University of Twente
Enschede - The Netherlands

Faculty of Electrical Engineering, Mathematics and Computer Science
Databases Department

Life-Cycle Privacy Policies for the Ambient Intelligence

By
Harold van Heerde

February 15, 2006

Graduating committee:
Dr. Ling Feng
Dr. Nicolas Anciaux
Dr. Maarten Fokkinga

Abstract

Ambient Intelligence (AmI) environments continuously monitor individuals' context such as locations, activities, *et cetera*. The purpose of this is to make existing applications smarter, so they can make decisions without requiring user interaction. Such AmI smartness ability is tightly coupled to quantity and quality of the available context. Keeping in mind that there is a chance that their privacy is violated, it is not likely that people are willing to accept an environment in which many actions and the behavior of people are sensed to make a smart AmI possible. The goal of our research is to make a compromise between privacy and smartness in the AmI, by introducing policies in which donors can regulate the life cycle of their context data. We believe that, by giving the full control on their privacy to the donors of the context, will help the Ambient Intelligence being accepted by the public.

In this thesis, we propose to bind user specific Life-Cycle Policies (LCP) to context data regulating its progressive degradation. We investigate the problem of correctness of the LCP model when used to implant one-way degradation (ensuring that degraded information can no longer be recovered from the current database content). Finally, we show the feasibility of the proposed techniques by implementing a prototype on top of a traditional relational database.

Contents

Preface

Finally, the end of my study *Technical Computer Science* is coming to an end with the finishing of this thesis. I must say that after five years of studying, and having a great time at the University of Twente, I found myself lucky of being able to work on this great project with this thesis as a result. I even have a great opportunity to continue my time at the University with a PhD. position.

The subject of privacy in the Ambient Intelligence is or could be an 'hot topic' in the near future. In my opinion, the topic deserves this status as being crucial for future developments toward ubiquitous computing, smart surroundings and intelligent environments in which huge amounts of (possible privacy sensitive) data are collected and stored, which could have a great impact on social behavior. During the period I spend on this subject, I became more and more convinced that there is a serious lack of consciousness concerning privacy. Governments and service providers must be aware of the risks of collecting privacy sensitive data, otherwise the public may become to suspicious regarding new techniques like the Ambient Intelligence, if privacy violations are more common than rare. I hope our research and this thesis could be a good contribution for the Ambient Intelligence, and protecting privacy in common.

First of all I must give most credits to Nicolas Anciaux, member of my graduating committee, who has initiated the idea of life cycle policies, and has made it possible for me to participate in this research project. I have experienced a great time working with him, and learned a lot about the scientific approach of doing research. Therefor I want to thank him very much. Second I want to thank Ling Feng, first member of my graduating committee, for her help and advice during the project. Her enthusiasm gave me confidence, and helped me in making the decision to become a PhD student after finishing my study. Finally I want to thank Maarten Fokkinga, not only for the support during my writing of this thesis and following the progress of the project, but also for the support throughout almost my whole study. Besides giving me jobs to support some database related courses as a student assistant, he supervised my 'Panradio project' and helped me writing my first publication.

(To continue in Dutch...)

Naast mijn begeleiders wil ik natuurlijk ook mijn vriendin Ilse bedanken voor haar hulp bij het schrijven van dit afstudeerverslag, en het aanhoren van de voortgang van het hele project. Uiteraard kan ik mijn ouders niet vergeten die het überhaupt mogelijk gemaakt hebben om zover te komen in mijn studie, zonder al te veel druk te zetten. Ik heb daardoor kunnen genieten van mijn studie zonder mij al te grote (financiële) zorgen te hoeven maken, iets wat in mijn ogen erg belangrijk is. Als laatste wil ik iedereen bedanken die geholpen heeft

en interesse heeft getoond in mijn voortgang, ook het alleen maar aanhoren van schijnbaar onoplosbare problemen gedurende dit afstudeerproject hebben een bijdrage geleverd aan dit verslag.

(To continue in English again...)

I hope you enjoy reading this thesis. A paper on this subject, in which I have participated in writing it, is submitted to the SIGMOD 2006 conference and is currently under review by the conference committee.

Harold van Heerde,
Enschede, The Netherlands,
February 15, 2006

Chapter 1

Introduction

> *"Ambient Intelligence represents a vision of the future where we shall be surrounded by electronic environments, sensitive and responsive to people. Ambient intelligence technologies are expected to combine concepts of ubiquitous computing and intelligent systems putting humans in the centre of technological developments."[11]*

1.1 Background

Ambient intelligence is the future, or at least, it *could* be the future. An (electronic) environment can only be intelligent if it has enough data, or information, about the entities within that environment and especially the human being which should be in the centre of this. With different kinds of techniques, for example (RFID [23])-sensors and camera's, it is possible to *sense* a person's behavior, movements, et cetera. However, history shows a negative tendency concerning abuse of private information and a lack of security considerations while handling people's private data [2, 17].

In general, most people are not very willing to supply information to (for example) websites, because they are worried that their information will be used for *spam* and other kinds of abuse [9]. According to a poll in 2000, 84% of the Internet users are concerned privacy sensitive information is gathered and used for unknown matters [19]. There are policies which could be adopted by websites, which ensure privacy for anyone who supplies information (for example customers). A set of standards, called P3P [32], allows companies to declare these policies regarding privacy, upon which customers can agree (or disagree). Those companies are then responsible for conforming to their own policies, and therefor preserve the privacy of their customers. It is argued that such P3P protocol is not sufficient enough. You have to trust that the policy is clearly (and not ambiguously) specified, you are never sure if the company is really applying the policy, and you have to trust the company is technically capable of ensuring privacy at all [1]. A good survey of the disadvantages of the P3P platform can be found in the work of Bertino et al [5].

In the following section we will give some examples of (possible) privacy violations, which kind of impact they could have, and what caused the privacy violations. Those examples show that privacy could really be an issue. We think that privacy violations in the past could have an impact on how people will view the prospect of being monitored by a huge amount of sensors in the future.

This thesis will focus on the design and implementation of a data gathering and storage mechanism, in which donors of the data have full control over what data could be collected when and where, in what form the data is stored and what the life cycle of that data should be. In our research we will choose a different approach than the traditional *access control* mechanisms, because we think the only way to prevent privacy sensitive data to be retrieved by unauthorized individuals, is to remove the data when possible. This philosophy is based on a very simple example. In the real world, footsteps will leave a footprint. When the footprint is fresh, some details can be derived from the footprint, perhaps even the identity of a person who has left the footprint. Depending on time and the situation (like surface, density of people, et cetera),the footprint will fade away, leaving less detailed information behind. Our research will focus on this principle.

1.2 (Possible) Privacy violations

In this section we will try to sketch the problems with privacy in combination with services. First, we will show the dangers with respect to privacy violations that are issued by new webservices. Second, we will show, by means of some examples, the difficulties introduced by health care. Third, we will address some of the developments in the way the EU deals with privacy involving data storage. Those examples are meant to give an indication to what extend those possible privacy violations can influence future developments of the Ambient Intelligence. After that, we will have a look at possible black scenarios for the Ambient Intelligence itself.

1.2.1 'Traditional' privacy issues

Google

We start our survey of possible privacy violations with Google. Google is well known because of its successful search engine which is now one of the most used search engines in the world. Currently Google earns its money by selling *Adwords*, which are payed search engine results. If someone issues a query to Google's search engine, Google presents, together with the normal search results, also sponsored advertisements. The company which has the highest bid on a particular *keyword* will be placed highest in the search results on that keyword.

Since some years, Google is starting to launch a large number of new (free) services, like GMail, Google Earth, Google SMS, and so on [33]. Recently Google launched *Google Analytics*, a tool to monitor websites and measure the effects of promotional campaigns. With all those services, Google is able to collect a huge amount of data about their users [8], including privacy sensitive information. However, privacy advocates claim that Google does have the ability to build very detailed profiles of their users, which implies that there is a huge privacy risk [27]. For example: because of the free available Analatycs software, it is possible for website owners to store information about every visit of a website per individual. This information is stored on the servers from Google. If the monitored individual also has a GMail email account, Google is able to crosslink the visited websites with information from the individual's emails and account information. Moreover, if the same individual uses the search engine, the used queries can be stored in the profile.

Of course, Google itself disagrees with that they are up to violating the privacy of their users in their privacy statement [12]. Indeed, the commonly used slogan by Google is 'Don't be evil'. However, the same privacy advocates as mentioned above state that Google doesn't give any guarantees that they will not misuse the gathered information in the future. What will happen if the databases are hacked, or what will happen if an employee, with access to the data, is payed to handover the profiles? Huge amounts of privacy sensitive data could be disclosed to parties which can make misuse of it.

> *Example:* A student wants to have some information about HIV, because of a project on his school. He knows that HIV is a sexual exchangeable disease, and that the disease is relative often be spread by gay persons. So, unknown where to start, he types in the queries *'information about hiv'* and *'gay sexual'*. Indeed, the last query is not likely to give the results he looked for. Naive as the person is, he clicks on a result link and enters a gay porn site which is monitored by Google Analytics. The student also has a GMail account. With the GMail account, he has send an email with is curriculum vitae to a possible employer.
> Now imagine that the profile of the student has become publicly available, and a possible employer can search through this profile. Although, at least in the Netherlands, it is completely legal to be gay, and the student even isn't gay at all, to be gay is not always commonly accepted. The employer doesn't want to take any risks, and decides not to give the job to the student.

Very recently, the government of the USA demanded the disclosure of data stored by Google [25]. This data includes the queries which visitors of the Google search engine use to search the web. Currently, at the time of writing, Google refuses to disclose this information.

Health services

To be able to give persons the health care they need, a lot of privacy sensitive information could be needed and is therefor gathered by hospitals, insurance companies, et cetera. Information which could be misused by others, but also information which people simply doesn't want to share with others because it could have a negative social impact.

Regrettably, there are many examples of privacy violations in health and human services [16]. Some examples are (directly copied from the HIPAA privacy and security website):

- The medical records of an Illinois woman were posted on the Internet without her knowledge or consent a few days after she had been treated at St. Elizabeth's Medical Center following complications from an abortion at the Hope Clinic for Women. The woman has sued the hospital, alleging St. Elizabeth's released her medical records without her authorization to anti-abortion activists, who then posted the records online along with a photograph they had taken of her being transferred from the clinic to the hospital. The woman is also suing the anti-abortion activists for invading her privacy. (T. Hillig and J. Mannies, "Woman Sues Over Posting of Abortion Details," St. Louis Post-Dispatch, July 3, 2001, p. A1).

- New York Congresswoman Nydia Velasquez's confidential medical records – including details of a bout with depression and a suicide attempt – were faxed from a New York hospital to a local newspaper and television station on the eve of her 1992 primary. After overcoming the fallout from this disclosure and winning the election, Rep. Velasquez testified eloquently about her experiences before the Senate Judiciary Committee as it was considering a health privacy proposal. (A. Rubin, "Records No Longer for Doctors' Eye Only," Los Angeles Times, September 1, 1998, p. A1)

- In Tampa, a public health worker walked away with a computer disk containing the names of 4,000 people who tested positive for HIV. The disks were sent to two newspapers. (J. Bacon, "AIDS Confidentiality," USA Today, October 10, 1996, p. A1)

These examples, in which human mistakes or actions directly cause the privacy violations, make clear that privacy violations not always are due to a lack of technical security. Money, human emotions, career prospects, et cetera could be reasons to violate privacy of individuals (like political opponents). Only if privacy sensitive data is actually removed when it isn't needed anymore, above violations could have been prevented.

However, the question is to which degree parts of medical reports could be removed. A history of health related issues can be important for further treatments, so it is likely that it is needed to keep this privacy sensitive information to be able to ensure good services. *Details* of the treatment, or parts which are too privacy sensitive and which are not needed for future treatments could however be removed. People must be able to decide for themselves in which extent they want to exchange privacy for the best possible service. However, it has also been argued that you cannot know in advance wetter or not there will be situations in which you wanted your medical record had been available: *"Should I be knocked unconscious in a road traffic accident in New York please let the ambulance have my medical record."* [20]

Data storage law in the European Union

In 1949, George Orwell published his novel *1984* with it's central theme 'Big Brother is watching you'. In this book, a totalitarian state controls every aspect of life.

The European Union is not a totalitarian state, and hopefully it never becomes such a state. However, the recent developments around a law proposal which orders the storage of all telecommunication data are quite disturbing with respect to possible privacy violations. The proposal defines the storage of all telecom and Internet data, including calling, emailing, SMS-ing, geographical locations and so on of all citizens of the European Union for at least 12 months. With this proposal, a huge collection of privacy sensitive data about all people living in the EU could be build. With access to this collection, all social contacts between humans could be tracked. This data must be available to help in the efforts of the EU to fight against terrorism [24].

The EU itself recognizes the possible effect on privacy laws, but claims that *"the interference with these rights is justified in accordance with of Article 52 of the Charter on Fundamental rights. Specifically, the limitations on these rights provided for by the proposal are proportionate and necessary to meet the generally recognised objectives of preventing and combating crime and terrorism."* [24].

With respect to our research, one particular remark can be made from this statement. The EU offers their citizens the service *fight against terrorism* in return for privacy. Citizens do not have any control over how, how long, and which data with how much detail is stored. They have to trust that data is stored secure, and that only authorized institutes have access to the data. Which institutes are authorized is not clear. If the data will be misused, and privacy is violated for the wrong goals, people could possibly become very suspicious when the ambient intelligence is introduced, with even more sensors and collection of privacy sensitive data.

1.2.2 Black scenarios for the Ambient Intelligence

In the preceding section we gave examples of possible privacy violations. These scenarios are traditional in this sense that they are based on existing technologies and existing privacy threads. The Ambient Intelligence, and especially the usage of ubiquitous computing facilities, will bring along new privacy threads. We will describe those new privacy threads in section 2.2.1, in this section we limit ourself to give simple examples of privacy threads which could occur in future Ambient Intelligence environments. A comprehensive investigation of possible dark scenarios in the AmI is done by the SWAMI consortium [28]. We use some examples directly reproduced from their report.

> *Example:* We can imagine, in the future, everyone will have a 'friend-locater' function on, for example, his mobile phone. Now imagine the following situation which could occur to any fictive person:
>
> *"In Munich, I experienced an awkward situation after I located a former colleague of mine using the 'friend-locater' function (LBS) of my PWC.33 I just wanted to say hi, but when I walked up to him, I was surprised to see that he had a good-looking, younger woman with him who obviously was not his wife. He blushed, mumbled a few words and disappeared in the crowd."*

This example shows the difficulties with privacy. Suppose that there are standard privacy policies which state that your location may only be used by friends. In the above example, this policy would not been strict enough, because there are situations in which you don't want to be found by friends. To prevent those kind of privacy violations as showed in the example, you need more control over your data. The degree of information disclosure depends on person, context and situation [28].

> *Example:* In an AmI, a lot of services will become available to people, based on locations, activities, health information, et cetera. To make those services possible, privacy sensitive information must be supplied to those service. People may become dependent of those services, making it hard to switch back if you want more privacy, which is showed by the following (fictive) scenario:
>
> *"I began to make a point of switching off my AmI sensors in public places so that my preferences could not be revealed and monitored. I'll leave them on in the home where I control the environment, or at work where there are confidentiality clauses protecting us but the moment I step outside I switch them off. A cumbersome procedure, a real hassle, but it can be done. The downside is that I now find myself missing out on announcements including the emergencies or special promotions that I would have liked to take advantage of. Recently, I was in the airport where I actually found that I was banned from the frequent flyers' lounge because their sensors objected to my sensors opting out! Even though I showed them my card, they still wouldn't let me in. Can you believe that? Why should I be denied entry? Now I can see that if I have my AmI sensors off at the airport, there's a distinct risk that I'll be stopped and searched and maybe miss my flight."*

Above example suggest that without being monitored you will not be able to participate in any services. In this example, the person is concerned about his privacy and therefor he decides to stop donating information. If the person had full control over his privacy, he could decide which services he needed and which not, and be able to make a trade off between services and privacy.

1.3 Requirements upon privacy in the Ambient Intelligence

In the previous section we showed different kinds of (traditional) situations in which privacy violations could occur. In an environment in which sensors are collecting data from donors, it is perhaps even harder to adopt the right of individual donors to determine to what extent information about them is collected, stored and made useful to applications. A (database) system that wants to be aware of such a privacy definition, must have some mechanism to provide *limited retention* of data from donors. Limited retention means that the donor's data only remains in the system as long as the user wants the data to be in the system. Limited retention techniques are already applied in Hippocratic databases [2] by means of setting a date specifying when the data must be removed from the system. This principle is also adopted by the P3P project. More about these techniques in section 2.

A problem arises if we not only consider the need for privacy for the donor, but also the *usefulness* of the context data in order to allow applications to become as *smart* as possible. Although we want the donor to have control over his privacy, we also want the donor to hand over as much information as possible, *independent* of the purposes of the applications. Hence, we want to find a suitable *trade off* between privacy of the donor, and smartness of applications.

1.4 Research context

In the preceding sections so far, we showed some problems and requirements for the Ambient Intelligence. In our research, we will focus mainly on the trade off between smartness and privacy. One way to accomplish the privacy and smartness requirements of the Ambient Intelligence, is not to remove all data at once after a single retention period, but *degrade* the data in several steps after *several* retention periods, with a possible final degradation step leading to actual removal of the data. Moreover, in stead of retention periods we can imagine that data could be degraded after the occurrence of an *event* (e.g., leaving the building). This degradation steps specify the *life cycle* of the data. This leads us to the main problem which this thesis will focus on:

> *For each acquired piece of data, how to allow the donor to specify a policy, that specifies the life cycle of that data and how to implement this in an ambient environment by means of a privacy aware context database.*

The advantage of being able to specifying the life cycle of context data can best be illustrated with an example. Imagine a web shop where customers can order goods which will be delivered at the address specified by the customer. For this service, the customer (the *donor*) needs to hand over his privacy sensitive address information. After the delivery, this precise address is not needed anymore for delivery purpose, but a part of the address (for example the city) can still be useful for global marketing purposes. Moreover, the ID of the customer is not needed anymore and can therefor be degraded to only 'male' (or 'female'). In exchange of this information (which is much less privacy sensitive than the accurate address), the web shop can offer additional services (for example a discount on the next order). Both parties are now happy: the customer is more willing to give his privacy sensitive information, and the web shop is still capable of offering its services.

The introduction of life cycle policies will certainly have an impact on how data will be stored in a database. This data storage could be quite different from traditional database systems, which implies that access to this data will also be different. To provide a certain kind of transparency to applications which will use the data, we want to specify a schema of the database, with the relations in it as they would appear in a traditional database (for example, a relation *person* with attributes ($identifier$, *time*, *context*)). Queries executed on this relations must then be translated to queries on the actual data in the database. More specificly:

> *How to allow an application to query the data, without knowing the actual data structure.*

1.5 Scope of this study

Although tackling above problems will provide privacy control for the donors, some privacy principles are left out of consideration. This includes for example the principle of *limited disclosure*. There are many (traditional) techniques which could be applied to prevent illegitimate access to the database, all based on *access control* [18]. This is to prevent regular attacks on the database, but doesn't prevent human insiders (such as the database administrator) to simply disclose the data. If data is degraded according to privacy policies, such disclosure will not lead to privacy violation. Also principles like *limited use* will not be considered in this thesis. The limited use principles states that certain peaces of data should not be accessed more than a predefined number. Although data is degraded to a less accurate level, this degraded data could still be useful for some services and therefor increasing the overall usefulness of the data, making the Ambient Intelligence 'smarter'. Restricting the number of accesses to this data is in this context not an issue.

Finally, in this research we do not consider attacks on the database which could *alter* the data stored in the database. In theory, it is possible to replace the life cycle policies with less stricter ones, and so possibly violating privacy. This is specified by the *safety* principle of Hippocratic databases.

1.6 Performance and implementation requirements

Preserving privacy to the donors of data has our main concern. Maintaining some performance of the database (for querying, inserting, updating the data, ACID properties, et cetera) is also important if we want to make our approach feasible. A context database (and especially our privacy preserving context database) will have normally three tasks:

Inserting new context data, for example locations, mostly from sensors or other inputs connected to the database. The context database must be fast enough to handle incoming locations, so that no information is lost due to a denial of service of the context database.

Updating context data to comply with the limited retention principle. The context database must be able to handle the privacy policies specified by the donors at all time, with eventually an acceptable delay.

Querying the context data must be possible, so that application can use the context data and privacy policies are not offended.

After implanting life cycle policies, the context database must still be able to perform those task sufficiently.

1.7 Organization of the thesis

This thesis is organized as follows. We start with an overview of related work on the topic of privacy protecting, and some work from the ongoing ambient intelligence research. In Chapter 3 we give a detailed description of our approach by presenting the *life cycle policy* model. In this Chapter we will give a formalization of our model, some motivating examples and problems with our approach. In Chapter 4 we show the *feasibility* of our approach by presenting a performance and implementation study. We build a small prototype of LCP-enabled databased on top of a traditional relational database. We finalize with a discussion about open main issues and future work, followed by a conclusion.

Chapter 2

Related work

> *"Privacy is an Interaction, in which the information rights of different parties collide. The issue is of control over information flow by parties that have different preferences over 'information permeability'." – Eli Noam*

2.1 Related work based on access control

In this section, we will give a brief overview about the traditional work based on preserving privacy with the use of *access control*. Throughout this thesis, we will use the term *access control* to refer to this traditional techniques. Although we call these techniques 'traditional', still a lot of research is done to improve the techniques or to find new access control based methods.

2.1.1 P4P

The P4P framework returns the responsibility of maintaining privacy to the 'people'. Instead of providing information such as email addresses, an identifier to that information (which is maintained by a personal *agent*) is sent to the website/shop/whatsoever who asked for it. If the web shop wants to use the email address, it sends a request to the agent, which could forward the request to the owner, or reply according to a specified policy. The agent can determine if the request is valid, or, for example, an identifier is shared with other companies and therefor a limited disclosure property is violated. Although the P4P framework returns privacy control to the donors who supply the information, for several practical reasons it is very hard to adopt it in an ambient environment. The P4P architecture is client-server, or even client-client based, and therefor not centralized. If privacy sensitive data is needed from several persons, each client of the person has to agree with the data request. It isn't hard to see that such architecture is not sufficient for statistical surveys, or learning algorithms which need fast access to large data sets.

2.1.2 Encryption

One traditional approach for preserving privacy, is to store the data encrypted, in such a way that most of the querying could be done at the server (without decrypting the data) [14]. Encrypting is done by encrypting the tuple (represented as a string, with the use of standard

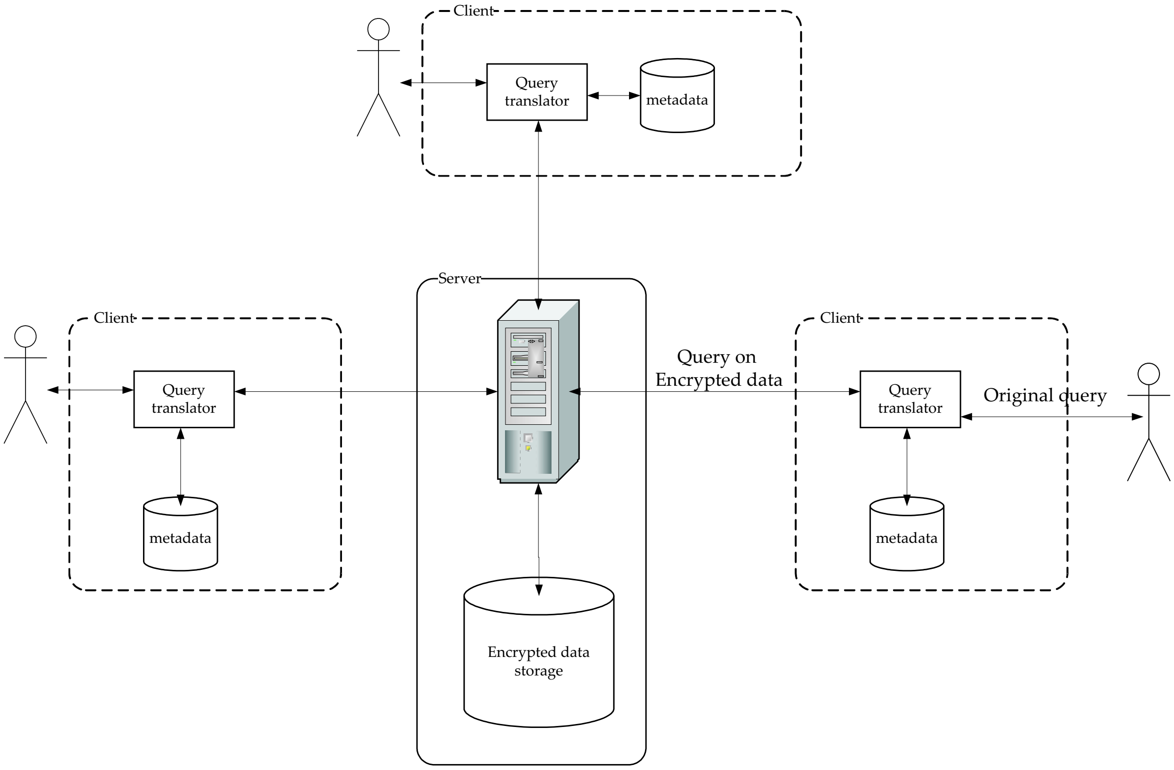


Figure 2.1: *Simplified architecture of a database storing encrypted data and a set of clients*

encryption techniques), and map the attributes of the tuple into non-overlapping 'buckets'. Only a client can decrypt the data, so privacy can be controlled by those who controls the client (which could be delegated to an agent of the donor of the data).

The architecture of such system consists of three fundamental entities: a *user*, a *client* and the *server*. The client encrypts data and stores this encrypted data on the server, and maintains *metadata* to be able to *translate* queries from the user into queries which can be executed over encrypted data. Result sets returned from the server will be post-processed by the client and returned to the user. A more comprehensive explanation can be found in the work of Hacigümüş [14].

As for the P4P approach, this approach is only useful when only data must be accessed from one client. In an environment with many users and queries which include data of many persons, this approach is not possible.

2.1.3 k-Anonymity

An other kind of privacy violation which we not mentioned before, but which has occurred many times in history, is the combining of two public databases, which on first glance does not contain data which could be tracked down to individuals. Combining of public databases could result in a privacy violation because of the possible presence of a shared *quasi-identifier*. A zip-code in combination with a birthday is one example of such a identifier. *k*-Anonymity algorithms could solve these kinds of problems [31].

Race	Year of Birth	Gender	ZIP	Problem
Black	1965	m	02141	short breath
Black	1965	m	02141	chest pain
Black	1964	f	02138	obesity
Black	1964	f	02138	chest pain
White	1964	m	02138	chest pain
White	1964	m	02138	obesity
White	1964	m	02138	short breath

Table 2.1: *Example of k-Anonymity, with k* = 2 *and the quasi identifier = (race, Birth, Gender, ZIP)*

The basic idea of *k*-Anonymity algorithms is to create an anonymized, public copy of the database. In this public copy, data is manipulated such that the data is not privacy sensitive anymore. A *k*-anonymized databases guarantees that a certain piece of privacy sensitive data can only be resolved to a possible set of *k* different persons. An example (from [30]) is given in Table 2.1. In this example, the tuples sharing the same quasi identifier ($QI = \{Race, Birth, Gender, ZIP\}$) have at least $k = 2$ occurrences.

2.1.4 Hippocratic databases

In 2002, Agrawal et al. provided a new vision about preserving privacy by means of a *hippocratic database* [2]. They provide a set of privacy principles, which should be adopted by a database which stores privacy sensitive data. Main focus is to return the control over privacy to the one who provides the data (in context of hippocratic databases called *donors*), because it is believed that ambient intelligence only will be accepted when donors don't have to be scared that all there actions, behavior and goings could be misused by the wrong people [10].

Hippocratic databases must comply with ten founding principles, which mostly refer to commonly used privacy statements of governments. We give a short overview of some of the ten principles mentioned. The ten principles are: *purpose specification, consent, limited collection, limited use, limited disclosure, limited retention, accuracy, safety, openness and compliance*.

Purpose specification simply states that the purposes for which applications are allowed to access the data must be attached to the data, which will have *consent* of the donor. *Limited collection* implies that the stored data is the minimal set of data needed to accomplish the specified purposes, and that unnecessary data will not be stored. The *limited use* principle states that only those queries which are consistent with the purposes could be executed. *Limited disclosure* will say that data could not been accessed or communicated outside the database if there is no consent of the donor. Data should only remain in the database until the purposes are fulfilled, which is specified by the *limited retention* principle.

The final principles to adopt are *openness* and *compliance*. Openness will say that donors must be able to access all information about them in the database. Finally, a donor should be able to verify *compliance* of the database with all ten principles.

2.1.5 Fine-grained access control

In recent privacy related research, the interest in fine-grained access control is more and more increasing. Recent work [6, 5] about this topic shows a *view-based approach* to accomplish fine-grained access control. With fine-grained access control, donors should be able to define which parts of data with which accuracy can be disclosed to whom with what purpose. According to these parameters, *views* of the data are made public to applications which could query those views. Data generalization techniques are used to decrease the accuracy of a value, but keeping the semantics of the data consistent.

The privacy policies are stored as *meta-data* together with copies of all levels of generalized values of the privacy sensitive data. When a query is submitted to the database, the data set on which the query is executed will be constructed according to this meta-data. The same queries, submitted by different applications, can therefor return different results.

Problems with this approach are the great amount of storage needed for all copies with different accuracy, the question on how to specify the policies, and how to generalize the data.

2.1.6 Platform for Privacy Preferences (P3P)

P3P is a standard, developed by the World Wide Web Consortium (W3C), which main purpose it is to make it possible for visitors of websites to have more control about the data provided to websites. P3P is a language which could be interpreted by (for example) browsers, making it possible to compare automatically donors' preferences and the specified policy.

In the most basic form, a P3P policy answers a set of multiple-choice questions covering a broad range of privacy principles. An example of a P3P policy is given in Appendix B. With this policy, a website could ask their visitors if they are allowed to set a *cookie*. With a cookie, it is possible to track a visitor, and to store some information for later purposes. The purposes are described in the policy, as is the retention time. More specific, in the example policy, the retention period is 'until the purposes are met'.

This example shows directly the weakness of P3P. The retention period is very ambiguous: when is the data not needed anymore to fulfill the purposes (according to the policy these are: *develop*, *pseudo-analysis*, *pseudo-decision*, *individual-analysis*, *individual-decision* and *tailoring*)? Indeed, when presented such policy, a visitor (e.g. the donor) can refuse the policy. But, with more some flexibility and more specific information, perhaps the donor *would* have accepted the policy.

P3P policies are a good way to make the privacy measures of companies more transparent and readable to their customers. The policies do not contain any enforcement mechanism to make sure companies really try to not violate their own policies.

Compared to P3P, the LCP model (which we will discuss in this thesis) shows some similarities. With P3P, it is possible to specify a retention time, after which data must be removed from the system. Our approach however will be more flexible: it will be possible to specify intermediate steps, letting the system remove *parts* of the data and/or change the accuracy of the data. However, with some changes, it could be possible that the P3P standard will make this kind of policies possible. Another difference is the *point of view* of the P3P standard compared with our LCP approach. P3P policies are initially specified by companies which offer a certain kind of service. A user can decide wetter or not he accepts the policy. With our LCP approach, we have a broader view: we let the donor specify a general policy which must be applied on his data. When specifying his policy, the donor must keep in mind which kind of applications could use his data, and how much he wants to give up his privacy in exchange for more possible services. This is more in the spirit of ubiquitous computing and the Ambient Intelligence which will discuss in the next section.

2.1.7 Privacy-Preserving Data Mining

Data mining is all about finding non-obvious information in large data sets. Those data sets could contain privacy sensitive information of individuals. For most data-mining applications, values in individual tuples do not have much interest, only statistical functions (aggregate functions like sum, avg, etcetera) are useful. If the individual values could be altered in such way that estimating the original value is nearly impossible, without having a (or only a little) affect on the statistical calculations, privacy could be preserved without losing functionality.

Agrawal and Srikant [3] describe methods to add a randomized value (from a uniform or gaussian distribution of values) to the original value, and/or divide the values into distinct non-overlapping classes. They describe algorithms to recover the original dataset, and show that this could be done with high accuracy and different gradations of privacy protection.

2.2 Ubiquitous computing and the Ambient Intelligence

In the following section, we will discuss ubiquitous computing and the ambient intelligence with descriptions of privacy threads and observations found in related work.

2.2.1 Ubiquitous computing

With the introduction of *ubiquitous computing*, new privacy related issues arose. One of the main difficulties with privacy in the ubiquitous computing, is the way how data is collected. When making a transaction with a webshop, it could be quite clear which kind of data is exchanged. Ubiquitous computing techniques however, such as small sensors or cameras equiped with powerful image recognizing algorithms, often collect data when people are not aware of it [19]. In that case it is possible that people *think* they are in a closed private area (such as coffee rooms), but in *reality* they could be monitored by sensors in that room where they not aware about. Moreover, due to increasing storage capabilities and capacity, it is easier to keep data longer, enabling more data mining possibilities, with the danger of making wrong interpretations of data because the data is far out of context.

Ubiquitous computing leads to an effect named *asymmetric information*, a term from economics which state that one side from a transaction has less information about the transaction and the information involved than the other side. In the context of ubiquitous computing, this will say that the *owner* of the data (the *donor*) has less information than the *collector* of the data [19]. Xiaodong et al gave an example which show the effect of asymmetric information:

> *Example:* Imagine a situation in which a neighbor makes loud music at 3 AM in the morning. In normal social environments, the neighbors will not tolerate this and will consider (social) sanctions. Loud music can be easily detected, making it possible to take action upon the violations of social norms.

With ubiquitous comping, it is more difficult to detect if, how, and why your privacy is violated. When violations are discovered late, it makes it harder to take action, which is a negative effect caused by the *asymmetric information* principle.

Xiaodong et al state that the presence of asymmetric information is the heart of the information privacy problem in ubiquitous computing. In environments with significant asymmetry between the information knowledge of *donor* and *collector/user*, negative side effects as privacy violations are much harder to overcome. Based on these observations, Xiadong derived the following principle called *the Principle of Minimum Asymmetry*:

A privacy-aware system should minimize the asymmetry of information between data owners and data collectors and data users by:

- *Decreasing the flow of information from data owners between data collectors and users*
- *Increasing the flow of information from data collectors and users back to data owners*

Also Langheinrich [20] specified in his work four properties of ubiquitous computing which make ubiquitous computing different from previous privacy threads. Shortly summarized they are:

Ubiquity: the goal of ubiquitous computing is to be *everywhere*, affecting large parts of people's life

Invisibility: sensors disappear from our view, making it hard to know when we are monitored or not

Sensing: due to increasing technical performance, sensing abilities are improved making it possible to sense even emotions and actions, et cetera.

Memory amplification: with increasing sensing capabilities, it is feasible that it will be even possible to build a record of people, allowing browsing in a fairly complete history, acting like a memory amplifier.

A concrete example of one form of ubiquitous computing is location tracking. Görlach et al made an survey about privacy threads in this specific range of ubiquitous computing [13]. With new and some older technologies like GPS in combination with PDA's, *active badges*, signal strengths in wireless LAN and triangulation measurements with cell phone networks, RFID chips, et cetera, it is nowadays not difficult to build tracking systems which could continuously monitor peoples' locations. Location information could be very privacy sensitive, for example, one's religion could simply be derived by knowing to which church he goes [13].

Görlach et al specified three kinds of privacy threads: by *first-hand communication*, by *second-hand communication* and by *observation*. With first-hand communication, an attacker makes use of vulnerabilities of a device and breaks into it, or because of the specification of device itself, some information can directly obtained from the device. With second-hand communication, information which is not longer under control of the owner is communicated to an unauthorized party. This happens when, for example, a web shop sells his customer information to third parties. The last privacy thread is caused by *observation*. With observation, an attacker makes observations himself with the use of camera's our other imaging devices.

Chapter 3

The Life Cycle Policy model

au · tom · a · ton

Latin, self-operating machine, from Greek, from neuter of automatos, self-acting

3.1 The Ambient Intelligence environment

3.1.1 Architecture description

As shortly mentioned in the introduction, in an ambient intelligence (AmI) people are sensed by sensors and therefor become *donors* of context data. In our vision, donors should be able to choose or specify their own policies, describing the *life cycle* of their sensed, privacy sensitive data. Those policies could then be bound to the data so they can be processed by the context database, but possibly also by other components in the ambient environment such as data caches of applications. In an environment in which applications query the database, it is likely that applications will cache the data for more performance. To make it possible that data, although not longer managed by the context database, could be degraded compliant with the specified policy, those policies must always stay bound to the data. How to *enforce* that applications or other components in the AmI apply the policies is a huge challenge.

Figure 3.1 shows a *possible* architecture of an AmI-space. In this architecture we suppose that there is one centralized context database storing all donors' sensed data per AmI-space. An AmI-space is one collection of connected sensors, applications and possible other components needed in an ambient intelligence belonging to one environment (e.g., a building). There are several AmI-spaces which may or may not overlap each other. If donors move to an other AmI-space, it must be possible to transport the corresponding data (and the corresponding policies) to the new AmI-space. This leads to distribution problems which are further discussed in section 3.8.

Instead of a centralized database we could also use an decentralized approach where context data is stored at devices of donors. This architecture is proposed by Aggarwal et al. in the P4P 'privacy for the paranoids' vision project [1]. However, from an application point of view a centralized approach is better for limiting the number of interactions with donors, which in an ubiquitous environment is desirable. Also in terms of performance is a centralized approach more likable.

Finally, a policy translator is placed before the context database to translate the policies to, for example, SQL language.

In this thesis we will focus mainly on the context database, although the LCP-model we propose should also be applicable to other components. We will present an evaluation of an implementation of the LCP-model on top of a traditional Postgres relational database management system in section 4.

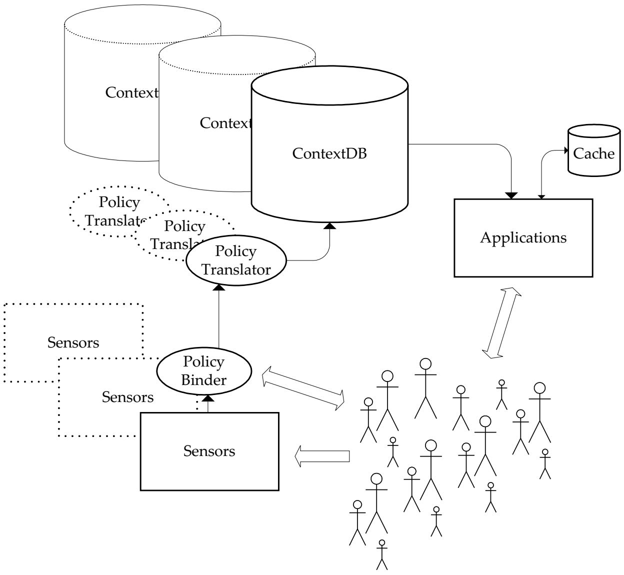


Figure 3.1: Possible general architecture of an AmI-space

3.1.2 Events

Before presenting a model for describing life cycle policies, we first make a distinction between different types of *events* which may occur. We will see that different types of events imply different kinds of problems. The three types of events are *external events*, *internal events* and *universal events*:

internal events are events which can be monitored within the boundaries of the current AmI-space. Those events are normally triggered by actions of donors.

external events are events which appear *outside* the current AmI-space.

universal events are events which can be monitored always and everywhere, with the nice property that there is no donor-specific context data needed to monitor such events. One example of universal events are *time events*.

Examples of internal and external events are events like 'I left the building', or 'some person is in the coffee room'. To be able to know when such an event has occurred, context data of the donors which are subject of the event must be available in the AmI-space, which is normally the case. However, if someone is within an AmI-space and specifies an event in his policy which can only be sensed in an other AmI-space (*external* events), a problem arises. More about this in section 3.8.

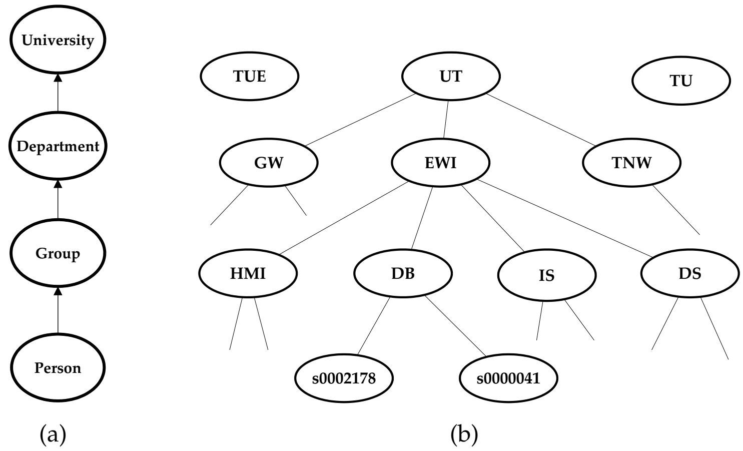


Figure 3.2: *Concept hierarchies of person, (a) shows the domain generalization, (b) shows possible instances*

3.2 Formalization

3.2.1 Context states and domain generalizations

Information sensed by the Ambient Intelligence can be presented by a triplet (*time,id,context*), where *context* represents context data like location, temperature, activity, etcetera. Each element of an instance of such a triplet can have a different level of *accuracy*. For example, one instance of the triplet (*time, id, location*) could be the triplet (*2005-12-06 12:15, 2178, Zilverling*), having accuracy *minute*, *id* and *building* respectively. A triplet containing the accuracy-levels of data corresponding to that triplet is called a *context state*. Each element of a context state is called a *dimension* of the context state. This term will get more body in the next section.

The representation of data with different levels of accuracy is also known as *data generalization* [15], which is applied in many traditional database systems. Figure 3.2 and Figure 3.3 show two *concept hierarchies* of the person and location dimensions. We assume that the knowledge needed to make a generalization step is contained in the ambient intelligence itself.

3.2.2 Cubical representation of context states

The complete set of possible context states can be represented by a *cube* (see Figure 3.4). This cube consists of three dimensions, with the first two axes representing the *time* and *id* dimensions. The third axis represents the *context* dimension. Throughout the thesis we will use *location* as the context dimension. The axes are divided into respectively n_t,n_{id} and n_l distinct regions, with each region $n+1$ representing a less accurate value than region n (we say that the *granularity* of the dimension is n). This will divide the cube into ($n_t \times n_{id} \times n_l$) context states representing different levels of accuracy. Each context state can now be identified with the triplet (t,id,l), with $0 < t \leq n_t$, $0 < id \leq n_{id}$ and $0 < l \leq n_l$.

> *Example:* The context state $S = (4, 1, 1)$ denotes the context state (*day,GUID,coordinates*) in the cube of Figure 3.4. A possible data triplet $t \in S$ could be (*2005-12-06,s0002178,(11,51)*).

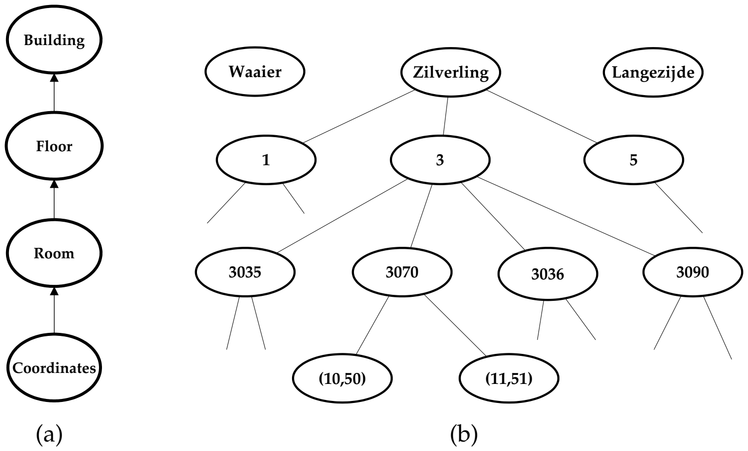


Figure 3.3: *Concept hierarchies of location*

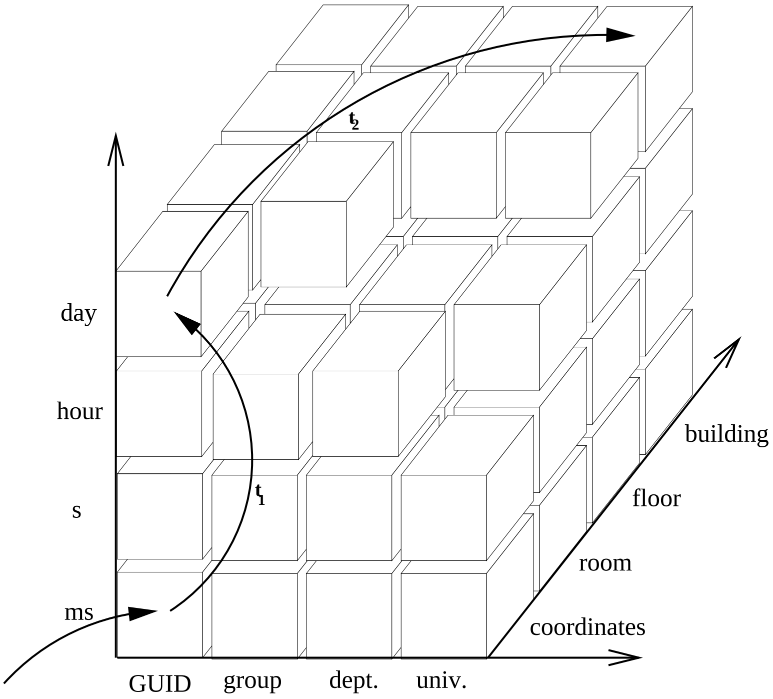


Figure 3.4: *A Cube with 3 dimensions and* 4 × 4 × 4 *sub-cubes. The arrows indicate the steps of a simple life cycle policy described in section 3.2.3*

Note that the cubic representation of the data is already used in, for example, data warehouses to represent the result of a query [7]. The main difference with our representation is that each dimension takes different data accuracies linked to a given domain of values, ordered from the more accurate (e.g., exact coordinates for location) to the less accurate (e.g., building), instead of representing an ordered set of discrete values (e.g., years) or interval (e.g., age between 0-10) having the same accuracy.

3.2.3 The LCP model

In this section we present a way to model Life Cycle Policies (LCPs). A LCP must have two main properties:

1. it must specify *how* data must be degraded
2. it must specify *when* data must be degraded

A LCP species when data must be degraded to which accuracy. We propose to implement a LCP as a set of context states, combined with descriptions about how and when a context state is reached. A step in the policy means normally (at least in an one-way policy [1]) degradation of data (decreasing the accuracy of data corresponding to the policy). More specific, a step in a LCP is defined as a transition from one context state to another context state. We say that state S is more accurate than state S', if at least one of the three dimension of S' has a lesser accuracy than state S, denoted as $S \trianglerighteq S'$:

$$S_i \longrightarrow S_i \ with \ S_j \trianglerighteq S_i$$

A transition may only occur when an *event* happens. Different types of events have already been described in section 3.1.2. We now first present an example of how to model a LCP with the use of a *deterministic finite automaton* [29]:

> *Example:* The following LCP specifies that at construction time, data (from a sensor) is stored in the most accurate form: time in milliseconds, a personal identification number and the exact coordinates of his position. After 10 minutes, at time t_1, the data is degradated to a less accurate form: now only the time in hours will be kept in the database. Again, after 1 day (t_2), the data will be degradated to another level. Now only the building where some person from a certain university was in a certain hour could be derived from the system.
>
> $S = \{s_0, s_1, s_f\} = \{(1,1,1),(4,1,1),(4,4,4)\}$
> $\Sigma = \{t_1, t_2\} = \{$ *after 10 minutes, after 1 day* $\}$
> $\delta(s_0, t_1) = s_1$
> $\delta(s_0, t_2) = s_f$ (*)
> $\delta(s_1, t_1) = s_1$ (*)
> $\delta(s_1, t_2) = s_f$
>
> *(*) Note that those transitions are only meant to make the automaton fully deterministic. However, from the nature of absolute time values we are sure that those transitions will never take place, and therefor can be omitted.*

The above LCP is one particular instance of an automaton. We now define a LCP as:

$$LCP = \left(S, \Sigma, \delta, s_0, s_f\right)$$

S is a set of context states written as a triplet (t, id, l), Σ a set of events, δ a set of *transition functions* $S \times \Sigma \rightarrow S$, s_0 the *start state* (also called the *construction state*) and s_f the *final*

[1] More about this property in section 3.2.4

state of the degradation process, where $s_f = \{\emptyset\}$ indicates removal of the context data from the system.

An automaton can be represented by a *labeled directed acyclic graph* [29]. The nodes of such graph are elements of the set of states Q, the labels are elements of Σ, in such way that an arc from s_i to s_j is labeled a if $s_j = \delta(s_i, a)$. A *DAG* of the LCP of above example is given in Figure 3.5. The arrows in Figure 3.4 of the previous section illustrate the path in the Cube corresponding to this LCP.

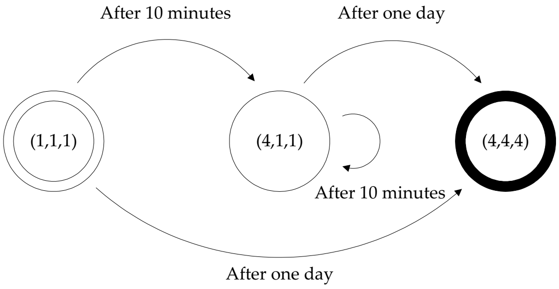


Figure 3.5: *Example of a degradation policy noted as a DFA*

3.2.4 The one-way property

In this thesis, we will only consider LCPs which hold the *one-way property*. The one-property has a syntactically meaning and a semantical one. The syntactical form states that it not possible to specify transition functions which not specify a degradation of data. From a once degraded value, the previous, more accurate values never can be derived, or stated in terms of the automaton:

$$\forall (s, e \rightarrow s') \in \delta : s \trianglerighteq s', \ \textit{with}\ e \in \Sigma$$

The semantical meaning of the one-way property is that an implementation of a LCP must respect the one-way property and may not violate it, thus a once degraded value may never return to its original state. In the following sections we will investigate the difficulties of realizing this non-violation. We will see that, among other problems, inference problems must be solved (e.g., inference by closely looking to the policies).

3.3 Motivating examples

In this section we present some motivating examples, proving the usefulness of our model. We present two different types of examples. First we describe an organization-oriented policy, a general policy specified by an organization and shared by all members of that organization. Second we present user-oriented policies, policies which are not shared by others unless two similar policies are specified by coincidence .

The examples given in this section will be used in the next section to investigate possible one-way property violations. We will see that especially user oriented policies, but also organization oriented policies, possibly lead to several violations of this property.

3.3.1 Organization-oriented policies

To achieve its privacy goal, an organization has to minimize the available (retained) data within its own information system. One reason to do this from an organization point of view is to protect against potential spying from other organizations. By restricting the life cycle of the sensed data, the risk of leaking privacy sensitive information could be decreased. By using life cycle policies, an organization can parameterize its own information system to only retain context information which is strictly required in providing the services still needed by the organization (and so making a compromise between privacy and smartness of services).

> *Example:* Although a company wants to have good privacy regulations, the company could still require the following services:
>
> 1. phone call redirection
> 2. automatic filling-in daily timetable forms
> 3. room availability forecasting for the next week
> 4. statistics in terms of visibility of different teams (for example, the number of days per week a team is represented by one of its members in the organization)

To provide the services of the above example with privacy in mind, a LCP (pictured in Figure 3.6) could regulate employees' location information acquired by the AmI-space. Following this LCP, the context databases keeps accurate location states (employee ID, precise acquisition time, and room-identifier) for a few minutes. This short history of still accurate data is needed for making phone call redirections possible. After a few minutes, the data is not considered precise enough (for this service only a short history is useful), and therefor the data could be degraded to hour of acquisition. This accuracy level is enough to allow automatic fill-in of daily timetables. One day later, the employee's ID is degraded to team identifier, still enabling room availability forecast for the next week, and finally, one week later, the room-identifier is deleted. Now it is still possible to generate statistics about what day the chance of meeting *someone* of a certain team is highest.

The last context state is considered as non dangerous for the organization's privacy, and can thus be durably kept in the system to enable further statistic computations and long term historical analysis. Although this LCP is shared by all the employees of the company, it reduces the amount of context information available in the AmI-space, which could be accessible in case of a spying attack. Also the employees themselves will feel more comfortable about being monitored, knowing that their goings and behavior could not be misused, thanks to the privacy policies of their organization.

3.3.2 User-oriented policies

The primary goal of this research is to increase the available context information to make an application smarter, by giving full privacy control to donors of data. Normally, when a donor

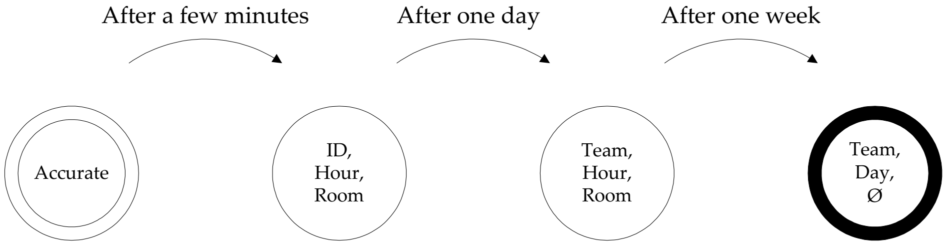


Figure 3.6: Example of an organization oriented policy

wants to protect his privacy, he simply objects being monitored, or wants his sensed data to be removed immediately. However, services available in the AmI-space could require more context information, which could include privacy sensitive data. For a service to be useful and to become available for the donor, it can notify a donor and ask for privacy sensitive data in exchange of the service itself. Donors can than accept or reject this offer, or even negotiate leading to a LCP which is acceptable for both donor and service. A resulting LCP can be very rich with much intermediate states, and be different for each donor.

> *Example:* Imagine a traffic environment with donors being drivers of a car. The AmI-space consists of several possible services, including:
>
> 1. a personalized road planner, including traffic jam warnings and directions
> 2. a carpool service, based on localization of colleagues
> 3. a general carpool service, with predictions of best places for being given a lift
> 4. a general statistic collection, enabling governmental organizations to efficiently plan work on the road, traffic jams, etcetera.

A particular donor could consent to the LCP shown in Figure 3.7. Sensed data is stored in the most accurate form, enabling precise calculation of position, speed and movement direction. After a few minutes, this accurate history is not needed anymore, and could be degraded. Now, the current road and time in minutes is available, making it possible for colleagues to predict carpooling options. This data is stored for one hour, assuming that the user waits for (a maximum of) one hour to get his car filled before moving on. After one hour, the personal ID is degraded to *type of car*. Now it is not possible anymore to track the movements of the donor, but it is still possible to make general one-week-in-advance carpool predictions, knowing which kind of cars normally are waiting on which road. After a week, also the type of the car is degraded (actually removed), keeping the data for gathering statistical information about road usage. Finally, after one month the data is removed from the system.

Note that there are many services, needing as much data as possible from as many donors as possible to become as smart as possible. Data of *one* particular donor is not necessarily required for a service to give for example traffic jam information to *that* donor. However, we

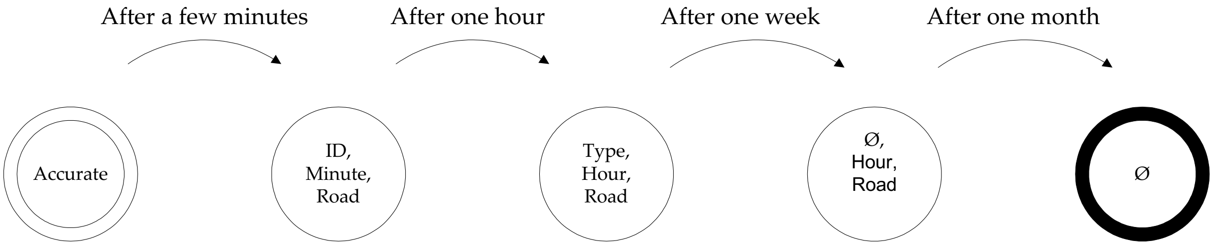


Figure 3.7: Example of an user oriented policy

assume that services will work with the commonly used principle that you can only make use of data from others if you are yourself also willing to share your context data.

3.4 Possible one-way property violations

3.4.1 Unexpected information disclosure

If a policy is not chosen carefully, the policy itself might disclose privacy sensitive information. This problem is inherited from the one-way property itself. Assume we take a snapshot of a context database, containing all tuples of a given donor with their corresponding LCP (obviously, this is only possible when the ID is not already degraded). The LCP of a particular tuple in the snapshot consists of n states $\{S_1, S_2, \ldots, S_n\}$, with $S_i \trianglerighteq S_{i+1}$ and $n-1$ *delays* $\{d_1, d_2, \ldots, d_{n-1}\}$ specifying when a transition from state S_i to state S_{i+1} takes place. Let t_i be a tuple in the snapshot, element of the set tuples corresponding to state S_i ($t_i \in S_i$). The following property $\forall t_j \in S_{i+1}, \bullet t_j.Time \leq t_i.Time$, states that an already degraded tuple is always inserted *before* a tuple with a higher accuracy. Knowing this, it is possible to *refine* information more than should be possible according to the LCP.

We illustrate this problem with an example, as pictured in Figure 3.8.

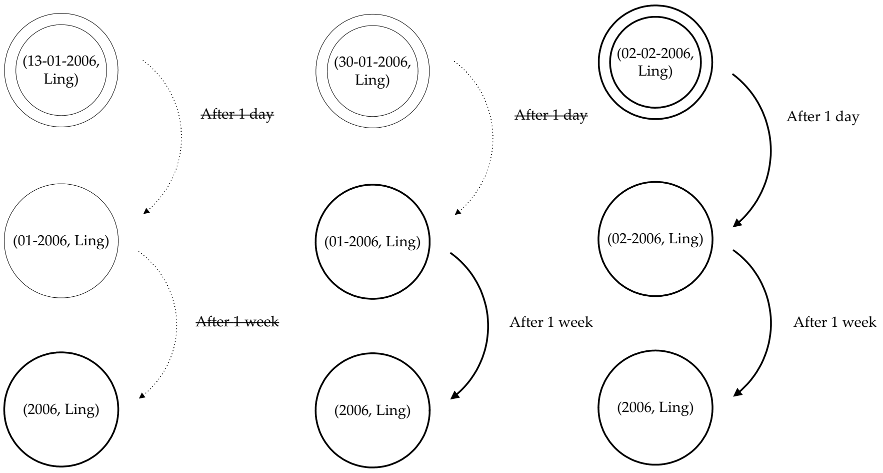


Figure 3.8: Demonstration of (unexpected) information disclosure

Example: The tuple that currently is most accurate has been inserted at the 2nd of February. Previous inserted tuples have already been degraded in such way that one can only derive the fact that the tuple was inserted in the month 'January'. More specificly, the policy specifies that the correct insertion date only can be derived with a chance of $\frac{1}{31}$. Because of the existence of the most accurate value, we know that the degraded tuple has been inserted somewhere *before* the 2nd of February, but we already knew that because the tuple has been inserted in January. We also know that the second degradation step will occur one week after the insertion of the original tuple. Since the second tuple is not degraded to the final state yet, we know that the tuple can not inserted before February 2nd minus one week. The tuple is thus inserted between 27th of January and 31th January.
Moreover, the least accurate tuple has been degraded to '2005', hoping that the chance to derive the correct month is $\frac{1}{12}$. The actual chance is $\frac{1}{1}$, because we know that the insertion date was before the 11th of the second month of the year, but due to the fact that more degraded values with the same LCP as lesser degraded values are always inserted earlier, also before or in January 2005.

Once you see this 'problem' it it is quite trivial. However, it is interesting to see that many transition times specified by the user must be altered by the system, in order to prevent information disclosure. One method is to *break* the property that a degraded value is always acquired *before* a non-degraded value. We are able to do this by adding a random value from the interval $\left[-\frac{L}{2}, \frac{L}{2}\right]$ to the specified transition time. With increasing or decreasing the transition time compared to transition times of tuples which are acquired later, it could be possible that an tuple which is acquired later is degraded earlier.

3.4.2 Inference

In the previous section we mentioned two different types of policies: organization oriented policies and user oriented policies. In the first case a policy is shared by all members of an organization. In the second case however, policies could be unique for every donor in the AmI-space, leading to a serious inference problem.

If every donor has a different policy, then the policy uniquely identifies the donor, making a policy a possible *quasi identifier* [31] of the context data. To illustrate this, consider the following table. In this example we use the policies like those shown in Figure 3.7. Remember that every donor can specify his own transition times and intermediate states. At first sight

id	Values (ID, Time, Value)	Current state	Remaining LCP
1	(Van, 15-10-2005 11, Street1)	(type,hour,road)	$\{s_3 \rightarrow s_4\}$
2	(car3, 15-10-2005 12:37, Street2)	(id,minute,road)	$\{s_1 \rightarrow s_5 \rightarrow s_6\}$
3	(car2, 15-10-2005 12:36, Street3)	(id,minute,road)	$\{s_1 \rightarrow s_3 \rightarrow s_7\}$
4	(car1, 15-10-2005 12:35, Street4)	(type,hour,road)	$\{s_1 \rightarrow s_3 \rightarrow s_4\}$
...	...	...	...

Table 3.1: Example of revealing privacy sensitive data

nothing happens: four tuples are stored, of which three (tuple 2, 3, and 4) are in an accurate state, and tuple 1 has been degraded. Because tuple 2, 3 and 4 are in an accurate state, the ID is available in the tuple, not violating any policy. The *ID* value of tuple 1 has been degraded to *type*, so it must not be possible anymore to identify the owner of that car. However, if we have a close look to the policies associated with the tuples, we see that the remaining policy of tuple 1 corresponds with the last part of the policy of tuple 4. Assuming that policies of every donor are unique, and that the same policies are used for each acquired tuple of one donor, there is a large chance that tuple 1 belongs to the same donor as tuple 4. In that case, we know that the actual owner of the *Van* was the owner of *car1*, violating the one-way property.

In general, joining on policies can result in a lot of one-way property violation candidates, even if the policies are not all unique. Using the generalization knowledge expressed in the policy, one should be able to decrease the possibilities. For example, this could happen when 10 people share the same policy, but only one of them is owner of a *Van*. Imagine that the following row is stored in the database, along with the rows from Table 3.1:

id	Values (ID, Time, Value)	Current state	Remaining LCP
5	(Truck, 15-10-2005 11, Street5)	(type,hour,road)	$\{s_3 \rightarrow s_4\}$

Table 3.2: Extension of the context data with one additional row

There are now two tuples with the same LCP. Recall that, to be able to degrade to a less accurate value, the generalization knowledge must be contained in the LCP, thus also in that of tuple 4. Suppose that the LCP of tuple 4 specifies that *car1* must be degraded to *Van*. Than we have a good chance that tuple 1 corresponds to tuple 4, and tuple 5 does not.

3.4.3 Database implementation issues

If the data with their corresponding LCPs will be stored in a traditional database (which is likely to happen), the logging process must be implanted carefully, to make sure that accurate values might not be recovered from the log files after degradation. In this section we mention three kinds of issues which must be at least taken into account. First, *undo logs* are usually required to provide transactional atomicity (for example, undo aborted transactions) while allowing a steal cache management policy (report modifications to disk before commit) when long transactions are performed [21]. However, the benefit of an undo log in the context of data degradation is limited, because only two processes write to the database. One is inserting incoming tuples issued from sensors (insertions are performed by the policy translator, see Figure 3.1) and the second is degrading tuples content as expressed by the LCPs. Both processes operate by short transactions involving a few statements, or even just a single statement. Hence, undo logs might be disabled in our settings.

Second, redo logs are required to enforce transaction durability while enabling a no-force cache management policy (i.e., changes operated by a committed transaction do not require to be reported to disk immediately, but when appropriate regarding the load on the disk controller). In our context, final states correspond to non-private data which might be stored durably in the database. Those states can thus be logged in the redo. However, logging only final states (and not intermediate states) leads to a reduce of the smartness of the system in case of media failure, in particular when a very slow degradation process occurs. Instead of using logs to achieve durability, we propose to make use of database redundancy, to duplicate the intermediate states (including their corresponding LCP) in a second database implementing degradation (without logging), the second database being able to recover intermediate states in case of media failure.

Finally, we have to be careful that internal indexing or numbering mechanisms don't lead to possible violations. For example, every row that is created in the commonly used PostgreSQL DBMS gets a unique OID, unless defined otherwise [26]. When those numbers are not chosen randomly, a DBA or someone who has hacked the databases could infer some knowledge from the *order* in which the rows are inserted in the database, knowing that some data precedes others in time, and possibly refine degraded data.

3.5 Complex policies

In our previous examples, we only used transitions based on *time*. With such automata, we know which states will be processed in which order. We know that transition t_i will take place after t_j if $j > i$. Thus, if we only use time values, we have a sequential chain of states, so that if we know the current state, we also know what the next state will be.

With sequential automata and transitions based on time, the *alphabet* of the automaton (Σ) is minimal and is equal to the set of all time values T. As defined before, transitions are functions $\delta : Q \times T \to Q$, with Q the set of states. We can sort the states $s \in Q$ as states $s_i, s_j, \ldots s_n$ with $n = |Q|,\ j > i$ and s_n the *final state*. We say that a state s_i is always processed *before* state s_j, if $j > i$. If we do the same for all transitions $t \in T$, we require that all transition functions hold the following properties[2]:

$$\delta(q_i, t_j) = q_{i+1},\ if\ j = i\ (i < n)$$
$$\delta(q_i, t_j) = q_{j+1},\ if\ j < i\ (j < n)\ (*)$$
$$\delta(q_i, t_j) = q_i,\ if\ j > i\ (*)$$

For a *complex automaton*, above definitions do *not* hold. With complex automata we mean automata which are *not* necessarily sequential, and thus *could* contain *branches*. An example of a complex automaton is given in Figure 3.9. In this example we assume that if event e_1 occurs, e_2 will *not* occur. As a consequence, the automaton is not fully deterministic, but this could easily be solved by introducing an error state or add transitions to a predefined other state (for example to the final state, or to the current state) [29].

Although we now have branches, we could sort the states so that for some states we could say wetter or not that state is processed before or after another state. This is not always true for all states, because we are not sure *if* a state is being reached. For example, if we look at the automaton of Figure 3.9, we cannot say that state $(3, 1, 1)$ is processed *before* or *after* state $(4, 1, 1)$. We only can say that if state $(3, 1, 1)$ is reached, state $(4, 1, 1)$ will *not* be reached.

Complex policies could be well used to expand the possibilities of organization oriented policies. Hence the example of section 3.3.1. This LCP is meant to comply to the needs of the services, and to comply with the minimal security and privacy demands of the company. It is however possible that some individual employees want a more strict degradation schema in some particular cases.

[2] Again, transitions marked with (*) are only meant to make the automaton fully deterministic. For time values, we are sure that those transitions will never take place.

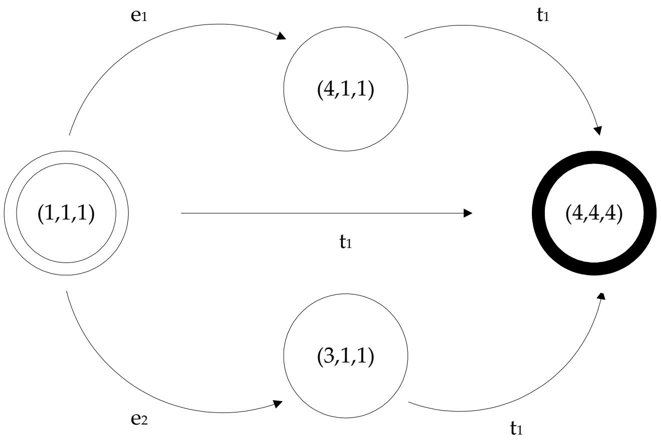


Figure 3.9: *Non-sequential degradation policy*

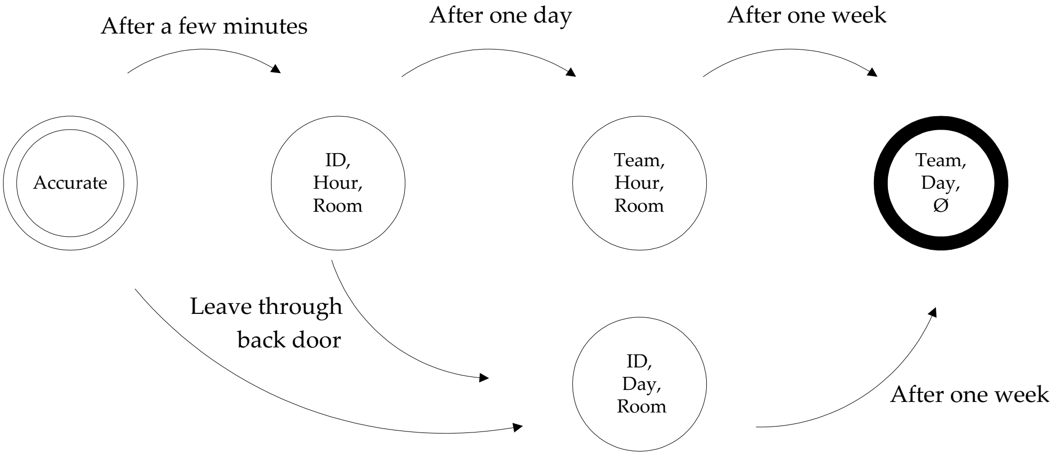


Figure 3.10: Example of an organization oriented policy extended with an user oriented part

> *Example:* The organization oriented LCP of section 3.3.1 is extended with a branch. A employee does not want that, if he departs earlier than 5 p.m. through the *back door*, his employer can see when he actually left. An extra condition is added to the LCP, so that the time field is degraded to *day* instead of *hour*. See Figure 3.10

We can make a conjecture regarding (most) complex policies:

A branch will mostly only occur when the ID field has not yet been degraded. If the ID field has already been degraded, a branch will normally not occur.

This conjecture is true for most policies, because events are normally tightly bound to the donor of the data and the owner of the policy. For example: to monitor that a donor left the building, and therefor degrade the data, the subject of the event (the donor) must be known. It is not likely to have a policy like: "When I (with ID 2781) leave the building, degrade my

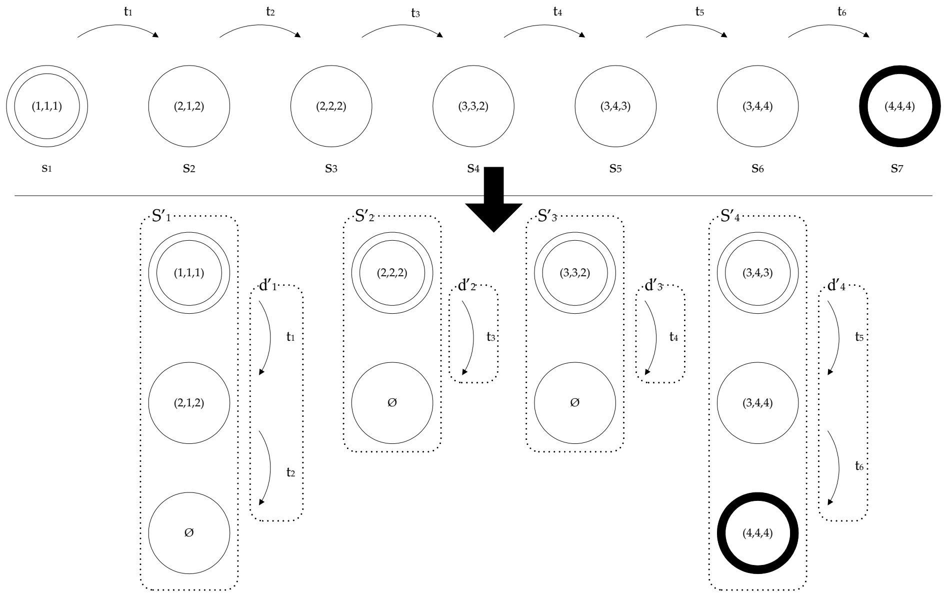


Figure 3.11: *Example of a decomposition of a sample policy. States are triplets (Time, ID, Location)*

data from state (*hour,group,room*) to (*hour,department,building*)". On the other hand, if the ID is not contained in the policy it is impossible to monitor such an event.

3.6 LCP decomposition

In section 3.4.2 we described the inference problem as a possible violation of the one-way property. We also defined 2 different kind of policies: *organization* and *user* oriented policies. The motivating examples we gave were all *sequential* policies, without having branches. In the previous section we showed how to extend the LCPs with branches, making even more richer *complex* policies. We also showed that the inference problem only occurs in the sequential parts of a complex policy, because after the degradation of the *ID* dimension there are no more branches possible.

We saw that the inference problem exists because we can join and match policies. If many donors share the same policy (as is the case with organization-oriented policies), a policy is not an identifier and there is therefor no inference problem. In the case of user-oriented policies it *is* possible that many donors specify a different policy, and therefor the inference problem is a serious problem.

In this section we will present a solution, although limited, to the inference problem. We show that with means of decomposition it is possible to tackle the one-way property violation. We will also show the limitations of decomposition.

The easiest way to explain decomposition is with an example, shown in Figure 3.11. The LCP consists of 7 context states $\{s_1, s_2, \ldots, s_7\}$, with $s_i = (Time, ID, Location)$ and s_7 being the final state. After transition t_2 the *ID* value is degraded. *ID* is further degraded after transitions t_3 and t_4. We therefor can *cut* the LCP in 4 distinct parts: $\{s_1, s_2\}$, $\{s_3\}$,$\{s_4\}$ and $\{s_5, s_6, s_7\}$, creating 4 new policies which can be inserted into the database separately, attached to the same piece of original data, as being four autonomous tuples with their corresponding LCP. As with every policy, the data is inserted with an accuracy as specified in the starting state of the policy. However, the final state of the first decomposed part has a far lesser accuracy than the specified final state of the original policy. To make sure no accurate data stays in the database forever, an additional state is attached to the decomposed parts of the policy, specifying when the data must be removed from the database. This is not needed for the last part of the policy, because the final state in that part is the same as for the original policy. In general, when a original transition specifies that the *ID* value must be degraded, that transition in the decomposed part will now lead to a removal of the complete triplet.

More formally, we can define the notion of decomposition as a function D with the following property: D maps (S, δ) into $\{\{S_1 \cup \{\emptyset\}, \delta_1\}, \{S_2 \cup \{\emptyset\}, \delta_2\}, \ldots, \{S_n, \delta_n\}\}$ where $\{S_1, \ldots, S_n\}$ is a partition of S. For each i, δ_i is the restriction of δ to S_i, extended with one pair $(s, \emptyset)$ for that $s \in S_i$ for which $\delta(s) \notin S_i$. In practice, we choose the partitioning in such a way that all $s \in S_i$ have the same accuracy in the ID dimension.

> *Example:* A decomposed version of the example given in section 3.4.2 is showed in Table 3.3. Although we still can make a join on *remaining policy*, this will not reveal more information because the remaining policy associated to a certain tuple is certainly not part of a policy corresponding to a more accurate value. If it would, than it may because the ID is not degraded at all and thus not violating the one-way property.

id	Values (ID, Time, Value)	Current state	Remaining LCP
1	(Van, 15-10-2005 11, Street1)	(type,hour,road)	$\{s_3 \to s_4\}$
2	(car3, 15-10-2005 12:37, Street2)	(id,minute,road)	$\{s_1 \to s_5 \to s_6\}$
3	(car2, 15-10-2005 12:36, Street3)	(id,minute,road)	$\{s_2 \to \emptyset\}$
4	(car1, 15-10-2005 12:35, Street4)	(type,hour,road)	$\{s_1 \to \emptyset\}$
5	(Van, 15-10-2005 12, Street4)	(type,hour,road)	$\{s_3 \to s_4\}$
6	(Truck, 15-10-2005 11, Street3)	(type,hour,road)	$\{s_3 \to s_4\}$

Table 3.3: Example of a dataset after decomposition

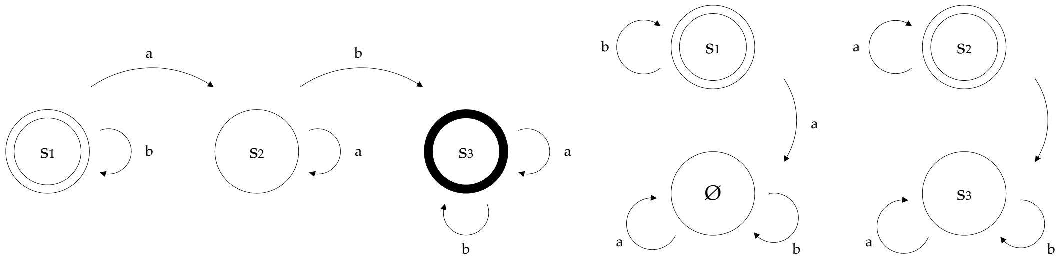
In the next section we will explain why decomposition is only a limited solution.

3.6.1 Additional condition

As said before, decomposition is only a limited solution and can only be applied when the LCP suits an important condition. The event which triggers a transition must occur in a sequential order specified in the LCP.

Let L be an LCP. L defines an *successor relation* between the transitions in L: t is followed by t' if there are transitions $t_1, \ldots, t_n$, such that $t = t_1, \ldots, t' = t_n$ is a *path* in L.

> *Definition:* Condition X states, that a LCP must hold the property that for every tuple t, t' (with t' a *successor* of t and corresponding events e, e', in the 'real world' the occurrence of e is *before* the occurrence of e'.



(a) Original (b) Decomposed

Figure 3.12: *Example of a fully deterministic automaton*

For some e the occurrence of e will not always be on a predefined absolute point in time, and therefor condition X will not hold. We will show this with an example. In this example we use a deterministic automaton with two variable events a and b. Hence Figure 3.12(a), in which $\Sigma = \{e_1, e_2\} = \{a, b\}$ and $T \subset \delta = \{t_1, t_2\} = \{\delta(s_1, a) = s_2, \delta(s_2, b) = s_3\}$. T is the set of transitions which must lead to a next state. The complete set of transition functions is showed in Table 3.4.

δ	**a**	**b**
s_1	s_2	s_1
s_2	s_2	s_3
s_3	s_3	s_3

Table 3.4: *Table of transition functions belonging to the DFA pictured in Figure 3.12(a)*

Now suppose that event a is the event *'leave coffee room'* and event b is the event *'leave the building'*. Imagine that event a never happens, and therefor state s_2 will never be reached. This implies that, although it is likely that the donor leaves the building and thus event b will happen, the transition to the final state will *not* happen. Figure 3.12(b) shows a decomposition of the DFA, resulting in two autonomous automata. Because of the decomposition, the transition between state s_2 and s_3 is *independent* of the transition between s_1 and s_2. This implies that, when the donor *'leaves the building'* although he hasn't been in the coffee room, his data will be degraded, which doesn't comply with the origignal LCP. This LCP violates condition X.

An example of events which do not violate the condition are *time events*. However, if relative events like *'after 10 minutes'* and absolute time events like *'at 5 p.m.'* are used in the same policy, the condition could easily been violated when the corresponding tuple is acquired at 4.55 p.m.

3.6.2 Decomposition of complex policies

So far we only described a solution for the inference problem for simple sequential policies. We decomposed the LCP in several partitions. For complex policies containing branches, this solution is not sufficient. However, a complex policy could be seen as a policy containing several sequential policies which can be decomposed separately. The problem is that it is not possible to predict which path of the policy will be followed. If we *not* decompose the policy, the inference problem and the violation of the one-way property still exists.

What we propose is to *hide* the complex, for everyone different and thus identifiable part of the policy. This is not very desirable, because in order to *hide* the policy, we also have to hide it for the system and thus making the policy not executable. So, we can only hide parts of a policy, leaving the current (and perhaps the next state) of the policy visible. It is, as we have seen before, possible to decompose the sequential parts of the policy. So, if we decompose the sequential part of the policy, that is, the part until a *branch* occurs in the policy, and *hide* the complex part of the policy (the part after the branch), we have are close to a solution.

We can use encryption to hide the complex part of the policy. If we encrypt the parts of the policies in such way that they are different for all tuples, they can not (like decomposition of policies) be used to find other tuples with the same policy. An overview of encryption techniques can be found in [22] and [4]. The keys and algorithms used for the encryption can be kept in hardware which can be trusted (there is no human being who has to know the keys).

When a branch is reached, and thus only the encrypted part of the policy is left, the policy translator can be used to decrypt the next part of the policy. This action is triggered by an event. The part which never will be reached can be removed. The part which must be decrypted is sent back to the policy translator, together with the current data tuple. The policy translator can decrypt the policy, and decompose the next sequential part, and attach this to the tuple. The first state in the policy will be the *construction state*, as if the tuple were inserted for the first time from a sensor.

Example

To illustrate the proposed solution, we present an example of the storage of a complex policy. Figure 3.13 shows a policy (modeled as an automaton) which could be divided in three main parts ($P\{0, 1, 4\}$). $P1$ can be divided in three subparts ($P1.\{1, 2, 3\}$). All sub policies are sequential, and thus can be decomposed.

When the policy arrives at the *policy translator*, the first step is to *decompose* the first policy until a *branch* has been reached. This is illustrated in Figure 3.14. The first part of the policy $P0$ consists of two states ($S0$ and $S1$), which are always reached (unless event e_0 never happens, but we keep this out of consideration). When state $S1$ is reached, there is a branch: depending on event e_1 or e_3 a different part of the policy will be reached (either $P1$ or $P4$). Because we don't know which one will be reached unless e_1 or e_3 happens, we will encrypt $P1$ and $P4$ to hide the complex part of the policy. When e_1 or e_3 occurs, we can submit either $P1$ or $P4$ to the policy translator, which will decrypt and decompose the next part of the policy.

Suppose $P4$ is reached (see Figure 3.15), $P4$ will be decrypted and decomposed. This could be done straightforwardly, because it is the last part of the remaining policy. The algorithm shown in 3.15(c) describes the decomposition.

Finally, the last example is similar to the first part of the policy (see Figure 3.16). With this technique, using a policy translator, all policies can be split in sequential parts which can be decomposed. By using decomposition in combination with encrypting, all policies and part of policies belonging to different tuples are distinct, and thus can't be used to match tuples to retrieve data.

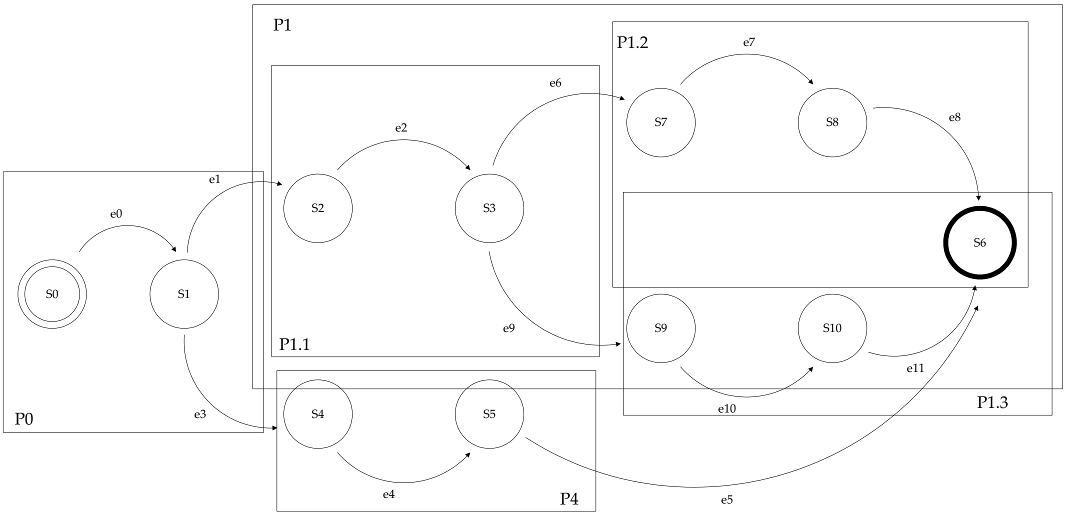
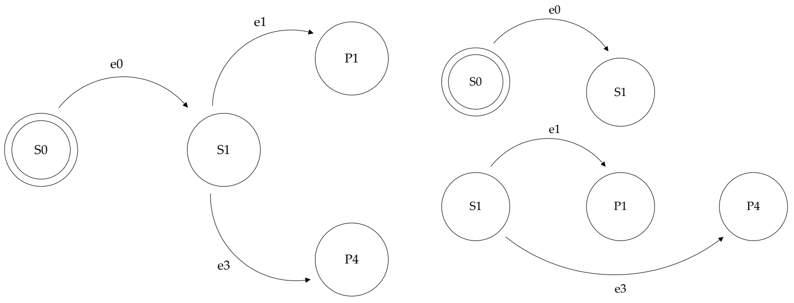


Figure 3.13: *Example of a complex policy. The policy is modeled as an automaton with two branches. The automaton is divided in three sub policies ($P\{0, 1, 3\}$), sub policy $P1$ is again divided in three sub-sub policies ($P1.\{1, 2, 3\}$).*



(a) First step

(b) Second step

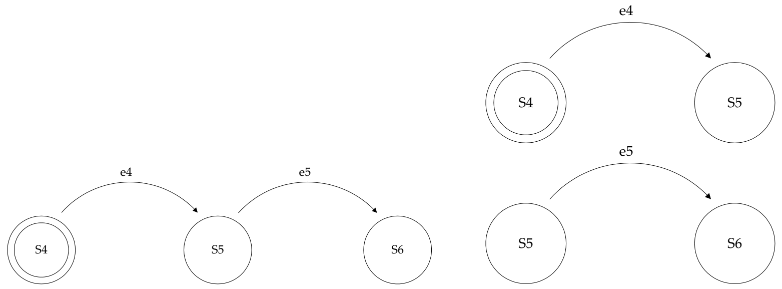
```
insert S0,S1
if e0
then
   delete(S0)
else if e1
then
   delete(S1)
   submit(P1)
else if e3
   delete(S1)
   submit(P4)
fi
```

(c) Pseudo code

Figure 3.14: *Decomposition of $P0$*

3.7 Post-processing of policies

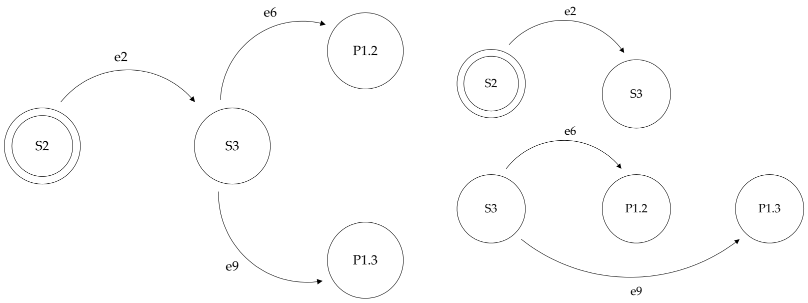
As briefly mentioned in the previous section, dependening on the moment when a LCP is attached to context data, the LCP may contain transitions or states which could cause troubles. This is clearly illustrated by Figure 3.17. In this example, a LCP is specified in which data is degraded twice: the first degradation step must be one hour after acquisition time, the second step at a fixed time (5 p.m.). If the data tuple is acquired and the specified LCP attached to it at 2 p.m., everything goes well. The first degradation step will take place at 3 p.m., the second step at 5 p.m.. But, if the data tuple is acquired *within one hour* before 5 p.m., the degradation step caused by the event '5 p.m.' will not take place. The automaton is still in the first state at the moment that it will become 5 o'clock, waiting for the event 'after 1 hour'. When this event finally happens, it is already past 5 p.m. so this event will not happen anymore. Hence this policy will not hold the condition stated in section 3.6.1.



(a) First step (b) Second step (c) Pseudo code

```
insert S4,S5,S6
if e4
then
   delete(S4)
else if e5
then
   delete(S6)
fi
```

Figure 3.15: *Decomposition of* $P4$



(a) First step (b) Second step (c) Pseudo code

```
insert S2,S3
if e2
then
   delete(S2)
else if e6
then
   delete(S3)
   submit(P1.2)
else if e9
   delete(S3)
   submit(P1.3)
fi
```

Figure 3.16: *Decomposition of* $P1.1$

To bother donors with this problem by letting themself be responsible for specifying a correct LCP is probably not the best solution. It is better to *post-process* policies at the moment that the data is acquired and the policy is attached to the context data. At that time, more information is available to be able to check the LCP for possible conflicts. In Figure 3.17, two possible solutions are proposed. Solution A is perhaps the most simple: an extra transition is added to make the automaton deterministic. When the automaton is deterministic, it will response always, no matter when the tuple is acquired. However, one major drawback of this solution is that a *branch* is introduced, still making the simple decomposition of sequential policies not possible.

Solution B is more satisfactory. This solution takes the time when the tuple is acquired into account, and, if needed, removes malicious states and transitions. If the tuple is acquired within one hour before 5 p.m., the first event to happen will be 5 p.m.. Because of the property that $S2 \trianglerighteq S3$ $(S2 is more accurate than S3)$, $S2$ can be removed without violating the purpose of the original LCP.

The LCPs specified by donors (or organizations) could been seen as a *mold*. The policy translator can be used to process the policies to remove possible conflicts. Conflict detection must then be possible. To make conflict detection possible, the semantics of the specified transitions must be clear. For time transitions, this is not to difficult, as shown in the next example:

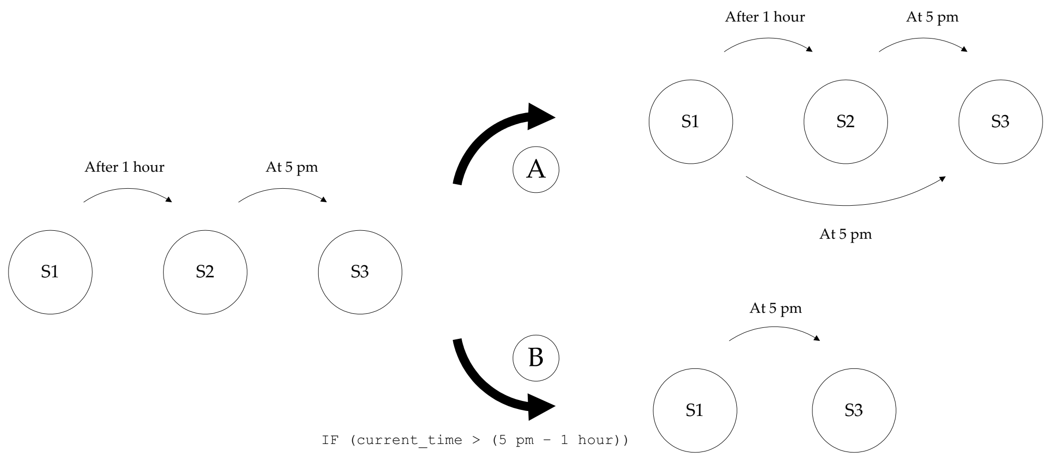


Figure 3.17: *Conditional transformation of a LCP. When a tuple is acquired within an hour before 4 p.m., the intermediate state could be skipped.*

Example: A simple mold could be the following LCP:

$Q = \{s_0, s_1, s_f\} = \{(1,1,1),(4,1,1),(4,4,4)\}$
$\Sigma = \{t_1, t_2\} = \{after\ 1\ hour, at\ 5\ p.m.\}$
$\delta(s_0, t_1) = s_1$
$\delta(s_1, t_2) = s_f$

This mold can be represented with the following encoding which can be understood by a policy translator:

```
S = {(1,1,1),(4,1,1),(4,4,4)}
T = {now()+01:00, 17:00}
```

The *keywords* $after$ and *at* are interpreted as possible modifiers, where the first denotes adding the specified time value to the current time. The keyword $after$ makes clear that the following time value is an absolute transition time.

The mold can now simply be processed as follows:

```
for i in (1 .. count(T)-1)
if (T(i) > T(j))
    remove(S(i));
    remove(T(i));
end
```

Also the *inference problem* can possibly be reduced by post-processing of policies. This is because the inference problem is caused by the fact that tuples can be identified by their policies, if the policy is unique (or almost unique). We can use *k-anonymity* algorithms to check if an incoming policy is sensible for the inference problem [31]. If the policy appears to be used by to view donors, the policy could be altered to met the *k*-anonymity requirements.

3.8 Distribution of policies

In section 3.1.1 we mentioned that an Ambient Intelligence environment could consist of many smaller AmI-spaces, each containing its own context database and sensors. This implies that data stored in context databases must be distributed across the AmI-spaces if a donor moves from one to another AmI-space. However, if data moves between databases, also their corresponding policies must be distributed, to make it possible that the new container of the data can degrade the data according to the LCP.

In this section we will not go into detail about how to distribute the data and the policies themselves. We will however take a look on one particular problem: how to 'monitor' events, if the events take place in an other AmI-space. In section 3.1.1 we specified three types of events: *internal*, *universal* and *external*. For this problem, *internal* are not an issue as they always take place within the AmI-space. *Universal* events can also be monitored everywhere (hence *time* events), and therefor don't form a problem.

External events are events which will take place *outside* the AmI-space in which the data is acquired. If a transition in a LCP must be triggered by such an external event, communication is needed between the current AmI-space and the AmI-space in which the event can be monitored. Otherwise the event will not be monitored at all, and the LCP will get stuck in a state so that the context data will never be degraded further.

> *Example:* Figure 3.18 shows two AmI-spaces, each 'covering' one floor in a building. Person *X* is on floor *A* at the moment when its location is monitored by the sensors of that floor, and his LCP is attached by the policy binder. The data is stored, together with the policy, in the context database of floor *A*. The LCP states that after a short period, the data must be degraded. When the donor *leaves the building*, the data must be degraded further to the final state. Now imagine that the donor moves from floor *A* to floor *B*, and finally leaves the building. With the movement from the first to the second AmI-space, the context data and its corresponding policies are copied between the two AmI-spaces (although it is not always necessary or there is always a reason to copy or move context data, we simply assume that it is the case now). The event *leave the building* is monitored at the border of the AmI-space of floor *B*, and is therefor an *external*. The AmI-space of floor *B* can degrade the data, but floor *A* can not because it doesn't know the event has occurred.

This problem could be solved with different strategies. We could for example broadcast all events to the AmI-spaces where the donor comes from. This implies that each AmI-space must keep a history of previous AmI-spaces of each donor. Besides the fact that this implies a lot of overhead, it is very privacy sensitive, something we want to prevent instead. Simply broadcasting all events to all AmI-spaces needs an infrastructure in which all AmI-spaces are connected with each other. Something which is not very likely to happen.

When a donor is out of sight for the sensors of the AmI-space, it not possible anymore to know when data must be degraded if external events are involved. When a donor crosses the *border* of the AmI-space, all events in the LCPs of the donor managed by the AmI-space could be scanned for possible problems. The events which cannot longer be monitored can now be processed as shown in Figure 3.19.

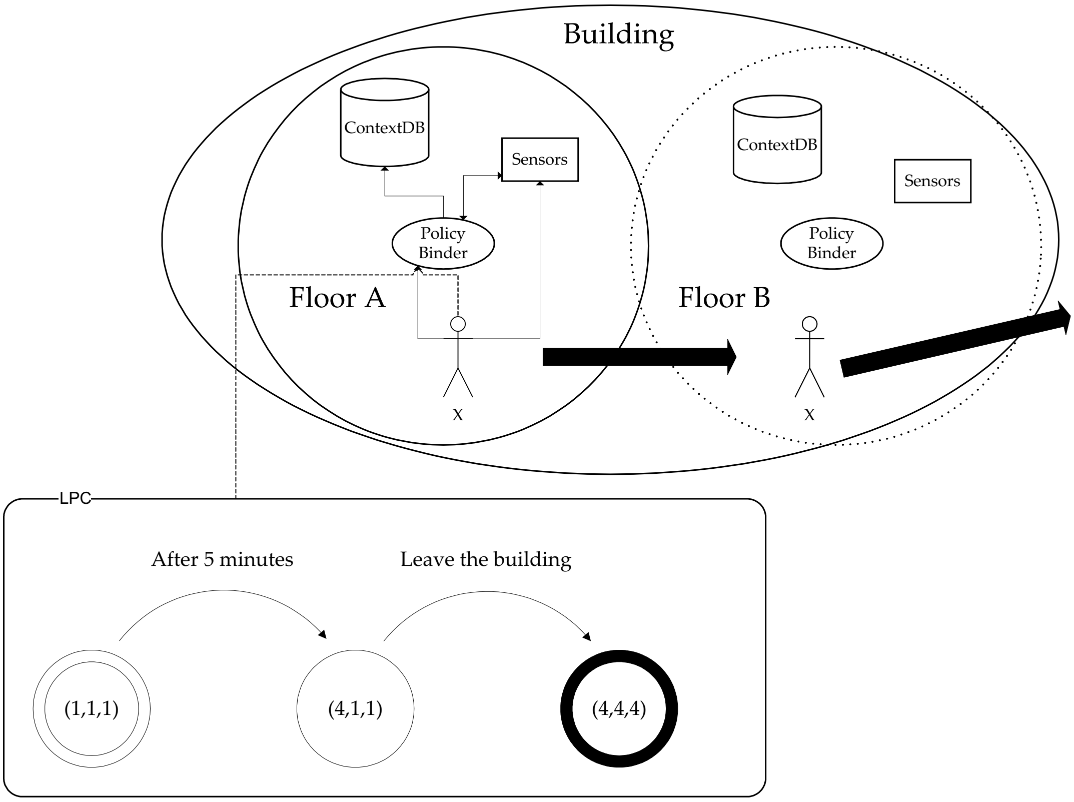


Figure 3.18: *Distribution op events through several AmI-spaces. Person X moves from the AmI-space on 'Floor A' to the AmI-space on 'Floor B'. Both floors are part of the same building.*

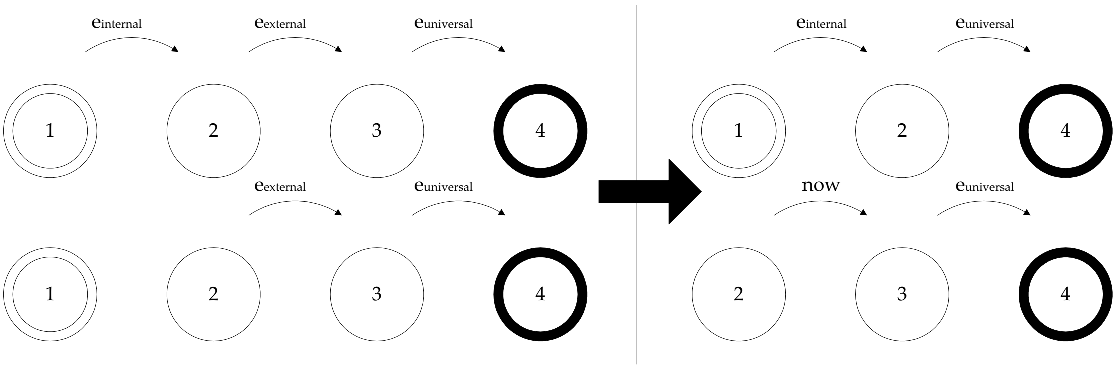


Figure 3.19: Transformation of LCPs containing external policies

Chapter 4

Performance evaluation of the LPC model

In this section, we present a brute-force instantiation of the LCP model on top of the PostgreSQL RDBMS. The goal of this implementation is to investigate the feasibility of the technique and to make the LCP model hopefully more understandable; the goal is not an in-depth performance evaluation.

4.1 'Brute force' implementation

We define a *brute force* implementation as an implementation which is not optimized in any way, although the brute force approach still seems to be useful as a 'base' for further implementations. It is 'not-optimized' in the sense that it does not contain optimization techniques on top of the normal use of indices and de-normalized schemas.

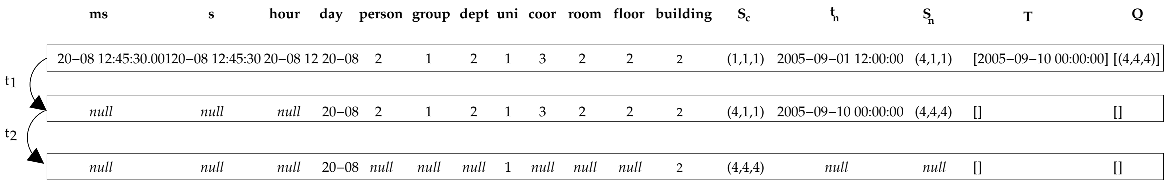
The *core* of our brute-force approach is to pre-compute all degraded fields for each newly incoming triplet according to donor-specified life-cycle policies, and store them together with the most accurate original ones. For performance consideration, we use a flat *fact table* to accommodate different-leveled context states within one tuple. A fact table is a table storing all context data.

As said a few times before, it is necessary to know the relations between GUID, group, department and university in order to do the degradation. This knowledge is not privacy sensitive and in most cases freely available to everyone, and therefor can be attached to the policies (for example by the policy binder, see section 3.1.1). Thus, if we are allowed (according to the LCP) to store the GUID of a donor, we can also store the group value, department value, etcetera. This is shown in table 4.1 with example data corresponding to states $(4,1,4)$,$(1,1,1)$,$(4,4,4)$ and $(2,2,2)$ respectively. Hence the attributes of the fact table

ms	**s**	**hour**	**day**	**GUID**	**group**	**dept**	**uni**	**coor**	**room**	**floor**	**building**
null	*null*	*null*	20-08	0109	DB	EWI	UT	(30,50)	3070	3	ZI
20-08 12:45:30.001	20-08 12:45:30	20-08 12	20-08	0109	DB	EWI	UT	(30,50)	3070	3	ZI
null	*null*	*null*	20-08	*null*	*null*	*null*	UT	*null*	*null*	*null*	ZI
null	20-08 12:45:30	20-08 12	20-08	*null*	DB	EWI	UT	*null*	3070	3	ZI

Table 4.1: *Example of a filled fact table*

consist of the dimensions and their granularity of the cube from section 3.2.2. This approach



ms	s	hour	day	person	group	dept	uni	coor	room	floor	building	S_c	t_n	S_n	T	Q
20–08 12:45:30.001	20–08 12:45:30	20–08 12	20–08	2	1	2	1	3	2	2	2	(1,1,1)	2005–09–01 12:00:00	(4,1,1)	[2005–09–10 00:00:00]	[(4,4,4)]
null	*null*	*null*	20–08	2	1	2	1	3	2	2	2	(4,1,1)	2005–09–10 00:00:00	(4,4,4)	[]	[]
null	*null*	*null*	20–08	*null*	*null*	*null*	1	*null*	*null*	*null*	2	(4,4,4)	*null*	*null*	[]	[]

Figure 4.1: *Example of the degradation process. S_c is the current state, t_n the next transition time, S_n the next state, T a set of upcoming transition times and Q a set of upcoming states*

leads to a lot of redundancy. However, for simplicity and especially for performance matters, it is a useful approach. It certainly makes data degradation quite easy. When degradation happens, we only have to delete the accurate value from the database by setting *NULL* to the corresponding columns, and keep the less accurate ones. This is illustrated in Figure 4.1. The additional columns denote the current, next and following states.

To know when and which tuples must be degraded and how to perform the degradation, we need to attach life cycle policies to individual tuples. There are many ways yo accomplish this, for example, by encoding the current, next, and following states and transitions as strings and push them into one or more attributes of the fact table. However, we simulate such a degradation behavior through an additional script table, in which we store a list of SQL update statements for tuple's future degradation. Along with the insertion of every new incoming context tuples into the Context-DB, a sequence of update statements plus their trigger events for its degradation will be recorded in the script table according to the donor's life cycle policy. By querying, fetching, and executing the corresponding update statements from this script table, the degradation process can then be implemented:

```
while(1){
        updates [] = "select update_statements
                from script_table
                where event has taken place";
        foreach update (updates){
                        execute update;
        }
}
```

Listing 4.1: *Pseudo-code of the degradation process*

4.2 Simulation studies

To investigate the feasibility of the proposed techniques, we performed several simulations to evaluate the performance of a DBMS adopting the life cycle policies, where the extra cost increase mainly comes from the degradation process for every incoming context tuple, i.e., updating the existing context tuple according to a life cycle policy when certain events happen until the final context state has been reached. The simulations simulates as close as possible the brute force approach as described in the previous section. For simulation purposes, we choose random tuples which must be degraded instead of using real events.

The overall workload of the database system includes dealing with users' normal query requests, plus continuously insertion of new context tuples as well as their degradation updates. We interleave database query requests with insertion and update operations. The whole stream of SQL statements processed by the database, consisting of N arrived tuples, Q times N queries and N times the number of steps in the policy S statements can be presented as follows:

$$\left(\left(i_{fact}; i^{S}_{script}; q^{Q}_{fact}\right)^{U}; q_{script}; u^{U\times S}_{fact}; d_{script}\right)^{\frac{N}{U}}$$

i_{fact} = insert statement (for inserting a new context tuple into the fact table)
i_{script} = insert statement (for writing an update statement into the script table)
q_{fact} = query (for querying the fact table (a users' normal query requests))
q_{script} = query (for querying the script table to fetch a update statement)
u_{fact} = update statment for updating a tuple in the fact table (for degradation)
d_{script} = delete statement (deleting an update statement from the script table)
U = Number of inserts preceding a degradation run
S = Number of degradation steps of the policy
Q = Number of queries after each incoming context tuple
N = Total number of incoming context tuples

> *Example:* In Appendix A an example of a SQL stream is shown, with the parameters $N = 4$, $U = 2$, $S = 2$ and $Q = 2$. This results in 4 insert statements in the *fact table* (simulating 4 newly acquired tuples), 8 insert statements in the *script table* (each tuple will be updated $S = 2$ times, because the LPC has 2 degradation steps), 8 queries on the *fact table* (with every acquired tuple 2 queries will be executed), 2 select statements on the *script table* (after every 2 acquired tuples there is an 'degradation run'), 8 update statements on the *fact table* (in each 'degradation run' 4 tuples are updated) and 2 delete statements in which each time 4 tuples are removed from the *script table* (in each 'degradation run' a delete statement is executed).

Now assume T is the time needed for the execution of all the above mentioned SQL statements. The number of inserts per second can now be calculated as $\frac{N}{T}$, and the number of queries per seconds as $\frac{N\times Q}{T}$. Considering that querying is one of the most fundamental operations in any database system, the performance measure we used is the number of queries per second (query throughput) which can be executed with a given number of inserts per second (arriving rate of context data).

We compared the performance with that of a traditional database system without degradation policies, whose main workload consists of querying and inserting into the fact table. In our tests we set $U = 5$, $N = 5000$ and vary the value of Q over $[0 \ldots 64]$.

The first test simulates the degradation process on top of a databases filled with 200.000 tuples. The degradation process processes LCPs with 3 states (2 degradation steps, $S = 2$). We compared the performance of the database with and without degradation. Results are shown in Figure 4.2. The performance of the system is negatively influenced due to the extra updating workload for degradation. Of course this is not surprising. The second test

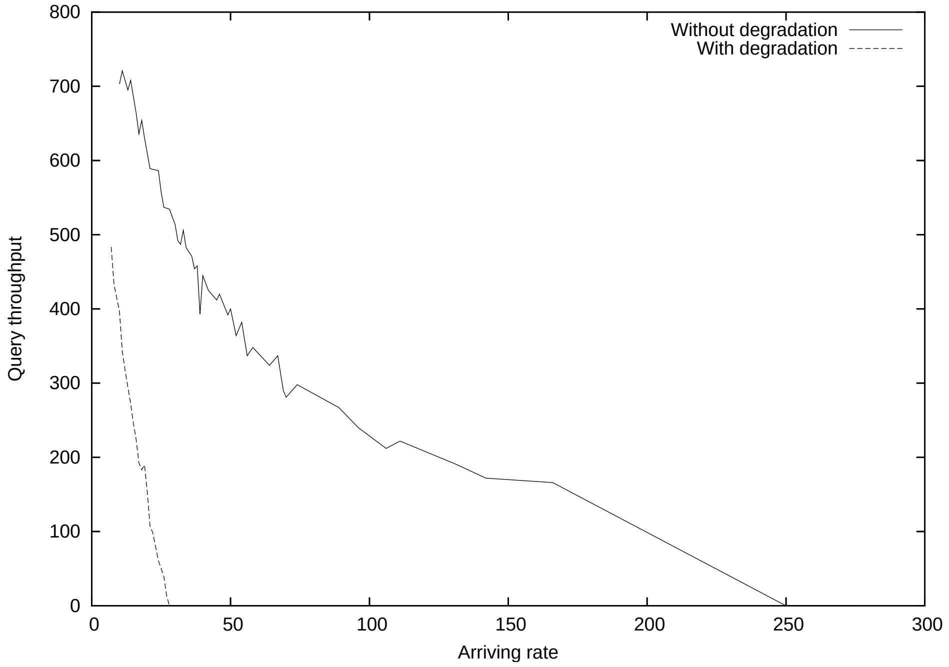


Figure 4.2: *Comparison between a medium sized database implementing LCPs with $S = 2$, and a database without data degradation.*

simulates different complexities of policies, varying the number of degradation steps on [1 . . . 4]. Figure 4.3 clearly shows that the higher the complexity, the worse the performance. This is easily explained because of the larger number of updates which must be performed. The third test simulates the degradation process on different sized databases with 30.000, 200.000, 400.000 and 750.000 tuples. The results (shown in Figure 4.4) give an indication of the *scalability* of the system, which is quite linear.

In Section 3.6 we described a method to overcome the inference problem as described in Section 3.4.2. Decomposition has an impact on the stream of SQL statements needed to simulate the degradation process. First, decomposition implies that tuples are inserted multiple times, depending on the number of partitions the original LPC is cut. In the extreme this will be S parts, which we will simulate here. Second, in stead of *updating* tuples in the fact table, tuples are now *deleted*. This results in the following stream of statements:

$$\left(\left(i^{S}_{fact}; i^{S}_{script}; q^{Q}_{fact}\right)^{U}; q_{script}; d^{U\times S}_{fact}; d_{script}\right)^{\frac{N}{U}}$$

We compared a simulation with the decomposed LPCs with the original simulations. Results are shown in Figure 4.5. Surprisingly, results are similar, indicating that decomposition does not lead to a degradation of performance. This could possibly be explained with assuming that the database management system efficiently buffers the new tuples, writing them at once to the hard disk. It is also possible that, if the DBMS is ‘smart’, it processes inserts and delete statement in one I/O operation.

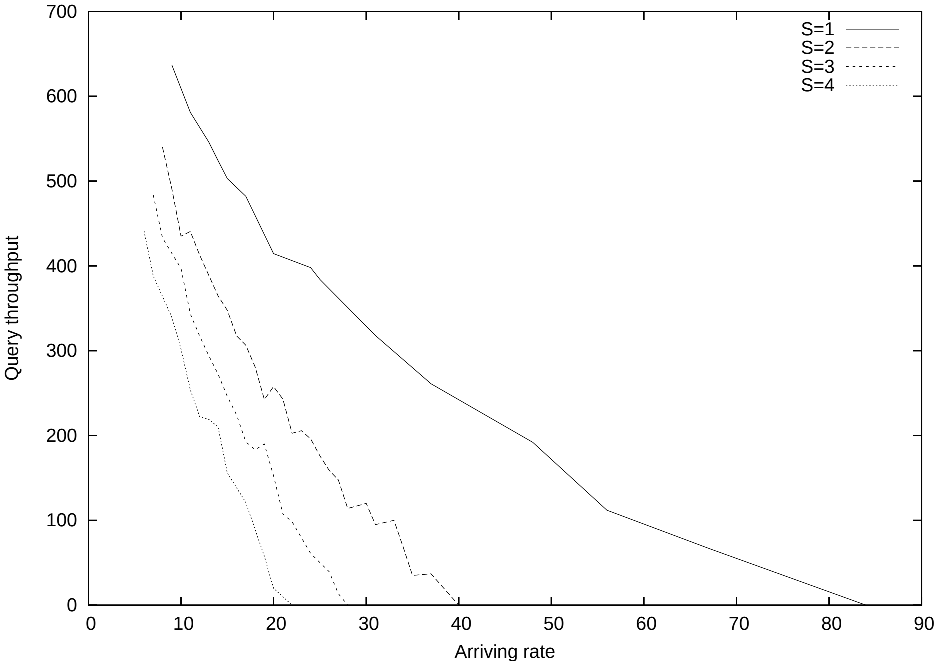


Figure 4.3: *Comparison of a medium sized databases with* $S = [1 \ldots]$

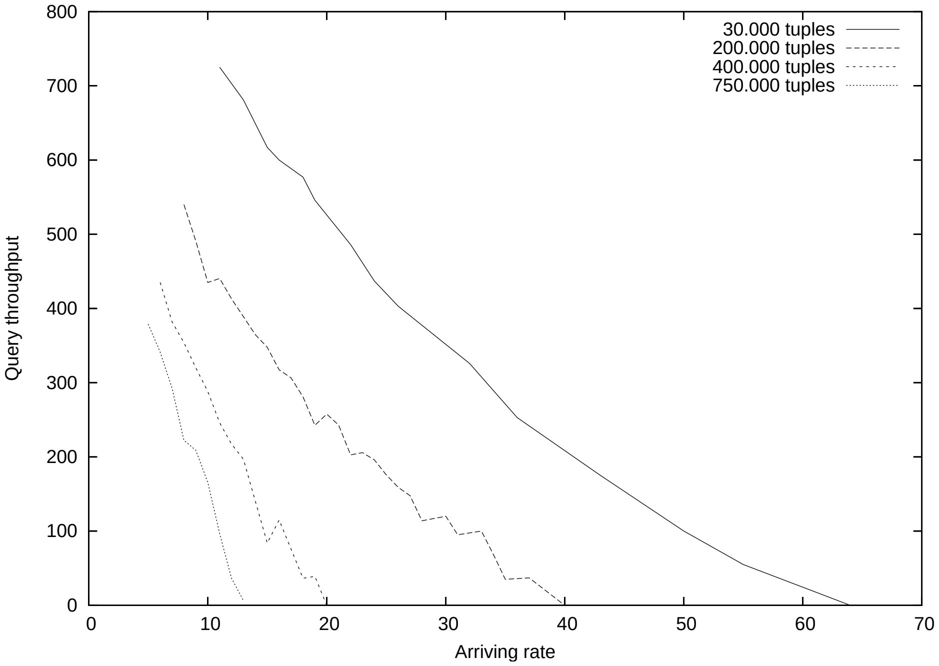


Figure 4.4: *Comparison of different sized databases,* $S = 2$

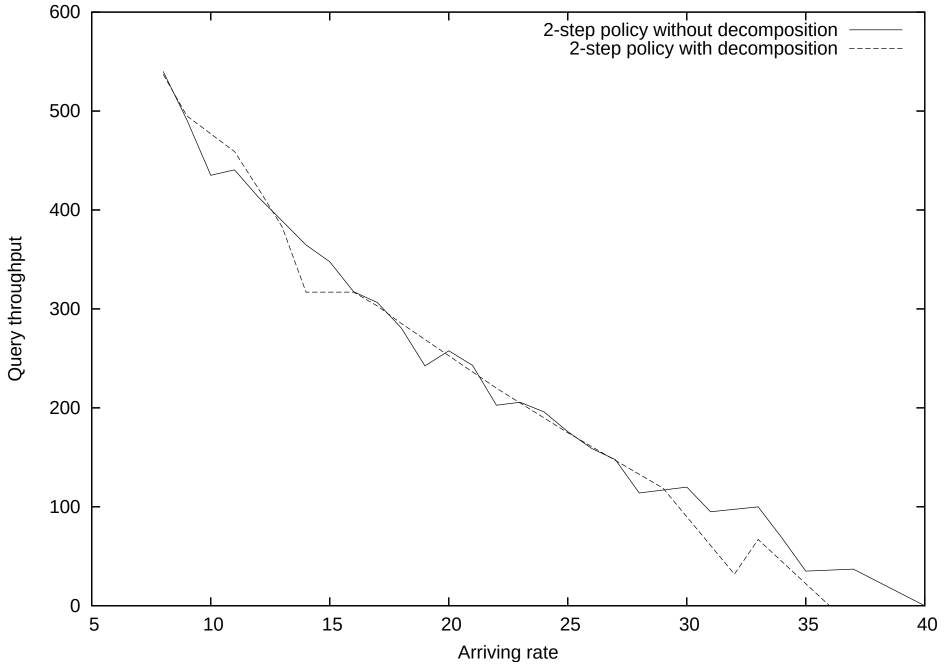


Figure 4.5: *Comparison of degradation process with and without decomposition of LCPs*

4.3 Implementation studies

Besides the simulation studies we also built an actual implementation of the brute force technique, in both the languages Perl and Java, to study the impact of degradation from an implementation point of view. Although the LPC model itself supports events other than only *time events*, we will only use those time events in our implementation.

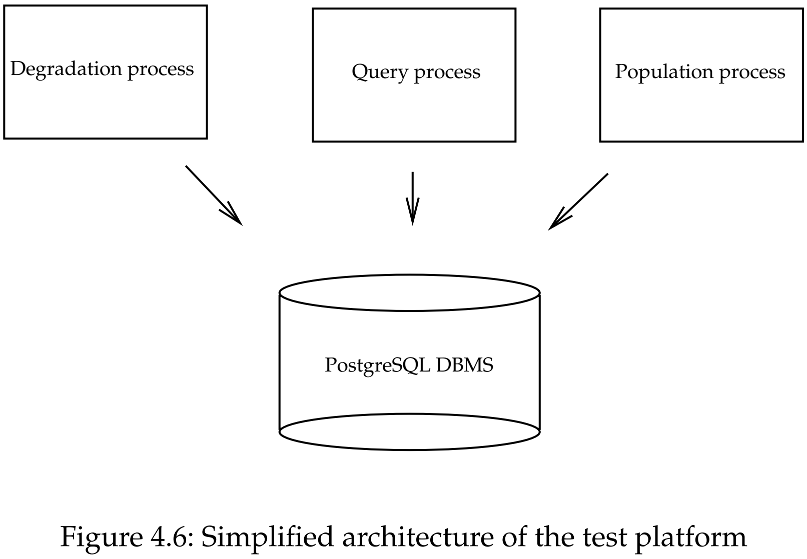


Figure 4.6: Simplified architecture of the test platform

Three external scripts execute statements on the PostgreSQL database management system. According to the brute force technique, the context database itself is implemented by two separated tables: one to contain the location data (the fact table), and one for storing *update statements* and time values to specify when the update statements have to be executed by the degradation process. The database schema is:

> *fact_table (fact_id, ms,s,hour,day,GUID,group,dept,univ,coor,room,floor,building)*
> *script_table (fact_id,time,update_statement)*

As you can see, we don't attach the automaton directly to the fact tuples. Instead, we use a script table in which we store all updates that have to be performed to comply with the degradation policies. These policies are represented as an array of states and transition times. States are expressed as tuples containing the accuracy levels of the three dimensions *time,id* and *location*. For each policy, the degradation steps have to be precomputed so they can be contained in SQL update statements. The following pseudo-code shows how an array of values ($t_1, t_2, t_3, t_4, id_1, id_2, id_3, id_4, l_1, l_2, l_3, l_4$) is degraded:

```
function degrade(state, values [] org){
        time_state = state(1);
        pers_state = state(2);
        loc_state = state(3);
        values [] newValues = {};
        i = 1;
        while(length(values) >= 0){
                my value = pop values;
                if(i < time_state){
                        push newValues, null;
                }elsif(i > 4 && i < pers_state){
                        push newValues, null;
                }elsif(i > 8 && i < loc_state){
                        push newValues, null;
                }else{
                        push newValues, value;
                }
                i++;
        }
        return newValues;
}
```

Listing 4.2: *Pseudo-code of degrading context data*

All values t_n,p_m,l_k, with $n \leq 4$, $4 < m \leq 8$ and $8 < k \leq 12$ are made *null*, if the corresponding index is smaller than the specified state (*time_state*, *pers_state*, *loc_state*). These values could now be used to create an update statement:

```
attributes = {ms, s, hour,day,guid, group, ...}
sql = "update fact table set "
for i=0, i < length(values), i++{
        sql += attributes[i] + "=" + values[i] + ","
}
chop(sql)
```

Listing 4.3: *Pseudo-code of creating an update statement*

The degradation process could now simply be expressed as an continuous loop that checks transition times, and executes corresponding update statements:

```
while(1){
        updates [] = "select update_statements
                        from script_table
                        where time < now()";
        foreach update (updates){
                execute update;
        }
}
```

Listing 4.4: *Pseudo-code of the degradation process*

Querying the database will be done by a multi-threaded application. Each thread simulates an application which queries the database. The scripts will run on the same server as the database server.

4.3.1 Expectations

If we insert, query and update the database always as fast as possible, e.g. apply a maximum load on the database without the use of scheduler, the following equation always applies

$$cost(T_A(t)) = maximum\ load = cost(T_B(t))$$

This means that the *cost* of the total number of transactions T at time t without degradation (A) must be equal to the number of transactions at time t *with* degradation (B). If we divide the transaction into queries, insertions and updates, we get:

$$\underbrace{a * q_F^A(t) + b * i_F^A(t)}_{without\ degradation} = \underbrace{a * q_F^B(t) + b * i_F^B(t) + c * q_S^B(t) + d * i_S^B(t) + e * u_F^B(t)}_{with\ degradation}$$

with

a, b, c, d, e =costs of transaction
$q_F(t)$ =total number of query's on the fact table at time t
$i_F(t)$ =total number of inserts in the fact table at time t
$i_S(t)$ =total number of inserts in the script table at time t
$u_F(t)$ =total number of updates of the fact table at time t
$q_F(t)$ =total number of query's on the script table at time t

We can find the performance loss due to the degradation process with the following equation:

$$loss = \frac{(a*q_F^A(t)+b*i_F^A(t))-(a*q_F^B(t)+b*i_F^B(t))}{a*q_F^B(t)+b*i_F^B(t)}$$

In this statement we have to specify an a and b, to calculate the actual performance loss. An approach could be to specify the ratio (α) between querying the fact table and inserting into the fact table. Without any insertion, the number of transactions per second (Tps) will be:

$$Tps = \frac{q_{without\ inserting}}{t}$$

If we, while querying, also insert tuples in the fact table, the total number of transactions per seconds will be:

$$Tps = \frac{q_{with\ inserting}}{t} + \alpha \cdot \frac{i_F}{t}$$

This equation thus only defines that inserting will be α times slower than querying. We can find α with:

$$q_{without\ inserting} = q_{with\ inserting} + \alpha \cdot i_F \Leftrightarrow$$
$$\alpha = \frac{q_{without\ inserting} - q_{with\ inserting}}{i_F}$$

Say we found that querying the database, without inserting, is performed with 500 queries per second. When we ran the test again, this time with inserting tuples, we found that querying was performed with 400 queries per second, and inserting with 10 tuples per second. We now are able to calculate α:

$$500 = 400 + \alpha \cdot 10 \Leftrightarrow$$
$$\alpha = \frac{500-400}{10} = 10$$

This will tell us that inserting a tuple is 10 times more costly than querying. We now can use our α to determine the performance loss when we use a context database with degradation of data. The following holds:

$$q_F^A + \alpha \cdot i_F^A \ \rangle \ q_F^B + \alpha \cdot i_F^B$$

This statement says that querying and inserting into the fact table *without* the degradation process running will always be faster (in terms of queries and insertions per second) than *with* degradation.

Example: from some of our tests, we know that without degrading 50 tuples per second are inserted in the fact table, and 500 queries per second are executed. With degradation however, insertion speed is much lower (9 tuples per second) but querying is much faster: 650 queries per second. If we fill in this values, and use an $\alpha = 10$, we will find the performance loss:

$$loss = \frac{(q_F^A + \alpha \cdot i_F^A) - (q_F^B + \alpha \cdot i_F^B)}{q_F^B + \alpha \cdot i_F^B} \Leftrightarrow$$
$$loss = \frac{(500 + 10 \cdot 50) - (650 + 10 \cdot 9)}{650 + 10 \cdot 9} \approx 35\%$$

4.3.2 'Implementation studies' test results

The results of the test performed by the scripts shows us the behavior as predicted by our database model, as could be seen from Figure 4.7. At first glance one could say that a context database *with* degradation techniques for preserving privacy has better performance than a traditional database. Indeed, the total number of transactions of the database with degradation is higher than without degradation: about 675 queries plus 10 inserts per second against about 500 queries plus 80 inserts per second for the traditional database without degradation. But, if we apply these numbers into our model, the following must be true otherwise our model is not correct:

$$500 + \alpha \cdot 80 \ \rangle\ 675 + \alpha \cdot 10$$

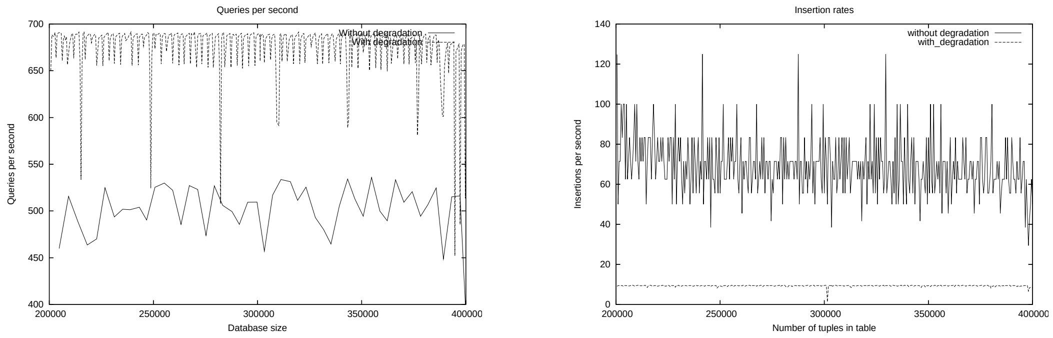
We didn't measure the correct value of α yet, but perhaps we can derive it from traditional database theory. If we imagine that the (simple) query can use the index to find the correct tuple, and if we assume the index is not in cache memory, it takes the depth of the B-tree index in I/O costs to find the address of the tuple, and one additional I/O to retrieve the result [21]. Total I/O operations will thus be 3. The insert operation however, has to update several indices (one primary key index on *fact_id*, and the twelve indices for the dimensions and their granularities of *the cube*). Again, assuming that the indices are not in cache and that they will be written back immediately to disk, this will take at most 2 times 13 I/O operations. The write of the insert itself will take one additional I/O. The total cost could thus be 27 I/O's, which is 9 times more than the cost of the query. If we now say that $\alpha = 9$, we can validate the test results:

$$500 + 9 \cdot 80 \ > \ 675 + 9 \cdot 10 \Leftrightarrow$$
$$1220 \ > \ 765$$

Whit this test we show that our model is not be invalidated by the test results. Nevertheless, these test results are not very useful to specify a *baseline* of the performance of the brute force degradation process. We could say, if we compare the above costs, that with degradation we lose about 37%. However, we are, if we rely on the Postgres load balancing, not able to measure the number of queries per second with changing insertion rates. In theory, we can therefor use the scheduler technique as will be described in Section 4.3.3.

4.3.3 Scheduler

Querying, inserting and degradation are three processes which will concurrently send statements to the databases. Normally the scheduler of the DBMS chooses which statements are



(a) Querying (b) Insertion

Figure 4.7: *Queries per second and insertion rates with and without degradation, using Perl and Java scripts to simulate the degradation process, inserting and querying.*

executed in which order, without giving priority to one or another process. However, we have to ensure that the degradation policies are applied, so we have to ensure that the update statements always are executed on time. This is why we need to implement a scheduler ourselves. A scheduler can be used to give priority to the degradation process when needed, and allow queries to be executed if there is enough room to do so.

To implement a scheduler, we will use a *query buffer*. All incoming queries will be stored in the buffer, until the buffer is full. Queries in the buffer will be consumed by the scheduler, if the degradation process has nothing to do, or when a similar but less strict criterion holds. The use of a buffer and scheduler is illustrated in Figure 4.8.

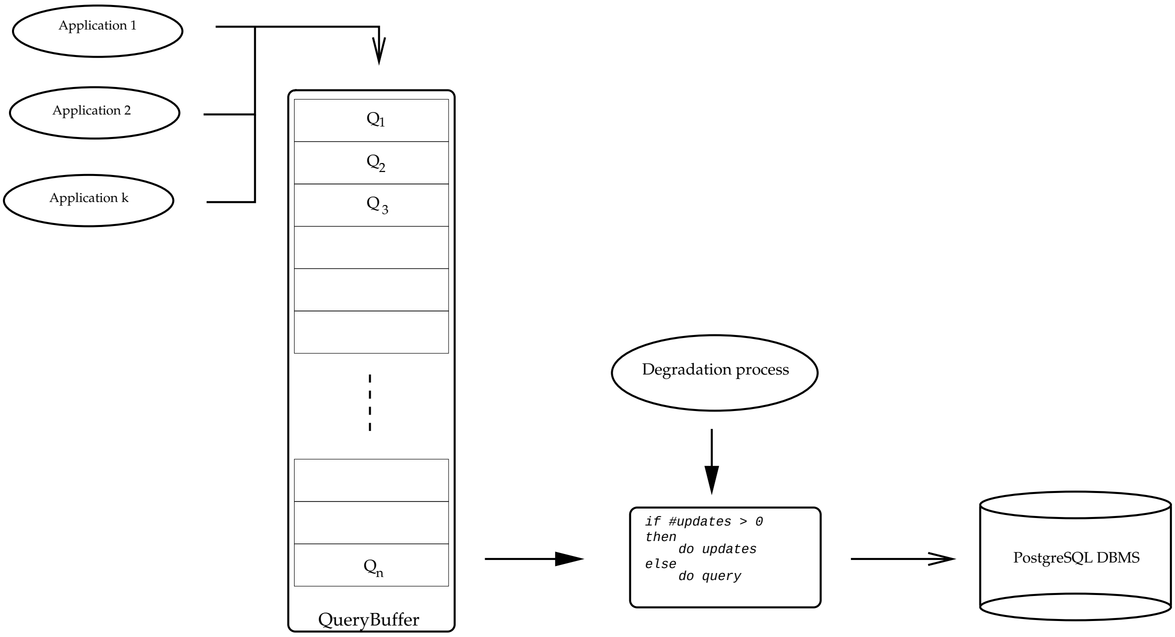


Figure 4.8: *Schematic overview of a schuduler and a query buffer. Applications are issueing queries, which will be pushed on a query buffer. The schedular consumes queries in the buffer according to a first in first out (fifo) strategy.*

Chapter 5

Future work

Although we already addressed several problems with the LCP model, there are a number of open issues which must be solved in future work. In this section we briefly summarize the main issues, starting with querying a multi-accurate data set.

5.1 Querying the database

Although we mentioned the problem in the problem definition, we were not able to give much attention to how to query data with different accuracies. In an ideal situation, the applications (the *users* of the stored data) are unaware of the fact that they use a privacy preserving context database. Thus, we want to create a situation in which the privacy preserving capabilities are transparent for the applications.

> *Example:* Let us assume that the application only sees a fact table with the three attributes *time,person* and *location*, but has the same *environment knowledge* as the context database. Thus, we assume that applications also know what buildings there are, what the relation is between group and department, etcetera. Consider the following query:
>
> ```
> select person
> from fact_table
> where building = Zilverling
> and time between 10 pm and 11 pm
> ```
>
> (select all persons who are in building 'Zilverling' between 10 and 11 pm)

This query could produce different answers, highly depending on the states of the donor's degradation policies. We could for example return a *mixed* result set, with the most accurate values of which are available from a donor. An example is given in Table 5.1. This approach is not conform our 'transparency' statement, because we have to send accuracy information along with the actual results. How to solve this problem is subject to future research. The decomposition techniques as described in section 3.6 make things even worse. Copies of data are inserted multiple times in the database, making aggregations like 'give me the number of persons in the building' even more hard.

accuracy	person
GUID	2178
dept	EWI
dept	EWI
GUID	0109
univ	UT
univ	UT
dept	GW
group	DB

Table 5.1: *Mixed result set of persons*

5.2 Rich generalization schemes

The generalization schema as contained in Life Cycle Policies are focused on one particular group of purposes. Different application domains could require different generalization schemes.

> *Example:* The generalization schema of the identification value can start with the license plate number of a car. This value could be degraded to 'brand of the car' (e.g. Toyota, Mercedes, DAF, etcetera), and finally to the type of the car (e.g. Truck, Van, Car, etcetera). For the carpooling example given in section 3.3.2, this schema is sufficient. If the same location data is used by other kind of applications, the generalization schema for the identification value is likely to be different, for example: personal identification number → team → group → university.

How to specify (richer) generalization schemes and put them into the Life Cycle Model could be a topic for future research.

5.3 Enforcing the degradation process

In the introduction section we mentioned the ten principles of Hippocratic databases. One of the most important, but perhaps also the most difficult one to adopt, principles is the *compliance* principle. In this thesis we proposed techniques which regulate the life cycle of a policy, which could enhance privacy. But how can we make sure that databases will correctly adopt this techniques, and not only promise to comply with the specified LCPs?

LCPs also do not protect against attacks on the database itself. If someone gains access to the database and alters the LCPs, a serious privacy problem can arise. *Openness* for the donors, to check their data and LCPs could therefor be very important. How to accomplish this is subject to future research.

5.4 Two-way LCP policies

In our research we specified the *one-way property*, not allowing to regenerate data when it is already degraded to a less accurate level. This implies that it is not possible to *temporarily* degrade data. For example, it could be useful to hide the current location for one week, allowing, after that week, that particular location to be visible again. This flexibility is missing in the current LCP model, but in the spirit of life cycle policies, could be a very useful feature for the future. Moreover, access control mechanisms based on fine-grained micro-views support this feature already.

5.5 Integrating the LCP model in a prototype

In this thesis, we already showed the feasibility of the LCP model in terms of performance. This however will not guarantee that the LCP model can be adopted by a real ambient intelligence environment. Building a prototype and integrate this into a prototype of an ambient intelligence application could resolve many issues which must be investigate to make the LCP model really useful.

A good test framework could be the 'Soft meeting planner', a project of the University of Twente which is part of a research group doing research toward the Ambient Intelligence.

Chapter 6

Conclusion and recommendations

In this thesis we proposed a new approach toward privacy for the Ambient Intelligence. We specified a LCP model, in which it can be specified how and when data must be degraded. We used automata to specify transition functions which are triggered by different kinds of events. Each state of the automata represents a certain degree of accuracy.

We succeeded in specifying a rich model which is able to manage the life cycle of data in order to improve protection against violating of fundamental privacy principles. We discovered however that such model implies some difficulties due to the model itself. Inference and information disclosure are problems for which we supplied solutions, although limited. In order to support our proposal and making it more understandable, we performed an implementation and simulation study, which showed the feasibility of our approach.

Some open issues are left out of consideration in this thesis, but are quite important for the usability of our model in order to play a significant rôle in the development of an Ambient Intelligence. How to query the data, how to supply richer generalization schemes are, among other issues, main future research questions.

To investigate the real usability of the model in the Ambient Intelligence, it is useful to build a realistic model (of a part) of an Ambient Intelligence and integrate a database using our LCP technique. Requirements from the Ambient Intelligence in terms of the required smartness of applications in conjunction with the possibilities and the restriction of our life cycle privacy model can then be investigated.

Our model will not comply with every privacy requirement in the AmI, but will certainly be a step forward toward a situation in which control of their own privacy is returned to the people which otherwise are subject to possible privacy violations. Although many issues must be solved, people are, with our LCP model, able to specify what must happen to their data. To conclude, we are convinced that our approach toward solving privacy problems in the Ambient Intelligence, using our LCP model, paves the way to new solutions to respect individuals privacy in autonomous systems.

Bibliography

[1] G. Aggarwal, M. Bawa, P.Ganesan, H. Garcia-Molina, K. Kenthapadi, N. Mishra, R. Motwani, U. Srivastava, D. Thomas, J. Widom, and Y. Xu. Vision Paper: Enabling Privacy for the Paranoids. In *Proceedings of the 30th VLDB Conference*, 2004.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *28th Int'l Conf. on Very Large Databases (VLDB), Hong Kong*, 2002.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450, New York, NY, USA, 2000. ACM Press.

[4] Nicolas Anciaux. *Systèmes de gestion de base de données embarqués dans une puce électronique (Database Systems on Chip)*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, 2004.

[5] Elisa Bertino, Ji-Won Byun, and Ninghui Li. Privacy-preserving database systems. *Lecture Notes in Computer Science*, 3655:178–206, 2005.

[6] Ji-Won Byun and Elisa Bertino. Micro-views, or on how to protect privacy while enhancing data usability. Vision paper CERIAS Tech Report 2005-25, Center for Education and Research in Information Assurance and Security, West Lafayette, IN 47907-2086, 2005.

[7] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.

[8] CNN. How much does google know about you? `http://www.cnn.com/2005/TECH/internet/07/18/google.privacy.ap/index.html`.

[9] L. Cranor, J. Reagle, and M. Ackerman. Beyond concern: Understanding net users attitudes about online privacy, 1999.

[10] Ernesto Damiani, Sabrina De Capitani di Vimercati, and Pierangela Samarati. Towards privacy-aware identity management. *Ercim news*, 63:23–24, 2005.

[11] EUSAI-2004. Second european symposium on ambient intelligence. `http://www.eusai.net`.

[12] Google. Gmail and privacy. `http://gmail.google.com/gmail/help/more.html`.

[13] Andreas Görlach, Andreas Heinemann, and Wesley W. Terpstra. Survey on location privacy in pervasive computing. In Philip Robinson, Harald Vogt, and Waleed Wagealla,

editors, *Privacy, Security and Trust within the Context of Pervasive Computing*, The Kluwer International Series in Engineering and Computer Science, 2004. Workshop at Pervasive 2004: `http://www.pervasive2004.org/program'_w6.php`.

[14] Hakan Hacigümüş, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database service provider model. In *SIGMOD Conference*, 2002.

[15] Robert J. Hilderman, Howard J. Hamilton, and Nick Cercone. Data mining in large databases using domain generalization graphs. *J. Intell. Inf. Syst.*, 13(3):195–234, 1999.

[16] HIPAA. Examples of privacy violations from heath and human services. `http://www.hipaaps.com/examples.html`.

[17] Jason I. Hong and James A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 177–189, New York, NY, USA, 2004. ACM Press.

[18] Sushil Jajodia, Pierangela Samarati, V. S. Subrahmanian, and Elisa Bertino. A unified framework for enforcing multiple access control policies. In Joan Peckham, editor, *SIGMOD Conference*, pages 474–485. ACM Press, 1997.

[19] Xiaodong Jiang, Jason I. Hong, and James A. Landay. Approximate information flows: Socially-based modeling of privacy in ubiquitous computing. In *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing*, pages 176–193, London, UK, 2002. Springer-Verlag.

[20] Marc Langheinrich. Privacy by design - principles of privacy-aware ubiquitous systems. In *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, pages 273–291, London, UK, 2001. Springer-Verlag.

[21] Philip M. Lewis, Arthur Bernstein, and Michael Kifer. *Databases and transaction processing - An Application-Oriented Approach*. Addision-Wesley Pearson Education International, Upper Saddle River, NJ, USA.

[22] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[23] David Molnar, Andrea Soppera, and David Wagner. Privacy for RFID through trusted computing. In *Workshop on Privacy in the Electronic Society – WPES*, Alexandria, Virginia, USA, November 2005. ACM, ACM Press.

[24] Commission of the european communicties. Directive of the european parliament and of the council on the retention of data processed in connection with the provision of public electronic communication services and amending, 2005.

[25] Andrew Orlowski. Us gov demands google search records. The Register, `http://www.theregister.co.uk/2006/01/19/feds_subpoena_google_search_records/`, January 2006.

[26] PostgreSQL. Documentation. `http://www.postgresql.org/docs/`.

[27] Electronic privacy information center. Gmail privacy page. `http://www.epic.org/privacy/gmail/faq.html#1`.

[28] Y. Punie, S. Delaitre, I. Maghiros, and D. Wright. Dark scenarios on ambient intelligence: Highlighting risks and vulnerabilities, swami deliverable d2. a report of the swami consortium to the european commission under contract 006507. `http://swami.jrc.es`, 2005.

[29] Thomas A. Sudkamp. *Languages and machines: an introduction to the theory of computer science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[30] Latanya Sweeney. Achieving *k*-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty Fuzziness and Knowledge-based Systems*, pages 571–588, 2002.

[31] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty Fuzziness and Knowledge-based Systems*, pages 557–570, 2002.

[32] W3C. A P3P preference exchange language 1.0 (APPEL1.0). `http://www.w3.org/TR/P3P-preferences/`, April 2002.

[33] Wikipedia. List of google services and tools — wikipedia, the free encyclopedia, 2006. [Online; accessed 1-February-2006].

Appendix A

Example of simulation SQL stream

Example stream of statements with parameters $N = 4$, $U = 2$,$S = 2$ and $Q = 2$.

```
insert into fact_table values( ... )
insert into script_table values( ... )
insert into script_table values( ... )
select coor, room, floor, building from fact_table where fact_id = 136380
select coor, room, floor, building from fact_table where fact_id = 111959
insert into fact_table values( ... )
insert into script_table values( ... )
insert into script_table values( ... )
select coor, room, floor, building from fact_table where fact_id = 195960
select coor, room, floor, building from fact_table where fact_id = 165948
select * from script_table where fake_id >= 2730 and fake_id < 2734
update fact_table set ms = null, ...
update fact_table set ms = null, ...
update fact_table set ms = null, ...
update fact_table set ms = null, ...
delete from script_table where fake_id >= 2730 and fake_id < 2734
insert into fact_table values( ... )
insert into script_table values( ... )
insert into script_table values( ... )
select coor, room, floor, building from fact_table where fact_id = 192738
select coor, room, floor, building from fact_table where fact_id = 195489
insert into fact_table values( ... )
insert into script_table values( ... )
insert into script_table values( ... )
select coor, room, floor, building from fact_table where fact_id = 116315
select coor, room, floor, building from fact_table where fact_id = 9418
select * from script_table where fake_id >= 4313 and fake_id < 4317
update fact_table set ms = null, ...
update fact_table set ms = null, ...
update fact_table set ms = null, ...
update fact_table set ms = null, ...
delete from script_table where fake_id >= 4313 and fake_id < 4317;
```

Listing A.1: *SQL simulation stream*

Appendix B

Sample P3P policy

The following P3P policy is generated with a free available P3P editor from IBM (`http://www.alphaworks.ibm.com/tech/p3peditor`).

```
<?xml version="1.0"?>
<POLICIES xmlns="http://www.w3.org/2002/01/P3Pv1">
<!-- Expiry information for this policy -->
<EXPIRY max-age="86400"/>
<POLICY xml:lang="en">
        <!-- Description of the entity making this policy statement. -->

        <ENTITY>
                <DATA-GROUP>
                </DATA-GROUP>
        </ENTITY>

        <!-- Disclosure -->
        <ACCESS>
                <nonident/>
        </ACCESS>

        <!-- No dispute information -->

        <!-- Statement for group "Cookies" -->
        <STATEMENT>

                <EXTENSION optional="yes">
                        <GROUP-INFO name="Cookies"/>
                </EXTENSION>

                <!-- Consequence -->
                <CONSEQUENCE>
                        Cookies are used to track visitors to our site,
                        so we can better understand what portions
                         of our site best serve you.
                </CONSEQUENCE>

                <!-- Use (purpose) -->
                <PURPOSE>
                        <develop/>
                        <pseudo-analysis/>
                        <pseudo-decision/>
                        <individual-analysis/>
                        <individual-decision/>
                        <tailoring/>
                </PURPOSE>

                <!-- Recipients -->
                <RECIPIENT>
                        <ours/>
                </RECIPIENT>

                <!-- Retention -->
                <RETENTION>
                        <stated-purpose/>
                </RETENTION>

                <!-- Base dataschema elements. -->
                <DATA-GROUP>
                        <DATA ref="#dynamic.cookies" optional="yes">
                                <CATEGORIES>
                                    <content/>
                                    <navigation/>
                                    <purchase/>
                                    <uniqueid/>
                                </CATEGORIES>
                        </DATA>
                </DATA-GROUP>
        </STATEMENT>

<!-- End of policy -->
        </POLICY>
</POLICIES>
```

Listing B.1: *P3P policy*