a total of 1125 rooms inside the building. The total of bits needed to clearly identify 1 room out of the 1125 is "S":

$$S = -\log_2(\frac{1}{1125}) \approx 10.1357 \tag{6.2}$$

So we need approximately 10.1357 bits of information to clearly identify one specific room. Following the information retrieval bits for our other granularity steps:

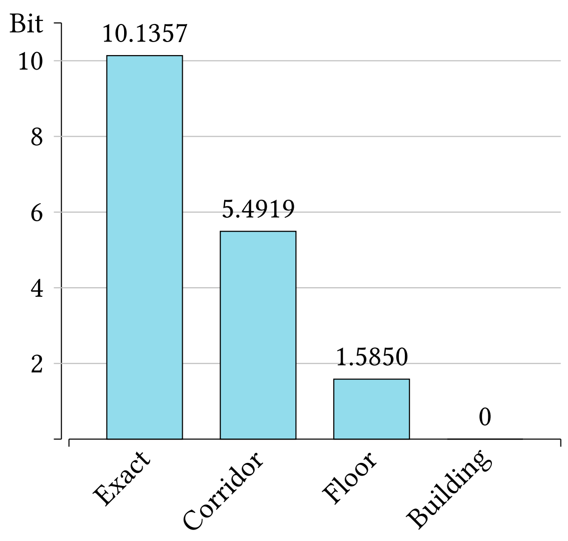

Figure 6.1: Comparison of the granularity steps.

$$S_{Corridor} = -\log_2(\frac{1*25}{1125}) \approx 5.4919 \tag{6.3}$$

$$S_{Floor} = -\log_2(\frac{1*25*15}{1125}) \approx 1.5850 \tag{6.4}$$

$$S_{Building} = -\log_2(\frac{1*25*15*3}{1125}) = 0 \tag{6.5}$$

As shown in figure 6.1, the values of information for corridor and floor, measured in bits, are less than the amount for room. Building has value zero as it contains all rooms, such we gain no information out of the response "Building" as far as we assume that the person is already inside this building. In this calculation we take for given, that the employee's would never leave the building. To make the calculation more realistic we have to add other rooms outside of the building (e.g. the employees homes). With this calculation we can show, that our granularity steps greatly contribute to the privacy of the users. Even the step from room to corridor cut the bits of information gained from the answer in half. The step from room to floor even let us receive only about one-tenth of the original information. Since we have great gaps between the granularity steps we should consider to refine our granularity steps and create intermediates.
It should be added here, that these entropy values will stay the same as long as the proportion of the granularity steps stays the same, even when the count of rooms, corridors and floors change.

6.1.2.2 Comparison to the previous system

After we calculated the Bits of information value for each granularity step we can now try to evaluate our security enhanced systems average disclosure of information, compared to the former system, without security features. Therefore we simulate exemplary usage of the person search service. Special requests like an emergency inside the building and a self-search are excluded. The sequence will be as followed ("querier" ⟶ "queried"): Alice ⟶ Bob; Alice ⟶ Charlie; Alice ⟶ Bob; Alice ⟶ Eve; Alice ⟶ Bob; "One token-recovery-step"; Alice ⟶ Dave; Alice ⟶ Bob; Alice ⟶ Charlie. Our [recoverValue] = [maxGranularity] = 4.

Previous System without security

- Alice ⟶ Bob : Access (Room)
- Alice ⟶ Charlie : Access (Room)
- Alice ⟶ Bob : Access (Room)
- Alice ⟶ Eve : Access (Room)
- Alice ⟶ Bob : Access (Room)
- "Token recovery"
- Alice ⟶ Dave : Access (Room)
- Alice ⟶ Bob : Access (Room)
- Alice ⟶ Charlie : Access (Room)

System with security features

- Alice ⟶ Bob : Access (Room)
- Alice ⟶ Charlie : Access (Corridor)
- Alice ⟶ Bob : Access (Room)
- Alice ⟶ Eve : Access (Room)
- Alice ⟶ Bob : No Access (-)
- "Token recovery"
- Alice ⟶ Charlie : Access (Corridor)
- Alice ⟶ Dave : No Access (-)
- Alice ⟶ Bob : No Access (-)

For the following calculation we have to consider that the query answer we retrieve are momentary, thus we don't gain any new information in comparison to the previous query. To be able to better compare the systems with and without security features, we calculate the average of the bits of information we retrieve from each answer given back to us. We summarize the bits of information gained for each query, depending on the previously computed values for "Room-", "Corridor-", "Floor-" and "Building-" granularity and divide it through the count of queries we have. We take the assumption, that a person doesn't leave its room while our measurement. Therefore we gain the following average bits of information value for the previous system without security features ($S_{previous}$):

$$S_{previous} = \frac{8 * 10.1357}{8} = 10.1357 \tag{6.6}$$

And an average bits of information value for the current system with security features ($S_{current}$) of:

$$S_{current} = \frac{10.1357 + 5.4919 + 10.1357 + 10.1357 + 0 + 5.4919 + 0 + 0}{8} = 5.1739 \tag{6.7}$$

As shown in figure 6.2 our average bits of information received from the new system (red line), in comparison to the old one (black line), dropped from nearly 10 to about 5 bit. Thus, for this example, we only reveal half of the information with the current system than we did with the previous. Obviously the information retrieval depends on the preferences of the user and the user's interaction with the system. As indicated by step 3 to 4 (see blue line), our system would not contribute to the privacy of the users, when we just used this two steps as measurement. Although this is only an example, we have shown that in this and similar cases our current system design greatly contributed to the protection of the user's privacy. For a better analysis may a field study over a longer time is necessary.

6.2 System design evaluation

We will now take a look at our general system design, starting with the Security Control. The heart of the privacy prevention in the system's design is the Security Control. It's the decision-making body. The separation of point of decision and execution has positive as well as negative aspects. On the one hand, we have an independent position that can be outsourced into a network and can be easily adjusted without intervening with other system components, meaning we only have to adjust one component and all other parts can stay the way they are. But this centralised design is at the same time its problem. Every output plug-in of the system has to wait for the decision of our Security Control and if it's occupied the plug-in has to wait, till it receives a free slot. In addition to that if the Security Control gets compromised, the whole system is compromised. So there is a small line between the advantages and disadvantages of this structure.
This small line also counts for the token system inside the Security Control. The advantage of this token system is its customisability. Depending on the field of application one has to customise the amount of tokens available for the user to spend, the amount of tokens spend as well as the amount of tokens regained for specific actions and communication with the system and functions calculating the new values. On the other side, the problem with an token based system is, that it won't be able to really think

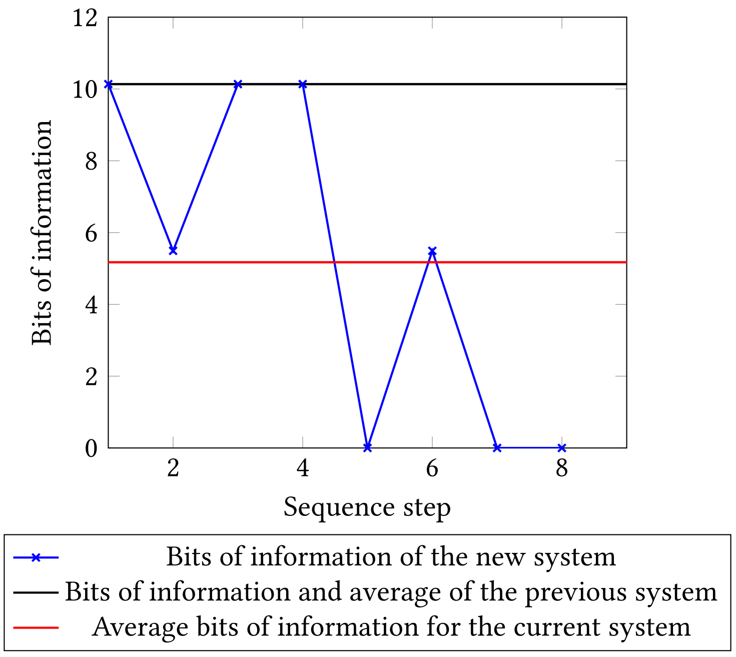

Figure 6.2: Comparison of the example sequence.

about knowledge gained from the answers given to the user. It just helps to identify and block direct attacks or miss-usage of our knowledge base, like the case of iterative querying.

We should also discuss the filtering plug-ins. On the output-side of the system we have the same advantages and disadvantages as all centralised and decentralised system structures. We decided for a decentralised design as we have the advantage, that we already have a centralised decision point so we can split and perfectly customize our plug-ins for the specific output service. The same goes for the input plug-ins. They are customizable as well and as the decision for what information are to be stored in the knowledge base is already determined in the preference database. The preference database itself has the advantage that we analysed our minimum and optimum of the output services, so there is a guarantee that our output services have enough information to work correctly. Furthermore we can guarantee the user of the system, that the privacy preferences, within the minimum and optimum analysis, are fully respected.

Last but not least we take a look at the accounting and logging section. Logging is a great tool to discover flaws and bugs of the software after and while the system is running and to trace malicious behaviour of the user. While the ability to trace behaviour is good for a detective's work, it also greatly invades the privacy of our user. In this case we have to make sure, that the log files can only be accessed by "trusted people". For example, the police or a elected person from the building. The definition of "trusted people" is very imprecise and generally has to be discussed elsewhere or in a following

thesis. It should also be discussed how many times and how long the log files should be available.
Finally it should be noted that Java is not adapted to handle database similar structures. For this problem we may have to add a real database for the *ContextHandler* and *Preference Database* classes to manage the user's save data permanently. For example, MySQL by Orcale, as there is already a connector implemented. [21]

Chapter 7

Conclusion and Outlook

In the beginning of this thesis in chapter 1.1 we spoke of a perfect system, "a working system, that respects privacy of the users, is able to block attacks from outside and within itself and is although user-friendly". We aren't able to design a perfect system, but at leased mitigate the mentioned problems and we have shown that it is possible to create a privacy preserving ontology-based knowledge base. We found out that there is no problem with the ontologies security itself and applying privacy preserving mechanism to it. Actually the current system even did not differentiate a lot from a database implementation in a black box view. As of the time limitation of a bachelor thesis we weren't able to fully use the knowledge of the ontology to our advantage. The current "Security Control" is very limited in its function and only blocks penetration or simple strategic attacks (e.g. iterative querying). Additionally it is strictly based on the user's privacy preferences, what in general is an advantage as we respect the user's wishes. But we are unable to block more strategic attacks with previously gained knowledge and experience. For example, we can track person A exact and person B only in the granularity of "corridor". After a longer period of time (e.g. a couple of weeks) we gained the fact, by information intersection, that A and B are always in the same corridor. From this fact we could infer B's exact position, assuming A and B are always together.
A possible solution to this exact problem could be to add a small ontology for each user to the Security Control. So the small ontology can reason about the previously retrieved answers and what information can be gained out of them. It then can restrict access to future queries when they seem to reveal to much information. As this system alone may not be able to detect the iteration attacks a combination of the token security and this ontology-based security is recommended, thus creating a third instance "Ontology Decision". One more possibility to work with this small ontologies is to compare the small one with the actual ontology within our knowledge base. So we can get the difference of them and find out which information is still not revealed to a user. Thus we may implement a threshold that must not be undercut. An ontology itself give the system designer a lot of possibilities with its ability to reason about itself and gaining

knowledge from the inserted information.
Another option that is available to the current system design is to outsource the Security Control into a network. Obviously we have to consider network security if we take on this task. We have to make sure, that the connection between the Security Control and the output plug-ins is not compromised, altered or blocked and we have to implement a rule for the plug-ins, if the Security Control is not available. For example, to block all communication with the knowledge base.

Bibliography

[1] Verizon Enterprise Solution, “2015 DATA BREACH INVESTIGATIONS REPORT,” Verizon, Report, 2015.

[2] Wolfram Research. (2015) Wolfram Alpha. https://www.wolframalpha.com/input/?i=Wolfram+Knowledgebase™. [Online]. Available: http://www.wolframalpha.com/faqs9.html

[3] A. Kemper and A. Eickler, *Datenbanksysteme: eine Einführung*. Oldenbourg, 2006. [Online]. Available: http://books.google.de/books?id=YezXpIacjkgC

[4] G. Sailer, *Wissensdatenbanken bei IT-Projekten: Projektdokumentation mit Wissensmanagement 2.X*. Diplomica Verlag, 2013. [Online]. Available: https://books.google.de/books?id=TcXriUelVk0C

[5] H. Willke, *Einführung in das systemische Wissensmanagement*, ser. Carl-Auer compact. Auer, 2007. [Online]. Available: https://books.google.ca/books?id=ACM8cAAACAAJ

[6] Noy, Natalya F and McGuinness, Deborah L and others, “Ontology development 101: A guide to creating your first ontology,” 2001.

[7] T. Gruber, “Ontology,” *Encyclopedia of database systems*, pp. 1963–1965, 2009, available online: http://tomgruber.org/writing/ontology-definition-2007.htm.

[8] Sebastian Bendeich, “Data Models and Conict Resolution in Smart Building Environments,” Master’s thesis, Technical University of Munich, October 2014.

[9] Horridge, Matthew and Drummond, Nick and Goodwin, John and Rector, Alan L and Stevens, Robert and Wang, Hai, “The Manchester OWL Syntax,” in *OWLed*, vol. 216, 2006.

[10] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, “Hippocratic Databases,” in *Proceedings of the 28th International Conference on Very Large Data Bases*, ser. VLDB ’02. VLDB Endowment, 2002, pp. 143–154. [Online]. Available: http://dl.acm.org/citation.cfm?id=1287369.1287383

[11] L. SWEENEY, “k-ANONYMITY: A MODEL FOR PROTECTING PRIVACY,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, pp. 557–570, 2002. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0218488502001648

[12] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, “L-diversity: Privacy Beyond K-anonymity,” *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 1, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1217299.1217302

[13] N. Li, T. Li, and S. Venkatasubramanian, “t-Closeness: Privacy Beyond k-Anonymity and l-Diversity,” in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, April 2007, pp. 106–115.

[14] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song, “Privacy-Preserving Aggregation of Time-Series Data.” in *NDSS*, vol. 2, 2011, p. 3.

[15] D. J. Bizer, “Sieben Goldene Regeln des Datenschutzes,” *Datenschutz und Datensicherheit - DuD*, vol. 31, no. 5, pp. 350–356, May 2007, Print-ISSN 1614-0702.

[16] Jaap-Henk Hoepman, “Privacy Design Strategies,” *ICT Systems Security and Privacy Protection*, vol. 428, pp. 446–459, June 2014, Print ISBN 978-3-642-55414-8.

[17] R. Kainda, I. Flechais, and A. Roscoe, “Security and Usability: Analysis and Evaluation,” in *Availability, Reliability, and Security, 2010. ARES’10 International Conference on*. IEEE, 2010, pp. 275–282.

[18] B. Saidi, “User controlled privacy settings for Smart Environments,” June 2015.

[19] P. ECKERSLEY. (2010, January) A Primer on Information Theory and Privacy. EFF. [Online]. Available: https://www.eff.org/deeplinks/2010/01/primer-information-theory-and-privacy

[20] C. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal, The*, vol. 27, no. 3, pp. 379–423, July 1948.

[21] Oracle. (2015) Mysql database connector/j. [Online]. Available: http://dev.mysql.com/downloads/connector/j/