Privacy of Mobile Users in Context-aware Computing Environments

Author:
Alireza Behrooz

Academic Supervisor:
Matei-Dimitri Ciobanu-Morogan

Industrial Supervisor:
Alisa Devlic

June, 2010
Abstract

This thesis provides a solution to address the difficulty of design and development of privacy-sensitive context-aware applications that can be used in peoples’ everyday life without the concerns regarding potential abuse. Users would like to be able to control who can access their contextual information, with what granularity, and in which situations. These users’ privacy preferences must be considered when distributing context information in a context-aware environment.

The current privacy policy languages have not been tailored to the specific requirements of context-aware applications, because they are mostly used to create static rules for personal information that rarely changes, while the situation of mobile users in a context-aware environment frequently changes and their privacy preferences must be updated accordingly. The thesis introduces Context Privacy Policy Language (CPPL) as a context-aware language that maps different situations of the context owner to a set of privacy rules that must be applied in the corresponding situation.

Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environment (MUSIC) provides an open technology platform that makes it technically and commercially feasible for the wider IT industry to develop innovative mobile applications which are context-aware, self-adapting, and inherently distributed. This thesis investigates, designs, implements, and evaluates a privacy management system based on CPPL, that is integrated to the context distribution architecture of the MUSIC project.

The evaluation of designed context privacy management system revealed that the MUSIC Context distribution service is highly scalable and can respond to hundreds of subscriptions per second and enabling the privacy adds a delay of 131µs (4 %) to the average subscription response time (3,263ms). Moreover, the scalability tests showed that the context-aware privacy management system is highly scalable in a sense that increasing the number of privacy rules from 1 to 1000 has a slight affect on performance of the system by increasing the context publication time from 284,55 ms to 287,34 ms (0,9 %).
Acknowledgement

It would not have been possible to write this thesis without the help and support of my industrial supervisor, Alisa Devlic, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject.

I would also like to thank my academic supervisor, Matei-Dimitri Ciobanu-Morogan, for his invaluable assistance, support and guidance.

Last, but by no means least, I offer my regards and blessings to my parents who supported me in any respect during the completion of my studies.
Table of contents

1 INTRODUCTION .................................................................................................................. 1
  1.1 BACKGROUND AND MOTIVATION ............................................................................. 1
    1.1.1 The MUSIC project ............................................................................................... 2
  1.2 PROBLEM STATEMENT ............................................................................................... 3
  1.3 GOAL ............................................................................................................................ 3
  1.4 PURPOSE ...................................................................................................................... 4
  1.5 METHOD ....................................................................................................................... 4
  1.6 AUDIENCE .................................................................................................................. 5
  1.7 LIMITATIONS .............................................................................................................. 5
  1.8 THESIS STRUCTURE ................................................................................................. 5

2 BACKGROUND.................................................................................................................. 7
  2.1 WHAT IS PRIVACY? .................................................................................................... 7
    2.1.1 The history of privacy ........................................................................................... 7
    2.1.2 Privacy Definition ............................................................................................... 8
    2.1.3 Types of privacy .................................................................................................. 9
  2.2 CONTEXT-AWARENESS ............................................................................................ 10
    2.2.1 What is context? ................................................................................................... 10
    2.2.2 Context-aware computing .................................................................................... 14
  2.3 MUSIC ARCHITECTURE ........................................................................................... 15
    2.3.1 Distribution Service ............................................................................................ 17
    2.3.2 Context Middleware ............................................................................................ 19
    2.3.3 Context distribution ............................................................................................ 20

3 MOTIVATION SCENARIOS AND REQUIREMENTS .......................................................... 23
  3.1 SCENARIOS TO HIGHLIGHT THE PRIVACY CONCERNS IN MUSIC ...................... 23
  3.2 PRIVACY REQUIREMENTS FOR CONTEXT-AWARE COMPUTING ENVIRONMENTS ...... 26

4 RELATED WORK ............................................................................................................... 29
  4.1 DISCRETE BOX AND PRIVACY MANAGER ............................................................... 29
4.2 INTELLIGENT PRIVACY AGENT ................................................................. 31
4.3 CONFab TOOLKIT .................................................................................... 32
4.4 CONTEXTWARE ....................................................................................... 34
4.5 COMPARISON OF STUDIED SOLUTIONS ............................................. 35

5 PRIVACY POLICY LANGUAGES ................................................................ 36
  5.1 P3P AND APPEL AS PRIVACY POLICY LANGUAGE ............................. 36
    5.1.1 How P3P Works? ............................................................................ 36
    5.1.2 How APPEL Works? .................................................................... 38
    5.1.3 P3P and APPEL for our scenarios .............................................. 40
  5.2 COMMON POLICY AS PRIVACY POLICY LANGUAGE ...................... 45
    5.2.1 Conditions .................................................................................. 46
    5.2.2 Actions and Transformations ....................................................... 48
  5.3 COMMON POLICY FOR OUR SCENARIOS ......................................... 48
    5.3.1 Common Policy strengths ............................................................ 52
    5.3.2 Common Policy Limitations ......................................................... 53

6 A CONTEXT-AWARE PRIVACY POLICY LANGUAGE ............................. 54
  6.1 REQUIREMENTS .................................................................................... 54
  6.2 EXISTING PRIVACY POLICY LANGUAGES ......................................... 55
    6.2.1 Houdini ....................................................................................... 55
    6.2.2 UbiCOSM .................................................................................... 56
    6.2.3 CoPS .......................................................................................... 57
    6.2.4 Context Privacy Engine ............................................................... 58
    6.2.5 SenTry ......................................................................................... 59
    6.2.6 Summary ..................................................................................... 59
  6.3 CPPL .................................................................................................... 60
    6.3.1 CPPL examples ........................................................................... 64

7 CONTEXT PRIVACY MANAGER ARCHITECTURE AND DESIGN ............. 66
  7.1 CONTEXT PRIVACY MANAGER ARCHITECTURE .................................. 67
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2 CONTEXT PRIVACY MANAGER DESIGN</td>
<td>69</td>
</tr>
<tr>
<td>7.3 CONTEXT PRIVACY MANAGER SOFTWARE ARCHITECTURE</td>
<td>72</td>
</tr>
<tr>
<td>7.4 CONTEXT PRIVACY MANAGER BEHAVIOR</td>
<td>74</td>
</tr>
<tr>
<td>8 CONTEXT PRIVACY APPLICATIONS AND TOOLS</td>
<td>77</td>
</tr>
<tr>
<td>8.1 UPDATING PRIVACY POLICIES</td>
<td>78</td>
</tr>
<tr>
<td>8.2 PRIVACY POLICY STORAGE</td>
<td>78</td>
</tr>
<tr>
<td>8.3 PRIVACY POLICY EDITOR TOOL</td>
<td>79</td>
</tr>
<tr>
<td>8.4 USER PRIVACY MANAGER TOOL</td>
<td>80</td>
</tr>
<tr>
<td>9 PERFORMANCE EVALUATION</td>
<td>83</td>
</tr>
<tr>
<td>9.1 SYSTEM BOUNDARIES</td>
<td>83</td>
</tr>
<tr>
<td>9.2 SERVICES PROVIDED BY SYSTEM:</td>
<td>83</td>
</tr>
<tr>
<td>9.3 MEASUREMENT DESCRIPTION</td>
<td>86</td>
</tr>
<tr>
<td>9.4 TEST SETUP</td>
<td>88</td>
</tr>
<tr>
<td>9.5 EVALUATION RESULTS</td>
<td>90</td>
</tr>
<tr>
<td>9.5.1 Subscription response time</td>
<td>90</td>
</tr>
<tr>
<td>9.5.2 Subscription throughput</td>
<td>91</td>
</tr>
<tr>
<td>9.5.3 Context publication time</td>
<td>93</td>
</tr>
<tr>
<td>9.5.4 Situational context publication time</td>
<td>96</td>
</tr>
<tr>
<td>10 CONCLUSIONS AND FUTURE WORK</td>
<td>98</td>
</tr>
<tr>
<td>10.1 CONCLUSIONS AND DISCUSSIONS</td>
<td>98</td>
</tr>
<tr>
<td>10.2 SUGGESTIONS FOR FUTURE RESEARCH</td>
<td>100</td>
</tr>
<tr>
<td>11 REFERENCES</td>
<td>101</td>
</tr>
</tbody>
</table>
## List of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overview of MUSIC architecture</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>Overview of the functionality offered by distribution service</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>The high-level view of MUSIC context middleware</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>Context distribution in MUSIC</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>The resource list returned by XCAP server</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>DISCREET BOX</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>USER PRIVACY MANAGER</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>Context-awareness architecture prototype and privacy components</td>
<td>32</td>
</tr>
<tr>
<td>9</td>
<td>AN INFOSPACE (CLOUDS) contains contextual data and tuples (Squares), being contained in infospace servers (rounded rectangles)</td>
<td>33</td>
</tr>
<tr>
<td>10</td>
<td>MAPPING GEOPRIV onto ContextWare architecture</td>
<td>34</td>
</tr>
<tr>
<td>11</td>
<td>A simple HTTP transaction with P3P added</td>
<td>36</td>
</tr>
<tr>
<td>12</td>
<td>Basic operations in common policy</td>
<td>66</td>
</tr>
<tr>
<td>13</td>
<td>CPM as an internal component of the context middleware</td>
<td>68</td>
</tr>
<tr>
<td>14</td>
<td>Structure of context middleware</td>
<td>69</td>
</tr>
<tr>
<td>15</td>
<td>Context Privacy Manager</td>
<td>70</td>
</tr>
<tr>
<td>16</td>
<td>ICONTEXTPRIVACYSERVICE INTERFACE</td>
<td>70</td>
</tr>
<tr>
<td>17</td>
<td>ICONTEXTPRIVACYMANAGEMENT INTERFACE</td>
<td>71</td>
</tr>
<tr>
<td>18</td>
<td>Internal structure of Context Privacy Manager</td>
<td>73</td>
</tr>
<tr>
<td>19</td>
<td>Behavior of the CPM after receiving a subscription request</td>
<td>74</td>
</tr>
<tr>
<td>20</td>
<td>Behaviour of Context Privacy Manager after a context changed event</td>
<td>75</td>
</tr>
<tr>
<td>21</td>
<td>System boundaries</td>
<td>83</td>
</tr>
<tr>
<td>22</td>
<td>Successful subscription without privacy and plug-in is activated</td>
<td>84</td>
</tr>
<tr>
<td>23</td>
<td>Publishing context updates to permitted context requestors</td>
<td>85</td>
</tr>
<tr>
<td>24</td>
<td>Subscription response time</td>
<td>87</td>
</tr>
<tr>
<td>25</td>
<td>Publication time</td>
<td>88</td>
</tr>
<tr>
<td>26</td>
<td>Test bed configuration</td>
<td>89</td>
</tr>
</tbody>
</table>
FIGURE 27.  THE AFFECT OF VALID PRIVACY RULES ON SUBSCRIPTION RESPONSE TIME.......................... 91

FIGURE 28.  THE SUBSCRIPTION THROUGHPUT FOR A FIXED NUMBER OF REQUESTS WITH DIFFERENT TIME INTERVALS ......................................................................................................................... 92

FIGURE 29.  THE SUBSCRIPTION THROUGHPUT FOR A BURST OF SUBSCRIBE MESSAGES ......................... 93

FIGURE 30.  THE CHANGE OF CONTEXT PUBLICATION TIME DUE TO THE CONTEXT MANAGER'S INTERNAL HANDLING OF CONTEXT CHANGED EVENTS .................................................................................................................. 94

FIGURE 31.  CONTEXT PUBLICATION TIME FOR DIFFERENT NUMBER OF VALID PRIVACY RULES ........ 95

FIGURE 32.  CONTEXT PUBLICATION TIME FOR DIFFERENT NUMBER OF SUBSCRIBERS ....................... 95

FIGURE 33.  THE AVERAGE CONTEXT PUBLICATION TIME FOR MULTIPLE SUBSCRIBERS ..................... 97
List of tables

TABLE 1. PRIVACY CONCERNS IN OUR SCENARIO ................................................................. 25
TABLE 2. THE COMPARISON OF RELATED WORK AGAINST OUR PRIVACY REQUIREMENTS ............ 35
TABLE 3. COMPARISON OF CONTEXT-DEPENDENT PRIVACY POLICY LANGUAGES .................. 60
TABLE 4. CONSTRAINT OPERATORS ..................................................................................... 62
TABLE 5. TEST BED HARDWARE SPECIFICATION ................................................................... 90
TABLE 6. SUBSCRIPTION RESPONSE TIME FOR A SINGLE REQUEST ...................................... 90
TABLE 7. CONTEXT PUBLICATION TIME FOR A SINGLE SUBSCRIBER ...................................... 94
<table>
<thead>
<tr>
<th>Listing</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Listing 1.</td>
<td>A P3P file showing policies of an online shopping website ........................................... 37</td>
</tr>
<tr>
<td>Listing 2.</td>
<td>An APPEL file representing the privacy preferences of a user ................................. 39</td>
</tr>
<tr>
<td>Listing 3.</td>
<td>XML file to define location as a new data schema in P3P ........................................ 41</td>
</tr>
<tr>
<td>Listing 4.</td>
<td>Referencing a data element inside P3P ........................................................................ 41</td>
</tr>
<tr>
<td>Listing 5.</td>
<td>Using the &quot;base&quot; attribute to reference a newly created data schema ...................... 42</td>
</tr>
<tr>
<td>Listing 6.</td>
<td>Embedded DATASHEMA element .................................................................................. 42</td>
</tr>
<tr>
<td>Listing 7.</td>
<td>Extension of the &lt;RECIPIENT&gt; element .................................................................. 43</td>
</tr>
<tr>
<td>Listing 8.</td>
<td>Bob’s Privacy Preferences in APPEL (privacy concern 3) ........................................ 43</td>
</tr>
<tr>
<td>Listing 9.</td>
<td>Bob’s Privacy Preferences in APPEL (privacy concern 2) ........................................ 44</td>
</tr>
<tr>
<td>Listing 10.</td>
<td>Bob’s Privacy Preferences in APPEL (privacy concern 1) ........................................ 44</td>
</tr>
<tr>
<td>Listing 11.</td>
<td>The XML schema of conditions element in Common Policy ....................................... 47</td>
</tr>
<tr>
<td>Listing 12.</td>
<td>A location based condition defined by Geolocation Policy ..................................... 47</td>
</tr>
<tr>
<td>Listing 13.</td>
<td>XML Schema of actions and transformations in Common Policy ................................ 48</td>
</tr>
<tr>
<td>Listing 14.</td>
<td>The common policy rule representing Privacy Concern 1 ....................................... 49</td>
</tr>
<tr>
<td>Listing 15.</td>
<td>The Common Policy rule representing Privacy Concern 2 ...................................... 50</td>
</tr>
<tr>
<td>Listing 16.</td>
<td>The Common Policy rule representing Privacy Concern 4 ....................................... 51</td>
</tr>
<tr>
<td>Listing 17.</td>
<td>The Common Policy rule representing Privacy Concern 5 ....................................... 51</td>
</tr>
<tr>
<td>Listing 18.</td>
<td>The Common Policy rule representing Privacy Concern 7 ....................................... 52</td>
</tr>
<tr>
<td>Listing 19.</td>
<td>XML schema representation of the root element of a CPPL policy ................................ 60</td>
</tr>
<tr>
<td>Listing 20.</td>
<td>A situation element representing Bob’s health emergency ..................................... 61</td>
</tr>
<tr>
<td>Listing 21.</td>
<td>An example of the use of the TimeConstraint element ........................................... 63</td>
</tr>
<tr>
<td>Listing 22.</td>
<td>XML schema representation of Ruleset element ....................................................... 63</td>
</tr>
<tr>
<td>Listing 23.</td>
<td>Privacy rule allowing Alice’s friends to access her city scope ................................ 64</td>
</tr>
<tr>
<td>Listing 24.</td>
<td>Using Expect element to exclude an individual from a group .................................. 64</td>
</tr>
<tr>
<td>Listing 25.</td>
<td>A CPPL example that allows full access to all requestors ....................................... 65</td>
</tr>
<tr>
<td>Listing 26.</td>
<td>The CPPL file representing Privacy Concern 1 ....................................................... 65</td>
</tr>
<tr>
<td>Listing 27.</td>
<td>A situational context .......................................................................................... 96</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background and motivation

Less than 20 years ago, Mark Weiser introduced a new radical perspective of computing and networking in his article “Computer for the 21st Century”[1]. He coined the term “ubiquitous computing” and envisioned invisible computers serving people in their everyday lives at home and at work, functioning invisibly and inconspicuously in the background. Over the last decade, a tremendous number of researches have been done to realize Wieser’s vision of ubiquitous computing.

Although ubiquitous computing offers a wide range of attractive possibilities, it introduces many new privacy risks because it decreases the cost of data collection for interested parties such as advertising companies and surveillance systems. In addition, the more invisible and extensive a computational system is, the more complex controlling the flow of information becomes. These concerns make the privacy one of the greatest challenges of ubiquitous environments.

Interestingly, the privacy has been a concern from the first day the concept of ubiquitous computing has been introduced as Wieser noted in his article:

*“Hundreds of computers in every room, all capable of sensing people near them and linked by high-speed networks, have the potential to make totalitarianism up to now seem like sheerest anarchy”*

However, surprisingly, ubiquitous researchers, even knowing the importance for privacy issues, have not actively pursued the privacy implications of their systems. To find the reason of this ignorance, Langheinrich [2] interviewed a number of active researchers involved in “Disappearing Computer Initiative” which was an European Union initiative funding 17 ubiquitous projects. The summarized opinions of researchers were the following:

- Working on privacy is not necessary at all because existing security mechanisms are adequate protection from privacy abuses.
Tackling the privacy issues is not necessary for research prototypes since they do not work in real world and real data.

Privacy issues will only become relevant when initial issues are solved.

Too complex of a problem to be solved technically

Although traditional security mechanisms such as anonymity and secrecy are important to protect the personal information of users, they do not cover many situations of every day life in which the owner of personal information would like to share his/her information with others, but wants to control who can access which data and in which situation.

Many research projects become the basis for real products through different stages where researchers hand over the prototype to product engineers. Developing and deploying privacy-sensitive systems considering the privacy principles at initial design, can maximize the real benefits while minimizing the risks. When the technologies become widespread, it becomes very difficult, if not impossible to change them.

1.1.1 The MUSIC project

Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environment (MUSIC)[3] is an integrated project founded by the European Commission under the Information Society Technology (IST) priority. The main objective of MUSIC is to provide an open technology platform that makes it technically and commercially feasible for the wider IT industry to develop innovative mobile applications which are context-aware, self-adapting, and inherently distributed. Applications developed by MUSIC platform, aimed primarily at mobile users, adapt their functionality and internal implementation mechanisms to highly dynamic user and execution context while maintaining a high level of usefulness across context changes. They address operational aspects such as security, dependability, etc. according to the user needs and circumstances.

The MUSIC project builds upon the basis of other European research projects such as: Middleware for Adaptive Applications (MADAM)[4], Semantic Interfaces for Mobile Services (SIMS) [5], and Middleware platform for development and deploying advanced mobile services (MIDAS)[6].
1.2 Problem Statement

During last decades, many researchers have worked in context-aware computing, but as mentioned in Section 1.1, the majority of the researchers have ignored the importance of protecting the privacy of users, and as a result some projects are rejected by users because of privacy concerns [7][8]. The difficulty of designing and developing privacy-sensitive context-aware applications by application developers and expressing the privacy preferences by users must be addressed to ensure that context-aware technologies can be used in peoples’ everyday life without the concerns regarding potential abuse.

There have been various approaches to tackle the issue of privacy in context-aware environments. Some researches, such as the Context Toolkit [9], PARCTab system[10], and iROS [11], provide support for building context-aware applications, but do not provide features for managing privacy. Some others, such as [12][13] propose the separation between the identity and contextual information, protecting the privacy by hiding the identity of users. Using this approach the owner of context information cannot share his/her information with others.

The latest attempts to address the issue of privacy in context-aware environments, such as[14][15][16][17], are mostly policy-based privacy frameworks which control the flow of context information based on the privacy policies specified by users. However, the proposed approaches do not support context-aware privacy preferences, where the users' preferences depend on their current context. In highly dynamic environments that the context changes very fast, the large user's involvement might affect the usability of the applications because they will have to frequently change their preferences as their situation change. Therefore, there is a need for a privacy management system which dynamically adapts the privacy preferences of users based on their context information.

1.3 Goal

The goal of this research is to identify the privacy requirements for context-aware environments, discuss shortcomings of existing context-aware systems; and as a result design, implement, and evaluate the performance of a new solution that addresses these shortcomings. The proposed context-aware privacy system is implemented as part of the
MUSIC framework, and protects the user's personal integrity and privacy when sharing the user's sensitive data. However, since it is important to deliver target context to a user in a timely fashion (i.e., before the current context changes), the cost of using these privacy policies has to be minimized. Therefore, another goal of this thesis is to determine this cost and show if this cost is acceptable, thus enabling delivery of the restricted target context to the requested user in a timely fashion.

1.4 Purpose

Having a context-aware privacy management system that dynamically adapts to the current user's context will enable the application developers to protect users' privacy. Users can define their privacy preferences using context-aware privacy policies. Additionally, this privacy system will provide users a graphical tool to add, delete, and modify their context privacy rules at runtime without interrupting the application execution.

1.5 Method

In order to understand the theoretical background of the problem and identify requirements for enabling privacy in context-aware environments, a literature review was performed to elaborate three concepts: context, privacy, and context-awareness.

Next, the existing solutions were studied and evaluated based on the identified privacy requirements. We also compared some existing privacy policy languages based on several privacy scenarios.

Based on the privacy requirements and analysing previous works, we designed our privacy manager and introduced a new policy language. Then we implemented the solution and integrated it into the MUSIC context middleware. Additionally, a simulation tool was developed to test the privacy component.

Finally, we did performance measurements to test our system scalability and to evaluate the cost of adding privacy component to the context distribution service.
1.6 Audience

The targeted audience of this thesis is mainly researchers seeking solutions for the privacy concerns in context-aware environments. Another targeted audience is the MUSIC application and tool developers interested to know more about the theory behind the Context Privacy Manager component of MUSIC context middleware.

1.7 Limitations

While the goal of this thesis is providing a privacy solution which fulfills the privacy requirements of context-aware environments, there are some limitations that might affect the design of such solution. The fact that our privacy solution must be designed as a part of MUSIC middleware imposes a number of limitations:

- Privacy and security is not the main focus of MUSIC project, and therefore the privacy will be considered as an optional feature of context middleware.
- The context model is limited to the MUSIC context model which might possibly prevent us from defining a new context model suitable for privacy protection.
- The investigation about the privacy feature started at the latest phase of MUSIC project, which imposed some limitation due to time and resource constraints.

1.8 Thesis structure

The rest of this report is structured as follows:

**Chapter 2**: Gives an overview of privacy, context, and context-aware computing. Then it describes the MUSIC architecture and the specifically the components involving in distributing context information.

**Chapter 21**: Provides some scenarios to highlight the privacy concerns of the mobile users of context-aware applications. Then, based on provided scenarios, it discusses the high level requirements to design a privacy framework for context-aware environments.

**Chapter 4**: Describes four related works in the area of privacy of context-aware applications, lists the advantages and shortcomings of proposed approaches, and analyzes them against the privacy requirements described in Chapter 3.
Chapter 6: Studies the current privacy policy languages and discusses the two most commonly used languages, P3P and Common Policy. Next, it describes CPPL as a new privacy policy language that was created having the limitations of the current languages in mind.

Chapter 7: Describes the architecture, design, structure and behavior of the context-aware privacy management system.

Chapter 8: Describes the functionalities provided by the context privacy manager component and describes the tools and applications that can employ these functionalities to help protecting their users’ privacy.

Chapter 8: Evaluates the context-aware privacy management by measuring different metrics regarding the context distribution with and without privacy and compares the measurement results.

Chapter 10: Presents conclusions and suggested future work in the area.
2 Background

2.1 What is privacy?

There is no straightforward answer to this question. There are many definitions of privacy from different outlooks such as law, sociology, human right, etc. In this section, we provide a historical view on privacy. We discuss different aspects of privacy based on the existing definitions in literatures. Understanding different perspectives of privacy will help us identifying potential privacy risks and finding suitable approach to address them.

2.1.1 The history of privacy

The legal protection of privacy had been considered in some western countries from hundreds years ago. According to Langheinrich[2], the first reference to privacy in common law is the English Justices of the Peace Act of 1361 in which eavesdropping was considered a crime. In 1776, the Swedish Parliament ordained that all government-held information be must used for legitimate purposes. In 1858, France prohibited the publication of private facts. In 1889, the Norwegian Criminal Code prohibited the publication of personal or domestic affairs [18].

However, in most privacy literature, a well-known 1890 article by Samuel D. Warren and Louis D. Brandeis entitled “The Right to Privacy”[19] is cited as one of the first articles that defines the privacy. Warren and Brandies defined privacy as “The right to be alone”. Their work was motivated by new technologies such as photography that could compromise the personal life of people.

“Instantaneous photographs and newspaper enterprise have invaded the sacred precincts of private and domestic life; and numerous mechanical devices threaten to make good the prediction that “what is whispered in the closet shall be proclaimed from the house-tops.”[19]

The modern privacy benchmark is the Universal Declaration of Human Rights, adopted by the United Nations in 1948, which states in its Article 12 that:
“No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honour and reputation. Everyone has the right to the protection of the law against such interference or attacks” [20]

There are various regional and international treaties that recognize the privacy as a right [21][22]. But all of them prohibit the unlawful interference with the privacy of individuals without giving a precise definition of privacy.

2.1.2 Privacy Definition

Different disciplines have their own definition of privacy and their meaning varies. As there is no universal definition of privacy, we can analyse the existing definitions to figure out which one is more appropriate for our research area.

According to Altman in his book “The Environment and Social Behavior”, definitions of privacy can be divided into two groups [23]. First group emphasizes isolation, seclusion, withdrawal, and avoidance of interactions. For example:

“A value to be oneself-relief from pressure a pressures of the presence of others” [24]

“Avoiding interaction and intrusion by means of visual, auditory, etc. channels and combinations thereof” [25]

Another group of definitions emphasizes the control of openness of the self to others and freedom of choice regarding personal accessibility. For example:

- Freedom to choose what, when and to whom one communicates [26][27]
- Personal control over personal information [28][29]

It is noticeable that the definitions in the first group are relatively older than the second group. This might be because of the fact that new technologies made it easier to control the level of individual’s interaction to the society; therefore privacy was not about isolation from society, but having the freedom to communicate and share personal information with selected individuals without a fear that someone who is unauthorized will obtain and misuse their personal information.

The most common definition especially in IT researches is the one that Alan Westlin in his book “Privacy and Freedom” introduced:
"Privacy is the claim of individuals, groups and institutions to determine for themselves, when, how and to what extent information about them is communicated to others." [28]

This definition is currently the most common definition especially in IT research, and it will be considered as the definition of privacy in the scope of this research.

2.1.3 Types of privacy

From a theoretical point of view, privacy can be seen in different ways. In general, most of the researchers has analysed the privacy as interactive condition of person and environment. For example, Westlin has provided a systematic analysis of privacy from this perspective. He has introduced four types of privacy [28]:

- **Solitude**: person is alone and free from observation by others.
- **Intimacy**: a small group separate themselves from outsiders in order to be alone.
- **Anonymity**: when a person is lost in the crowd
- **Reserve**: establishment of psychological barriers against intrusion.

This categorization shows how different social units are involved in privacy. Privacy is a personal affair that dynamically changes based on the situation and circumstances. Sometimes we might need just to be alone (Solitude), but we as human can not isolate ourselves from other people all the time and should interact with our friends, family, etc (Intimacy). In some situations, despite of physical presence of a person in a crowd, person privacy should be protected in a sense that others are not engage in more causal interaction (anonymity). The importance of anonymity has been increased dramatically with the growth of internet and online interactions and social activities. Finally, one of the most interesting types of privacy is the one that a person is not interested to be overwhelmed by interactions from outside his or her personal boundaries. Westlin has defined Reverse privacy as a psychological state that someone is listening to others but psychologically ignores them. This kind of privacy might not be a big concern at the time that Westlin defined it, but today this conception of privacy is more important because of the different kind of electronic spams that might compromise the privacy of people.
From a practical point of view, the concept of privacy has the following aspects:

- **Territorial privacy**: Protection of domestic and other physical area surrounding a person such as home, workplace or public space.
- **Bodily privacy**: Protection of a person against undue interference such as genetic tests, physical searches, drug testing or information violating his/her moral sense.
- **Information privacy**: establishment of rules to control how personal data can be gathered, stored, processed, or selectively disseminated.

Back in the 18th century, the protection of home and personal territory was the most common aspect of privacy. This aspect is known as territorial privacy now, and has the same importance as bodily privacy. These aspects together are also known as physical or local privacy. The information privacy as defined above is equivalent to Westin's definition of privacy described in Privacy Definition.

One of the most important aspects of privacy which is not mentioned in most of the literature is “communication privacy” which covers the security and privacy of mail, telephones, e-mail, and other forms of communication. Development of telecommunication and computer technology considerably facilitated and expedited the possibilities to communicate, thus it has increased the amount of sensitive personal information sent in computer networks and telecommunication systems. Therefore the communication privacy or “data protection” turned into one of the most significant aspects of privacy.

### 2.2 Context-awareness

#### 2.2.1 What is context?

In order to understand the concept of context-awareness, we need to define what the context is, and how it can be used to protect user's privacy. In this section, first we review the various definitions of context. Then we summarize the attempts to categorize the context in current literature.
2.2.1.1 Definition of context

The following definitions from dictionaries provide a general understanding of context:

- “the circumstances that form the setting for an event, statement, or idea”[32]
- “the interrelated conditions in which something exists or occur”[33]
- “That which surrounds, and gives meaning to, something else”[34]

Context is also defined in different fields of science and technology such as artificial intelligence, computer vision, and natural language processing. In context-aware computing, the concept of context is of great importance and traditional definitions are inadequate for designing context-aware applications.

At the early age of mobile computing, context was typically defined as an enumeration of examples. In 1994, first definition of context was proposed by Schilit et al [35]. Although not defined explicitly, they referred to context when defining context-aware applications as applications that adapt according to the user's current location, nearby people, and devices in his/her vicinity. Similarly, Brown [36] defines context as location, time of day, season of the year, and temperature. These definitions are very specific and not applicable to other domains and systems. For example, Brown definition is not applicable to a system in which the behaviour of applications depends on the user activity.

The first attempt to propose a formal definition which is generalizable was made by Rodden et al [37]. They defined the context as:

“The devices relationship with the technical infrastructure, the application domain, the socio-technical system in which it is situated, the location of its use and the physical nature of the device”

This definition is application specific and can possibly be interpreted as the application’s settings. Some others [38][39] have defined context as user’s environment rather than application’s environment. Schmidt et al [40] proposed a three dimensional space with dimensions: Environment, Self, and Activity. Their definition was based on general definition of context in dictionaries. These definitions are more general than a
simple enumeration of examples, but they only provide a conceptual synonym for context and it is practically hard to identify the context based on this type of definitions.

The most popular definition of context which is widely used in different literature is defined by Dey and Abowd in [41]. They reviewed the previous definitions and proposed a new definition:

“Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.”

In this thesis, we also follow Dey’s definition because it is general enough to be functional in different application domains and it is widely used as the base definition of context in most of context-aware research projects in context-aware computing.

However, some authors have claimed that the Day’s definition is not precise enough and therefore proposed new definitions. For example, Chen and Kotz [42] believe that a comprehensive definition of context should include both active and passive context. The former influences the behaviour of an application while the latter is relevant but not critical to an application. For example, in a GPS application, the location is active context because it determines the behaviour of the application, but the temperature is a passive context that can be displayed to inform the user. The following statement outlines their understanding from context:

“The set of environmental states and settings that either determines an application’s behaviour or in which an application event occurs and is interesting to the user”

During development of context-aware applications, we realized that context is a changing concept because by increasing the capability of mobile devices to sense more environmental and user characteristics, consequently researchers started to include this new kind of information into their definition of context.

From another perspective, a definition of context can be specific or general. The former refers to certain type of context information (i.e location, temperature) even implicitly or explicitly while the latter defines context as a general concept. There is a trade-off between choosing a definition. More specific definitions make it easier to understand and implement context in a specific domain, but it might not be applicable to
other domains. On the other hand, general definitions are applicable to more domains, but harder to implement.

### 2.2.1.2 Categories of context

When Schilit et al. [43] defined the context for the first time; they indicated three important aspects of context as where you are, who you are with, and what resources are nearby. But this categorization could not be useful for application designers because it does not cover some essential contexts like activity and time. Therefore, researchers tried to categorize context in a way that leverages the design and development of context-aware applications.

There are two different approaches to categorize context types. First approach is based on characteristics of context information and typically indicates the context types which are used more frequently than others. Second approach categorizes the context based on the entity that context information belongs to.

Ryan et al. [44] introduce context types of *location, environment, identity, and time*. Dey and Abowd [41] suggest a similar categorization in which status (or activity) is mentioned rather than environment. They believe that environment is a synonym for context, while activity answers a fundamental question of what is occurring in the situation.

The original categorization of context is introduced by Schilit [35]. He divides context into three categories: *computing context, user context, and physical context*. Cohen et al. [42] proposed to add time as a fourth context category.

The most recent categorization of context is done by Marc et al. [45]. They suggest five categories of context: *environmental, user, hardware, temporal, and geographical*. *Environmental* context is the information captured by sensor from external environment. *User* context is derived from user interaction with application. *Hardware* context gives information on available resources or hardware specification of system. *Temporal* context preserve and manage the history. And *geographical* context represents the geographical information such as GPS data, horizontal/vertical moving, speed etc.
Since our focus in this work is proposing a privacy framework to preserve the privacy of users in context-aware environments, we distinguish between contexts whose access is controlled by the privacy framework (i.e., the *sensitive context*) from contexts which are used to determine a user's situation (i.e., the *situational context*).

### 2.2.2 Context-aware computing

Having a clear definition of context, effectively using the context in mobile computing is a challenge which is addressed by context-aware computing. Although the first work in context-aware computing is *Olivetti Active Badge* in 1992[46], the first definition of context-aware computing was proposed by Schilit et al [35] in 1994:

> “Context-aware computing is the ability of mobile user's applications to discover and react to changes in the environment they are located in.”

Schilit categorized the context-aware applications to four categories [43]:

- **Proximate selection**: applications that emphasize the nearby located objects or make them easier to be chosen. For example, showing nearby printers and order them by distance from user.

- **Automatic contextual reconfiguration**: automatically adding new components to a system, remove existing component, or alter the connection between components based on context.

- **Contextual information and commands**: applications that can produce different results or execute different commands based on the current context.

- **Context triggered actions**: simple IF-THEN rules used to specify how context-aware systems should adapt.

Pascoe [47] introduced a set of core generic capabilities to describe context-awareness:

- **Contextual sensing**: applications detect various environmental states and presenting them to the user.

- **Contextual adaptation**: applications adapt their behaviour to the current context information.
• **Contextual resource discovery**: applications use context information to discover available resources and exploit them. For example, a mobile device with limited display capabilities may discover an unused display screen and can temporarily use that screen to display information on.

• **Contextual augmentation**: associating digital data with a particular content that is related to or execution of a process in a specific situation.

Although Schilit and Pascoe have developed their taxonomies from two different perspectives, we can find a mapping from generic capabilities in Pascoe’s taxonomy to corresponding categories in the Schilit’s classification. Reviewing these taxonomies, Dey [48] proposed a categorization that combines the ideas of Schilit and Pascoe and includes a list of context-aware features that context-aware applications may support: 

- presentation of information and services to a user;
- automatic adaptation of an application execution based on the current context;
- tagging of context to information for later retrieval.

In the scope of this research, we define the context-aware computing as defined in MUSIC project [49]:

“Context-aware computing is an area which studies methods and tools for discovering, modelling and utilizing contextual information."

### 2.3 MUSIC architecture

The MUSIC platform has a layered architecture with the intention of portability and separation of concerns between different layers. It consists of two main building blocks, the *MUSIC Studio* and the *Runtime environment*. In the following, we provide a high-level description of the components depicted in Figure 1
The MUSIC Studio contains a set of tools and software libraries which provide support to application developers for building their applications. These tools are essential for linking the applications with MUSIC platform. In addition, they can be used for simulation, testing, and tuning.

The Application layer of the MUSIC platform is responsible for the management of MUSIC applications. It can be seen as an interface between the third party MUSIC-enabled applications and the MUSIC middleware.

The Middleware layer collects a set of services providing the core capabilities of the MUSIC middleware. The Context Middleware is responsible for collecting, organizing, managing and sharing the context information and making it available to adaptation.
middleware and context-aware applications. The *Adaptation Middleware* analyzes the changes of the context and its impact on the application(s) in order to adapt the set of running applications to the current circumstances (e.g. resource situation). The *Profile Manager* provides support for a dynamic platform configuration to optimize the scarce resources of mobile devices.

The *System Services* layer includes the following system-level services:

- **Communication service**: exports remote-enabled services at the service provider side and binds to these services at the client side.

- **Distribution service**: provides the required transport infrastructure for efficiently disseminating arbitrary information types among various devices and MUSIC middleware instances.

- **Resource Management service**: manages the low-level resources available in an adaptation domain.

- **Security Management**: provides a middleware-level security management for the middleware services.

The *Core Environment* layer is the backbone of the MUSIC platform. It contains a set of services, which provide the low-level operations to deploy and to easily retrieve the various kind of services (either middleware or application) hosted by the MUSIC platform, and the *Information Model*, which defines the MUSIC information model which provides the information about the data structure of a data type as well as the relationship between the data types.

### 2.3.1 Distribution Service

Based on the MUSIC architecture specification [50], the DS must be a system service for the exchange of arbitrary information types between networked hosts. Therefore, it is decoupled from the middleware and implemented as a library transfer any kind of information between MUSIC nodes.

Figure 2 illustrates a high level functionality provided by the DS. In this example, an application running on Alice’s device needs to have access to Bob’s calendar information and Charly’s location information.
Figure 2. **Overview of the functionality offered by distribution service**

The MUSIC middleware running on Bob’s and Charly’s devices retrieves the calendar and location information respectively through context providers installed on their devices. These context providers are MUSIC context plug-ins which act as wrappers around hardware and software sensors to sense the raw context information. The DS running on Bob’s and Charly’s devices is leveraged to make information provided by context providers available to other MUSIC nodes (e.g. Alice’s device).

The MUSIC middleware running on the Alice’s device leverages the DS to obtain the context information from the devices that produces this information.

In described scenario, the Bob’s and Charly’s devices are producer devices and Alice’s device is a consumer device.

The DS provides the following functionalities:

1. **Registration**: registers the type of available information provided by the context-providers (e.g. context plug-ins) to a public registrar.

2. **Resource location**: discovers the available types of information provided by other nodes in the network.

3. **Distribution**: enables the distribution of actual information from consumer to producer.

In the given example, the DS on Bob’s and Charly’s devices registers the calendar and location information as the provided information types in the context distribution platform. Alice’s DS discovers the availability of Bob’s calendar information and
Charly’s location information and subscribes to these information types resulting the delivery of the actual information producer device (Bob’s or Charly’s) to consumer device (Alice’s).

2.3.2 Context Middleware

In MUSIC the contextual sensing and resource discovery is provided by context middleware and adaptation and augmentation is provided by adaptation middleware. The context middleware is composed of several components as shown in Figure 3.

![Context Middleware Diagram](image)

Figure 3. The high-level view of MUSIC context middleware

The main component is Context Manager (CM) which coordinates the other components and manages the context providers (i.e., the context plug-ins) and the context consumers (i.e., context-aware applications). It is also responsible for modelling context data, managing context events, and serialization of context data.

The Context Query Processor (CQP) processes the context queries received by the CM. The Context Repository component allows persistent storage and retrieval of context elements. The Context Distribution Manager (DM) component leverages the DS
to access context information provided by DM component in context middleware of other MUSIC nodes. DM acts as a sink component consuming other components offered services and communicate with DS.

When an application requires to retrieves context information, it queries the CM. Consequently, CM leverages the CQP to check if the requested context information is locally available. Otherwise, the DM will be triggered to find remote nodes that provide such context information through DS and subscribe for them to be notified about the new values whenever the queried context type changes on the remote nodes.

### 2.3.3 Context distribution

In order to understand how DM and DS coordinate to distribute the context information among the MUSIC nodes we describe the steps to be performed to distribute a context data from node A to node B:

1. The CM on node “A” detects the Location Context Plug-in as it is deployed and installs it.

2. The CM informs the DS about all context types that become available. When a new plug-in is installed, the CM registers the provided context types through DS.

![Context Distribution in MUSIC](image)
3. As soon as a new register call is made to the DS generates a metadataId which uniquely identifies the context information type (e.g. location.country ). The generated metadataId is sent to the XCAP server as an HTTP PUT request.

4. The XCAP server updates the global resource list in which a mapping between nodes and context types that they provide is stored. Then it user the inter-process communication to invoke a method at the SIP server that notifies watchers about the changes in the resource list. However, in our example, at this point, there are no watcher subscribed to changes in resource list.

5. On Node B, the application issues a context query to the CM.

6. After determining that the context cannot be retrieved locally, the CM issues a request to DS to search for remotely available sensors.

7. The DS sends a HTTP GET request containing the metadataId of the requested context to the XCAP server.

8. The XCAP server returns the list of nodes providing the requested context type as a resource list. Figure 5 shows the resource list returned by XCAP server.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns="urn:ietf:params:xml:ns:resource-lists"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <list>
    <entry uri="sip:alpha@music.org"
      metadata_id="location.country" />
  </list>
</resource-lists>
```

Figure 5. The resource list returned by XCAP server

9. The CM gets informed about the nodes (if any) that provide the requested information type. If this resource list is empty, the nodes will subscribe to receive notifications about changes in resource list (i.e., when a new context plugin gets available or the existing ones get removed from this resource list).

10. The CM selects a subset of these sensors and sends it to the DS.

11. DS uploads the so called private resource list to the XCAP server using the HTTP PUT method. XCAP server serves as an XML repository of documents which can be accessed by multiple entities. It is used by the DS to maintain and obtain public and private resource lists that contain all available nodes providing
a particular context type and selected subset of these available nodes, respectively.

12. The DS on node B sends a subscription message to the SIP server. The SIP:Subscribe request contains the URI of the private list corresponding to the requested context type.

13. The SIP server processes the incoming subscribe request and sends individual subscription messages to all sensors selected by node B in step 10. In our example only the sensor on node A is available and selected.

14. On node A, once a SIP SUBSCRIBE request is received from the SIP server, if the plug-in providing the requested context type is not activated, an activation command is propagated to location sensor plug-in on Node A.

15. As soon as the plug-in is activated, it enables the context flow from the GPS HW to the CM, and from there to the DS.

16. When the location context data changes, the location context sends a context changed event to the CM. The rate of sending this event depends on the implementation of context plug-in and the actual context change rate.

17. Context middleware sends a notification to the DS, containing the value of context information that has been changed.

18. The DS sends a SIP NOTIFY to the SIP server.

19. The SIP server gets all the NOTIFY messages (which is only one in our example) from different nodes, aggregates them into a single aggregated NOTIFY (according to RFC 4662 [51]) and sends the result back to the DS of node B, which initiated a query for this context.

20. The DS on node B notifies the context distribution manager about the incoming notify message, which extracts the value of context information and delivers it to the context middleware,

21. The CM sends the value of context information to the application.
3 Motivation Scenarios and Requirements

In this chapter, first, we present some user scenarios that motivate the need for context-aware privacy in daily lives of average mobile users; then, based on the described scenarios and studied literature; we specify the privacy requirements in context-aware computing environments.

3.1 Scenarios to highlight the privacy concerns in MUSIC

In this chapter we present four scenarios that highlight the privacy concerns of mobile users in their everyday lives. The scenarios described here take place in Paris and Alice and Bob are two mobile users that carry MUSIC-enabled devices to benefit from the MUSIC context/aware applications in their daily life.

In the first scenario, Bob, an elderly man with sight and walking difficulties, decides to make a travel to Hôpital Cochin hospital by metro. He needs assistance to get in and off the train. Upon the entrance of Bob to the station, RATP, the major Paris transit operator, offers him to install a MUSIC application, called RATP Travel Assistant, which can help him during his journey. Bob accepts and installs the application. Application requires from Bob to input the desired destination, then suggests him the best itineraries. The application has a GUI that can be used for configuring a user’s privacy preferences. Bob can decide how his location should be available to RATP agents, to what extent, and for how long. In our scenario, Bob decides to give his exact location to RATP agents, while he is in a confirmed itinerary, without revealing his real identity to RATP agents. If he cancels his journey and chooses a new itinerary, his privacy preference will automatically be replaced by a new one based on the new itinerary. When he arrives to intermediate or destination station and needs assistance, the station agent will be notified and receives Bob’s information along with his location. If the agent is not available, an automated component will be used to highlight the closest elevators, escalators, and exit routes in the station.
In the second scenario, Bob has installed a MUSIC application, called *Health Terminal*, on his device that collects body temperature, heart rate, blood pressure, and body motion, making these data available to external requestors. A *Healthcare Institution (HCI)* collects all the important health information of the people using the *Health Terminal* application. When a health emergency situation occurs (e.g., the body temperature exceeds a predefined threshold), the *Health Terminal* will detect it and Bob’s *health status* will be changed from “normal” to “emergency”. The nearest nurse to Bob will be called by HCI. She will read the sensor data and decide to visit Bob. Bob will be notified about Alice visiting him in 15 minutes. This time is calculated based on Alice and Bob location and their average speed of motion.

In the third scenario, *Alice* is a nurse working for HCI. She carries a mobile device having a MUSIC application, called *HCI Assistant Console*. HCI centre can locate Alice whenever she has the device, but only during working hours which is stored in Alice’s calendar. Alice visits patients in their homes, during which time her company can locate her with an average accuracy of 100 meters. Her company uses this information to inform patients of her likely arrival times and to maintain her schedule.

In the fourth scenario, Alice has installed a MUSIC application, called *Friend Finder*, on her device that provides social networking and is integrated with her Facebook account. Alice usually synchronizes her calendar with her husband. She agrees to give him access to her current activity, (e.g., in a meeting, driving, etc.) while she is in Paris. When Alice is on vacation, (which can be inferred from her calendar), she wants to inform her friends about the city she is visiting. She is willing to share her location information at the street level to her friends on weekends so they can find each other and go out together.

From the mentioned scenario, the following privacy concerns could be identified:

- **Privacy Concern 1:** Bob’s health information should only be sent to HCI and HCI does not have permission to send Bob’s information to 3rd parties.

- **Privacy Concern 2:** Bob does not want to disclose his location when his health status is normal. His location should only be available to HCI when health terminal detects an emergency situation (detected by high blood pressure, high
body temperature, or high heart rate). The normal range of these health parameters is defined in the *Health Terminal* application and might be changed by a user based on his/her health profile.

- **Privacy Concern 3:** Bob would like that RATP agents can track him only during his journey in Paris metro. He would like to use RATP service without revealing his identity.

- **Privacy Concern 4:** Alice would like that HCI centre can track her only during working hours with a resolution of 100 meters.

- **Privacy Concern 5:** Alice would like to inform her husband about her current activity only while she is in Paris.

- **Privacy Concern 6:** Alice would like to share her location at the "city" granularity to her friends while she is on vacation.

- **Privacy Concern 7:** Alice would like to share her location with her friends at the "street" level on weekends.

Table 1 describes privacy concerns in our scenario. One of the objectives of this thesis is to define a privacy policy language which is machine readable, to enable the MUSIC application developers to specify their context-aware privacy policies that will be enforced by the privacy manager component of MUSIC middleware.

<table>
<thead>
<tr>
<th>Concern</th>
<th>Disclosed Info</th>
<th>Recipient</th>
<th>Circumstance</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concern</td>
<td>Bob's health info</td>
<td>HCI Centre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concern</td>
<td>Bob’s location</td>
<td>HCI</td>
<td>Emergency</td>
<td>Building</td>
</tr>
<tr>
<td>Concern</td>
<td>Bob’s location</td>
<td>RATP Agents</td>
<td>Traveling</td>
<td>Exact</td>
</tr>
<tr>
<td>Concern</td>
<td>Alice’s location</td>
<td>HCI Centre</td>
<td>Working</td>
<td>100 meters</td>
</tr>
<tr>
<td>Concern</td>
<td>Alice’s activity</td>
<td>Her husband</td>
<td>Paris</td>
<td>Activity</td>
</tr>
<tr>
<td>Concern</td>
<td>Alice’s location</td>
<td>Friends</td>
<td>Vacation</td>
<td>City</td>
</tr>
<tr>
<td>Concern</td>
<td>Alice’s location</td>
<td>Friends</td>
<td>Weekend</td>
<td>Street</td>
</tr>
</tbody>
</table>

Table 1. **Privacy concerns in our scenario**
3.2 Privacy requirements for context-aware computing environments

As mentioned in Section 2.1.1, privacy has been always a social concern for people. By growing the communication technologies during last century, more concerns have been raised. In context-aware computing environments, the privacy and trust aspects are more important because of the possibility of automated and invisible collection and processing of personal information. According to Lawrence Lessig[52], practical privacy is shaped by four strongly interacting forces: markets, social norms, legislation, and technology. All of these forces should be considered to determine the privacy requirements. In this chapter, the privacy requirement of a context-aware system from the perspective of the end users is discussed.

In order to specify the user needs regarding privacy a wide range of source were studied. We did not limit ourselves to the technical literature and studied other sources such as European Union Directive on data protection [53], and Consumer Privacy Protection Act of 2005 [54] as well as some publications about the experience and feedback of context-aware systems deployed in real environments [55][56][57]

The first and most important user requirement is the usability of the system in a sense that users might want to have control over to whom they want to disclose their context information and in which circumstances. The lack of control might let users think that their information is being monitored and misused by others. Jiang et al[58] have introduced a model for privacy in context-aware computing in which they have categorized the control mechanisms to three themes: prevention, avoidance and detection. The goal of prevention theme is to ensure that the undesirable attempt to access context data does not occur by reducing the accuracy of data to an acceptable level (i.e., video data can be masked to prevent recognizing a person), or by eliminating privacy risky operations. The avoidance theme aims to minimize the risk while maximizing benefits of sharing data with others. In this theme, users must predict who might want to use their personal information and then set the access privileges accordingly which raise the issue of updating the permission as the situation changes. In detection theme, users reveal the context information to everyone but some detection mechanisms are in place to detect the abuses. For example, a users might allow
everyone to have access to his/her location, but by using a detection mechanism identifies if someone is continuously tracking his/her location. Detection theme is useful in cases where openness and availability are more important than complete enforcement of access.

Two lessons can be learned from the themes introduced by Jiang et al. First, in avoidance theme, in order to update the access privileges when situation changes (which is very common for context-aware application), the design of context-aware systems should contain a mechanism to update the access privileges automatically when the situation changes. Second, in detection theme, an appropriate notification and feedback mechanism is required to detect and report abuse.

During last decades, context-aware computing systems have been designed and experimented. The proposed systems have either a centralized architecture in which the user data is stored centrally and can be accessed by others or a decentralized architecture in which keeps the users’ data on the mobile devices. While the centralized solutions have many benefits, users which are the real owners of their sensitive data have limited control over it. Even if very strong security and privacy protection mechanisms are enforced on the data, a problem of trust between end-users and system administrators which have access to central data remains. Moreover, the centralized servers are attractive targets for attackers. On the other hand, a fully or partially decentralized architecture, if designed properly, has the advantage of keeping the sensitive data at the user’s device as long as the owner of those data likes to keep them secret. Therefore, having a decentralized architecture must be considered as an important requirement when designing a context-aware system.

The users are interested to have a full control over the precision of their disclosed personal information. For example, in the fourth scenario in section 3.1, Alice wants to show his location information with the precision of city to her friends. The privacy framework should provide the possibility to specify privacy preferences that allows controlling the granularity of disclosed context information.

In summary, the following requirements must be considered when designing a context-aware system:
- **Usability**: Users should have simple control over their personal data.

- **Dynamic privacy preferences**: Privacy preferences of user should automatically adapt to the current user's situation.

- **Notification and feedback**: Users should have the option to get notified when someone accesses their sensitive information.

- **Decentralized architecture**: The users' personal information should not be collected and processed centrally.

- **Fine-grained access**: Users must be able to control the granularity of disclosed information to others.
4 Related work

In this chapter, we provide an overview of existing solutions and discuss their advantages and limitations. Then we compare them based on the requirements described in Section 3.2. There has been a great deal of work in the area of context-aware computing, but the majority of them ignored the issue of privacy. In the following sections we compare four solutions that have addressed the privacy issue.

4.1 Discrete Box and Privacy Manager

The Discreet Box is introduced in [59]. It proposes a middleware architecture that acts as a three way mediator between the users, the service providers, and the privacy regulations. It has a centralized architecture that allows various parties to communicate while enforcing the privacy regulations.

Figure 6. Discreet Box

Figure 6 shows the architecture of the Discreet Box. The Discreet Box is a privacy proxy installed at the service provider side but controlled by a Privacy Authority whose responsibility is supervising the observance of privacy provisions. The operation of
Discreet Box is similar to a proxy server. The application server submits requests for personal data to the Discreet Box through dedicated APIs. The Discreet Box performs the proper actions based on the regulations stored in the *Regulations Policies Repository* and the user privacy preferences which are communicated by user’s terminal as metadata of the personal information. The *Notification and Consent Manager (NCM)* component is responsible for sending notification or consent to the user.

Another important component is the *User Privacy Manager (UPM)*, showed in Figure 7. It constitutes the user-side component of the architecture and resides in user’s terminals. The User Privacy Manager has three main responsibilities. Firstly, it assists the users regarding privacy issues (i.e., to set privacy preferences, edit a privacy profile, notify the user of privacy related alerts generated by Discreet Boxes, etc.). Secondly, it leverages a policy framework to protect the personal data before leaving the data equipment. Thirdly, it employs a Notification and Consent Manager (NCM) module for the interaction with the Discreet Box.

![Diagram of User Privacy Manager](image)

**Figure 7. User Privacy Manager**

Analyzing the Discreet approach based on our requirements, we conclude that Discreet fulfills the *usability* requirement since it allows users to define their privacy preferences through the Privacy Manager Console in the User Privacy Manager. However, the dynamic privacy preferences requirement is not supported by Discreet since the Discreet Privacy Language does not provide the possibility of defining privacy
preferences that depend on the user context information and the framework does not have any mechanism to adopt the preferences based on the current situation of the user.

Discreet fulfills the notification and feedback requirement through NCM components installed on both Discreet Box and User Privacy Manger. For example, users can define some rules in their preferences to get notified whenever someone accesses their information. However, the user information are stored centrally which is in contradiction with the decentralized architecture requirement. Moreover, fine-grained access is not supported by Discreet

4.2 Intelligent privacy agent

Using a privacy agent in context-aware environments is proposed in [60]. Their objective was to help notify mobile users of relevant information disclosure and to enable them to manage their privacy by expressing the privacy preferences in different privacy levels.

They have developed a privacy-respecting context-aware architecture prototype showed in Figure 8. The Context Processing Layer processes the raw context data into a higher-level context data with the desired level of abstraction. These context data are stored and managed by the Context repository Layer. The Context Association Manager is responsible for collection and dissemination of context information. The Context Coordinator serves an interface to context clients to perform the basic access control. Finally, the Privacy Agent performs a fine-grained privacy check based on individual user's privacy preferences.

In this solution, an ontological modelling technique for privacy related interactions is used and the privacy syntax for specifying the privacy preferences uses Platform for Privacy Preference (P3P)[61].
The intelligent privacy agent fulfills the decentralized architecture by introducing the context-aware middleware. In addition, it fulfills the fine-grained access requirement. However, in terms of usability it has some weaknesses due to using P3P as the privacy policy language. We will discuss why P3P is not suitable for context-aware environments in section 5.1. The notification and feedback, and dynamic preferences are not supported in this solution.

4.3 Confab Toolkit

The Context Fabric (Confab), proposed by Hond Na Landay [62], provides an architecture for developing privacy-enabled context-aware applications. The Confab data model structures context information as infospaces, network-addressable logical storage units that store context information (see Figure 9). These infospaces are populated by context sources such as sensors, managed by infospace servers, which can be either distributed across a network or managed centrally, and queried by context-aware applications.
Figure 9. An infospace (clouds) contains contextual data and tuples (squares), being contained in infospace servers (rounded rectangles)

Infospaces contain Context Tuples that represent the contextual information of infospaces. Each tuple contains the metadata (e.g., data type, data format, etc.), one or more values and one or more sources describing how data is gathered and transformed. The context tuples can also have privacy tags which contains end-user preferences on how the data should be used when it flows to other infospaces. The confab framework adds privacy tag to all outgoing tuples from an infospace, and the privacy rules are enforced on all incoming and outgoing tuples. Hong and Landay have implemented their model using Web technologies, such that infospaces are identified by URLs and tuples are exchanged in an XML format.

Confab has a decentralized architecture and notification and feedback is supported by specifying a notification receiver (e.g., email address) in the privacy tag. However, despite the fact that the fine-grained access and usability are identified as their requirements, they are not fully supported in their solution. Users can define their privacy preferences as privacy tags for context tuples, but in practice defining the user preferences is complicated for end users unless an application hides the complexity.
4.4 ContextWare

ContextWare is a component of the Ambient Control Space [63] that provides distributed storage and reasoning about an entity’s context information. Raffaele Giaffreda et al [64] have proposed a privacy and authorization framework based on IETF GEOPRIV [65] privacy framework to restrict and control the distribution of context information. The GEOPRIV framework protects the location information against unauthorized access to preserve the privacy of the owner of this information.

Figure 10 shows the overall architecture of the ContextWare framework. It includes two functional areas, the Context Coordinator (ConCoord) for supervising access to and the Context Manager (CM) for managing the content of a distributed Context Information Base (CIB).

![Figure 10. Mapping GEOPRIV onto ContextWare architecture](image)

The ContextWare framework provides two levels of authorization: coarse access control and refined access control. A basic authorization mechanism enforces the coarse access control by applying the basic policies that are valid regardless of who is the owner of the requested context information. For example, deny access to context information for all clients within an IP range. The fine grain access control is enforced at the source of the context information or alternatively by a context manager (CM in Figure 10) on behalf of the context source.
ContextWare fulfills the *decentralized architecture* requirement because the proposed architecture in ContextWare is a protocol independent architecture that can be implemented either centralized or decentralized. For example, the ConCoord component can be either in central system or be located on mobile devices.

The GEOPRIV framework aims at protecting location information against unauthorized access. ContextWare reuses this framework, but extending the privacy policies to support other types of context information is considered as a future work. Therefore, the usability requirement is not fulfilled by ContextWare because it is not really clear that how users can define their privacy preferences.

After a context source provides the value of context information to the Context Manager, the clients can access the context information many times and context source will not be notified about these accesses. Therefore the Notification requirement is not fulfilled.

ContextWare does not fulfill the anonymity and dynamic preferences requirement.

### 4.5 Comparison of studied solutions

The evaluation of the discussed solutions against the privacy requirements that we described in section 3.2 is demonstrated in Table 2.

<table>
<thead>
<tr>
<th></th>
<th>Discrete Box</th>
<th>Privacy agent</th>
<th>Confab</th>
<th>ContextWare</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Usability</strong></td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Notification</strong></td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Decentralized</strong></td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Quality control</strong></td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Dynamic preferences</strong></td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Table 2. *The comparison of related work against our privacy requirements.*
5 Privacy policy languages

This chapter reviews the existing privacy policy languages and analyzes them by encoding the privacy concerns described in Chapter 21 using these languages.

5.1 P3P and APPEL as Privacy Policy Language

Platform for Privacy Preferences (P3P)\[61\] is a privacy policy language designed by W3C to be used by websites in order to transform their privacy policies into machine readable XML files. These files are stored at the web server, having unique URI. Users can decide what kind of information about them will be disclosed to particular websites. They can also decide whether the disclosed information can be revealed to other parties or not. These decisions are encoded using P3P Preference Exchange Language (APPEL) and stored as privacy policies.

5.1.1 How P3P Works?

When a user visits a website, the browser obtains a privacy policy proposal of the website, compares it with a user’s privacy preferences, and performs the appropriate action. It might reject the privacy proposal of website, accept it, or prompt user for making the final decision.

A P3P Policy includes the following information:

- Who is collecting data?
- What information is being collected?

![Figure 11. A simple HTTP transaction with P3P added.](image-url)
For what purpose?

Which information is being shared with others?

Who are these data recipients?

Can users access their identified data?

Can users make changes in how their data is used?

Figure 11 shows an example of P3P policy file that can be used by an online shopping website. The root element is `<POLICIES>` element that gathers one or more policies in a single P3P policy file. This example contains only one `<POLICY>` element for shoppers. The website could have another policy for users that only visit the website without doing any purchase.

```xml
<POLICIES xmlns="http://www.w3.org/2002/01/P3Pv1">
  <POLICY name="forShoppers"
    discuri="http://www.catalog.example.com/Shopping.html"
    opturi="http://example.com/preferences.html" xml:lang="en">
    <ENTITY>
      <DATA-GROUP>
        <DATA ref="#business.name">CatalogExample</DATA>
        <DATA ref="#business.contact-info.postal.street">4000 Lincoln Ave.</DATA>
        <DATA ref="#business.contact-info.postal.city">Birmingham</DATA>
        <DATA ref="#business.contact-info.postal.stateprov">MI</DATA>
        <DATA ref="#business.contact-info.postal.postalcode">48009</DATA>
      </DATA-GROUP>
    </ENTITY>
    <ACCESS><contact-and-other/></ACCESS>
    <STATEMENT>
      <CONSEQUENCE>
        We use this information when you make a purchase.
      </CONSEQUENCE>
      <PURPOSE><current/></PURPOSE>
      <RECIPIENT><ours/></RECIPIENT>
      <RETENTION><stated-purpose/></RETENTION>
      <DATA-GROUP>
        <DATA ref="#user.name"/>
        <DATA ref="#user.home-info.postal"/>
        <DATA ref="#user.home-info.online.email"/>
        <DATA ref="#dynamic.miscdata"/>
        <CATEGORIES><purchase/></CATEGORIES>
      </DATA-GROUP>
    </STATEMENT>
  </POLICY>
</POLICIES>
```

Listing 1. A P3P file showing policies of an online shopping website
The `<POLICY>` element contains one `<ENTITY>` element, one `<ACCESS>` element, and one or more `<STATEMENT>` elements. The entity element shows a precise description of a legal entity representing this policy.

The access element indicates whether the website provides access to personal data or not. This element can take a `<nonident/>` range, in which the website does not collect any user’s sensitive data to `<all/>` range, in which access to all sensitive data is given to the website. In the given example, the access element contains the `<contact-and-other/>` element, which indicates that requested access to sensitive online and physical contact information has been granted. The statement element is the main part of every P3P policy file.

The statement element represents a privacy decision specified in the privacy policy of the website. The `<CONSEQUENCE>` element explains the statement in a human-readable form, using sentences. The `<PURPOSE>` element shows how data can be used by the website. In the example in Listing 1, the accessed data is used to complete the purchase activity for which it was provided. The `<RECIPIENT>` indicates whether the collected data will be distributed to a third party or not. In this example, `<ours/>` element indicates that only the shopper, the website, and all the entities acting as shoppers’ agents can have access to the collected data. The `<stated-purpose/>` element shows that the collected data will be retained to meet the stated purpose. The `<DATA-GROUP>` element describes the data that needs to be transferred to the website. The `<CATEGORIES>` element gives additional hints such as what type of information is requested from a service. The `<purchase/>` element indicates that collected information is actively generated by purchase of a product or a service.

### 5.1.2 How APPEL Works?

APPEL [66] is a privacy rule language that specifies how to process a P3P proposal of a service. A service is a program that issues policies and (optionally) data requests. This language is used to show privacy preferences of a user who is interested in a particular service. The P3P user-agent is a software (e.g. browser) that on behalf of the user checks the APPEL privacy preferences that define an appropriate behaviour for the requested service. This behaviour might be interpreted as one or a combination of the following actions:
- **“request”**: The provided service is acceptable.
- **“limit”**: The provided service is acceptable to a certain extent (e.g. all except the necessary request headers should be suppressed).
- **“block”**: The provided service is not acceptable.

Each behaviour has an optional “prompt” attribute. If the prompt is set to “Yes”, the user agent should display a message and ask the user for decision. The prompt can also be a warning message. Listing 2 shows an example of APPEL rule set.

```xml
  <appel:RULE behavior="block" description="Service collects personal data for 3rd parties">
    <p3p:POLICY>
      <p3p:STATEMENT>
        <p3p:DATA-GROUP>
          <p3p:CATEGORIES appel:connective="or">
            <p3p:physical/></p3p:CATEGORIES>
          </p3p:DATA-GROUP>
          <p3p:RECIPIENT appel:connective="or">
            <p3p:same/></p3p:RECIPIENT>
        </p3p:STATEMENT>
      </p3p:POLICY>
    </appel:RULE>
  <appel:RULE behavior="request" description="Service only collects clickstream data">
    <p3p:POLICY>
      <p3p:STATEMENT>
        <p3p:DATA-GROUP appel:connective="or-exact">
          <p3p:DATA ref="#dynamic.http.useragent"/>
          <p3p:DATA ref="#dynamic.clickstream.server"/>
        </p3p:DATA-GROUP>
      </p3p:STATEMENT>
    </p3p:POLICY>
  </appel:RULE>
  <appel:RULE behavior="limited" prompt="yes" promptmsg="Suspicious Policy. Do you want to continue (limited access)?">
    <appel:OTHERWISE/>
  </appel:RULE>
</appel:RULESET>
```

**Listing 2.** An APPEL file representing the privacy preferences of a user

The first rule is a “**block rule**” that prevents information disclosure to third parties. APPEL rule usually contains a P3P policy element that is compared to the service’s proposal P3P policy. If the embedded policy matches the service’s P3P policy, the
“block” rule will reject the service policy without any notification to user. The embedded policy defines the personal data using <DATA> elements in the categories <physical/>, <demographic/>, or <uniqueid/>. These data types are defined in the P3P basic data schema. The policy element defines third parties as <RECIPIENT> matching <same/>, <other-recipient/>, <public/>, <delivery/>, and <p3p:unrelated/> covering all recipients except <ours/>.

The second rule is a “request rule” containing a P3P policy element with a data element showing user agent and click stream data. The “or-exact” element indicates that the ”request rule” will match if the statement in the policy does not contain any additional data references that are not contained in the rule.

The third rule is a “limit rule” that will be considered if all the preceding rules fail to match. It will show a message to a user to inform him/her that the proposed policy does not match his/her preferences. In general, a ruleset should be written such that there is always a rule that will be triggered. In APPEL changing the ordering of rules also changes the preferences behaviour. For example, if we put the limit rule as the first rule the prompt message will always be displayed.

### 5.1.3 P3P and APPEL for our scenarios

As mentioned earlier, P3P and APPEL are created to protect privacy of users’ data on the Internet. We need to extend both P3P and APPEL languages to support privacy concerns described in section 3.1. For example, in the standard format of P3P the location is not defined as one possible data type. In the following sections we show how we can extend P3P to support context privacy.

#### 5.1.3.1 Context Information in P3P

The first step in customizing P3P to support our scenarios is to extend the P3P to represent context parameters. P3P has defined a P3P base data schema that includes a large number of data elements commonly used by services. These data elements are organized into a hierarchy. P3P provides a mechanism to define new data structures. There are two methods for defining data structures: standalone XML files and embedded data structure.
A standalone XML file method enables storing of a new data schema in an XML file, with a root element called DATASCHEMA. This data schema should be referenced in a policy file. We created a new data schema to express a user’s location information. A data element automatically includes all the data elements below this element in the hierarchy. Using this nesting representation, we defined country, city, street, and building as sub-elements of location data structure in order to enable different granularity of location information. Listing 3 shows an XML file showing the location data structure.

```xml
<DATASCHEMA xmlns="http://www.w3.org/2002/01/P3Pv1">
  <DATA-STRUCT name="location.country" short-description="Country">
    <CATEGORIES><location/></CATEGORIES>
  </DATA-STRUCT>
  <DATA-STRUCT name="location.city" short-description="City">
    <CATEGORIES><location/></CATEGORIES>
  </DATA-STRUCT>
  <DATA-STRUCT name="location.street" short-description="Street No">
    <CATEGORIES><location/></CATEGORIES>
  </DATA-STRUCT>
  <DATA-STRUCT name="location.building" structref="http://www.w3.org/TR/P3P/base#postal"
    short-description="Construction Place">
    <CATEGORIES><location/></CATEGORIES>
  </DATA-STRUCT>
  <DATA-DEF name="currentLocation" structref="#location"/>
</DATASCHEMA>
```

**Listing 3.** XML file to define location as a new data schema in P3P

The data schema defines a data structure and a data elements that comprise a data structure using DATA-STRUCT and DATA-DEF elements, respectively. In our example, we defined four data structures to describe location data elements and a separate currentLocation data element, representing a user’s current location.

Listing 4 shows how a user’s current location is referenced in a P3P file. In this example, only a user’s country and are referenced.

```xml
<DATA-GROUP>
  <!-- First, the "currentLocation.country" data element, whose definition is in the data schema at http://www.example.com/models-schema -->
  <DATA ref="http://www.example.com/models-schema#currentLocation.country"/>

  <!-- And second, the "currentLocation.city" data element, whose definition is the data schema at http://www.example.com/models-schema-->
  <DATA ref="http://www.example.com/models-schema#currentLocation.city"/>
</DATA-GROUP>
```

**Listing 4.** Referencing a data element inside P3P
When there is a large number of data elements in a P3P file, the "base" attribute of the DATA-GROUP element can be used to reference a newly created data schema (as depicted in Listing 5).

```xml
<Data-Group base="http://www.example.com/models-schema">
  <Data ref="#currentLocation.country"/>
  <Data ref="#currentLocation.city"/>
</Data-Group>
```

Listing 5. Using the "base" attribute to reference a newly created data schema

A newly created DATASCIEMMA is embedded in a P3P policy file (as depicted in Listing 6).

```xml
<appel:RULESET xmlns:appel="http://www.w3.org/2002/04/APPELv1"
  xmlns:p3p="http://www.w3.org/2000/12/P3Pv1" crtdby="W3C"
  crtdon="1999-11-03T09:21:32-05:00">
  <appel:RULE behavior="request"
    description="HCI has access to Bob’s location in the case of emergency">
    <p3p:POLICY>
      <p3p:STATEMENT>
        <Datascema xmlns="http://www.w3.org/2002/01/P3Pv1">
          <Data-Struct ... />
          <Data-Def ... />
        </Datascema>
        <p3p:DATA-GROUP/>
        <p3p:RECIPIENT />
      </p3p:STATEMENT>
    </p3p:POLICY>
  </appel:RULE>
</appel:RULESET>
```

Listing 6. Embedded Datascema element.

In described examples we described how P3P can be extended to represent the location context information. The same procedure should be followed to describe all desired context parameters. A more sophisticated approach would be to create a data schema based on a context model. Such a data schema would contain the same context information as defined in the context model.

5.1.3.2 Applying P3P to our scenarios

In order to control access to a user’s sensitive information P3P has a <RECIPIENT> element. The <RECIPIENT> element defines one or more recipient categories. We can use the <RECIPIENT> element to specify who can access what context parameter, defined in the <DATA-GROUP> element. Since build-in recipients are generic and are designed for access to services on the Internet, we created a new, mandatory recipient
type called \textit{CONTEXT-REQUESTOR} that should be used by context-aware applications that implement the support for this P3P extension.

Listing 7 shows an example of defining a new \textit{<RECIPIENT>} element, called HCI, whose namespace is "http://www.example.com/P3P/REQUESTOR".

\begin{verbatim}
<p3p:RECIPIENT appel:connective="and-exact">
  <EXTENSION optional="no">
    <CONTEXT-REQUESTOR type="include"
      xmlns="http://www.example.com/P3P/REQUESTOR">
      <HCI/>
    </CONTEXT-REQUESTOR>
  </EXTENSION>
</p3p:RECIPIENT>
\end{verbatim}

\textbf{Listing 7.} Extension of the \textit{<RECIPIENT>} element.

Listing 8 represents Bob’s privacy concern about not revealing his identity to RATP agents.

\begin{verbatim}
<appel:RULESET xmlns:appel="http://www.w3.org/2002/04/APPELv1"
  xmlns:p3p="http://www.w3.org/2000/12/P3Pv1" crtdby="W3C"
  crtdon="1999-11-03T09:21:32-05:00">
  <appel:RULE behavior="block" prompt="yes"
    description="RATP agents collects Bob’s location(Scenario5)"
    <p3p:POLICY>
      <p3p:STATEMENT>
        <p3p:DATA-GROUP><p3p:DATA><p3p:CATEGORIES appel:connective="or">
          <p3p:physical/>
          <p3p:online/>
          <p3p:uniqueid/>
          <p3p:financial/>
          <p3p:other-category/>
        </p3p:CATEGORIES></p3p:DATA></p3p:DATA-GROUP>
        <p3p:RECIPIENT appel:connective="and-exact"> CONTEXT-REQUESTOR type="include"
          xmlns="http://www.example.com/P3P/REQUESTOR">
        </p3p:RECIPIENT>
      </p3p:STATEMENT>
    </p3p:POLICY>
  </appel:RULE>
</appel:RULESET>
\end{verbatim}

\textbf{Listing 8.} Bob’s Privacy Preferences in APPEL (privacy concern 3)

The second rule makes Bob’s current location available to HCI in form of street address.
Listing 9. Bob’s Privacy Preferences in APPEL (privacy concern 2)

The third rule permits RATP agents to access Bob’s location with resolution of building.

Listing 10. Bob’s Privacy Preferences in APPEL (privacy concern 1)
When an application wants to obtain Bob’s current context information, it sends a request to his user agent. The request has to contain a P3P file to inform the user agent about the requestor, the requested context information, and the required level of granularity. The P3P file can either be generated by a requestor or by a privacy manager on behalf of the requestor. The latter approach is preferred since it prevents generation of malicious requests with false information. User agents receive the P3P file, check the file contents against Bob’s privacy preferences, and perform the appropriate action.

Our extended P3P should be able to represent the following features:

- What is sensitive information?
- Who is recipient of a particular sensitive information?
- What is the scope of a sensitive information?
- In what circumstances should sensitive information be disclosed to a particular requestor?

We have extended P3P to show all these features, except specifying the circumstances of disclosing the sensitive information. Because of the static nature of P3P and APPEL, circumstances cannot be represented in it P3P policy files.

From this discussion we can conclude that customizing P3P and APPEL for context-aware solutions needs a context-aware privacy mechanism to manage modify privacy policies each time the user’s context changes. In the following, we demonstrates Bob’s privacy preferences (described in section 3.1) encoded in APPEL.

### 5.2 Common Policy as Privacy Policy Language

*Common Policy* is a document format for expressing privacy preferences[67]. The current version of common policy is enhanced by location-specific and presence-specific policy documents. The *Common Policy* contains a number of rules each of which has the following elements:
5.2.1 Conditions

Conditions can be applied to the requestor identity or the context owner information. A rule matches if all conditions are evaluated as “true”. The default elements of conditions are:

- **identity**: restricts the matching of the privacy rule with a single entity or a group of entities.
- **sphere**: indicates a state of the context owner such as “work”, “home”, or “meeting”.
- **validity**: specifies the timeframe when the rule is valid.

The **identity** element can be used if the requestor is identified using a URI as a unique identifier. In the current version of MUSIC middleware, SIP URIs are used to identify users or devices. Therefore, we decided to use the same SIP URIs to check the requestor’s identity. The **sphere** element matches if the context owner is in the state indicated inside the sphere element. This state is defined by an application and can be configured manually or be extracted from the user’s context information. The **validity** element matches if the request is made in the timeframe specified by this element. This element can be used for automatic removal of expired rules, offloading the application from removing it manually. However, this feature requires time synchronisation between different nodes to be implemented.

The default condition elements are not sufficient for a wide range of context information that can be used to define a user’s situation (such as location, temperature, activity, etc.). Common Policy is extensible to create new conditions, but these conditions should be created in a new namespace.

Listing 11 shows the XML schema of the conditions element in common policy. An unlimited number of extensions can be added to the condition element of a rule.
The **Geolocation Policy**[65], a document format for expressing Privacy Preferences for Location Information, has extended the condition element of common policy to create location-based conditions. It has defined different location condition profiles such as civic condition and geodetic condition. The former is used for a civil address and the latter is used for defining a user’s geographical position. These extensions can be used to create a rule for **Privacy Concern 3** described in section 3.1 by specifying that Bob would like to disclose his location information to RATP agents while he is in a Paris subway. Listing 12 demonstrates a location condition that returns true if a user’s location is within 1500 radius of a circle from the geographical position of “Boulevard St. Michel”.

**Listing 11.** The XML schema of conditions element in Common Policy

```xml
<xs:element name="conditions" type="cp:conditionsType" minOccurs="0"/>
<xs:complexType name="conditionsType">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:choice maxOccurs="unbounded">
        <xs:element name="identity" type="cp:identityType" minOccurs="0"/>
        <xs:element name="sphere" type="cp:sphereType" minOccurs="0"/>
        <xs:element name="validity" type="cp:validityType" minOccurs="0"/>
        <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
      </xs:choice>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Although location condition can be used in many scenarios, other context parameters (such as health information, activity, mode, etc) can also be used as a condition. For each of these parameters, a new condition should be defined in the extension of the Common Policy. We will discuss the extensions required for our scenarios in Section 5.3.
5.2.2 Actions and Transformations

The actions and transformations elements specify how a request to access user data should be processed. Actions are operations that need to be performed after a request is evaluated (e.g., permit or deny access to the requested information), while transformations modify data before it is presented to recipient. Union of all matching transformation creates a “mask” that will be applied on data.

The actions and transformations elements are defined as extensible types in the common policy XML schema (as illustrated in Listing 13).

```
<xs:element name="actions" type="cp:extensibleType" minOccurs="0"/>
<xs:element name="transformations" type="cp:extensibleType" minOccurs="0"/>
<xs:complexType name="extensibleType">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

**Listing 13.** XML Schema of actions and transformations in Common Policy

Geolocation Policy has extended the location transformation using the `<provide-location>` element. It can tune the accuracy of location information based on a given radius or restrict the location up to certain level such as country, region, city, or building.

5.3 Common Policy for our scenarios

This section analyzes the Common Policy rules by trying to encode them according to the privacy concerns defined in our scenarios. Based on the results of this analysis we will determine if by extending the Common Policy the requirements for the context-aware privacy policy language can be met.

In the **Privacy Concern 1** (PC1), the sensitive context information is the user’s health data. The following information is collected from different sensors and combined for inference of a user’s health status:

- user.bloodPressure
- user.heartRate
- user.bodyTemperature
- user.bodyMotion

The inferred user’s health status should be sent to the HCI centre (as shown in Listing 12).

```xml
<rule id="privacyConcern1">
    <!-- HCI centre can have access to Bob's health information retrieved from Health Terminal -->
    <conditions>
        <!-- Only HCI centre -->
        <identity>
            <one id="sip:admin@HCI.com"/>
        </identity>
    </conditions>
    <actions/>
    <transformations>
        <cm:provide-bloodPressure>YES</cm:provide-bloodPressure>
        <cm:provide-heatRate>YES</cm:provide-heatRate>
        <cm:provide-bodyTemperature>YES</cm:provide-bodyTemperature>
    </transformations>
</rule>
```

Listing 14. The common policy rule representing Privacy Concern 1

In the condition part the HCI centre identity should be checked. Common Policy supports the identity in the condition element. However, Common Policy has to be extended to support transformation of the new context information. The granularity of the context information is not important in this scenario. Note that the ‘cm’ namespace indicates our transformation extension, which will be based on the MUSIC context model.

In the Privacy Concern 2 a user wants to send his health information to HCI employees only in emergency cases. In the <conditions> element, we need to create a new condition type that contains parameters defining a user’s health status. This new extension includes two profiles: a user-defined profile and a default profile. In a user-defined profile the emergency situation is defined by the minimum and maximum values of health parameters, while in a default profile the health status context reasoner determines if the situation is emergency or not. Context reasoner is a context plug-in that aggregates and processes context information obtained from sensors. Listing 15 shows a common policy rule containing both user-defined and default profiles in two
different conditions, but in practice, only one of them can be used at a time. In the transformations element depicted in Listing 15 access to Bob’s location is restricted to the building granularity. To represent the location condition the existing extension defined in the Geolocation Policy [65] is used.

```xml
<rule id="privacyConcern2">
  <!-- HCI employees can have access to Bob's location in emergency -->
  <conditions>
    <!-- Emergency situation -->
    !-- A user defines parameters of emergency situation-->
    <cm:health-condition
        profile = "user-defined">
      <cm:bloodPressureMin>8</cm:bloodPressureMin>
      <cm:bloodPressureMax>13</cm:bloodPressureMax>
      <cm:heartRateMin>72</cm:heartRateMin>
      <cm:heartRateMax>110</cm:heartRateMax>
      <cm:bodyTempMin>35</cm:bodyTempMin>
      <cm:bodyTempMax>39</cm:bodyTempMax>
    </cm:health-condition>
    <!-- Emergency situation is detected by the context reasoner-->
    <cm:health-condition
        profile = "default">
      <cm:healthStatus>emergency</cm:healthStatus>
    </cm:health-condition>
    <!-- Only HCI employees -->
    <identity>
      <many domain="HCI.com"/>
    </identity>
  </conditions>
  <actions/>
  <transformations>
    <!-- The location resulostion is decreased to building level -->
    <gp:provide-location
        profile="civic-transformation">
      <gp:provide-civic>building</gp:provide-civic>
    </gp:provide-location>
  </transformations>
</rule>
```

Listing 15. The Common Policy rule representing Privacy Concern 2

In the Privacy Concern 3 a location condition based on the Geolocation Policy format can be used to describe an itinerary planned by Bob.

In the Privacy Concern 4 there is a need to represent the time condition, such as Alice’s working hours. Although the time condition is not explicitly provided by Common Policy, a rule can be defined that is valid during a time period using the Common Policy’s `<validity>` element. For example, if Alice is working from Monday to Friday from 8:00 to 16:00, then for each day a rule has to be created containing the `<validity>` element for that particular day. An alternative solution will be suggested when describing the Common Policy rule for Privacy Concern 7. The transformations
element contains a geodetic location transformation to represent Alice’s location with a resolution of 100 meters.

```xml
<rule id="privacyConcern4">
  <!-- Alice's manager in HCI can have access to her location with resolution of 100 meters only during working hours-->
  <conditions>
    <validity>
      <from>2010-05-23T08:00:00.000-01:00</from>
      <until>2010-05-23T16:00:00.000-01:00</until>
    </validity>
    <identity><one id="sip:manager@HCI.com"/></identity>
  </conditions>
  <actions/>
  <transformations>
    <!-- Set the location resolution -->
    <gp:provide-location
      profile="geodetic-transformation">
      <gp:provide-geo radius="100"/>
    </gp:provide-location>
  </transformations>
</rule>
```

Listing 16. The Common Policy rule representing Privacy Concern 4

In the Privacy Concern 5 Alice’s husband can see her activity while she is in Paris. Alice’s location is represented in the civic form of the Geolocation Policy, while her activity can be represented by the presence information. To describe Alice’s presence information extensions defined in RFC 5025 [68] as presence rules will be used. As Listing 17 shows, Alice’s activity and mood are allowed to be showed to the watchers that meet the specified location condition.

```xml
<rule id="privacyConcern5">
  <!-- Alice's husband (Joe) know about her activity and mood when she is in paris-->
  <conditions>
    <identity><one id="sip:joe@example.com"/></identity>
    <gp:location-condition
      profile="civic-condition"
      xml:lang="en"
      label="Paris"
      xmlns="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr">
      <country>FR</country><A1>Île-de-France</A1><A3>Paris</A3>
    </gp:location-condition>
  </conditions>
  <actions><pr:sub-handling>allow</pr:sub-handling></actions>
  <transformations>
    <!-- Restricts presence information to activity and mood -->
    <pr:provide-activities>true</pr:provide-activities>
    <pr:provide-mood>true</pr:provide-mood>
  </transformations>
</rule>
```

Listing 17. The Common Policy rule representing Privacy Concern 5

In the Privacy Concern 6, Alice’s location can be revealed at the street granularity to her friend list on weekends. The validity element cannot be used to define the "on
weekends" condition since periodic time cannot be expressed as validity nor time frames can be enumerated separately as validity elements connected using the logical operation "AND". Furthermore, defining more than one timeframe will always return "false". A possible solution can be to define a so called "periodic-time" -condition as a new condition type, using the sub-elements such as year, month, week, and day to define time periods. This condition should return "true" if the request is sent during the specified time periods. Listing 18 shows the proposed periodic time condition to define a weekend.

```xml
<rule id="privacyConcern6">
  <!-- Alice's friends can see her location at the street granularity on weekends-->
  <conditions>
    <!-- It is a weekend -->
    <xx:periodic-time-condition>
      <xx:day>sunday</xx:day>
      <xx:day>saturday</xx:day>
    </xx:periodic-time-condition>
    <identity>
      <!-- List of alice's friends -->
      <one id="sip:friend1@example.com"/>
      <one id="sip:friend2@example.com"/>
      <one id="sip:friend3@example.com"/>
      <one id="sip:friend4@example.com"/>
    </identity>
  </conditions>
  <actions/>
  <transformations>
    <!-- Set the location resolution -->
    <gp:provide-location
      profile="civic-transformation">
      <gp:provide-civic>street</gp:provide-civic>
    </gp:provide-location>
  </transformations>
</rule>
```

Listing 18. The Common Policy rule representing Privacy Concern 7

5.3.1 Common Policy strengths

After representing privacy concerns using the extensions of Common Policy in the previous subsection, we observed the following advantages of the Common Policy language:

- **Simple and easy to understand**: The common policy rule set is well structured and easy to understand. However, there is a lack of tools providing a graphical user interface to users to create, modify, and delete their privacy preferences.
• **Easy to change:** Common Policy is extendable, enabling easy creation of new conditions and privacy rules. A tool support is needed to transform the preferences that the users have specified using the GUI into the Common Policy privacy rules. However, advance users can edit their privacy rules directly using XML.

• **Suitable for dynamic environments:** The condition element can be specified for different context parameter value pairs, which upon the situation change are matched against the user’s current context in order to find the valid rules in a new situation.

• **Standard:** Common Policy is standardized by the IETF working group.

• **Extensible:** Common Policy language is easily extensible for specific application domains (e.g., location and presence). Any new application domain should be defined in a new namespace.

### 5.3.2 Common Policy Limitations

Common Policy has the following disadvantages.

• **Lack of enforcements:** Common Policy is not widely implemented.

• **No guarantee for some privacy transformations:** There are some privacy transformations that prevent retransmission of data or which specify the maximum retention time of particular data. However, there is no mechanism that can ensure a user that the watchers are following these privacy transformations.

• **Very generic:** Being a generic privacy language makes Common Policy usable by different applications that can customize it for their own purpose. However, consequently there will be many extensions created for each of these application domains, which can be seen as a disadvantage by application developers.
6 A Context-aware Privacy Policy Language

In this chapter, first, we discuss the requirements of a privacy policy language for context-aware environment. Then, we analyze the existing privacy policy languages designed for context-aware environments. As a result of this analysis, we introduce a new context-aware privacy policy language that enables users to define their privacy preferences indicating who can access their information, in which situation, and at what level of detail.

6.1 Requirements

From scenarios described in 3.1, we identified the requirements that should be considered when selecting an existing or designing a new context-aware privacy policy language:

- **User-defined situation:** A privacy policy language should enable users to define privacy preferences that are valid in specific situations. A situation should be specified by users based on their available context (via a tool with a graphical user interface), using parameters defined in the context model.

- **Rich context model:** A context model should be rich enough to allow users to define any situation, while at the same time it should be customized for use by applications in a particular domain.

- **Periodic-time:** A privacy policy language should enable the definition of privacy rules that are valid in periodic time intervals (e.g., on weekends, on work days from 12:00 to 13:00, etc.).

- **Social relationship:** Users should be able to define their privacy rules based on the social relationship that they have with other people. By maintaining social contacts in a small number of groups and using these groups to specify privacy rules can significantly reduce the number of rules that need to be specified by users.

- **Fine-grained access:** Users should be able to specify granularity of context information that they want to disclose to others in a privacy rule.

- **Context-awareness:** Privacy policy rules should be context-aware, thus they should be evaluated when a user’s situational context changes (rather than when a request
for sensitive information arrives). As a result, only a list of privacy rules that are valid in the user's current situation will be checked by the privacy mechanism when a context request arrives.

- **Conflict-handling:** There is a potential risk that more than one privacy rule is valid and can be applied in a particular situation. In some cases, the effect of the valid rules can indicate different actions. For example, one rule might grant access to the requested context, while another rule can deny it. A selected privacy policy language must provide a mechanism to handle such conflicts.

### 6.2 Existing privacy policy languages

There are existing policy languages that can be used to represent a user's context-dependent privacy preferences. However, in these languages, a user's context is used as an additional condition parameter based on which the decision for granting or denying access to the requested context is made. Since a condition is specified as part of the rule, privacy mechanisms that support these context-dependent privacy languages have to evaluate all the specified privacy rules upon receiving a context request, regardless of the fact that only some of these rules might be valid in the user's current situation. Evaluating a large number of privacy rules upon receiving a context query can potentially increase the time that is needed for decision making about granting or denying access to a user's sensitive context (as described in [69]).

In the following, we review the state-of-the-art context-dependent privacy languages according to the requirements identified in the previous section.

#### 6.2.1 Houdini

Houdini [70] is a context-aware privacy framework that enables users to specify their privacy preferences through web-based forms. The privacy preferences can be defined based on the users’ current situation, their social relationship with the requestor or the requestor's identity (e.g., Bob), and the relation of the requestor's current situation with respect to their own current situation (e.g., if they are located on the same street). However, if a user's privacy preferences are based on this last condition and if the access to the requestor’s current situation is denied to a user, the effect of all the rules...
that are dependent on this particular condition could be to deny access to the requested user data.

The privacy preferences defined through the web forms are translated into privacy policy rules and stored in a policy repository. Users can define potential situations also through the web-based forms. Situations cannot be defined or modified using the privacy policy language, because the privacy language is decoupled from the context model. Instead, a user's situation is a single variable that is used in the privacy policy rules and whose value must be calculated before evaluating the corresponding rule. Hence, when a request to access user context data arrives, all the situations have to be inferred from the latest context values and all the policy rules have to be evaluated, which reduces the time needed to retrieve the required context data and send the context response back to the requestor. The number of rules to be evaluated when a context request arrives is equal to the number of a user’s defined situations multiplied by the number of permissions that should be applied in the particular situation.

Conflict handling is supported in Houdini by assigning priorities to the rules. If there is a conflict between actions of different rules, the rule with the highest priority will be considered.

There is no support in privacy policies for periodic time conditions. Additionally, the granularity of disclosed context information cannot be controlled.

6.2.2 UbiCOSM

The Ubiquitous Context-based Security Middleware (UbiCOSM)[71] introduces an access control model that considers the user context as the foundation for policy specification and enforcement. In contradiction to traditional access control models which are tightly coupled to the requestor identity or role, permissions to access user context information are directly associated to a user's context.

The UbiCOSM categorizes context information as physical and logical context. The former identifies physical spaces delimited by specific geographical coordinates, while the latter identifies the logical properties such as a user's identity, activities, and roles.

The UbiCOSM policies are expressed as tuples of one or more contexts that are associated to a set of privacy permissions. Privacy permission determines what kind of
operation can or cannot be performed on a particular resource. Privacy permissions are not directly assigned to particular users. Instead, when a user enters a particular context (e.g., a physical location), the associated permission becomes applicable to this user. Permissions have a property that can be assigned either a positive or negative value indicating that access to the requested data is granted or denied, respectively. Therefore, it is not possible to control the granularity of disclosed information.

UbiCOSM allows mobile users to define situations based on their context and map their privacy permissions to these situations. The requestor identity/role is specified by the user as an additional context parameter along with other situational conditions within a rule, resulting in a large number of rules to be specified by a user. However, the UbiCOSM middleware updates the set of valid permissions whenever the user's situation changes, which decreases the policy evaluation time when a context request arrives. There is no explicit support in UbiCOSM for defining a user's situation based on periodic time interval. Additionally, conflict handling is not supported. Finally, a user's social relationship cannot be used (instead of a requestor's identity) to define privacy rules.

### 6.2.3 CoPS

The Context Privacy Service (CoPS)[72] provides a fine-grained access control mechanism allowing mobile users to control who can access their context data, when, and at what level. CoPS does not enable specification of a user's situations and rules based on a user's context information. Instead, CoPS uses optimistic or pessimistic approach to define a default policy in which all requests are granted or denied, except those that match one of the rules specified by a policy maker. Therefore, a policy maker needs to define only the rules specifying under which circumstances their context information should be disclosed or not, depending on the chosen default policy approach. This default policy approach reduces the number of rules that need to be specified and evaluated.

A policy maker can specify that he/she is "not available" (in which case the access to desired context is implicitly denied, but the requestor does not know whether this is due to the technical failure or access restriction) or that he/she wants to be asked, on the fly,
about the request. Access to context can be granted for the restricted time (e.g., only once, for 2 hours, always, or never allow). Granularity of disclosed context information can be specified using a spatial precision (e.g., "Room 123"), temporal restriction, or freshness of the disclosed context information (e.g., to disclose the user’s location 15 minutes ago). The context model in CoPS is limited to the context variables provided by the MoCA middleware (e.g., a user's device location, a device's CPU usage, remaining battery power, etc.). However, to enable users to specify granularity of their disclosed context information, a hierarchical syntax can be used (e.g., “campus.building” or “campus.building.room”).

CoPS implements definition of a groups and access control based on the membership in the specified groups, which decreases the effort of specifying and evaluating the policy rules. Groups can reflect the organization structure or can be defined by a user. The structure of privacy rules is specified through several fields. Each privacy rule has to be associated to a chosen default privacy policy in order to determine the basic evaluation algorithm for each request. If more than one rule matches the request, conflict handling is performed using the CoPS specificity algorithm that identifies the most specific rule from the matching rules set by comparing their structure fields in the specified order of priority. If there are two or more rules with the same field priority, the more specific one will be applied. CoPS lacks a standard format to represent the privacy policies, which restricts its use to the CoPS privacy service.

6.2.4 Context Privacy Engine

The Context Privacy Engine (CPE) [73] extends the traditional Access Control List (ACL) mechanism with a set of context constraints that have to be evaluated to validate a particular privacy policy. Context constraints are used to define context conditions that are associated to either the context owner or the context requestor, using XQuery expressions. However, a user's situation defined in the policy is not reusable in other rules. Therefore, if more than one policy should use a particular situation, then this situation definition must be repeated in each of these policies.

A subject and a requestor in CPE policies can be individuals or groups of people. However, a group (e.g., defining a user’s social relationships) is expected to be created by an application. CPE supports conflict handling by considering a policy level, which...
is an optional field in the policy. A policy at a higher level overrides all policies at lower levels. If there are multiple policies with the same level, the most specific one will be applied.

The CPE policy language does not provide a means to control the granularity of disclosed context information. This policy language is context-dependant, thus when a context request arrives, all the privacy policies have to be evaluated regardless of the user's current situation. Additionally, evaluating context-dependent privacy policies requires retrieving context data upon arrival a context query, which can be time consuming.

### 6.2.5 SenTry

The SenTry language [74] is designed as a combination of a user-centric privacy ontology (called SeT Ontology, written in Web Ontology Language (OWL)[75]) and the Semantic Web Rules Language (SWRL) [76] predicates. For each context entity a policy instance is defined which contains the associated privacy rules (defined as SWRL predicates). The SenTry language supports two categories of rules: Positive Authorization Rules (PAR) and Negative Authorization Rules (NAR). NAR rules can only have “deny” effect, while PAR rules can either allow access to the requested context or transform the context information according to the specified granularity.

Privacy rules are context-dependant, thus a context-awareness requirement is not met. Situations can be defined using the SWRL predicates, however SWRL does not support more complex logical combinations of predicates than the conjunction, which makes is difficult to define arbitrarily complex situations. SenTry provides the “grant override” combination algorithm to handle conflicts among different rules, which is an optimistic algorithm that grants access if at least one rule grants access to the requested context information. Different combination algorithms can be defined to handle conflicts, but it is up to the privacy framework to decide what algorithm should be applied for all the policies in the system.

### 6.2.6 Summary

Table 3 shows that none of the existing privacy languages fully meets the identified requirements. Most of these languages enable definition of situations and support the
rich context model, but neither of them enables definition of situations based on periodic time intervals. Languages that satisfy the context-awareness and/or the social relationship requirement consequently have a small number of privacy rules.

<table>
<thead>
<tr>
<th></th>
<th>Houdini</th>
<th>UbCOSM</th>
<th>CoPS</th>
<th>CPPL</th>
<th>SenTry</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-defined</td>
<td>+/-</td>
<td>+</td>
<td>-</td>
<td>+/-</td>
<td>+/-</td>
</tr>
<tr>
<td>Rich context</td>
<td>+/-</td>
<td>+/-</td>
<td>-</td>
<td>+</td>
<td>+/-</td>
</tr>
<tr>
<td>Periodic-time</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Social relationship</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Fine-grained</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Context-awareness</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Conflict-handling</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+/-</td>
</tr>
</tbody>
</table>

Table 3. Comparison of context-dependent privacy policy languages

6.3 CPPL

This section introduces a novel context-aware privacy policy language (CPPL) that enables mobile users to define their context-aware privacy preferences in a specific granularity based upon the social relationship of a user with a context requestor. By defining different parts of the language, we explain how CPPL meets all the identified requirements. CPPL specifies context-aware privacy rules by mapping a set of privacy rules to one or more user's situations, in which these privacy rules are valid (as shown in Listing 19). A CPPL policy contains one or more context-aware privacy rules.


```
<xs:element name="ContextPrivacyRules">  <xs:complexType>
    <xs:element ref="cppl:ContextPrivacyRule"/>
    <xs:attribute name="combinationAlg" type="xs:string"/>
  </xs:complexType></xs:element>
<xs:complexType name="ContextPrivacyRuleType"><xs:sequence>
  <xs:element type="cppl:Situations"/>
  <xs:element type="cppl:RuleSet"/></xs:sequence>
<xs:attribute name="contextPrivacyRuleId"/></xs:complexType>
```

The Situations element contains one or more Situation elements each of which is defined by one or more context conditions that must apply to an entity (as depicted in Listing 20). For an entity to be evaluated if it is in a particular situation, all the
conditions in the **Conds** element have to be evaluated to true. The **Entity** element represents a context owner, which can be an *environment*, a *device*, or a *user*. For example “**<Entity user|Bob</Entity>**” refers to Bob as a user. The entities in CPPL are expressed using the MUSIC context model [77].

Each time a user's situation changes, the list of valid privacy rules are updated that will be checked upon receiving a context query. This design makes the CPPL context-aware as it is elaborated in section 6.1.

The **Conds** element contains either a single condition (**Cond**) or a condition operator (**CondOp**) that performs a logical operation on two or more single conditions. Operators provided in the current version of CPPL are logical **AND** and **OR**. The **Cond** element can be defined as a single constraint or a logical combination of constraints.

```xml
<Situation situationId="Emergency">
  <Entity>#user|Bob</Entity>
  <Conds>
    <CondOp op="OR">
      <Cond>
        <Logical op="OR">
          <Constraint param="#healthInfo.bloodPressure.systolic" op="NEQ" value="105" delta="15.0"/>
          <Constraint param="#healthInfo.bloodPressure.diaстolic" op="NEQ" value="70" delta="10.0"/>
        </Logical>
      </Cond>
      <Cond>
        <Constraint param="#healthInfo.heartRate" op="NEQ" value="75" delta="25.0"/>
      </Cond>
    </CondOp>
  </Conds>
</Situation>
```

**Listing 20.** A Situation element representing Bob's health emergency

In the example in Listing 20, an "OR" logical combination of the abnormal blood pressure and abnormal heart rate is defined to determine an emergency situation. The former is a logical combination of two constraints while the latter is a single constraint. Note that two kinds of operators are used in this example. One is applied to constraints or conditions, while another operator is applied to context parameters to define a constraint.

The **Constraint** element (illustrated in Listing 20) is used to specify the set of context parameters that define a condition. It specifies five attributes:
- **entity**: An optional attribute that is used to specify an entity to which the context parameters belong to. If it is not specified, the default entity of parent situation element will be used.

- **param**: It refers to the semantic concept (e.g., "location") that groups context values belonging to the same context domain (e.g., "longitude" and "latitude"). Note that this notation of context parameters is adopted from the MUSIC context model [77].

- **op**: The operator applied to one or two context parameters for constraint verification. Table 4 shows the constraint operators that are supported in CPPL. The definition of these operators is adopted from [78].

- **value**: The value of the context parameter.

- **delta**: This attribute is used for continuous parameters. It shows the acceptable range of context parameters values for a given constraint.

<table>
<thead>
<tr>
<th>GT</th>
<th>Greater</th>
<th>NGT</th>
<th>Not greater</th>
<th>ST</th>
<th>Starts with</th>
</tr>
</thead>
<tbody>
<tr>
<td>LT</td>
<td>Lower</td>
<td>NLT</td>
<td>Not lower</td>
<td>EN</td>
<td>Ends with</td>
</tr>
<tr>
<td>EQ</td>
<td>Equals</td>
<td>NEQ</td>
<td>Not equals</td>
<td>NS</td>
<td>Not starts</td>
</tr>
<tr>
<td>CO</td>
<td>Contains</td>
<td>NCO</td>
<td>Not contains</td>
<td>NE</td>
<td>Not ends</td>
</tr>
</tbody>
</table>

**Table 4. Constraint operators**

The CPPL time constraint element is used to specify any (periodic) time constraint. It contains either a *DateRange* element or an *Interval* element. The former defines a time range that *begins* and *ends* at the specified date and time. The latter specifies an interval that has the following attributes:

- **daysOfWeek**: denotes week days in the form of numbers or words, representing one or more days or a range of days (e.g., notations "mon,wed,fri" or "1-3" can both be used to represent Monday, Wednesday, and Friday).

- **months**: denotes months of the year by their names or numbers. Months can enumerated or represented by a range (e.g., "2,4,7" or "may-aug").

- **daysOfMonth**: Numbers between 1 and 31 that indicate days in a month.

The *Interval* element contains an optional "*TimeRange" element for specifying time periods with "*startTime" and "*endTime" attributes. An example of using the *TimeConstraint* element to represent working hours is shown in Listing 21.
Listing 21. An example of the use of the TimeConstraint element

Listing 22 shows the definition of Ruleset element, a collection of Rule elements that are mapped to one or more situations. The Rule element (see an example in Figure 7) describes who (Identity element) can access what kind of context information (ContextParams element). The effect of a privacy rule is to permit or deny access to the requested context information. When there is more than one rule in a rule set, a combination algorithm has to be used to evaluate the final effect of the RuleSet. The value of the combination algorithm is either “denyOverrides” or “permitOverrides”. The former will set the final effect to deny if at least one rule in the set evaluates to deny while the latter sets the final effect to permit if all rules in the set evaluates to permit. However, since a CPPL policy contains multiple Context Privacy Rules, the same algorithms can be used to combine the effect of multiple rule sets (see Listing 19).

Listing 22. XML schema representation of Ruleset element

The ContextParams element specifies sensitive context parameters, defined using an entity and one or more context scopes. Using an AnyContextParam element within the ContextParams element indicates that all context parameters can be accessed by all potential requestors.

Privacy rules can be defined based on the user's social relationship with a context requestor, enabling a user to specify a rule for a class of requestors instead for each requestor individually. The Identity element enables different ways of representing a
context requestor that can be used in a privacy rule, depending on which of the following elements is used within the Identity element:

- **One**: an individual represented by an id (e.g. `<one id="sip:admin@HCI.com"/>`).
- **Many**: a group of users in the same administrative domain. (e.g. `<Many domain="HCI.org"/>`).
- **Relation**: a group of people having a specific relation to the context owner (e.g. `<Relation relation="spouseOf"/>`).
- **AnyIdentity**: is used for rules that should be applied to all the requestors.

Listing 23 shows an example of RuleSet element that permits Alice’s friend to access her current location at the city level.

```xml
<Rule effect="Permit">
  <Identity>
    <Relation relation="friendOf"/>
  </Identity>
  <ContextParams>
    <ContextParam>
      <Entity>$user|Alice</Entity>
      <Scope>$civilAddress.city</Scope>
    </ContextParam>
  </ContextParams>
</Rule>
```

**Listing 23.** Privacy rule allowing Alice’s friends to access her city scope

When a group of people are selected using the Many or Relation elements, it is possible to exclude one or more individuals from the selected group using the Except element (as depicted in Listing 24).

```xml
<Identity>
  <Many domain="HCI.org">
    <Except id="sip:Alice@example.com"/>
  </Many>
</Identity>
```

**Listing 24.** Using Expect element to exclude an individual from a group

### 6.3.1 CPPL examples

In this section some simple examples is described to show how CPPL can be used in practice. The simplest case is a CPPL file that permits any access to any kind of information from any one. Listing 25 shows such CPPL file. Having this CPPL file as the only privacy preference would be equal to have no privacy.
Listing 25. A CPPL example that allows full access to all requestors

The example illustrated in Listing 26 shows a CPPL file that might be used by HCI Terminal application as mentioned in section 3.1. It contains a context privacy rule that represent the privacy concern 1 in which Bob wants to reveal his health information to HCI center.

Listing 26. The CPPL file representing Privacy Concern 1

Appendix B contains the CPPL file of all privacy concerns described in section 3.1.
7 Context Privacy Manager Architecture and Design

This section discusses the architecture of the context privacy component and its integration into the MUSIC architecture. The main part of the context privacy component is the context privacy manager (CPM) that is responsible to check any incoming request to access or any outgoing attempt to publish a user’s sensitive context information. According to the CPPL specification described in section 6, the following information is required by the context privacy manager to perform this check:

- A user’s defined privacy policies
- A requestor’s identity
- The time when the request is sent
- A user’s context information that will be used to determine the user’s current situation

The basic operation of applying common policy rules is described in RFC 4745 [67] and illustrated in Figure 12. A Policy Server (PS) receives a request for context information from a Watcher/Recipient (WR). A Rule Maker provides the user-defined privacy policies to the Policy Server. A Presentity/Target (PT) sends data to the PS, which based on this data evaluates the context request and sends the response back to the WR. In our context privacy component design a context requestor acts as the WR, providing its identity in the context request. The PT in our design is the node providing a user’s context information that is used to determine this user’s current situation.

![Diagram of basic operations in common policy](image)

Figure 12. Basic operations in common policy
The location of the elements in the CPM can be different based on the proposed architecture. Putting the CPM inside the context middleware makes it easier to access the context information, but it might not have access to the requestor’s identity. On the other hand, putting privacy manager outside the context middleware reduces the accessibility to context information. Whenever it needs a data from the PT to be used as a condition in a privacy rule, the CPM would send a request for this data to the PT.

Another important architecture concern is where to store a user’s privacy preferences. The user’s privacy preferences should be stored and managed in a repository. This repository can be centralized or distributed. If we put the CPM in the context middleware, the privacy preferences can be stored locally because the only component that needs them is CPM. Otherwise, if we put the privacy manager on the server, a centralised repository would be used to store the user’s preferences.

Users should be able to create and edit their privacy rules either directly (if the users are familiar with the privacy policy language) or using the graphical user interfaces.

### 7.1 Context Privacy Manager Architecture

The following section proposes the CPM architecture. As discussed in Section 3.2, one of the privacy requirements of context-aware applications is a decentralized design. To satisfy this requirement, the CPM has to be located in the context middleware and the privacy policies have to be stored locally. This approach has the following advantages:

- It decreases the signalling because local calls are used instead of remote calls.
- Since privacy policies are stored locally, there is no security concern about the confidentiality of the privacy preferences.
- All the privacy related functionalities can be implemented in an independent component, which can be an optional component in the middleware runtime configuration that can be selected by the application developers when their applications need to implement context privacy

This approach also has the following disadvantages:
Putting the CPM inside the context middleware might be a problem for resource constrained devices because CPM needs to parse and evaluate the privacy policies which for large number of policies can be computationally expensive.

A user’s CPM should receive all the requests and analyse them in order to decide whether to grant or deny access to the requested context information. Therefore, a malicious unauthorized node can easily do a DOS attack by sending a large number of requests for accessing context information.

The Figure 13 shows the overall architecture of the context middleware containing the CPM component, the Distribution Manager (DM), and the Context Management (CM) component.

![Figure 13. CPM as an internal component of the context middleware](image)

The Policy Administration Point (PAP) obtains the user privacy rules from the GUI, transforms them into the CPPL format, and sends them to the CPM.

When the DS receives a subscription request for a desired context, it leverages the CPM to check the user’s permission for the requestor and the requested context type. If requestor is permitted to access the requested context, it queries the DM for this context, following the same steps as described in Section 2.3.3.
The software component design of the context middleware is illustrated in Figure 14. As it can be seen in Figure 14, the CPM internally uses IContextAccess interface to query for desired context types which are required to determine the user’s current situation. These context types are called situational context types. When a situational context changes, the CM determines the current situation and notifies the CPM about it, which in turn updates the privacy rules based on the new situation.

Figure 14. Structure of context middleware

7.2 Context Privacy Manager design

This section describes the CPM software design in more detail. As illustrated in Figure 15, the CPM provides the IContextPrivacyService and IContextPrivacyManagement interfaces to other components in the context middleware and requires the IContextAccess interface that is provided by the CM middleware component. The detail functionalities provided by these interfaces will be explained in the remaining text of this section.
The IContextPrivacyService interface, illustrated in Figure 16, has one method called checkPermission(). The DS calls this method to check if the incoming requests from other nodes in the network are authorized and if so, at what granularity should the requested context be provided to the requestor. It passes two parameters to this method: the type of requested context information as IEntityScopePair and the requestor’s identity as IContextRequestor. The return value of this interface is an IPermissionResponse object containing a string called permission and potentially the restricted type of the context that can be accessed by the requestor (if the access to this context is granted to the requestor). The permission can have three different values: "permit", "deny", or "notApplicable". The IPermissionResponse interface specifies the permitted context granularity using the IScope interface.

### Figure 16. IContextPrivacyService interface
The `IContextPrivacyManagement` interface, illustrated in Figure 17, provides the methods to retrieve, create, delete, or update the context privacy rules, which are stored as CPPL files in the Policy Repository.

In order to create a new privacy rule for a specific situation and add it to the repository, the (PAP) uses the `addContextPrivacy()`. The first parameter of this method is the policy name and the second parameter is an object containing the real context privacy rule. The return value of this method indicates if a new CPPL file has been added to the repository or not.

![Figure 17. IContextPrivacyManagement interface](image)

The `IContextPrivacyPolicy` interface represents a CPPL file object, providing four methods. The `getContextPrivacyRules()` method returns an array of the `IContextPrivacyRule` objects representing all `ContextPrivacyRule` elements in the CPPL file. The `getCPPLDoc()` method returns an XML document containing the CPPL file content. It is also possible to obtain the XML representation of the context privacy rule by invoking the `getXmlContextPrivacyRule()` method.
The `IContextPrivacyManagement` interface also provides two methods; `removeContextPrivacyRules()` and `updateContextPrivacyPolicy()` for removal from and update of the existing context privacy rules in the policy repository.

Additionally, `IContextPrivacyManagement` provides a method called `getContextPrivacyRules()` that returns a mapping of policy names to the corresponding context privacy policy objects. It is also possible to obtain only one rule if the policy name is specified as a parameter of the `getContextPrivacyRules()` method.

### 7.3 Context Privacy Manager Software Architecture

Figure 18 shows software architecture of the Context Privacy Manager. The main subcomponent is the Policy Decision Point (PDP) that implements the `IContextPrivacyService` interface. The PDP performs the following three functions:

1. It retrieves the context privacy rules from the policy repository and extracts the situational context parameters.

2. It obtains the values of the extracted context parameters by listening to context changes using the `IContextAccess` interface. Next, it determines if the current situation has changed based on the obtained values, and if so, updates the list of valid privacy rules.

3. It checks among the valid rules that are valid at the time of receiving the context request which of these rules relate to the requestor’s identity. The resulting rules are evaluated and if the access to the requested context is granted, the PDP retrieves this current context and optionally performs the transformation that is needed to restrict this context scope (if such a transformation is specified in the privacy rule).
The Social Relation Model is a file that contains a user’s contacts classified in different social relations groups (such as family, friends, and colleagues). The Social Relation Model is described using the FOAF (Friend Of A Friend)[79] ontology is often used by social networks to specify user profiles containing their personal information & activities and to link each user’s profile to the profile of his/her friends and acquaintances. Thus, in their FOAF profiles, users do not only describe their personal information and activities, but also the relations to other people (i.e., who they know). However, in FOAF there is only a “knows” relationship defined. In order to cover also family and colleagues relationships, we used an extended version of FOAF, proposed in [80], in which “knows” relationship is extended by sub-properties such as: “colleagueOf”, “friendOf”, etc.

The defined social relationships can be used by a user to specify privacy rules for a class of requestors, instead of specifying a separate rule for each requestor. For example, Alice can specify that she wants to reveal her location information to her friends when she is on vacation, without need to create a separate rule for each individual friend and without specify who these friends are in the rule. When defining a context privacy rule, Alice specifies the “FriendOf” relation as the requestor’s identity in the privacy rule condition.

When the CPM receives a context request which the requestor is “Bob”, the PDP checks if the Alice’s social relation model contains a `<rel:friendOf rdf:nodeID="Bob"/>` element indicating that the Bob is Alice’s friend, resulting in granting to Bob access to desired Alice’s context.

Figure 18. Internal structure of Context Privacy Manager
The Context Model defines a user’s context parameters that can be used to determine a situation or a sensitive context. Context parameters have to be modelled in such a way to allow different granularity levels of particular context information (e.g., a user’s location can have different levels of detail, such as address, city, country, or building).

The Policy Repository is responsible for storage and management of privacy rules. It provides the IContextPrivacyManagement interface. The Policy Repository checks the syntax of privacy rules and stores them as XML policy files. A simple file manager is created to store, modify, or delete policy files from the disk as requested by the user.

### 7.4 Context Privacy Manager Behavior

This section describes the behaviour of the Context Privacy Manager in two different phases: 1) after receiving a context request and 2) after receiving a context update.

![Diagram](figure19)

**Figure 19. Behavior of the CPM after receiving a subscription request**

When DS receives a subscription request from the SIP server (as illustrated in Figure 19), it invokes the `checkPermission()` method of the CPM, which in turn evaluates the
valid privacy rules, creates a `permissionResponse` message, and sends it back to the DS. If the permission response to access context has been granted, a SIP 200 OK message will be sent back to the SIP server, otherwise an Unauthorized 401 message will be returned.

Assuming that the requestor is permitted to subscribe, the DS sends a notification message with a PENDING status in its header indicating that the context middleware is processing the request. Next, the DS sends an activation message to the DM containing the type of the requested context information (metadataId) and the requestor’s identity. The DM queries the CM for the requested context information that are requested and after obtaining the context information sends a notification to the DS containing the obtained context value. Finally, the DS sends a SIP NOTIFY message to the SIP server containing the context information. Note that in this SIP NOTIFY message the status field in the header changed to ACTIVE, representing the successful retrieval of context information.

![Context Privacy Manager Diagram](image)

**Figure 20. Behaviour of Context Privacy Manager after a context changed event**

When a context information changes, assuming that at least one context requestor has successfully subscribed to this context type, the CM fires a `ContextChangedEvent`,
which is received by the Distribution Manager. The DM notifies the DS about this context change. The DS extracts the context data from the notification message and asks the DM for the list of authorized requestors for this context type (represented by the metadataId). The DM invokes the checkPermission() method at the CPM for each of the requestors, creates a list of authorized requestors, and sends it back to the DS. The DS sends individual SIP NOTIFYs containing the context data to all the authorized subscribers via the SIP server.
8 Context Privacy applications and tools

Based on the nature of each application the privacy preferences can be defined, updated, and enforced in different phases. In design phase, the *application developers* decide whether they need a privacy service or not and create an application-specific privacy preference for their application. In runtime, *end users* can update the policies created by application developers. Furthermore, the end users can create and update *general privacy policies* to be applied regardless of what kind of application is running on the device.

Figure 21 shows how application developers and end users can *create* their privacy preferences. The application developer creates a default privacy preference based on the specifications of the application (e.g., *RATP travel assistant* application should have a default privacy preference for all RATP users). These privacy preferences should be created as XML files according to the CPPL XML schema. Therefore, a MUSIC Studio tool, called *Privacy Policy Editor*, is used by application developers to create valid privacy policies according to CPPL XML schema. Additionally, end users can employ a MUSIC application, *User Privacy Manager (UPM)*, to create their own privacy policies.
8.1 Updating privacy policies

Figure 22 shows how end users can update their privacy preferences at runtime. The application developer designs a user interface (UI) to enable users to change their privacy preferences. For example, the RATP travel assistant application shows by default the user’s personal information (name, address, & disability attributes) and location to RATP agents, however some users might not like to reveal their personal information at all times to RATP agents. They can change their privacy preferences to show only their location and disability type to RATP agents while they are commuting in the Paris metro. Moreover, end users can update their general privacy policies through the UI provided by the User Privacy Manager application.

![Diagram](image.png)

**Figure 22. Updating the privacy preferences in runtime by user**

When an application starts, it reads the default privacy preferences from the XML file and adds them to the policy repository through the middleware (MW) privacy API. Whenever the user modifies his/her privacy rules through the UI, the application employs the MW privacy APIs to update the list of privacy preferences. When the application stops, the privacy rules are deleted from the policy repository.

8.2 Privacy policy storage

The user privacy manager transforms the user’s privacy setting to well-formed context privacy rules and employs the Context Privacy Manager to store the rules in the policy repository. Inside the repository, there is a general privacy policy file
(GenerealPreferences.xml) that includes all the rules defined by the user privacy manager application. There are also some privacy policy files representing the applications’ privacy preferences. These files are created by application developers using the MUSIC studio privacy policy editor and stored in the policy repository (as shown in Figure 23).

**Figure 23. Storage of Privacy Policy Files**

### 8.3 Privacy policy editor tool

A privacy policy editor tool is a MUSIC Studio Tool that enables application developers to create a default privacy policy for their applications. This tool is similar to the Context Query Language (CQL) editor described in [81]. The privacy policy XML schema is embedded in the tool according to which policy XML files are created (as depicted in Figure 24).

**Figure 24. Privacy Policy Editor Tool**
8.4 User privacy manager tool

Users might want to specify the general privacy preferences that should be enforced regardless of what kind of applications are running on their devices. A User Privacy Manager (UPM) application has been developed to provide this function. However, having a general privacy preferences defined by user may cause some conflicts with applications. For example, Alice creates a general privacy preference to prevent everybody from knowing her location, then, she installs an application which reveals her location information to her friends. As a result, this application will not be able to access Alice’s current location due to Alice’s general privacy preference. A possible solution to this problem is to notify Alice about this problem and prompt her if she agrees to override her privacy preference for this specific application.

The user privacy manager application is create to assist MUSIC middleware users to create and update their general privacy preferences. It has a graphical UI that allows users to edit their preferences. This GUI shows a list of context privacy rules each of which contains a description and a mapping from a situation to one or more privacy rules. Users can add new context privacy rules and edit or delete the existing rules. The rules that are selected by the user will be applied as general privacy preferences. For example, Figure 25 illustrates the case where the user has defined three context privacy rules, but has selected only “Context Privacy Rule2” which states that when the user’s activity is clubbing and if it is a weekend that his/her location should be revealed to Bob and her mood information can be shown to her friends.

![Figure 25. The General Privacy Preferences UI in User Privacy Manager Application](image)

Figure 25. The General Privacy Preferences UI in User Privacy Manager Application
Figure 26 shows how a user can define a new context privacy rule or edit an existing one. First, the user should select the *situation* in which a rule must be applied. If the situation has not been defined yet, the user can create a new situation. We will explain later how a situation can be defined. The next step is mapping of one or more privacy rules to the selected situation. If an appropriate rule has not already been defined, the user can add a new rule or edit an existing one. As showed in Figure 26, Alice has mapped two privacy rules to the “ClubbingOnWeekends” situation.

A *situation* can be defined as a logical combination of different context parameters values. Figure 27 shows a user-defined situation which indicates that this situation occurs when either the user is clubbing or when it is Saturday night. Using the GUI, the user can add an arbitrary number of conditions to define a situation.
A privacy rule can be defined by specifying the users’ identities that are granted or denied access to one or more context parameters. Figure 28 shows the GUI that enables users to define new or edit the existing privacy rules. The user can select to specify a rule for a group of people (that are retrieved from the user’s social relationship model) or particular individuals from a list. Next, the user selects the sensitive context parameters and potentially the scope of these parameters if the rule specifies to grant access to the selected context requestors. Finally, the user selects the type of permission which can be either to “Allow” or “Deny” access to the selected context information. The user chooses a name for this privacy rule and saves it to the policy repository.

Note that the list of context parameters depends on the context information that can be provided by context plug-ins running locally on the device. For example, the privacy rule illustrated in Figure 28 indicates that Bob is allowed to access the user’s current location at the building level.

Figure 28. Defining a privacy rule
9 Performance evaluation

The Context Privacy Manager is designed and implemented as an optional feature of MUSIC context middleware. Application developers can employ this feature to let the users decide who can access their information, in what situation and with what granularity. The main goal of this performance evaluation is to measure the cost of enabling privacy feature in form of delay that it adds to the services provided by context middleware at different stages and verify that these delays are acceptable compared to delays added by other systems such as SIP server.

9.1 System boundaries

Since the CPM is an internal component of context middleware, we can see the MUSIC middleware as the system under test and CPM as component under study, therefore we would only see the MUSIC Middleware as a system that receives the subscriptions and reply to it or send a notification to context subscribers through SIP server when a context change event occurs. Therefore, the System Under Test (SUT) is MUSIC Context Middleware and DS, and the Component Under Study (CUS) is Context Privacy Manager.

9.2 Services provided by system:

Two main functions of context-aware policy manager are: 1) to grant or deny access to particular context information after receiving a subscription request from a SIP server

![Diagram of system boundaries]
and 2) to update valid rules after situational context changes. These functions will be elaborated in more detail in the reminder of this section.

When an application subscribes to particular context information identified by an <entity, scope> pair, it sends a SUBSCRIBE message to a SIP URI of a resource list composed as: sip://<entity>..<scope>@[<domain>]. After receiving this subscription request, the SIP server will obtain the list of context source nodes SIP URIs to which it will send individual SUBSCRIBE messages, thus maintaining separate dialogs with each context source node. The individual subscription request will be intercepted by Distribution Service at the context source node, invoking the context privacy manager to check permission for the requested context specifying the context requestor identity and the desired <entity, scope> pair. The context privacy manager checks the valid privacy rules, creates a permission response message, and sends it back to the Distribution Service. If the permission response allows a distribution of requested context, the Distribution Service will send a SIP 200 OK response message back to the SIP server (as shown in Figure 30); otherwise an Unauthorized 401 message will be returned.

If the context requestor is permitted to subscribe to the desired context, the Distribution Service will return an immediate zero-length NOTIFY request, containing a PENDING state in its header to indicate that the middleware processes the request.

Figure 30. Successful subscription without privacy and plug-in is activated
Then it will send an *activate* message to the Distribution Manager, containing the `<entity, scope>` pair of the requested context and the identity of the context requestor. The Distribution Manager will subsequently ask the Context Manager to retrieve desired context and activate context plug-ins on context sources nodes to start publishing context data (unless they have already been activated, periodically publishing the desired context updates). Thus we distinguish between two situations here: when a context plug-in is activated and when it is not activated. Such a distinction will be important for the evaluation of the cost of context privacy, since the activation of context plug-ins adds a substantial delay (as described in Section X).

Figure 31. *Publishing context updates to permitted context requestors*

Figure 31 shows the process of publishing context updates to permitted context requestors. Context plug-ins will publish their context updates to the context manager, which will propagate these updates in the form of *context changed events* to all the context listeners, thus these events will subsequently be received by the Distribution Service. The Distribution Service will extract the data and the `<entity, scope>` pair from the notification message and will ask the Distribution Manager which of the context requestors (that have previously subscribed to this context) are authorized to receive this
context update. The Distribution Manager will in turn ask the Context privacy manager to check permission of the specified context requestors and after receiving the white list of authorized requestors, it will create an authorization response and send it back to the Distribution Service. The Distribution Service will send individual NOTIFY messages (containing these context updates) for each of the authorized context requestors back to the SIP server. The RLS module of the SIP server will wait for a predefined time (which is by default 1 second) to receive individual NOTIFYs from the context distribution user agents in the resource list, then it will aggregate them in a separate NOTIFY message, and send this aggregated NOTIFY message back to the context requestor/watcher.

9.3 Measurement description

The main goal of performance evaluation is to evaluate the cost of introducing context-aware privacy in the context middleware in terms of the delay added to the services provided by this middleware. These services are: (1) subscription to context information and (2) publication of context information. In a subscription service we distinguish two cases: when a context plug-in is activated and when it is not, in which we measure the subscription response time and the subscription throughput.

The **subscription response time** (as depicted in Figure 32) denotes the time from receiving the subscription request from the SIP server until sending the first notification with a content length greater than zero back to the SIP server. The possible outcomes of processing a subscription request are:

- **unsuccessful subscription** (identified by the SIP 404 Not Found response), which is caused by an unspecified <entity, scope> pair in a subscription request, an unknown requestor's identity, or a lack of registered context plug-ins that can provide the requested context information;

- **unauthorized subscription** (identified by the SIP 401 Unauthorized response) due to the enabled privacy feature which has prevented the subscriber, based on the valid privacy rules, to access the requested context information;
successful subscription (identified by the SIP 200 OK response) which can happen due to privacy being disabled or privacy being enabled and according to the valid privacy rules, subscriber is permitted to subscribe to the requested context.

![Subscription Response Time Diagram]

**Figure 32. Subscription response time**

The subscription throughput represents the rate at which the subscription requests can be correctly processed by the context middleware.

In the publication service, we measure the context publication time in two scenarios: when privacy is enabled and when it is disabled. The context publication time (as shown in Figure 33) is the time passed from receiving the context changed event (fired upon the change of the value of context information) until sending of notification containing this information to the SIP server. The publication service assumes that the subscribers have already successfully subscribed to the desired context information. The possible outcomes of this service are:

*All subscribers are notified*, where SIP notification is sent to all subscribers because the privacy feature is disabled or because the privacy feature is enabled and all the subscribers are permitted to subscribe to this context information.

*Some subscribers are notified*, where privacy feature is enabled and SIP notification is sent only to permitted subscribers. Note that some of the subscribers who have earlier successfully subscribed to this context may not be allowed to further receive
notifications because the user's situation has changed, updating the privacy preferences that prevent some subscribers to be informed about context changes.

*None of subscribers is notified*, where the privacy feature is enabled and none of the subscribers is further permitted to receive notifications about this context changes (according to the updated privacy rules upon the change of a user's situation).

![Context Privacy Manager Behaviour](image)

**Figure 33. Publication time**

### 9.4 Test Setup

To perform the measurements we developed a Load Generator that acts as a fake SIP server. It sends the subscription messages to the node that MUSIC middleware is running on it and responses to all SIP NOTIFY messages with proper 200 OK. The following characteristics of load generator were configurable:

- Number of subscribers
- Time interval between subscribe messages
- Activate plug-in before sending subscribe message
- Number of times the operation must be repeated

As described in Section 2.3.3, when the middleware starts, it sends the context types that it provides to the XCAP server. Since we did not need to fake the behaviour of XCAP server and it is only used at start-up, we used a real XCAP server. All the tests were performed in an isolated wired local area network having a switch (a Netgear fast Ethernet switch model FS108) between the XCAP Server, load generator and the MUSIC middleware. The test bed configuration is shown in Figure 34 and the hardware specification of the test bed is described in Table 5.

![Test bed configuration](image)

**Figure 34. Test bed configuration**

In order to measure the subscription response time and context publication time, we traced the logs of the timestamps of specific events into files with comma separated format. We also used `System.nanoTime()` rather than `System.currentTimeMillis()` to make the measurements more accurate.

<table>
<thead>
<tr>
<th>Load generator (SIP Server)</th>
<th>Context Source (MUSIC Middleware)</th>
<th>XCAP Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>Operating System</td>
<td>Processor</td>
</tr>
<tr>
<td>Acer Aspire 5030</td>
<td>Microsoft Windows XP</td>
<td>AMD Turion64@1.6 GHz</td>
</tr>
<tr>
<td>Dell XPS M1530</td>
<td>Microsoft Windows 7</td>
<td>Intel Pentium 4 @ 2.60 GHz</td>
</tr>
<tr>
<td>Fujitsu Siemens M420</td>
<td>Ubuntu (Linux)</td>
<td>Intel Pentium 4 @ 2.60 GHz</td>
</tr>
<tr>
<td>Operating System</td>
<td>Processor</td>
<td>RAM Memory</td>
</tr>
<tr>
<td>Microsoft Windows XP</td>
<td>AMD Turion64@1.6 GHz</td>
<td>512 MB</td>
</tr>
<tr>
<td>Microsoft Windows 7</td>
<td>Intel Pentium 4 @ 2.60 GHz</td>
<td>3 GB</td>
</tr>
<tr>
<td>Ubuntu (Linux)</td>
<td>Intel Pentium 4 @ 2.60 GHz</td>
<td>3 GB</td>
</tr>
<tr>
<td>Processor</td>
<td>RAM Memory</td>
<td>Network Adapter</td>
</tr>
<tr>
<td>AMD Turion64@1.6 GHz</td>
<td>512 MB</td>
<td>Broadcom NetXtreme Gigabit Ethernet</td>
</tr>
<tr>
<td>Intel Pentium 4 @ 2.60 GHz</td>
<td>3 GB</td>
<td>Marvel Yukon PCI-E Fast Ethernet Controller</td>
</tr>
<tr>
<td>Intel Pentium 4 @ 2.60 GHz</td>
<td>3 GB</td>
<td>Broadcom NetXtreme Gigabit Ethernet</td>
</tr>
<tr>
<td>Network Adapter</td>
<td>IP Address</td>
<td>IP Address</td>
</tr>
<tr>
<td>Broadcom NetXtreme Gigabit Ethernet</td>
<td>192.168.1.102</td>
<td>192.168.1.102</td>
</tr>
<tr>
<td>Marvel Yukon PCI-E Fast Ethernet Controller</td>
<td>192.168.1.202</td>
<td>192.168.1.100</td>
</tr>
</tbody>
</table>
9.5 Evaluation results

All the measurements results are presented as median values and are repeated 100 times in order to minimize the random error.

9.5.1 Subscription response time

In the first test we measured the subscription response time when a watcher subscribes to context information from a single context plug-in, in order to obtain a reference value which will serve for comparison with more complex evaluation scenarios.

<table>
<thead>
<tr>
<th></th>
<th>Privacy enabled</th>
<th>Privacy disabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plug-in activated</td>
<td>2.905 ms</td>
<td>2.883 ms</td>
</tr>
<tr>
<td>Plug-in not activated</td>
<td>491.692 ms</td>
<td>448.018 ms</td>
</tr>
</tbody>
</table>

Table 6. **Subscription response time for a single request**

As it can be seen from Table 6, the delay that the privacy adds to the subscription response time is 22 µs. Note that during this time we only check the valid privacy rules in the current situation, because a context plug-in is activated, the last known context value is stored in the cache, and the valid privacy rules in the current situation are updated. However, the subscription response time increases significantly if the context plug-in is not activated. In this case the cost of adding privacy feature becomes 43.674 ms. During this time, when the plug-in is activated, it fires a context changed event which is subsequently received by the context distribution manager, checking the permission of watchers that should be notified about this context change. Thus, the cost of privacy in the case a plug-in is not activated, includes both checking the valid privacy rules after receiving the subscription request and updating the list of valid privacy rules after receiving the context changed event. This cost of privacy (i.e., 43.674 ms) is minor when compared to the delay added by other context middleware components (i.e., 445.135 ms) when a plug-in is not activated.
The first test was repeated for different number of watchers in order to measure how the number of watchers affects the subscription response time. However, the result did not change when compared to the single watcher's results.

Next, we measured how the number of valid privacy rules affects the subscription response time. Note that privacy preferences are evaluated each time the user's situation changes, resulting in the list of privacy rules that are valid in the current user's situation. In this test we varied the number of valid privacy rules and measured the subscription response time. The results of this test are shown in Figure 35.

As it can be seen from Figure 35, when the number of privacy rules is less than 100, the overhead of processing these extra privacy rules is less than 1 ms. Even for large number of privacy rules (500 and 1000), the overhead takes less than 2 ms. This is a good result for the cost of context privacy, since in a real scenario, the number of valid rules in one situation will rarely (if ever) exceed 50 or 100.

![Figure 35. The affect of valid privacy rules on subscription response time](image)

### 9.5.2 Subscription throughput

The subscription throughput was measured in two different tests, with and without privacy enabled, both with an activated context plug-in:

By sending a fixed number of subscription requests to the SIP server at different time intervals (starting with 10 ms and reducing this interval by 1 ms each time until we
reach 1 ms) and measuring the percentage of successfully responded requests with a NOTIFY message.

By sending a burst of subscription requests to the SIP server and increasing the number of requests each new iteration (starting from 50 until reaching 1000 requests).

Using the first test, we determined the minimum inter-arrival time of subscription requests that can be successfully responded by the middleware. The number of successfully responded subscriptions was measured by the number of SIP NOTIFYs received by the load generator. This test was performed with and without privacy feature enabled, while the context plug-in was activated. The results of this test are shown in Figure 36. As it can be seen, subscription requests arriving in time intervals longer than 6 ms will be successfully responded, regardless of whether the privacy is enabled or not.

However, for inter-arrival times equal to 6 ms, when the privacy is enabled causes the loss of 2 out of 1000 notifications. For inter-arrival times less than 6 ms, there were subscription requests lost even without privacy enabled. Considering 7 ms as the minimum inter-arrival time at which arriving subscription requests can be successfully responded, we can calculate the maximum subscription throughput for our testbed to be 143 requests/second.

![Figure 36. The subscription throughput for a fixed number of requests with different time intervals](image)

The second test measured the subscription throughput for a burst of subscribe messages, sent without any delay between them (i.e., this is emulated by a single
process sending these requests in a for loop). This test was performed for different number of subscribers, each of which sent one request in a burst. The results of the second test are shown in Figure 37. It can be seen that when the privacy is disabled, the maximum number of concurrent subscription requests that can be received and successfully responded by the middleware is 50. With privacy enabled, this number increases to 70. This result is interesting because it shows that enabling the privacy increases the subscription throughput. The reason for such a behavior is processing of subscription requests that consumes more time than when there is privacy enabled, thus increasing the time in which the middleware sends NOTIFY requests back to the load generator. Consequently, less NOTIFY messages are lost on the way back to the load generator.

![Figure 37. The subscription throughput for a burst of subscribe messages](image)

**9.5.3 Context publication time**

The first test measures the context publication time for a single subscriber where the context plug-in is activated after receiving the subscription request. The current time is inserted as a context information in a body of the NOTIFY message. Upon the change of context value, the context distribution service sends this NOTIFY message to the load generator (acting on behalf of the SIP server). The context publication time is calculated as a difference of the current time and the context value (the context changed time). Table 7 depicts this test's results. The average context publication time is 6.2 ms larger when the privacy is enabled. However, observe the large difference between the minimum and maximum values in both cases. The reason for this phenomenon is in the way how context manager handles the change of context information. This is better
illustrated in Figure 38. The first time the context changes, the context publication time is the shortest, but it linearly increases until it exceeds 500 ms when the garbage collector is called, resulting in decreasing the context publication time to the initial value.

<table>
<thead>
<tr>
<th>Context publication time</th>
<th>Without privacy</th>
<th>With privacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>33.056 ms</td>
<td>30.78 ms</td>
</tr>
<tr>
<td>Maximum</td>
<td>503.913 ms</td>
<td>505.339 ms</td>
</tr>
<tr>
<td>Average</td>
<td>275.121 ms</td>
<td>281.344 ms</td>
</tr>
<tr>
<td>Median</td>
<td>272.2 ms</td>
<td>289.748 ms</td>
</tr>
</tbody>
</table>

Table 7. **Context publication time for a single subscriber**

![Context publication time graph](image)

**Figure 38. The change of context publication time due to the context manager's internal handling of context changed events**

The second test measures how the number of valid privacy rules impacts the context publication time, when a watcher is subscribed to a single context. The results are illustrated in Figure 39. As it can be seen, the overhead of checking the large number of valid privacy rules when compared to only one rule is small (i.e., up to 2.8 ms), which confirms the result of the test that measured the impact of the number of valid privacy rules to the subscription response time (depicted in Figure 35).
The third test measures how the number of subscribers affects the context publication time. After successfully subscribing watchers to a particular context type, we change the context and measure the context publication time. Note that when a context changed event occurs, the context distribution service receives a notification that contains the context type whose value has been changed and this context value. As the privacy was enabled, the context distribution service retrieves from the context privacy manager a list of subscribers that are authorized to receive these context updates and sends a SIP NOTIFY message to each of these subscribers. Results of the third test are shown in Figure 40.
9.5.4 Situational context publication time

By *situational* context we refer to the context information, which when changes also changes the situation and consequently updates the valid privacy rules of a user. For example, Alice defines a context privacy rule that is depicted in Listing 27. This rule specifies that when her activity is clubbing, Alice's friends can see her activity and her location. In this example “status.activity” is a situational context because when it changes the valid privacy rules must be updated. The location information is not situational because there is no situation that depends on Alice’s location.

```xml
<ContextPrivacyRule contextPrivacyRuleId="Scenario1">
  <Situations>
    <Situation situationId="clubbing">
      <Entity>#user|Alice</Entity>
      <Conds><Cond>
        <Constraint param="status.activity" op="EQ" value="clubbing"/>
      </Cond></Conds>
    </Situation>
  </Situations>
  <RuleSet>
    <Rule effect="Permit">
      <Identity><Relation relation="friendsOf"/></Identity>
      <ContextParams><ContextParam>
        <Entity>#user|Alice</Entity>
        <Scope>#status.activity</Scope>
        <Scope>#location.street</Scope>
      </ContextParam></ContextParams>
    </Rule>
  </RuleSet>
</ContextPrivacyRule>
```

*Listing 27. A situational context*

A test is performed to determine the cost of updating valid privacy rules due to publication of situational context. The overhead of publishing situational context is 14 ms when there is only one context privacy rule. With the second test we measured how the number of privacy rules impacts the publication time of the situational context. This test’s results are shown in Figure 41. As it can be seen, the publication time linearly increases with the number of context privacy rules. The time to process one context privacy rule is $(375.73-308.42)/99=0.68$ ms. Assuming that an average mobile user will have less than 100 context privacy rules, this is an acceptable result.
Figure 41. **The average context publication time for multiple subscribers**

However, for a large number of privacy rules, we might lose some context if it changes more frequently than once in a second. For example, having 1000 privacy rules will add a delay of 680 ms which along with the context publication time becomes 1 second. Therefore, if the context changes more frequently than once in a second, there is a potential risk of not being able to inform all the watchers about the context change(s).
10 Conclusions and future work

This chapter draws the final conclusion of our work bringing to focus the achievements of the thesis in terms of the reviewed literature, the analyzed approaches, and the contributions made. It also contains the suggestions for future work.

10.1 Conclusions and discussions

The goal of this thesis was to design and implement a privacy management architecture within the MUSIC project for controlling access to context information of mobile users. After an extensive review of the literature to understand the concepts of privacy, context, and context-aware computing, we extracted the privacy requirements for context-aware applications from scenarios that motivate the need for context-aware privacy in daily lives of average mobile users. We discussed and evaluated the current approaches based on our privacy requirements. Through this discussion, we realized that none of them satisfies our requirements and therefore we decided to introduce a new approach to address the problem.

We introduced a context-aware privacy policy language (CPPL), which can be used to represent privacy preferences of mobile users in context-aware environments. A user's context is used in context-aware privacy rules to specify which of these rules are valid in particular situation(s). CPPL enables a user to define situations using a set of context parameters that are defined in a context model. When a user's current situation changes, a list of valid rules is updated by a privacy mechanism, thus leaving only a smaller subset of relevant rules to be evaluated upon arrival of a context query. CPPL enables a user to specify privacy rules based on a social relationship with a context requestor, thus reducing the number of rules that need to be specified by the user and that consequently need to be evaluated by the privacy mechanism. In the existing context-dependant privacy policy languages a user's context is used as an additional condition in a rule (along with a requestor's identity). In order to process a context request, the privacy mechanisms supporting these languages need to process all the specified rules and retrieve all context values that are used to define privacy
preferences, which can in the case of evaluation of large number of rules, significantly increase a privacy policy evaluation time.

In order to enforce the privacy preferences of MUSIC users, defined as CPPL files, we had to design a privacy policy management system. We analyzed the possible architectures that can be integrated to the MUSIC architecture based on our privacy requirements and introduced a decentralized architecture which is described in details in Chapter 7. We added a new component to the MUSIC context middleware called Context Privacy Manager that can be employed by other components at any time to check the permission to access context information for a requestor. The CPM is an optional component which is implemented as an OSGi bundle. In order to enable the privacy feature of the Context Middleware, it is enough to deploy the CPM bundle together with other bundles such as: DS, DM, and CM.

The CPM provides the programming APIs that can be used by MUSIC application developers to add, remove or update the context privacy rules at runtime. Therefore, they can let their application users to change their privacy preferences when the application is running.

In order to test the functionality of CPM two sample MUSIC application were developed. The first application was a simulation application that controls the value of eight simulated context plug-ins and has a GUI to change the context values. The second application was a simulated client that subscribes for context information provided by the first application. These simulation applications were used in different phases of implementation and improved the process of debugging. A load generator was also developed to act as a fake SIP server that sends subscription requests to the MUSIC middleware.

The evaluation of designed context privacy management system revealed that enabling the privacy adds a delay of 131µs (4 %) to the average subscription response time (3,263ms). Moreover, the scalability tests showed that the context-aware privacy management system is highly scalable in a sense that increasing the number of privacy rules from 1 to 1000 has a slight affect on performance of the system by increasing the context publication time from 284,55 ms to 287,34 ms (0,9 %)
Another result of evaluation was measuring the precise subscription throughput of the MUSIC middleware, that is, the number of subscribe messages per second that can be responded without dropping any request. The subscription throughput of MUSIC middleware was 143 requests/second. Enabling the privacy did not change the throughput of the system.

Evaluating the results from the measurements, we can see that the Context Privacy Manager doesn't add a considerable delay to the context distribution.

10.2 Suggestions for future research

Some of the suggestions for extending and enhancing the work of this thesis are:

Implementation of CPM: In the current implementation the following features are missing and can be implemented in future:

- Time constraints: As described in Section 6.3, CPPL situation conditions can have a time constraint to validate a condition based on the current time. The CPM which is an implementation of CPPL language should support time constraints.

- Multiple situations: In the current implementation of the CPM, only one situation can be mapped to a rule set. The implementation can be improved to support multiple situations.

- Rule evaluation combining algorithms: In the current implementation of the CPM, the combination algorithm of privacy rules is hardcoded to be "deny-overrides" which allows a single evaluation of deny taking precedence over any number of permit, not applicable or indeterminate results. More combination algorithms can be implemented in future.

- Standalone privacy framework: The proposed architecture in this thesis is coupled to the MUSIC middleware architecture and therefore is not general enough to be used in any platform. However, the CPPL is generic enough to be applied on different platforms. A possible future work is design a privacy framework for context-aware environments that employs the CPPL for representing the privacy preferences of mobile users and is general enough to be applied on different domains.
11 References


[52] Lessig, L. The Architecture of Privacy. In Taiwan NET'98. 1998. Taipei, Taiwan


[66] Lorrie Cranor, Marc Langheinrich, and Massimo Marchiori. A P3P preference exchange language 1.0 (APPEL1.0). See www.w3.org/TR/P3P-preferences/, April 2002.


Appendix A  CPPL XML Schema

<?xml version="1.0" encoding="utf-8" ?>
<!--Created with Liquid XML Studio - FREE Community Edition 7.1.4.1284
(http://www.liquid-technologies.com)-->
<xs:schema xmlns:cppl="http://ContextPPL/1.0"
attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://ContextPPL/1.0"
xmllns:xsi="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Description" type="xs:string" />
  <xs:element name="ContextPrivacyRules">
    <xs:annotation>
      <xs:documentation>Root of a Context Privacy Policy document</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" ref="cppl:Description" />
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="cppl:ContextPrivacyRule" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ContextPrivacyRule" type="cppl:ContextPrivacyRuleType" />
  <xs:complexType name="ContextPrivacyRuleType">
    <xs:sequence>
      <xs:element minOccurs="0" ref="cppl:Description" />
      <xs:element ref="cppl:Situations" /> 
      <xs:element ref="cppl:RuleSet" />
    </xs:sequence>
    <xs:attribute name="contextPrivacyRuleId" type="xs:ID" use="required" />
    <xs:attribute name="active" type="xs:boolean" />
  </xs:complexType>
  <xs:element name="Situations" type="cppl:SituationsType" />
  <xs:complexType name="SituationsType">
    <xs:choice>
      <xs:element maxOccurs="unbounded" ref="cppl:Situation" />
      <xs:element ref="cppl:AnySituation" />
    </xs:choice>
  </xs:complexType>
  <xs:element name="Situation" type="cppl:SituationType" />
  <xs:complexType name="SituationType">
    <xs:sequence>
      <xs:element minOccurs="0" ref="cppl:Description" />
      <xs:element ref="cppl:Entity" />
      <xs:element ref="cppl:Conds" />
    </xs:sequence>
    <xs:attribute name="situationId" type="xs:ID" use="required" />
  </xs:complexType>
  <xs:element name="Entity">
    <xs:annotation>
      <xs:documentation>characterized entity</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:sequence> 
            <xs:element ref="cppl:Description" />
            <xs:element ref="cppl:Entity" />
            <xs:element ref="cppl:Conds" />
          </xs:sequence>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
<xs:attribute name="ontConcept" type="xs:string" use="optional"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>

<!-- /situation/conditions -->
<xs:element name="Conds">
<xs:annotation>
<xs:documentation>complex condition</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:choice>
<xs:element ref="cppl:Cond"/>
<xs:element ref="cppl:CondOp"/>
</xs:choice>
</xs:complexType>
</xs:element>

<!-- /situation/conditions/condition -->
<xs:element name="Cond">
<xs:annotation>
<xs:documentation>simple condition</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:complexContent mixed="false">
<xs:extension base="cppl:CondType"/>
</xs:complexContent>
</xs:complexType>
</xs:element>

<!-- /situation/conditions/condition/condType -->
<xs:complexType name="CondType">
<xs:annotation>
<xs:documentation>Type Cond</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element minOccurs="0" ref="cppl:Description"/>
<xs:element minOccurs="0" ref="cppl:TimeConstraint"/>
<xs:choice minOccurs="0">
<xs:element ref="cppl:Constraint"/>
<xs:element ref="cppl:Logical"/>
</xs:choice>
</xs:sequence>
</xs:complexType>

<!-- /situation/conditions/condition/conditionType -->
<xs:simpleType name="ConditionType">
<xs:annotation>
<xs:documentation>Values for condition types</xs:documentation>
</xs:annotation>
<xs:restriction base="xs:string">
<xs:enumeration value="ONCHANGE"/>
</xs:restriction>
</xs:simpleType>

<!-- /situation/conditions/condition/timeConstraint -->
<xs:element name="TimeConstraint" type="cppl:TimeConstraintType"/>
<xs:complexType name="TimeConstraintType">
<xs:sequence>
<xs:element minOccurs="0" ref="cppl:DateRange"/>
<xs:element minOccurs="0" ref="cppl:Interval"/>
</xs:sequence>
</xs:complexType>

<xs:element name="DateRange" type="cppl:DateRangeType"/>
<xs:complexType name="DateRangeType">
<xs:attribute name="from" type="xs:dateTime" use="required"/>
<xs:attribute name="to" type="xs:dateTime" use="required"/>
</xs:complexType>
<xs:element name="Interval" type="cppl:IntervalType" />
<xs:complexType name="IntervalType">
  <xs:sequence minOccurs="0">
    <xs:element ref="cppl:TimeRange" />
  </xs:sequence>
  <xs:attribute name="daysOfWeek" type="cppl:DaysOfWeekType" />
  <xs:attribute name="months" type="cppl:MonthType" />
  <xs:attribute name="daysOfMonth" type="cppl:DaysOfMonthType" />
</xs:complexType>
<xs:element name="TimeRange" type="cppl:TimeRangeType" />
<xs:complexType name="TimeRangeType">
  <xs:attribute name="startTime" type="xs:time" use="required" />
  <xs:attribute name="endTime" type="xs:time" use="required" />
</xs:complexType>
<xs:simpleType name="DaysOfWeekType">
  <xs:restriction base="xs:string">
    <xs:pattern value="(((1-7),(1-7))|(1-7)|((MON|TUE|WED|THU|FRI|SAT|SUN),)*(MON|TUE|WED|THU|FRI|SAT|SUN) (C)?)|((MON|TUE|WED|THU|FRI|SAT|SUN) (-) (MON|TUE|WED|THU|FRI|SAT|SUN) (C)?)|((1-7)|(MON|TUE|WED|THU|FRI|SAT|SUN))(L)?)|((L)?)" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="MonthType">
  <xs:restriction base="xs:string">
    <xs:pattern value="((((0?[1-9]|1[0-9])|0?[1-9]|1[0-9]),)*(0?[1-9]|1[0-9]))|((0?[1-9]|1[0-9]))|((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC),)*(JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC) |((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC) |((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC))|((L)?)|((L)?)|((L)?)|((L)?)" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DaysOfMonthType">
  <xs:restriction base="xs:string">
    <xs:pattern value="((((0?[1-9]|1[0-9]),)*(0?[1-9]|1[0-9]))|((0?[1-9]|1[0-9]))|((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC),)*(JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC) |((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC) |((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC))|((L)?)|((L)?)|((L)?)|((L)?)|((L)?)|((L)?)|((L)?)" />
  </xs:restriction>
</xs:simpleType>
<xs:enumeration value="NLT" />
<xs:enumeration value="CONT" />
<xs:enumeration value="NCONT" />
<xs:enumeration value="STW" />
<xs:enumeration value="NSTW" />
<xs:enumeration value="ENW" />
<xs:enumeration value="NENW" />
<xs:enumeration value="EX" />
<xs:enumeration value="NEX" />
</xs:restriction>
</xs:simpleType>
<!-- /situation/conditions/condition/logical -->
<xs:element name="Logical">
  <xs:complexType>
    <xs:complexContent mixed="false">
      <xs:extension base="cppl:LogicalType" />
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<!-- /situation/conditions/condition/logical/logicalType -->
<xs:complexType name="LogicalType">
  <xs:sequence>
    <xs:element minOccurs="2" maxOccurs="unbounded" ref="cppl:Constraint" />
  </xs:sequence>
  <xs:attribute name="op" type="cppl:LogicalOperator" use="required" />
</xs:complexType>
<!-- /situation/conditions/condition/logical/logicalOperator -->
<xs:simpleType name="LogicalOperator">
  <xs:annotation>
    <xs:documentation>Values for logical operators</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="AND" />
    <xs:enumeration value="OR" />
  </xs:restriction>
</xs:simpleType>
<!-- /situation/conditions/conditionOp -->
<xs:element name="CondOp">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="2" maxOccurs="unbounded" ref="cppl:Cond" />
    </xs:sequence>
    <xs:attribute name="op" type="cppl:LogicalOperator" use="required" />
  </xs:complexType>
</xs:element>
<!-- /situation/AnySituation -->
<xs:element name="AnySituation" />
<!-- /privacyRule/ruleSet -->
<xs:element name="RuleSet" type="cppl:RuleSetType" />
<xs:complexType name="RuleSetType">
  <xs:sequence>
    <xs:element maxOccurs="unbounded" ref="cppl:Rule" />
  </xs:sequence>
</xs:complexType>
<!-- /ruleSet/rule -->
<xs:element name="Rule" type="cppl:RuleType" />
<!-- /ruleSet/rule/ruleType -->
<xs:complexType name="RuleType">
  <xs:sequence>
    <xs:element minOccurs="0" ref="cppl:Description" />
    <xs:element ref="cppl:Identity" />
    <xs:element ref="cppl:ContextParams" />
  </xs:sequence>
  <xs:attribute name="effect" type="cppl:EffectType" />
<xs:simpleContent>
  <xs:extension base="xs:string">
    <xs:attribute name="ontConcept" type="xs:string" use="optional" />
    <xs:attribute name="ontRep" type="xs:string" use="optional" />
  </xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
</xs:complexType>
</xs:schema>
Appendix B  CPPL Examples

This appendix contains CPPL examples for the motivation scenario described in Chapter 21.

Appendix B.1  CPPL Files on Bob’s device

Bob has installs two MUSIC applications on his device. This appendix contains the CPPL files used by each application.

HCI Health Terminal

```xml
<?xml version="1.0"?><ContextPrivacyRules xmlns="http://ContextPPL/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Description>HCI-Terminal application Context Privacy Policy</Description>
  <ContextPrivacyRule contextPrivacyRuleId="Scenario1" active="true">
    <Description>Always show my health information to HCI center</Description>
    <Situations>
      <AnySituation/>
    </Situations>
    <RuleSet>
      <Rule effect="Permit">
        <Description>The requestor is HCI admin (HCI Center)</Description>
        <Identity>
          <One id="sip:admin@HCI.com"/>
        </Identity>
        <ContextParams>
          <ContextParam>
            <Entity>http://www.ist-music.eu/Ontology_v0_1.xml#user|Bob</Entity>
            <Scope>http://www.ist-music.eu/Ontology_v0_1.xml#healthInfo</Scope>
          </ContextParam>
        </ContextParams>
      </Rule>
    </RuleSet>
  </ContextPrivacyRule>
  <ContextPrivacyRule contextPrivacyRuleId="Scenario2" active="true">
    <Description>Show my location and my health information to HCI employees in Emergency situation</Description>
    <Situations>
      <Situation situationId="Emergency">
        <Description>Health status is not normal</Description>
        <Entity>#user|Bob</Entity>
        <Conds>
          <CondOp op="OR">
            <Cond>
              <Description>Abnormal blood pressure</Description>
              <Logical op="OR">
                <Constraint param="#healthInfo.bloodPressure.systolic" op="NEQ" value="105" delta="15.0"/>
                <Constraint param="#healthInfo.bloodPressure.distorlic" op="NEQ" value="70" delta="10.0"/>
              </Logical>
            </Cond>
          </CondOp>
        </Conds>
      </Situation>
    </Situations>
  </ContextPrivacyRule>
</ContextPrivacyRules>
```
<Cond>
    <Description>Abnormal heart rate</Description>
    <Constraint param="#healthInfo.heartRate" op="NEQ" value="75" delta="25.0"/>
</Cond>

<Cond>
    <Description>Abnormal body temperature</Description>
    <Constraint param="#healthInfo.bodyTemp" op="NEQ" value="36.8" delta="0.7"/>
</Cond>

</ContextOp>
</Conds>
</Situations>
</RuleSet>
</ContextPrivacyRule>
</ContextPrivacyRules>

RATP Travel Assistance

<?xml version="1.0" encoding="utf-8"?>
<ContextPrivacyRules xmlns="http://ContextPPL/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Description>RATP travel assistant Context Privacy Policy</Description>
    <ContextPrivacyRule contextPrivacyRuleId="Scenario3">
        <Description>Show my exact location to RATP agents</Description>
        <Situations>
            <AnySituation/>
        </Situations>
        <RuleSet>
            <Rule effect="Permit">
                <Identity>
                    <Many domain="www.RATP.org"/>
                </Identity>
                <ContextParams>
                    <ContextParam>
                        <Entity>#user|Bob</Entity>
                        <Scope>#location.longitude</Scope>
                        <Scope>#location.latitude</Scope>
                    </ContextParam>
                </ContextParams>
            </Rule>
        </RuleSet>
    </ContextPrivacyRule>
</ContextPrivacyRules>
Appendix B.2 CPPL files on Alice's device

Alice has installs two MUSIC applications on her device. This appendix contains the CPPL files used by each application.

HCI Assistant Console

```xml
<ContextPrivacyRules xmlns="http://ContextPPL/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Description>HCI Privacy Policy For Employees</Description>
  <ContextPrivacyRule contextPrivacyRuleId="Scenario4">
    <Description>Show my location information to HCI center during my working hours</Description>
    <Situations>
      <Situation situationId="workingHours">
        <Description>Working hours situation</Description>
        <Entity ontConcept="prefix:music:environment.dateTime">#dateTime</Entity>
        <Conds>
          <Cond>
            <TimeConstraint>
              <Interval daysOfWeek="MON-FRI">
                <TimeRange startTime="09:00:00" endTime="18:00:00"/>
              </Interval>
            </TimeConstraint>
          </Cond>
        </Conds>
      </Situation>
    </Situations>
    <RuleSet>
      <Rule effect="Permit">
        <Description>The requestor is HCI admin (HCI Center)</Description>
        <Identity>
          <One id="sip:admin@www.HCI.com"/>
        </Identity>
        <ContextParams>
          <ContextParam>
            <Entity>#user|Alice</Entity>
            <Scope>#location.longitude</Scope>
            <Scope>#location.latitude</Scope>
            <!-- The transformation of location to a circle of 100 meters is not supported in this version! -->
          </ContextParam>
        </ContextParams>
      </Rule>
    </RuleSet>
  </ContextPrivacyRule>
</ContextPrivacyRules>
```

Friend Finder

```xml
<ContextPrivacyRules xmlns="http://ContextPPL/1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Description>Alice general privacy policy</Description>
  <ContextPrivacyRule contextPrivacyRuleId="Scenario5">
    <Description>Show my mood to my husband when I am in Paris</Description>
  </ContextPrivacyRule>
</ContextPrivacyRules>
```
<Situations>
  <Situation situationId="inParis">
    <Description>Paris situation</Description>
    
    <Entity ontConcept="prefix:music:username">#user|Alice</Entity>
    
    <Conds>
      <Cond>
        <Constraint param="civilAddress.city" op="EQ" value="Paris"/>
      </Cond>
    </Conds>
  </Situation>
</Situations>

<RuleSet>
  <Rule effect="Permit">
    <Description>Show my location to my husband</Description>
    
    <Identity>
      <Relation relation="spouseOf"/>
    </Identity>
    
    <ContextParams>
      <ContextParam>
        <Entity ontConcept="prefix:music:username">#user|Alice</Entity>
        
        <Scope ontConcept="prefix:music:status:mood">#status.mood</Scope>
      </ContextParam>
    </ContextParams>
  </Rule>
</RuleSet>

<ContextPrivacyRule contextPrivacyRuleId="Scenario6">
  <Description>Show my city location to my friends when I am on vacation</Description>
  
  <Situations>
    <Situation situationId="onVacation">
      <Description>Vacation Situation</Description>
      
      <Entity ontConcept="prefix:music:username">#user|Alice</Entity>
      
      <Conds>
        <Cond>
          <Constraint param="status.activity" op="EQ" value="Vacation"/>
        </Cond>
      </Conds>
    </Situation>
  </Situations>

  <RuleSet>
    <Rule effect="Permit">
      <Description>Show my city to my friends</Description>
      
      <Identity>
        <Relation relation="friendOf"/>
      </Identity>
      
      <ContextParams>
        <ContextParam>
          <Entity ontConcept="prefix:music:username">#user|Alice</Entity>
          
          <Scope ontConcept="prefix:music:civilAddress">#civilAddress.city</Scope>
        </ContextParam>
      </ContextParams>
    </Rule>
  </RuleSet>
</ContextPrivacyRule>
<ContextPrivacyRule contextPrivacyRuleId="Scenario7">
  <Description>Show my street location to my friends when it is weekend</Description>
  <Situations>
    <Situation situationId="weekends">
      <Description>Weekends situation</Description>
      <Entity>#dateTime</Entity>
      <Conds>
        <Cond>
          <Logical op="OR">
            <Constraint param="date.weekday" op="NEQ" value="saturday"/>
            <Constraint param="date.weekday" op="NEQ" value="sunday"/>
          </Logical>
        </Cond>
      </Conds>
    </Situation>
  </Situations>
  <RuleSet>
    <Rule effect="Permit">
      <Description>Show my street to my friends</Description>
      <Identity>
        <Relation relation="friendOf"/>
      </Identity>
      <ContextParams>
        <ContextParam>
          <Entity ontConcept="prefix:music:username">#user|Alice</Entity>
          <Scope ontConcept="prefix:music:civilAddress">#civilAddress.street</Scope>
        </ContextParam>
      </ContextParams>
    </Rule>
  </RuleSet>
</ContextPrivacyRule>