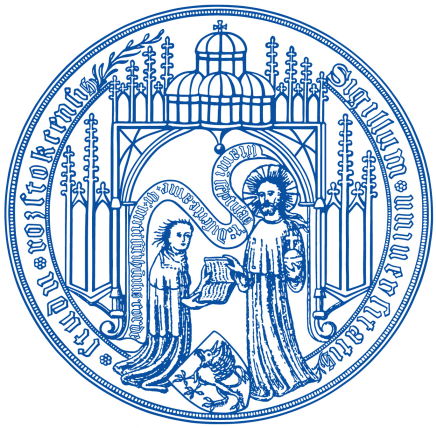
Technical Report CS-01-15

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik
Lehrstuhl für Datenbank- und Informationssysteme



Generating Privacy Constraints for Assistive Environments

Hannes Grunert, Andreas Heuer

Database Research Group
University of Rostock
18051 Rostock
(hg|ah)(at)informatik.uni-rostock.de

Long version of a four-page conference paper for PETRA 2015,
published as ACM 978-1-4503-3452-5/15/07,
DOI: http://dx.doi.org/10.1145/2769493.2769542

20. May 2015

Abstract: Smart environments produce large amounts of data by a plurality of sensors, which constantly track our activities and desires. To support our daily life, assistive environments process these data to calculate our intentions and future actions. In many cases, more information than required are generated and processed by the assistive system. Thereby, the system can learn more about the user than intended. By this, the users' right to informational self-determination is injured, because they lose control how their data is used.

In this paper, we present a model to let the user formulate requirements to protect his privacy in smart environments. These requirements are transformed into multiple integrity constraints, which ensure privacy.

1 Introduction

Assistive systems are designed to support the user at work (Ambient Assisted Working) and at home (Ambient Assisted Living). Sensors collect information about the current situation and actions of the users. These data are stored by the system and linked to other data, for example social network profiles. Based on the obtained information, preferences, patterns of behavior and future events can be calculated. Furthermore, intentions and future actions of the users are derived, so that the smart environment can react independently to satisfy their needs.

Assistive systems [Wei91] often collect much more information than needed. In addition, the user usually has no or only a very small effect on the storage and processing of his personal data. As a result, his right to informational self-determination is violated. In extending an assistive system by a data protection component, which checks the privacy claims of the user against the required information of the system, this problem can be resolved.

Figure 1: Use case scenario: The Smart Appliance Lab (SmartLab) at the University of Rostock. Various, partially invisible, sensors are tracking the users to support team meetings and lectures.

Two main principles of data protection are data minimization and data avoidance. Section 3a of the German Federal Data Protection Act [Bun10] defines data avoidance as the requirement to collect, process and use as little personal information as possible. This includes the design of information systems as well as the data processing itself. By means of a data-avoiding sharing of sensor and context information towards the analysis tools of the assistive system, not only the privacy-friendliness of the system is improved. By reducing the data by selection, aggregation and compression at the sensor itself, the efficiency of the system can be increased. The privacy claims and the required information of the analysis tools can be implemented as integrity constraints in the database system that stores the data collected by the sensors. Due to the integrity constraints, the necessary algorithms for preprocessing and anonymization can be run directly on the database.

The privacy module examines the users' requirements and compares them with the required information of the system. Instructions for selection, compression and aggregation are generated so that only the required data will be stored and processed in the system. A query can be decomposed into multiple subqueries which can achieve sub-goals already at the sensor level. Thus, a transfer of all data to a superior computing unit, which executes the actual analysis algorithms (see Figure 2), can be dispensed.

Our research is motivated by the following questions:

1. Can privacy techniques be directly executed in database systems?

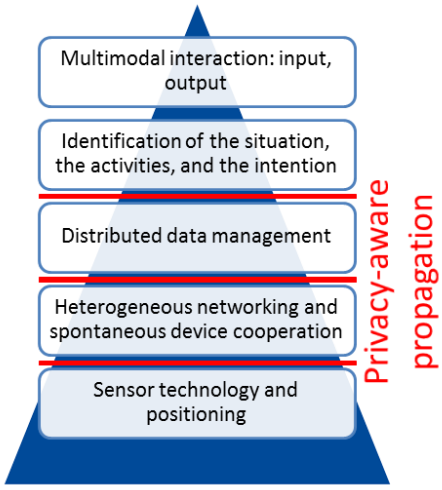


Figure 2: Data processing in assistive systems can be divided into five steps. Information is passed from the bottom (data collection) to the top (data analysis). The red lines indicate where the level of information detail should be reduced.

2. Is it possible to execute them together with analysis functionalities without losing too much precision?

3. Can the execution and response time of the (assistive) system be reduced by processing less data?

The aim of our work is to develop a privacy-friendly query processor that implements the aspects of data minimization and data avoidance. The processor is integrated within the PArADISE [1] framework. The evaluation of our framework is based on the sensor and context information collected at the Smart Appliance Lab of the graduate program MuSAMA [2].

In this article we present the language used to formulate the data protection requirements. In addition, we give an overview of the concept of the query processor. The rest of the paper is organized as follows: In the second section we present the architecture of our privacy framework. Our Privacy Policy for Smart Environments (PP4SE) and mechanisms for its automatic generation is outlined in section three. Section four gives a brief overview of the related work. Finally, we draw our conclusions and an outlook on future work.

[1] Privacy-aware assistive distributed information system environment
[2] Multimodal Smart Appliance ensembles for Mobile Application

2 PArADISE

PArADISE is a tool to support developers of assistive systems and –later on– users of assistive systems by performing queries and analyzing large amounts of sensor data. The system is privacy-aware by pooling existing data protection concepts and algorithms as well as developing and implementing new ideas into a database framework. For example, we developed an algorithm to ensure k-anonymity over multiple queries [Gru14a] and an algorithm to efficiently detect quasi-identifier attributes in high dimensional databases [GH14].

We introduced the basic idea of the framework briefly in [Gru14b]. In this paper, we present some of the individual components of the system in detail.

2.1 Background

PArADISE implements a privacy-aware query-processor-engine, which is used in dynamic ad-hoc sensor-networks. The basic architecture of the framework is illustrated in Figure 3. The user can set up privacy policies which are compared with the queries of the assistive system. The information flow is checked before and after the data is queried.

Important aspects are the time and space complexity, since the sensors may not have enough processing power or main memory to modify the data in soft or even hard real time. Accordingly, it must be decided at runtime, a) which algorithms are applied and b) whether preprocessing on a particular node can be computed at all. In general, the compression of the data should take place as close as possible to the sensor; in the best case where the data is generated.

Non-anonymized data should only be transferred if this is explicitly required or the modification on the affected node is not possible (aspect of data minimization). In order to decide if the anonymization can be computed on a specific sensor node, the processor consults hardware properties like

- CPU-performance and -usage,
- available and maximum main memory and
- available and maximum hard disc memory.

These properties are checked and evaluated against the time and space complexity of the used algorithm. By this, a selection of suitable algorithm and load balancing can occur at runtime.

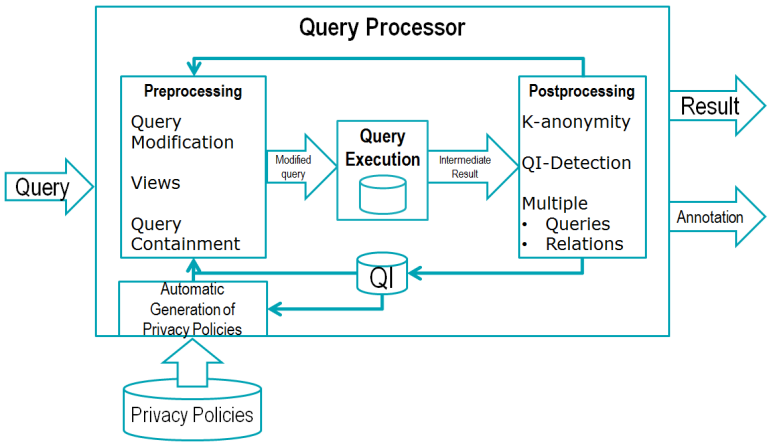


Figure 3: The query processor

2.2 Architecture

The basic architecture of our developed privacy-aware query-processor is shown in Figure 3. The processor works in three stages and can be seen as a black box[3].

The system can send queries towards the data storage. During the query processing, the query and the result will be modified under integrity constraints to achieve privacy while maintaining a certain, predefined degree of veracity. The modified results are masqueraded as normal results.

2.2.1 Preprocessing

During the preprocessing stage, the preliminary query is analyzed and checked against the privacy policy of the affected user. The involved personal information queried by the system is monitored, whether it is uncovered by the user at all (projection) and if it can be used under user defined constraints. These constraints can be used to decide if the revealed information will be preselected or aggregated. Furthermore, it is also checked if the processing node has enough capacity (sufficient CPU-power, free main memory). Also, it is tested if the information system could gain enough information to produce satisfactory results. To determine how satisfactory the result is, we are using the information loss metric based on the Kullback Leibler divergence [KL51], which has been shown to be a good approximation to determine how much information remain [HS10].

Finally, the constraints are used for a modification of the query which fulfills all needs. For modification, we analyze the different parts of the (SQL-)queries. Attributes in the *SELECT* clause are removed, if the user does not want to reveal specific information. If one sensor releases too much information, another sensor is queried by changing the

[3]The system can submit queries and receives back result sets as usual. The process between is hidden.

relation in the *FROM* clause. In case that not all tuples shall appear in the result, the WHERE condition is combined with the user's integrity constraints and the system query conjunctively.

2.2.2 Intermediate Step

After preprocessing, the modified query is executed on a concrete database system[4] containing the personal data. The result of the query is passed to the postprocessor. In ad hoc networks, each node responsible for the data processing can be replaced quickly. It is not always ensured that the underlying storage format remains the same. For example, an old sensor node has its data stored in a pico database management system. During the evolution of the smart environment, it is replaced by a new sensor which stores its data in a semi-structured XML file. By changing the file system, the corresponding query must be replaced in accordance with an equivalent construct in another query language.

A precondition consists of the knowledge about how the information in both formats can be mapped into each other. For this, techniques from the area of information integration can be applied. To handle the complexity of this problem, we are using plain old java objects (POJOs) as a meta model for the mapping of the queries and the results. This meta model is also used in the middleware [BK12] of the Smart Appliance Lab.

2.2.3 Postprocessing

Taken the preliminary result from the intermediate stage, the postprocessor checks the required information of the system and the privacy settings of the user. At this time, the result is modified with privacy-preserving algorithm like k-anonymity [Sam01] or data slicing [LLZM12], if and only if the processing unit has enough power. Thus, the modified result is sent back to the system. In case that a unit does not have enough power, the raw data will be sent to a more powerful node and anonymized later.

3 Policies

In order to collect, to process or to use personal data, the system has to specify (according to [Bun10]):

1. the purpose and duration of the contract,
2. the extent, nature and purpose of the proposed survey,
3. the processing and use of data,
4. the type of data,

[4] Our experiments are executed on MySQL, Postgres, DB2 and MonetDB databases

5. the range of stakeholders and
6. the origin of the data (provenance)

One aspect of the PArADISE framework intends to support the user of assistive systems to maintain his privacy. Therefore, the privacy-claims of the user have to be defined in a policy. In this chapter, we introduce the two main types of privacy constraints and their representation as a privacy policy for smart environments (PP4SE).

3.1 Sensitive Data

Privacy can be divided into two categories: anonymity and hiding of specific information.

3.1.1 Stay anonymous

For the protection of personal data, there exist concepts such as k-anonymity [Sam01], l-diversity [MKGV07] and t-closeness [LLV07]. These concepts divide the attributes of a relation into keys, quasi-identifiers, sensitive data and insensitive data. The aim is that the sensitive data cannot be clearly assigned to a particular person. Tuples can be uniquely determined by key attributes, so they should not be published together with sensitive attributes under any circumstances.

PArADISE implements concepts for relational databases to anonymize data as soon as possible. Similar to antivirus programs, the framework integrates new models as soon as possible to ensure privacy when new variants of attacks arise. Ubiquitous environments often use resource-limited devices with low CPU frequency or main memory. To cope this problem, we try to optimize the privacy-preserving algorithm with regard to such devices.

One example for this is our approach to find quasi-identifiers in high dimensional datasets. The term quasi-identifier was introduced by Dalenius [Dal86] and describes ”a subset of attributes which can uniquely identify most tuples in a table”. Quasi-identifiers are used to define which sets of attributes allow the re-identification of persons or activities, even if key attributes are removed. They have *similar* properties as keys: the identification of a tuple in a relation using these attribute combinations is still possible, but not 100% sure. QIs can be part of a key, but also combinations of sensitive and insensitive data.

Due to the high dimensionality and the large amount of data generated in an assistive system, it is not easy for an inexperienced user to recognize which attributes compromise his privacy. The same applies for the derivation of additional information from the existing data. In [GH14] we introduced an algorithm which reduced the time to find all quasi-identifiers by more than 95% (in contrast to the algorithm proposed by [MX07]). By this, the workload of sensors and other resource-limited devices is not affected too much. The basic idea of the concept is shown in Algorithm 1. The method alternates between a bottom-up- and a top-down-approach and shares the knowledge about (negated) quasi-identifiers (similar to the Sideways Information Passing approach [IT08]). Per calculation step either the top-down (testing attribute-combination with many attributes) or bottom-up

(small combinations) method is executed and the results of the step are passed to the other method. The algorithm terminates when all attribute levels were executed by one of the methods or the bottom-up approach has to check no more attribute combinations. More details are given in our previous paper [GH14].

Algorithm 1: bottomUpTopDown

```
Data: database table tbl, list of attributes attrList
Result: a set with all minimal quasi-identifier qiSet
attrList.removeConstantAttributes();
Set upperSet := new Set({attrList});
Set lowerSet := new Set(attrList);
// Sets to check for each algorithm
int bottom := 0;
int top := attrList.size();
while (bottom¡=top) or (lowerSet is empty) do
    calculateWeights();
    if isLowerSetNext then
        bottomUp();
        buildNewLowerSet();
        bottom++;
        // Remove new QI from upper set
        modifyUpperSet();
    else
        topDown();
        buildNewUpperSet();
        top--;
        // Remove new negated QI from lower set
        modifyLowerSet();
    end
end
qiSet := qiLowerSet ∪ qiUpperSet;
return qiSet;
```

As stated above, the data stored in assistive systems can be divided into four categories: key attributes, quasi-identifiers, sensitive data and insensitive data. Quasi-identifiers and key attributes allow the unambiguous identification of an object, such as a person or an action. These attributes make it possible to combine information from different tables, databases or with background knowledge. The key property of an attribute is usually defined within the database design process. In most cases, system-generated artificial attributes are created for keys, but several existing attributes can be recognized as a key. Keys should not be passed; they can be linked to information outside from the system and by this, new information on the affected object can be determined.

Sensitive data is data that needs to be protected. It depends on the application domain and the affected person to whom the data relate, which data is classified as sensitive. In the

Table 1: Device 1 stores the location (*x, y, z*) about the user (*firstname, lastname*) at a specific *time*. The user does not want to reveal his *lastname*.

attribute	*type*	*privacy*
lastname	string	PRIVATE
firstname	string	PUBLIC
x	integer	PUBLIC
y	integer	PUBLIC
z	integer	PUBLIC
timestamp	timestamp	PUBLIC

health sector for example, these are information about diseases. In contrast to sensitive data, insensitive data are those which do not necessarily have to be protected. Again, the domain and the personal preferences are critical of how the data will be classified.

3.1.2 Protection of secrets

In some assistive systems, it is necessary that the user has to be identified; so the data is not anonymized. Nevertheless, the user may not want to reveal all information about him or her. In this case, the user has to define his privacy requirements into a policy.

For an inexperienced user, it is not easy to make meaningful privacy settings. To make matters worse, in ad-hoc environments, the sensors and processing computer and the algorithms used can be replaced without the user even noticing it. For these reasons, it is important that the privacy component of the assistive system can generate useful integrity constraints for the user.

To overcome this problem, we are generating privacy policies for new devices based on the settings the user has specified for older devices before. We realize this in three steps. In the first step, schema mappings [MGMR02] between the new device and all known old devices are generated. The privacy settings of the best fitting devices are adopted for each mapped attribute. For those attributes on the new device without a compatible attribute, data mining approaches (clustering, classification) are used to find out which attributes with similar privacy settings are used together. From these attribute groups we adopt the settings for the unmapped attributes. In the third step, all attributes without a privacy mapping are set to *PRIVATE*, a state where the attribute value cannot be accessed.

The generated privacy settings for the new application/sensor are only of preliminary nature. It is up to the user to accept these settings or to modify them later to meet his desires. An example for the mapping process is shown in Table 4, where Tables 1-3 represent the old devices.

Table 2: Device 2 stores the location (*x, y, z*) and the *direction* of movement about the user (*id, firstname, lastname*). The user does not want to reveal his full name.

attribute	*type*	*privacy*
id	integer	PUBLIC
lastname	string	PRIVATE
firstname	string	PRIVATE
x	integer	PUBLIC
y	integer	PUBLIC
z	integer	PUBLIC
direction	vector	PUBLIC

Table 3: Device 1 stores the location (*position*) and the *direction* of movement about the user (*id, firstname*) at a specific time. The user does not want to reveal his name.

attribute	*type*	*privacy*
id	string	PUBLIC
firstname	string	PRIVATE
position	tuple(integer, integer)	PUBLIC
timestamp	timestamp	PUBLIC
direction	vector	PUBLIC

Table 4: The new device before and after generating privacy settings. After the mapping step, the *first-* and *lastname* are set to private; *id, x, y, z* and *direction* to public (based on the most similar device 2). *Timestamp* and *activity* are not mapped. After the data mining step, *timestamp* is mapped to public (based on the settings for device 1 and 3). Because *activity* has no corresponding attribute, it is set to private (default value).

attribute	*type*	*before mapping*	*after mapping*	*after data-mining*	*final*
id	string	???	PUBLIC	PUBLIC	PUBLIC
firstname	string	???	PRIVATE	PRIVATE	PRIVATE
lastname	string	???	PRIVATE	PRIVATE	PRIVATE
x	integer	???	PUBLIC	PUBLIC	PUBLIC
y	integer	???	PUBLIC	PUBLIC	PUBLIC
z	integer	???	PUBLIC	PUBLIC	PUBLIC
timestamp	timestamp	???	???	PUBLIC	PUBLIC
direction	vector	???	PUBLIC	PUBLIC	PUBLIC
activity	string	???	???	???	PRIVATE

3.2 Privacy policies for smart environments

Our data privacy policy is based on the draft of the W3C for Privacy Preferences Platform [W3C07], but leaves out browser-specific details, such as the management of cookies. In

return, our policy provides additional information for configuring data streams, such as the allowed query interval and possible aggregation levels.

Hitherto, security models and privacy policies focus on how whole data records can be protected from unauthorized access. Adding a more fine-granular access control allows the user to specify which information is revealed and how the data are processed. Data records can be modified by

- selection,
- projection,
- compression,
- aggregation and
- feature extraction (e.g. on motion images)

to reduce the amount of personal information send towards the tools for evaluation and analysis.

Our privacy policy is based on the draft of the W3C for Privacy Preferences Platform [W3C07], but leaves out browser-specific details, such as the management of cookies. In return, our policy provides additional information for configuring data streams, such as the allowed query interval and possible aggregation levels.

3.2.1 Internal Representation

In this chapter the policy is explained based on a small example, which is illustrated in Figure 4. The formal definition is given in the next section.

For internal representation and data exchange, we store the policy as an XML file. A policy is a set of applications (*apps*). An application can be a stand-alone computer program, a background process of an operating system or even a smart environment as a whole. Each application consists of a unique ID (in the example: *MyFridge*), which allows unique identification by its name. In addition, an application has a description and a list of functions (*modules*).

A module of an application is a well-defined task to solve a specific problem of the entire system. For example, a fridge has the basic functionality to check its contents (in the example: *CheckContent*) to determine how much food remains. Another module may implement the automation of buying new food. A module is characterized by its ID and its type. Each module consists of a description and an indication if the functionality is required to execute the application.

All modules provide an *attribute-list* containing all needed attributes. Attribute-lists can be connected with boolean operators (like *and* or *or*) to provide different alternatives if a user does not want to reveal a specific combination of personal information. An attribute is identified by its *name* and persists of different privacy settings. The access conditions for

```
<policy>
 <apps>
  <application id="MyFridge">
   <appDescription>
    This is a smart fridge.
   </appDescription>
   <modules>
    <module id="CheckContent" type="basic"
       required="yes">
     <moduleDescription>
      This module will check which
        amount of food is remaining.
     </moduleDescription>
     <attribute-list>
      <attribute name="amount">
       <allow>true</allow>
       <third-party-access>false</third-party-access>
       <condition>FAT <= 40</condition>
      </attribute>
     </attribute-list>
    </module>
    ...
   <modules>
  </application>
 <apps>
</policy>
```

Figure 4: Example for a privacy policy

the application are controlled by the *allow*- (the application itself) and *third-party-access*-tag (to shrink continued processing). The user can specify several *condition*s under which the information is disclosed. Additionally, settings regarding *aggregation* and *confidentiality* (privacy-level of detail) are specified. In the example, the application has access to the amount of food remaining for every item in the fridge, except of the food which has more than 40% of fat.

The separation of functions and applications is important, because this ensures that information can only be used for the intended purpose. Thus, the user may share his data for personalized advertisements. On the other side, the user does not want this information to be transmitted to his insurance company, so that his contribution rate will be adjusted according to his eating habits.

3.2.2 Formal definition

Our privacy policy model is a 9-tupel *P := (APP, M, ATTR, C, I, A, AGG, ANO, f)*, where

- APP := a set of applications
- M := a set of modules
- ATTR := a set of attributes
- C := a set of conditions

- I := a set of query-intervals
- A := a set of access conditions
- AGG := a set of aggregation functions
- ANO := a set of anonymization functions
- f := a partially defined function, which maps a combination of an application *app* $\in$ APP, a module *m* $\in$ M and an attribute *attr* $\in$ ATTR to a set of conditions *C'* $\subseteq$ C, as well as a query interval *i* $\in$ I, several access conditions *A'* $\subset$ A, an aggregate function *agg* $\in$ AGG and an anonymization function *ano* $\in$ ANO.
  $f := (app, m, attr) \rightarrow (C', i, A', agg, ano)$

The terms application, module, attribute and (access) condition have been introduced above. The term interval is related to data streams and their continuous collect of information. By specifying an interval, the user can decide in which frequency his data can be accessed. Aggregation functions allow the user to hide raw data. Typical functions are sums, minimum and maximum values and medians as well as complex, statistical constructs like correlation coefficients and regression lines. By selecting an anonymization function f, the user can decide which privacy model is used in the postprocessing stage. If no method is selected, an appropriate one is selected automatically.

Having selected the function f, the privacy requirements are transformed into Multiple Access-control Generated Integrity Constraints (MAGIC) for specific query languages and concrete (database) systems. With MAGIC, the privacy algorithms in the pre- as well as in the postprocessing stage (see Section 2) are parametrized.

4 Related Work

Due to the Snowden affair [Gre14], a hype around the field of privacy-enhancing technologies have arisen. In this chapter we present some proven techniques and systems, which dealt with the protection of privacy before Snowden.

4.1 Frameworks

In his paper [Bün09], Bünnig presents some approaches that implement the trusted exchange of whole documents between two parties. The sharing model uses machine learning techniques for automatic, context-based classification (release: yes or no) to carry out and transfer the old settings to new documents.

The PArADISE approach uses similar concepts, but is based on a more detailed release model and is not restricted to a document as a whole. Furthermore, PArADISE also analyzes and, if necessary, modifies the content of data sets before passing them to other processing units.

In addition to [Bün09], there is a vast amount of other research prototypes, like the framework developed at the University of Potsdam by Scheffler [Sch13] or the specialized privacy-aware XML framework developed by [lRA14]. Most approaches have in common that they are restricted to specific data formats such as relational structures, XML or JSON and do not consider the heterogeneity of data sources. In addition, they are limited to simple locking mechanisms, which do not provide advanced anonymity concepts.

4.2 Policies

Usually, simple access control lists (ACL) are often used for the formulation and enforcement of privacy rights. ACLs define the access rights only for files, but not for content. Besides pure rights management, privacy policies can be used as an advanced concept that allows a more detailed definition of privacy claims.

The World Wide Web Consortium (W3C) is proposing two standards for the formulation of privacy policies. On the one hand, there is the Platform for Privacy Preferences Project [W3C07], a markup language that is intended for use in browsers. The user can generally define what kind of websites can be accessed. The website provider can formulate what data he stores, e.g. by means of cookies, about the user. If these requirements contain contradictions, the user is warned.

In addition to the P3P, the W3C has suggested the Enterprise Privacy Authorization Language [W3C03], another language that is specifically designed for the exchange of data in business environments. Both language proposals are based on the XML format and provide a rule-based access mechanism in which roles and conditions are formulated to grant or deny write and read access rights.

Besides the above mentioned W3C standards, there exist many other language proposals, such as the eXtensible Access Control Markup Language [OAS13]. These do not differ substantially from the existing language proposals.

4.3 Bell-LaPadula security model

Security models are designed to prevent unauthorized data modification and access and to maintain consistency. One of the most frequently used models is introduced by Bell and LaPadula in [BL73]. The model provides three rules to achieve trust and privacy:

- simple security property: a subject with a lower security level than an object must not read the object
- star-property: a subject with a higher security level than an object must not write the object
- discretionary security property: an access control matrix is used to specify the relation between objects and subjects

Aside this model, there exist several other models like the model by Biba [Bib77] or the Chinese Wall Model [BN89] which offer similar approaches to provide access control mechanism.

5 Conclusions

The protection of privacy in ubiquitous environments is a challenging task. A variety of sensors capture every moment of our life. In this paper, we present a method to formulate privacy claims on an assistive system, so that we become the master of our own personal data. By reducing and preprocessing the data generated by the sensors and processed by the system, privacy can be ensured in smart environments.

We also presented a privacy-aware query processor. Powered by the data protection profiles of the users, the acquisitiveness of assistive systems is limited. Our privacy-aware query processor gives a comprehensive tool to ensure privacy by analyzing queries and optionally rewrites them.

At the present time, we integrate existing data protection techniques and analysis functions (in the context of student projects) into the query processor. It is recommended that further research should be undertaken in the following areas:

- Automatic comparison between the privacy requirements of the user and the required information of the system.
- Anti-virus programs offer protection against the newest malicious software. Can this procedure be adapted to data mining technologies and privacy protection mechanism?

Further studies, which take the comparison of the privacy constraints and the analysis functions into account, will need to be performed. We are in the process of investigating the transformation process of complex analysis functions into SQL-queries.

6 Acknowledgements

Hannes Grunert is funded by the German Research Foundation (DFG), Graduate School 1424 (Multimodal Smart Appliance Ensembles for Mobile Applications - MuSAMA).

References

[Bib77] Kenneth J Biba. Integrity considerations for secure computer systems. Bericht, DTIC Document, 1977.

[BK12] Sebastian Bader und Thomas Kirste. An overview of the helferlein-system. *Institut für Informatik, Universität Rostock, Rostock, Germany, Tech. Rep. CS-03-12*, 2012.

[BL73] D Elliott Bell und Leonard J LaPadula. Secure computer systems: Mathematical foundations. Bericht, DTIC Document, 1973.

[BN89] David F.C. Brewer und Dr. Michael J. Nash. The Chinese Wall Security Policy, 1989.

[Bün09] Christian Bünnig. Smart privacy management in ubiquitous computing environments. In *Human Interface and the Management of Information. Information and Interaction*, Seiten 131–139. Springer, 2009.

[Bun10] Bundesrepublik Deutschland. Bundesdatenschutzgesetz in der Fassung der Bekanntmachung vom 14. Januar 2003 (BGBl. I S. 66), das zuletzt durch Artikel 1 des Gesetzes vom 14. August 2009 (BGBl. I S. 2814) geändert worden ist, 2010. in german.

[Dal86] Tore Dalenius. Finding a Needle In a Haystack or Identifying Anonymous Census Records. *Journal of Official Statistics*, 2(3):329–336, 1986.

[GH14] Hannes Grunert und Andreas Heuer. Big Data und der Fluch der Dimensionalität: Die effiziente Suche nach Quasi-Identifikatoren in hochdimensionalen Daten. In *Proceedings of the 26th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken)*. `http://ceur-ws.org`, 2014. in german.

[Gre14] Glenn Greenwald. *No Place to Hide: Edward Snowden, the NSA, and the US Surveillance State*. Metropolitan Books, 2014.

[Gru14a] Hannes Grunert. Distributed Denial of Privacy. In *INFORMATIK 2014: Big Data Komplexität meistern*, Seiten 2299–2304. Springer, 2014.

[Gru14b] Hannes Grunert. Privacy-aware Adaptive Query Processing in Dynamic Networks. In *Proceedings of the 8th Joint Workshop of the German Research Training Groups in Computer Science*. Anja Jentzsch, Tobias Pape and Sebastian Pasewaldt (Edt.), 2014.

[HS10] Ayça Azgin Hintoglu und Yücel Saygın. Suppressing microdata to prevent classification based inference. *The VLDB Journal*, 19(3):385–410, 2010.

[IT08] Zachary G Ives und Nicholas E Taylor. Sideways information passing for push-style query processing. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, Seiten 774–783. IEEE, 2008.

[KL51] Solomon Kullback und Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, Seiten 79–86, 1951.

[LLV07] Ninghui Li, Tiancheng Li und Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *IEEE 23rd International Conference on Data Engineering*, Seiten 106–115. IEEE, 2007.

[LLZM12] Tiancheng Li Li, Ninghui Li, Jian Zhang und Ian Molloy. Slicing: A New Approach for Privacy Preserving Data Publishing. *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD)*, 24(3):561–574, March 2012.

[lRA14] Alberto De la Rosa Algarín. An XML Security Framework that Integrates NIST RBAC, MAC and DAC Policies, 2014.

[MGMR02] Sergey Melnik, Hector Garcia-Molina und Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, Seiten 117–128. IEEE, 2002.

[MKGV07] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke und Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[MX07] Rajeev Motwani und Ying Xu. Efficient algorithms for masking and finding quasi-identifiers. In *Proceedings of the Conference on Very Large Data Bases (VLDB)*, Seiten 83–93, 2007.

[OAS13] OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`, 2013.

[Sam01] Pierangela Samarati. Protecting respondents identities in microdata release. *Knowledge and Data Engineering, IEEE Transactions on*, 13(6):1010–1027, 2001.

[Sch13] Thomas Scheffler. *Privacy enforcement with data owner-defined policies*. Dissertation, Universität Potsdam, 2013.

[W3C03] W3C. Enterprise Privacy Authorization Language. `http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/`, 2003.

[W3C07] W3C. Platform for Privacy Preferences (P3P) Project. `http://www.w3.org/P3P/`, 2007.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

A PP4SE

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema
3     xmlns:xs="http://www.w3.org/2001/XMLSchema"
4     elementFormDefault="qualified"
5     attributeFormDefault="qualified">
```

Figure 5: The Privacy Policy for Smart Environments (PP4SE) is based on XML Schema.

```
1   <xs:element name="policy">
2       <xs:complexType>
3           <xs:sequence>
4               <xs:element ref="apps" minOccurs="0" maxOccurs="1"/>
5           </xs:sequence>
6       </xs:complexType>
7   </xs:element>
8
9   <xs:element name="apps">
10      <xs:complexType>
11          <xs:sequence>
12              <xs:element ref="application" minOccurs="0"
13                  maxOccurs="unbounded"/>
14          </xs:sequence>
15      </xs:complexType>
16  </xs:element>
```

Figure 6: Every Privacy Policy consists of a list (*apps*) of *application*.

```
1   <xs:element name="application">
2       <xs:complexType>
3           <xs:sequence>
4               <xs:element ref="app_description" minOccurs="0"
5                   maxOccurs="1"/>
6               <xs:element ref="modules" minOccurs="0" maxOccurs="1"/>
7           </xs:sequence>
8           <xs:attribute ref="app_ID" use="required"/>
9       </xs:complexType>
10  </xs:element>
11
12  <xs:attribute name="app_ID" type="xs:string"/>
13  <xs:element name="app_description" type="xs:string" />
14
15  <xs:element name="modules">
16      <xs:complexType>
17          <xs:sequence>
18              <xs:element ref="module" minOccurs="0"
19                  maxOccurs="unbounded"/>
20          </xs:sequence>
21      </xs:complexType>
22  </xs:element>
```

Figure 7: An application has a description (*app_description*), a unique ID (*app_ID*) and a list (*modules*) of its provided functionalities (*module*).

```
1   <xs:element name="module">
2       <xs:complexType>
3           <xs:sequence>
4               <xs:element ref="module_description" minOccurs="0"
5                   maxOccurs="1"/>
6               <xs:element ref="attributeList" minOccurs="0"
7                   maxOccurs="1"/>
8           </xs:sequence>
9           <xs:attribute ref="module_ID" use="required"/>
10          <xs:attribute ref="module_type" use="required"/>
11          <xs:attribute ref="module_required" use="optional"/>
12      </xs:complexType>
13  </xs:element>
14
15  <xs:attribute name="module_ID" type="xs:string"/>
16  <xs:attribute name="module_type" type="xs:string"/>
17  <xs:attribute name="module_required" type="xs:boolean"/>
18  <xs:element name="module_description" type="xs:string" />
19
20  <xs:element name="attributeList">
21      <xs:complexType>
22          <xs:sequence>
23              <xs:element ref="attribute" minOccurs="0"
24                  maxOccurs="unbounded"/>
25          </xs:sequence>
26      </xs:complexType>
27  </xs:element>
```

Figure 8: A module consists of its description (*module_description*), a unique ID (*module_ID*), the type of the module (*module_type*) and the declaration if this module is required for executing the application (*module_required*). Every module provides a list (*attributeList*) with its appropriate *attribute*s.

```
1   <xs:element name="attribute">
2       <xs:complexType>
3           <xs:choice>
4               <xs:sequence>
5                   <xs:element ref="allow" minOccurs="0"
6                       maxOccurs="1"/>
7                   <xs:element ref="third_party_access" minOccurs="0"
8                       maxOccurs="1"/>
9                   <xs:element ref="condition" minOccurs="0"
10                      maxOccurs="1"/>
11                  <xs:element ref="aggregation" minOccurs="0"
12                      maxOccurs="1"/>
13                  <xs:element ref="interval" minOccurs="0"
14                      maxOccurs="1"/>
15              </xs:sequence>
16              <xs:element ref="privacyLevel" minOccurs="0"
17                      maxOccurs="1"/>
18          </xs:choice>
19          <xs:attribute ref="name" use="required"/>
20      </xs:complexType>
21  </xs:element>
22
23  <xs:attribute name="name" type="xs:string"/>
24  <xs:element name="allow" type="xs:boolean"/>
25  <xs:element name="third_party_access" type="xs:boolean"/>
26  <xs:element name="interval" type="xs:string" />
```

Figure 9: An attribute is identified by its *name*. Furthermore, it is declared if the attribute can be accessed directly (*allow*) or indirect by third-party-applications (third_party_access). Furthermore, general *condition*s, level of *aggregation* and the query *interval* can be defined for the attribute. If the user don't want to give detailed privacy settings, he can specify a level of anonymization *privacyLevel* instead.

```
1   <xs:element name="privacyLevel">
2      <xs:simpleType>
3         <xs:restriction base="xs:string">
4           <xs:enumeration value="public"/>
5           <xs:enumeration value="confidential"/>
6           <xs:enumeration value="secret"/>
7           <xs:enumeration value="top-secret"/>
8         </xs:restriction>
9      </xs:simpleType>
10  </xs:element>
```

Figure 10: A simplified *privacyLevel* consists of a domain-specific enumeration of chosen values. Typical levels for such values are, for example *public*, *confidential*, *secret* and *top secret*.

```
<xs:element name="condition">
    <xs:complexType>
        <xs:choice>
            <xs:element ref="atomicCondition" minOccurs="1"
                maxOccurs="1"/>
            <xs:element ref="andCondition" minOccurs="1"
                maxOccurs="1"/>
            <xs:element ref="orCondition" minOccurs="1"
                maxOccurs="1"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

Figure 11: A release condition can be divided into *atomicCondition*s and composed conditions.

```
<xs:element name="atomicCondition" type="xs:string" />

<xs:element name="andCondition">
    <xs:complexType>
        <xs:choice minOccurs="2" maxOccurs="unbounded">
            <xs:element ref="atomicCondition" minOccurs="1"
                maxOccurs="unbounded"/>
            <xs:element ref="orCondition" minOccurs="1"
                maxOccurs="unbounded"/>
        </xs:choice>
    </xs:complexType>
</xs:element>

<xs:element name="orCondition">
    <xs:complexType>
        <xs:choice minOccurs="2" maxOccurs="unbounded">
            <xs:element ref="atomicCondition" minOccurs="1"
                maxOccurs="unbounded"/>
            <xs:element ref="andCondition" minOccurs="1"
                maxOccurs="unbounded"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

Figure 12: An atomic condtion can't be divided. To formulate such a condition, a SQL-WHERE-clause or an OCL-constraint can be used, depending on the domain. An *andCondition* consists at least of two atomic or *orCondition*s. The same applies for the *orCondition*. For internal use of the conditions the clauses are transformed int a disjunctive normal form for better evaluation.

```
1     <xs:element name="aggregation">
2         <xs:complexType>
3             <xs:sequence minOccurs="1" maxOccurs="1">
4                 <xs:element ref="aggregationType" minOccurs="1"
5                     maxOccurs="1"/>
6                 <xs:element ref="groupBy" minOccurs="1" maxOccurs="1"/>
7             </xs:sequence>
8         </xs:complexType>
9     </xs:element>
10
11    <xs:element name="groupBy" type="xs:string" />
12
13    <xs:element name="aggregationType">
14        <xs:simpleType>
15          <xs:restriction base="xs:string">
16            <xs:enumeration value="min"/>
17            <xs:enumeration value="max"/>
18            <xs:enumeration value="sum"/>
19            <xs:enumeration value="count"/>
20            <xs:enumeration value="avg"/>
21          </xs:restriction>
22        </xs:simpleType>
23    </xs:element>
24
25 </xs:schema>
```

Figure 13: To define an agggregation-constraint an aggregation function (aggregationType) and a groupping function (groupBy) are given.

Impressum

Universität Rostock
Institut für Informatik
Lehrstuhl für Datenbank- und Informationssysteme
Albert-Einstein-Straße 22
18059 Rostock

Vertreten durch: Prof. Dr. rer. nat. habil. Andreas Heuer