Context-addressed communication dispatch
Context-addressed communication dispatch

ALISA DEVLIĆ

Licentiate Thesis in
Communication Systems
Stockholm, Sweden 2009
Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologi licentiatexamen 29 May 2009 klockan 13.00 i sal C2, KTH-Electrum, Kungliga Tekniska Högskolan, Isafjordsgatan 22, Kista.

© Alisa Devlić, April 2009

Tryck: Universitetsservice US AB
ABSTRACT

This research concerns exploiting knowledge of the user's environment (i.e., context information) to enrich a user's communication making it more personal, by ensuring that the user receives only relevant messages and calls in his/her current context, and to facilitate more opportunities for communication interactions with people that are in the same context and that share the same interests as this user. We describe in this licentiate thesis the concepts of context-addressed messaging and context-aware session control that enable users to: (1) send messages to others based on their context, rather than their network address and (2) to initiate, adapt, and terminate user's communication sessions based on this user's current context, respectively. These concepts address questions such as: how to discover, select, and switch to an optimal communication means to meet varying user, contextual, communication, and device resource requirements and preferences. A key to solving these problems is to create a representation of the user's context-dependent preferences and to process the user's context-dependent preferences which are part of context triggers. These context triggers can initiate a communication event upon a particular context update. Additionally, in order to provide the described context-aware communication functions, these mechanisms need timely access to the acquired (desired) context information. This in turn raises a plethora of other questions, such as how to discover sensors that provide the desired context information; how to acquire raw context data from these sensors; how to abstract, process, and model this data to become "understandable" to applications and system components; and how to distribute this context to applications that are running on different nodes.

This research is split into three different parts. The first part concerns investigating and implementing context management functions. As part of this research we propose a novel approach for context synthesis using context operators. We also propose a design architecture for context-aware middleware that mediates between the sensors and applications, and that is able to share and retrieve context from other nodes in the network. The second part of our research concerns our proposed mechanism for context-addressed messaging. To implement this mechanism we designed our own message format, called the Common Profile for Context-Addressed Messaging (CPCAM) that is able to use any high level context to compose a context-based address. Additionally, we proposed to use context-based filtering to find the correct message recipients and determine if this message is relevant to these potential message recipients in their current context, as well as to deliver this message to the recipients' preferred device that is adapted using their preferred communication means. At the end of this second part we design context-addressed messaging system operations on top of a SIP and SIMPLE-based network infrastructure. The third part of our research describes context-aware session control mechanisms using context switch and context trigger constructs. A context-switch selects an action from a set of context-dependent actions upon an incoming communication event based on the receiver's current context. In contrast, a context trigger initiates an action based on a context update and the user's preferences that are specified in this updated context. This part illustrates in several examples the context-aware session control mechanisms, i.e. the initiation of a communication session based on the match of a user's preferences and current context, as well as adaptation and (if necessary) termination of an ongoing communication session based upon the user's context-dependent preferences.

The research leading to this licentiate has created network and system level models necessary for implementation of a context-addressed communication system that would enable users to easily design their own personalized, context-aware communication services. The necessary constructs and properties of these models are designed and analyzed in the
thesis, as well as in conference papers and other documents published in the process of doing the research for this thesis. A number of remaining open issues and challenges have been outlined as part of the future work.

**Keywords:** Context-addressed messaging, Context-aware session control, Context-aware communication, Context-aware call signaling, Context-based session initiation.
ACKNOWLEDGEMENTS

First and foremost, I would like to first my supervisor, Prof. Gerald Q. Maguire Jr., for his incredible guidance, support, valuable comments, and discussions, which helped me conduct research for this thesis.

I would also like to thank my company Appear Networks, especially the CEO Xavier Aubry, who gave me an opportunity to work in the research of context-aware services within the scope of European research projects and become an industrial doctoral student. I thank all the colleagues and members of the research team for all their comments and suggestions.

This project was partly funded by two EU FP6 projects: MIDAS (Middleware Platform for Developing and Deploying Advanced Mobile Services) and MUSIC (Self-adapting Applications for Mobile Users In Ubiquitous Computing Environments). I would also like to thank Swedish Institute for bringing me to Sweden as a guest researcher and financing my stay and research during nine months at Wireless@KTH.

Many thanks to my opponent Rasmus Olsen, assistant Professor at Aalborg University, for all the good comments he provided to this thesis, and to my committee members, Professor Mark Smith and Dr. Fredrik Kilander for their feedback and encouragement.

Last but not least I would like to thank my husband Alan for his inexhaustible support, love, and patience when I was working many late hours on my thesis; and to the rest of my family and friends, who always believed in me and supported me to pursue my goals.
# Table of Contents

Abstract ............................................................................................................................ i  
Acknowledgements ........................................................................................................ iii  
Table of Contents ........................................................................................................... iv  
Table of Figures ............................................................................................................. vi  

Chapter 1: Introduction ................................................................................................. 1  
1.1 Motivation ................................................................................................................ 1  
1.1.1 Example scenarios of context-aware communication ......................................... 2  
1.2 Problem statement .................................................................................................... 5  
1.3 Thesis overview ....................................................................................................... 6  
1.4 Contributions .......................................................................................................... 8  
1.5 Summary of Published papers and other documents .............................................. 8  

Chapter 2: Context management ................................................................................... 12  
2.1 Definition of context .............................................................................................. 12  
2.2 Context-aware system ............................................................................................ 14  
2.2.1 Context discovery vs. context distribution ......................................................... 15  
2.3 Context management ............................................................................................. 17  
2.3.1 Context sensing .................................................................................................. 17  
2.3.2 Context modeling .............................................................................................. 19  
2.3.3 Context synthesis ............................................................................................. 25  
2.3.4 Context distribution and querying .................................................................... 34  
2.4 Context-aware system architecture ....................................................................... 36  
2.5 Summary ................................................................................................................ 41  

Chapter 3: Context-aware communication ................................................................... 44  
3.1 Communication model .......................................................................................... 44  
3.2 Introducing context information into the communication model ......................... 46  
3.3 Summary ................................................................................................................ 50  

Chapter 4: Context-addressed messaging ................................................................... 52  
4.1 Introduction ............................................................................................................. 52  
4.2 Requirements ........................................................................................................... 52  
4.3 Application-level communication types ............................................................... 53  
4.4 Design of context-addressed messaging system infrastructure ......................... 54  
4.4.1 Context-addressed messaging mechanism ....................................................... 55  
4.4.2 Common Profile for Context-Addressed Messaging (CPCAM): Message format ........................................................................................................... 58  
4.4.3 Address resolution ......................................................................................... 60  
4.4.4 Context-based filtering ................................................................................. 60  
4.4.5 Context-addressed messaging system architecture ....................................... 62  
4.5 Related work on context-addressed messaging ................................................... 64  
4.5.1 Distributed location-based infrastructure ......................................................... 64
4.5.2 Content-based publish/subscribe ......................................................................................... 67
4.5.3 Variations of multicast ....................................................................................................... 69
4.5.4 Similarity-based profile matching ......................................................................................... 71
4.5.5 Restricted flooding (narrowcast) and ontology-based reasoning ............................................... 72
4.5.6 Preference rule-based reasoning ............................................................................................ 73
4.6 Implementing context-addressed messaging on top of SIP network infrastructure 74
4.6.1 Context-addressed messaging operations ............................................................................... 78
4.6.2 Context distribution operations ........................................................................................... 79
4.7 Summary ............................................................................................................................... 86
Chapter 5: Context-aware session control .....................................................................89
5.1 Introduction ............................................................................................................................. 89
5.2 Context switch .......................................................................................................................... 91
5.2.1 Context-aware VoIP prototype ........................................................................................... 92
5.2.2 Evaluation of context-based CPL scripts ............................................................................. 96
5.3 Context trigger .......................................................................................................................... 97
5.3.1 XML DTD for proposed CPL extensions ........................................................................... 99
5.4 Context-based session initiation ............................................................................................. 101
5.5 SIP network infrastructure for context-aware session control .............................................. 103
5.6 Context-aware session adaptation ............................................................................................. 108
5.7 Context-addressed messaging ............................................................................................... 110
5.8 Summary ............................................................................................................................... 111
Chapter 6: Conclusions ................................................................................................ 114
6.1 Conclusions ............................................................................................................................ 114
6.2 Open issues and future work ................................................................................................. 116
References ................................................................................................................................... 118
List of Acronyms, abbreviations, and standards ................................................................. 125
Appendix. PIM-SM: Calculation of the time to rebuild a multicast distribution tree 126
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Main components of context-addressed communication dispatch system</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Context entities and their relations</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Context-aware system consisting of the following entities: context sources, context space, and context consumers</td>
<td>14</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Context-aware system</td>
<td>17</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Context plug-ins</td>
<td>18</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Contextually extended ORM</td>
<td>21</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Cues</td>
<td>21</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Context modeling using situation theory</td>
<td>22</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Comparison of context modeling techniques</td>
<td>24</td>
</tr>
<tr>
<td>Figure 10</td>
<td>CyclistsInRange description</td>
<td>29</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Operator space file structure</td>
<td>30</td>
</tr>
<tr>
<td>Figure 12</td>
<td>This figure shows the algorithm itself, initiated by the user's context query, along with the invocation of the matched (specialized) operator.</td>
<td>31</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Operator space file structure designed for the purpose of MIDAS project</td>
<td>33</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Context middleware in interaction with applications, sensors, context provider, and context distribution</td>
<td>37</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Application retrieving high-level context information using context synthesis and retrieval of low-level context data from context plug-ins available on the same device</td>
<td>38</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Installation and un-installation of context plug-ins, triggering registration and deregistration of their provided metadata to the network</td>
<td>38</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Context synthesizer retrieves missing context information from context plug-ins available on remote nodes in the network</td>
<td>40</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Remote context information is no longer needed, removing context listener and unsubscribing from context plug-ins</td>
<td>41</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Communication model</td>
<td>44</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Introducing a context space around communicating entities</td>
<td>46</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Communicating entities can employ multiple devices for communication</td>
<td>47</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Communicating entities use their preferred devices in each Context space</td>
<td>47</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Receiver's infrastructure for context-aware communication</td>
<td>49</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Context-addressed messaging mechanism</td>
<td>56</td>
</tr>
<tr>
<td>Figure 25</td>
<td>The context-addressed message created by Bob</td>
<td>57</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Alice's subscription to the traffic information relevant to her current or near future location</td>
<td>57</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Context-based matching in case of context addressed message arrival</td>
<td>60</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Context-based filtering</td>
<td>61</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Sender's infrastructure for context-addressed messaging</td>
<td>62</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Receiver's infrastructure for context-addressed messaging</td>
<td>63</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Application and sensor device configurations with appropriate components</td>
<td>64</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Basic system architecture, with an instance of UMD message delivery illustrated by gray arrows</td>
<td>65</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Alice's and Bob's SIP network infrastructure for context-addressed messaging</td>
<td>77</td>
</tr>
<tr>
<td>Figure 34</td>
<td>SIP operations for context-addressed messaging</td>
<td>78</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Registration of context sensors metadata for context distribution</td>
<td>80</td>
</tr>
<tr>
<td>Figure 36</td>
<td>Synchronous resource location</td>
<td>81</td>
</tr>
<tr>
<td>Figure 37</td>
<td>Asynchronous resource location</td>
<td>82</td>
</tr>
</tbody>
</table>
Figure 38: Event notification of context information ............................................................. 82
Figure 39: Sensor deregistration when sensor plugin gracefully turns off .......................... 83
Figure 40: Sensor deregistration when context distribution UA fails ................................ 83
Figure 41: Hierarchy for storing public and private resource lists ..................................... 85
Figure 42: Context-based and context-triggered communication ....................................... 90
Figure 43: Call processing logic ......................................................................................... 92
Figure 44: SIP Express Router (SER)’s processing of CPL scripts ................................... 93
Figure 45: Context-aware VoIP prototype .......................................................................... 94
Figure 46: Client application .............................................................................................. 95
Figure 47: File chooser dialog ........................................................................................... 95
Figure 48: Comparison of (standard and context-dependent) CPL scripts response times .... 96
Figure 49: Comparison of different types of CPL scripts and their response times ............ 97
Figure 50: Communication initiated by a preference match ............................................... 102
Figure 51: Alice’s and Bob’s SIP infrastructure architecture for context-aware communication initiation ........................................................................................................................... 104
Figure 52: The action of uploading context-dependent preferences to trusted proxy .......... 104
Figure 53: The action of subscribing to context parameters upon which preferences are conditioned .................................................................................................................................. 105
Figure 54: Context provider retrieves information from two sensor devices ....................... 105
Figure 55: The action of activating a new preference in the current context and sending a query to a group of social contacts .............................................................................................................. 106
Figure 56: The action of matching Alice’s and Bob’s context and interests and returning the matching result ........................................................................................................................... 107
Figure 57: The action of initiation and establishment of communication session with Alice and Bob ............................................................................................................................................. 108
Figure 58: Context-aware session adaptation resulting in a call migration and establishment of video session ................................................................................................. 109
Figure 59: Instant messaging session between Alice and Bob ........................................... 110
Figure 60: Context addressed messaging scenario .............................................................. 111
Figure 61: PIM-SM building a multicast tree ...................................................................... 126
Figure 62: PIM-SM – Switching to source-based tree ......................................................... 127
CHAPTER 1

INTRODUCTION

The purpose of this chapter is to give a reader an understanding of a problem domain, the contribution of the thesis, the structure of the thesis, and to provide the summary of published papers and documents. The problem domain is initially explained through some motivating examples illustrating the difficulties of developing context-aware communication services and challenges that our system needed to overcome in order to be implemented. From these examples the key context-aware communication functionalities of our system are introduced, identifying the main system components which will be elaborated through the remainder of this thesis.

1.1 Motivation

Communication has always been an essential part of people's lives. They would meet to exchange goods, ideas, and socialize. Their senses helped them to share an awareness of their environment: people, places, and objects. With the appearance of mobile devices and advances in computing and internet technologies, people now use device-mediated communication to communicate globally. With this increasing globalization, more and more people travel both for business and private purposes. Because of this mobility, they want to use various types of mobile devices in order to be reachable by others and to access information and services irrespective of their current location. At the same time, mobile communications systems are evolving to IP based communications. This has resulted in a transformation of cellular phones into mobile Internet devices. Consequently, mobile devices are increasingly being used for accessing services on the Internet, such as m-commerce, location-based services, video conferencing, instant messaging, etc. These devices are increasingly powerful, with increasing CPU performance, increased memory & storage capacity, and multiple mobile and wireless network interfaces, such as General Packet Radio Service (GPRS), Enhanced Data rates for GSM Evolution (EDGE), 3G, High Speed Packet Access (HSPA), WiFi, Bluetooth, and Near Field Communication (NFC). Additionally, these devices are equipped with various sensors, such as GPS receivers, cameras, motion detectors, temperature sensors, ambient light sensors, etc. At the same time, more and more libraries have been developed to access the device's resources and networking functions, such as OpenNETCF's Smart Device Framework [9], a software library for Microsoft .NET Compact Framework application developers (on Windows Mobile devices); Java Specification Requests (JSRs) that have been defined for Java 2 Micro Edition (J2ME) platform are being implemented and built in as libraries in mobile devices; and SIGAR (System Information and Gatherer And Reporter) library [10] that gathers system information about device resources, networking interfaces, and connection status. The SIGAR library is implemented in C having bindings currently implemented for Java, C#, Python, PHP, and Ruby, supporting all major operating systems (i.e., Linux, Windows, Solaris, Mac OS X, AIX, HP-UX, FreeBSD, OpenBSD, and NetBSD). Consequently, more intelligence exists on the edges of the network (i.e., in the terminal), which is in opposition to the earlier telecommunications model where all of the intelligence was in the network and controlled by the operator. Today, the network is used simply for transport of data. However, not all parties are happy about this trend and there are major efforts to oppose this trend, such as the 3GPP IP Multimedia Subsystem (IMS) [11].

As a result of this trend, device manufacturers focus have shifted their development focus to mobile platforms that provide to application developers open access to a device low-level
and sensor information, thus enabling faster development and deployment of context-aware applications on a mobile device. These context-aware applications use context information and react to it, by adapting their behavior according to changes in context. As an example of a mobile platform developer, Nokia provides Web Runtime technology in their S60 device series that uses the core of the browser engine and provides tight integration with the S60 phone, allowing programs to access device-level data (such as location information, calendar, and contacts). Web widgets built as applications on top of this Web Runtime technology serve as front-ends to Web 2.0 services. These services enable accessing to Web content from the Web and presenting it in a personalized way to the user – thus creating a context-aware Web 2.0 service. Additionally, companies such as Google and Apple have recently entered the mobile device market releasing their own Internet-enabled smartphones. The Google phone and iPhone are built upon their own operating system bases (i.e., Google Android [12] and iPhone OS [13]). Both companies offer platforms for development of mobile applications, that are not only able to access low-level device and sensor information, but also have support for advanced features such as service publishing and discovery, DNS services, networking APIs, GSM/3G/… telephony Application Programming Interfaces (APIs), etc. The Android platform is the first complete, open, and free mobile platform developed by the Open Handset Alliance [14]. This alliance currently consists of 47 technology and mobile operators with a common goal – to build an advanced mobile phone that fosters third party development.

Millions of sensors are currently being deployed in sensor networks around the globe. These sensors are actively collecting an enormous amount of data (in the aggregate). Rapid deployment and significant research results in the field of wireless sensor networks have enabled a device to not only be part of a sensor network, but to act as a gateway between the sensor network and the Internet world, bringing the information from the environment to all kinds of devices (mobile, nomadic, and fixed). This vision leads to the "Internet of things", where all objects in everyday life will be equipped with (radio) tags or another form of communication interface, and will be uniquely identified, interconnected, self-configuring, and auto-organizing based on the context, circumstances, or environments. This phenomenon has brought benefits to society due to these technological developments, but users lost some of the emotional and intuitive aspects of their communication.

However, combining knowledge of the environment with ambient interaction (interaction between heterogeneous computing devices) and taking advantage of mobile platforms and their APIs to access device resources and capabilities (as well as stored user preferences) we can create new techniques to enhance people's communication experience. One way to achieve this is to create applications and systems for communication that are context-aware, meaning that communication with others could be initiated and adapted based upon our current context and our preferences set in this context. For example, sending messages to people addressed based upon their context information, instead of using their network address, delivering only relevant messages to users based on their context, and triggering communication between people based on a match between their preferences and their current context -- represent potential applications of context-aware communication. This can re-enable some of the intuitive aspects of personal communication; and with appropriate sensors might even provide some additional emotional feedback.

1.1.1 Example scenarios of context-aware communication

In this subsection, these context-aware communication services are motivated further by different use cases of the same "biking with friends" scenario illustrating why such services are difficult to implement and what challenges need to be addressed before context-aware communication services become a reality.
Finding friends for biking based upon a match of Alice's and her friends' preferences and their current context

A user Alice is currently available, i.e., she has no current activity or task assigned, and she enjoys biking with her friends in her free time. Therefore she uses her device to indicate in her preferences an interest in biking with her friends during her free time, who are located in the same city during that time. Now Alice needs to choose which of her friends she will call to join her in biking. Preferably, Alice would like to know which of her friends share the same current interest, location area, and/or activity in advance of calling them, because this would save her time to do what she is interested in doing - instead of spending time to call each of her friends and finding out this information herself. Luckily, Alice's device is equipped with sensors, such as GPS receiver, calendar application, and a motion detector, that are able to detect her current location, activity, and task(s). However, in order to provide such a service to Alice, there are many challenges that need to be addressed and provided by the system's point of view:

- Raw context information provided by Alice's device sensors needs to be provided to the system, modeled as context information in order to be unambiguously interpreted by application and system components, and provided to the service as the high-level context (e.g., "free time") in order to be utilized for context-aware communication;
- The system should provide Alice with a means to specify and upload her context-dependent preferences using an application running on her device. These preferences should become active when the specified context occurs. These preferences should trigger sending of a query to Alice's friends and match Alice's location, interest, and activity information with the same context parameters of her friends. Therefore, the system needs to implement a way to send a query to multiple recipients who are members of a group, in order to avoid unnecessary signaling that would be caused by a separate query/response communication with each group member;
- Having in mind to protect the privacy of Alice's friends' sensitive context information, the system should provide a means to Alice to implicitly give her permission to insert some of her private context (i.e., current location) along with her interest (i.e., biking) in a group query that will be sent to her friends. This inserted private context and interest should be matched at each receiver's side against the receiver's current interest and context. The result of this match should be sent back to Alice, offering her a possibility to initiate a call with her friend(s) who had the positive matching response(s);
- Alice and her friends' application and system components should share the same semantics (i.e., context modeling schema) in order to be able to understand each other's context. Additionally, the context modeling schema has to define different levels of information granularity in order to enable Alice (and other users who want to use this service) to reveal her current interests and location to her friends in a desired scope;
- Finally, most of the system components should be deployed on a server infrastructure, (which is preferably a stationary node accessible over Internet), because the functions these components provide cannot run on a mobile device (due to the constraints in the mobile devices' CPU capacity, battery power, etc.). Additionally, there should be a way to establish a trusted relationship between a user's mobile device and this server.
1.1.1.2  

**Initiation of session with Bob and Bob's call processing service that delivers only relevant calls to Bob while he is biking**

Let us assume (as a continuation of the previous scenario) that Bob, one of Alice's friends, is currently biking and is located in the same city as Alice. Therefore, this would yield a positive match by the system. This time when he went biking, Bob took his Bluetooth headset with him to be able to receive calls from his family and friends while biking with his phone in the backpack. Therefore, Bob would like to have an automatic call processing service, which would based upon the presence of this headset, his current context (e.g., activity, location, and task), and the friendship relationship with Alice, make a decision to accept the call from Alice. Otherwise, this call might be redirected to Bob's voicemail. In order to implement and provide such a service to Bob, the system would need to be able to:

- Enable Bob to specify his preferences about whether and/or how to accept the incoming calls in different situations, based upon his social relationship with the caller (e.g., whether Bob and the caller are friends, family, colleagues, or strangers);
- Enhance the call decision making process with context information in order to route an incoming call to the Bob's preferred device based on his current context;
- Detect Bob's current activity, location, and task, the presence of his Bluetooth headset, model this information as context, and provide this context to the call processing service.

1.1.1.3  

**Sending messages to people addressed based upon their context**

In another scenario, let us suppose that the system did not find anyone of Alice's friends with a current interest or activity in biking. However, the system could still notify Alice when someone with this interest appears (as long as Alice's current preference for biking is active).

Let us assume that after some time Alice's friend Ted decides to go biking and wants to send a message to all his nearby friends, who are currently biking in the same city. In order to achieve this, this message needs to be sent to an address specifying the target context of the Ted's friends. We refer to this kind of addresses as context-based addresses, and to the messages whose destination is indicated using context-based addresses as context-addressed messages. Since Alice's context matches the target context specified in the Ted's context-addressed message, this message will be delivered to Alice. Note that Alice prefers to receive messages in the form of text messages on her mobile phone while she is biking. After some time, Alice will reply to Ted that she will join him in biking, and this reply will reach Ted's device.

For realizing this context-addressed messaging functionality, there are many challenges that need to be overcome:

- The system needs to provide Ted an easy way to compose context-addressed messages;
- The system has to derive which destination should be the target of this message based on the context of the receiver as specified in the context-based address – while considering the context of all of the potential recipients;
- The system needs to determine whether this message is relevant to Alice in her current context and/or how to deliver this message to her, while protecting Alice's privacy. This process also has to include adaptation of the message to Alice's preferred format and delivery of this message to the Alice's preferred device in her current context.
- The system has to enable Alice to reply to the received context-addressed message.
1.1.1.4 Adapting Alice's current session with Bob when her context changes

If suddenly some of Alice's context changes (while she is in a call with Bob), such as: a change of Alice's location and activity from biking to working (from home or in the office) and higher bandwidth becomes available – her preferred device changes from a mobile phone to the desktop computer and her preferred communication means switches from audio to video calls. Therefore, a system needs to prompt Alice suggesting her to switch to a desktop device and start a video session. If Alice accepts this suggestion, the system should be able to migrate her call to a desktop device, establish a video session with Bob's device, and terminate the session with the mobile phone. Alternatively, Alice could prefer to switch to a messaging mode instead of switching to a video call after changing the context. In this case the system would need to establish a message stream instead of a video session and transmit a series of instant messages.

1.2 Problem statement

The problem to be addressed in this licentiate thesis is how to address (for the purposes of communication) people based on their context, rather than simply based upon their network address. The exchange of information (including different media types, such as images, audio, and video) is supported in different ways on various devices. Additionally, end users have their own preferences regarding how they would like to receive information from different parties (regarding both communication means and device; and policies regarding if they wish to communicate with the specific party). However, these preferences can change with time and the situation the user is in. These preferences are also dependent on the user's relationship (here in role of callee) with the caller. Thus, depending on whether the user and the caller are, for example, friends, colleagues, or family members, the user could have different requirements/wishes on how to receive a call/communicate with the particular caller. Moreover, these preferences have to be expressed in advance (before the call) – in order for the system to take these preferences into consideration before delivering the call (or other communication) to the callee.

The challenge is to overcome these difficulties and to create and build a model that has the capabilities to:

- address a message to recipients based on their context rather than their network address;
- route such a context-addressed message from the sender to the correct recipient(s) and to facilitate delivery of this message using the user's preferred communication means and device in the user's current context;
- initiate communication among users based on matching of their preferences and current context, taking the relation between caller and callee into account (i.e., context-based session initiation);
- enhance the session initiation decision making process with context information in order to route an incoming call to the callee's preferred device based on his/her current context (i.e., context-aware call signaling);
- adapt, modify, and manage user's communication sessions according to contextual parameters (i.e., context-aware session management); and
- enable a user to modify their preferences at any time during a communication session.

Different types of users have different preferences regarding the type of the communication and content they are interested to receive. These preferences may vary with time and the situation of the user. The user's situation can be described by the user's location, activity, or other context parameter(s). Therefore, the user's interest in a specific type of the content or communication could be triggered by a change in the user's current context. An example of a
user's interests in communication includes finding people (from a user's list of contacts that have the same relationship with the user, such as friends, family, colleagues) with the same interest or current context as the user and initiating a communication session with them. If during a session some of the context suddenly changes (e.g., a significant decrease in bandwidth or a match of the user's interests), new preferences (regarding device and communication means) will trigger a specific action (session initiation, adaptation, or termination). We unify the proposed modes of utilizing context information to manage the receiver's session as context-aware session control. Similarly, a change in the receiver's context (e.g., change of location from "office" to "home") could change new preferences regarding the content that he/she is interested to receive, triggering an action such as subscribing to a different type of topic (e.g., "sports" instead of "stocks").

Implementing the context-aware session control in this model includes discovering, selecting, and switching to an optimal communication means to meet varying user, contextual, communication, and device resource requirements and preferences. This assumes an understanding of the user's current context and their preferred communication means and device (i.e., that this user would like to use in this context). It is also important to understand the performance and cost of different communication means, as well as device capabilities (including which communication means are supported and what is the current state of each device's resources). Switching to another communication delivery technique needs to be done programmatically, by activating an appropriate application, which will establish a new session with the user or restore an ongoing session; and in the case of messaging – to complete the transfer of a message.

The solution to the problem of how to provide context-addressed messaging and communication to users, must consider how, when, and where to acquire raw data from sensors, model this data as context information to be unambiguously interpreted by applications and system components, process this information into high-level context, and exploit this context knowledge to enable context-addressed communication. The solution builds upon a distributed context model, while avoiding distributing context information to others in order to preserve each user's privacy. Modeling the relationship between caller and callee as part of the context knowledge will avoid the need to explicitly specify membership of each potential caller into a specific social group ("family", "friends", "colleagues", etc.).

This thesis (1) extends the earlier work of Theo Kanter [15] to enable context-aware communication adaptation during a session (as well as at the start of a session or for a message) and (2) because this adaptation depends upon collecting and utilizing context information for a receiver – enabling a new form of context-based addressing by allowing the sender to use a target context to specify the address of a message or session initiation request.

1.3 Thesis overview

This thesis presents a context-addressed communication system that consists of three functional blocks, as illustrated in Figure 1. First part is Context management that provides access to context information to any context-aware application. Second part is Context-addressed messaging and communication, which is built on top of and uses information provided by Context management to address and route messages/calls to the relevant recipient(s). Context-addressed messaging and communication also facilitates the delivery of messages/calls using the receiver's preferred communication means and preferred device in their current context. Context-aware session control is the third part of our system, which uses context information from Context management in order to initiate, adapt, and terminate user's session based on their current context.
The thesis is organized in six interrelated chapters.

Chapter 1 is the introduction. This chapter states the problem addressed and solved in the thesis. It also presents the thesis overview and outlines the contributions of the thesis.

Chapter 2 gives a definition of context, describes the background information about the context-aware systems, analyzes the state-of-the-art context management activities, presents our approach to the context synthesis using context operators, and finally proposes an architecture for context-aware systems that will be used throughout the thesis.

Chapter 3 gives an introduction into context-aware communication, analyzes the types of application-level communication, elaborates about the requirements for sender, receiver, and network infrastructure for context-aware communication, and introduces concepts of context-addressed messaging and communication.

Chapter 4 outlines the requirements for context-addressed messaging, analyzes the relevant related work performed in this area, and describes a design of context-addressed messaging infrastructure. In this chapter we propose our own format for composing context addresses, describe context-addressed messaging mechanisms & operations and their implementation based on the SIP network infrastructure, and finally present the idea of SIP based multicast that enables context distribution, group management, and group queries. SIP based multicast is used in our system for two purposes: (1) to group sensors providing the same context type in order to be able to provide event notification service about the context change and the sensors membership in the group, as well as (2) to group user's contacts that have the same social relationship with the user to send queries to the members of this group (containing information about the user's interest) in order to find the members whose interest or context matches the user's interest and initiate communication with them.

Chapter 5 describes context-aware session control and how it can be implemented using two types of constructs: context-switch and context trigger. A context-switch selects an action from the set of context-dependent actions upon an incoming communication event based on the receiver's current context, whereas a context trigger initiates an action based on a context update and preferences that have previously been expressed about this updated context. This chapter illustrates, using several examples, the initiation of a communication session based on the match of user's preferences and their current context, as well as adaptation and (if necessary) termination of the communication session based upon the user's context-dependent preferences.

Chapter 6 gives a summary of the thesis results, presents some conclusions, discusses the open issues, and suggests some future work.
1.4 Contributions
The major contributions of this thesis include:

- Context synthesis using context operators
- Introducing SIP based multicast for context distribution, group management, and group queries
- Introduction of context-addressed messaging and communication, including creation of a format for composing context addresses and the concept of inner-routing of this message to the correct recipient(s) (using context-based filtering in the receiver's infrastructure)
- Introducing context-aware call signaling based on the design of a context switch
- Introducing context-based session initiation triggered by the match of a user's current interest against the interest or the current context of other users
- Introducing context-aware session management based on a design using context triggers

1.5 Summary of Published papers and other documents
I am the main author of the following papers and publications. I have noted my specific contributions to each of them.


Note: The paper [1] presents an approach based upon exploiting context information to enhance the power of existing SIP call control services. These services are implemented using Call Processing Language (CPL), a language used to describe and control Internet telephony services. I extended CPL with context parameters to permit context-based decision making based on a context ontology. In these extensions, I defined a context-switch to support the services whose decisions are based on context information of an end user. I implemented a context-aware VoIP prototype, consisting of: (1) context extensions to the CPL-C module of an existing VoIP platform, called the SIP Express Router (SER); (2) a client application that enables a user to upload ontologies and CPL scripts to SER's database; and (3) a matching module that parses the uploaded ontology to extract the context values in order to determine the appropriate CPL script, then uploads this script via the SIP protocol to the SER. The goal of this work was to show how easy it is to add new context parameters to CPL and how complex decision criteria can be built using my solution.


Note: In the paper [2] I designed a model for retrieving and distributing context information in a mobile distributed environment. I proposed an approach to retrieve context information directly from its source only when it is needed, rather than simply when new value is available. The retrieved value is then cached in a database until its validity expires. To enable simple application requests for context information, while hiding from them the underlying transformation process, we have utilized context queries and context triggers. Context queries are used for stateless retrieval of context information, e.g. "What is the temperature of this room". Context triggers are queries for stateful context information, these trigger a predefined action when context information reaches a specified state, e.g. "Alert me when the temperature of this room reaches 28 degrees Centigrade". Context queries can be simple (i.e.,
requiring only a database query for the specific context information) or complex (i.e., requiring context synthesis).

Moreover, in this paper I introduced a novel approach to perform context synthesis using context operators. These operators serve two purposes. First, an operator provides a functional approach to context data simplifying context synthesis and programming of context-aware systems in general. Second, the context engine applies operators dynamically based on description of input and output types. Operators can invoke other simpler operators within their function, which is specified in their description using the operators' ontology schema. Operators can have different implementations of their functions, which can be added or removed at any time during context middleware runtime without changing the middleware source code. This results in system flexibility, extensibility, and enhances code reuse.

The co-author of this paper participated with me in the design of distributed context distribution architecture and wrote the conclusion of the paper.


Note: In the paper [3] I implemented and evaluated an approach for context synthesis using context operators that was described and designed in [2]. This paper illustrates the main advantages of context operators, which are: the reusability, extensibility, and interoperability, facilitated by ontology-based context modeling. For this purpose, a dedicated Lightweight Ontology library for representing and manipulating ontologies on mobile devices was created by Warsaw University of Technology. Additionally, this approach was used by Capgemini to develop a set of sport applications. These were demonstrated at a live sport race (in Super Prestige Cyclocross in Gieten, Netherlands) in order to provide a near real time virtual ranking service.

The evaluation of this operator-based context synthesis was performed in terms of response time to context query sent by the application. In this evaluation I showed that it is possible to perform context synthesis operation in near real time (i.e., with the average latency of 2 seconds) on the mobile device. Note that these 2 seconds of delay are not suitable for applications that require to context synthesized from very volatile information whose value changes more frequently than once in two seconds or for mission critical applications that need to have reliable information (e.g., if some person's life is in dangerous). However, in our case of a live sport race, where the position of cyclists in a group was presented to the spectators every 4 seconds, the spectators have reported that this delay did not affect their real-time experience.

The other two co-authors of this paper described the context modeling approach and the implementation & demonstration of the racing applications.


Note: In the paper [4] I introduced and evaluated context distribution methods in mobile systems environments using Bluetooth and WLAN technologies. The context distribution methods that I described in this paper are based on a simple idea: each device discovers other nearby devices, collects context information from these discovered devices, and distributes this information to all the discovered devices, such that they all share the same (most recent)
context information. As both technologies enable ad-hoc discovery & networking between heterogeneous devices, we evaluated the use of these technologies for context distribution within a local area (in this paper we considered a single hop). However, it is important to note that it is the distribution of the aggregated information which enables the discovery of devices and their context information beyond the single hop limit.

The evaluation of the proposed methods was performed by collecting and comparing battery power consumption measurements on two different handheld devices: HP iPAQ 4150 and 6915. The goal of this evaluation was twofold: (1) to determine whether it is more energy efficient to distribute context knowledge to other devices in advance of their arriving at a new location or having each device discovers this information itself; and (2) to compare energy consumption of Bluetooth and WLAN in context discovery and distribution operations. I obtained the following results: a) Bluetooth consumes 2-6 times more energy to send a file of 1MB size (containing aggregated context information) to two devices than to discover them – hence distributing this information via Bluetooth is more expensive than directly learning it (through the discovery procedure); b) if data is sent to more than three devices at once via WLAN multicast this is more energy efficient than using Bluetooth; c) it is more energy efficient to distribute (once discovered) context information to other devices in advance, rather than having all devices learn this information themselves.

The other co-authors of this paper provided a feedback to my design of context distribution methods, performed evaluation of context distribution methods on an HP iPAQ 6915 device, and wrote a section about results of this evaluation.

Based on these results, I decided to design and implement SIP based multicast for context distribution in my context-addressed communication dispatch system.


Note: This paper [5] is a magazine version of the previous paper.


Note: In the paper [6] I proposed implementing SIP based multicast for context distribution using resource lists to group sensors providing the same context information type. Therefore, SIP presence with extensions of SIP SUBSCRIBE message could be sent to members of a resource list and modified XCAP operations allow our customized authorization mechanisms for adding and removing sensor entries to and from a resource list. My colleague and I designed and implemented this context distribution service together in the scope of EU IST MUSIC project [23].


Note: In the paper [7] I proposed to infer a user's social relationships from his/her daily communication logs with others and to use these social relationships as a means to create user specific policies for granting access to a user's context information. This enables a user to specify different levels of access to his or her context information based on the relationship with another user (e.g., whether this other user is a friend, family member, colleague, or unknown). The decision about whether to grant access to context to the requesting entity, and at what granularity, is made based on the social relationship that this entity has with the context owner and the owner's current situation. However, these policy rules might depend
upon the situation the user might be in. Therefore, in our policy design I introduced context conditions to allow a user to define different rules (i.e., allowing distribution of a particular context to a specific scope or to deny distribution) based on his/her current situation. These context policies are utilized by multiple proxy servers which process queries for context information, thus improving scalability and avoiding single point of failure.

The proposed approach was evaluated on data obtained from three users monitored for a week, resulting in classification success rates of 87% for user1, 85% for user2, and 69% for user3, despite simple rules and very limited log-data. Note that there were not enough users to obtain statistically significant result. However, although the classification success of roughly 80% for all three users, the result is still promising, as it was achieved by applying very simple rules that reflect the common understanding of the communication characteristics of these categories of communication partners. We concluded that as the user's device performs the logging, the classification accuracy for those people with whom the user regularly has contact should quickly be correct, while only new contacts will be inaccurate.

The other co-authors of this paper implemented a Personal Information Management (PIM) sensor which acquires data about the user's communication activities and performed the logging of the communication data on three different users during a week; designed and evaluated an inference mechanism of user's social relationships using rule-based data mining approach, Bayesian network inference, and user feedback; and assisted in design of context access policies.

[8] A. Devlic, "Context-addressed communication dispatch", a poster presented at Wireless@KTH, as part of the KTH Research Assessment Exercise, 26 June 2008.

Note: In this poster [8] I illustrated how different parts of my work fit together and solve different problems of the system for context-addressed communication dispatch, which is a subject of this licentiate thesis.
CHAPTER 2

CONTEXT MANAGEMENT

This chapter gives an introduction into an area of context-aware systems, reviews previous definitions of context, and gives our own one. It also provides an overview of the context management activities while reviewing some of the important ongoing and previous research work in this area. Based on these reviews, we leverage some of the existing context management techniques, such as ontologies used for context modeling, and a context sensing mechanism that uses context plug-ins. Additionally, we describe our own approach for context synthesis using context operators that is used by applications when querying for high-level context information. Finally, we elaborate our design of context-aware system architecture that will be used by our high-level components for context-aware communication: context-addressed messaging & communication and context-aware session control.

2.1 Definition of context

There have been a number of attempts by researchers to define context and what information it should include. We review some of previous definitions of context, then give our own definition – that will be used for the rest of the thesis. Schilit and Theimer were first to define context and context-awareness (in their 1994's paper [16]) as the ability of applications to discover and react to changes in the environment (i.e., location, identity of people, nearby devices, and objects) they are situated in. The first and most frequently used type of context information is location, but over time the list of context attributes has grown to include: time, identity of people and objects in the user's environment, orientation, user's emotional state, activity, etc.

Two months later, Schilit, Adams, and Want stated that important aspects of context answer the following questions: where are you, whom you are with, and what resources are nearby [17]. They define context as constantly changing environment of people, places, and devices. Therefore they put an emphasis on context entities, but do not elaborate on their attributes which comprise the context.

In 2000, Chen and Kotz [18] have extended this version of context to include: computing context (network connectivity, bandwidth, and nearby resources such as printers, displays, or workstations), user context (the user's profile, location, nearby people, and current social information), physical context (lightning, noise level, traffic conditions, temperature), and extended it with the time context: time of the day, week, month, and seasons of the year. This classification simply groups several different context parameters, but does not associate these parameters with a context entity. However, this association is necessary to enable easy querying of a particular entity's information. This is illustrated in Figure 2 and elaborated in the associated text explaining this figure.

In 1998, Pascoe [19] defined context as the subset of physical and conceptual states of interest to a particular entity. For Pascoe context is a subjective concept for an entity that perceives it. He associates context entities with artifacts which have a name, type, and a set of contextual states (i.e., context parameters). Therefore, he did not consider the same context could be shared among and used by a set of applications or by different entities. In the same year, similarly to Pascoe, Dey [20] initially defined context as any information about the user and the environment that can be used to enhance the user’s experiences. This included data such as: the user’s physical, social, emotional, or informational state.
In 2000, Dey and Abowd [21] extended Schilit, Adams, and Want's classification of context and defined context as:

*Context is any information that can be used to characterize the situation of an entity. An entity can be a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves.* [21]

The important contribution of this definition is that the context information characterizes the situation from the perspective of an entity and contains attributes that are application domain-specific. Note that we are only concerned in this thesis with context that is relevant to the interaction with an entity. Thus, in this thesis we define context as:

*Context is any information that characterizes the situation of an entity, where an entity can be a user, his/her currently used device, the network this device is connected to and the status of this connection, physical locations in the user's surrounding, or nearby objects. Context consists of a number of attributes that can be used by an application to adapt its behavior in order to assist a user in his/her daily tasks. An assumption is that a suitable application exists and that the context attributes and their values are known to this application.*

Therefore, we distinguish between five types of entities which can be characterized as owners of context information: a person, a device, a network (interface or connection), a place, and an object. However, these entities are not independent of each other, having the following relations (illustrated in Figure 2): a person uses a certain device(s); this device is connected to a network; a person, device, and an object are located at a certain place; and a person and a device are somehow related to some other object(s). All entities are subclasses of the root class “Thing”, from which all other terms are derived. Thus, we assign all context information to a certain entity and we can query about information possessed by an entity, i.e. user context, device context, network context, place context, and object context.

![Figure 2: Context entities and their relations](image)

In this thesis we will focus on the context information that is relevant to initiating, routing, and adapting communication to a user. In the following section the fundamental concepts of a context-aware system are presented along with the approach we have used to model context information.
2.2 Context-aware system

We define context-awareness as:

A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the current user's task and/or state.

A system is context-aware if it can locate, extract, interpret, and use context information and adapt its functionality to the current context of use. There are two types of context-awareness:

Active context awareness an application automatically adapts to discovered context, by changing its behavior.

Passive context awareness an application presents the new or updated context to an interested user or a system component, or makes the context persistent to be retrieved later by this user or the system component.

A context-aware application can adapt itself based upon the context information, thus needing less explicit input from the user. Context-aware applications that can make the appropriate decisions based on the (context) information available to them, may cause the user to perceive the application as providing better results and enabling this user to be more productive (as the user spends less time interacting with the application for control – providing more time to do the actual task the user is interested in doing).

Context information should be collected from the environment (i.e., sensors) through some automated means. It should be modelled to hide low-level sensing details from the applications. This technique facilitates extensibility since the applications do not have to be modified and reusability of hardware-dependent sensor code is increased due to encapsulation. Both of these advantages ease application development. Application developers choose what information is relevant, then they must decide how to use this information within the application.

Figure 3: Context-aware system consisting of the following entities: context sources, context space, and context consumers

In a global sense, a context-aware system (shown in Figure 3) consists of: context sources, context space, and context consumers. Context sources provide context information to the context space. Context consumers use information that is stored in the context space to
accomplish different tasks. Context space is an abstraction of a distributed repository of context information which stores context information produced by context sources in order to allow queries by authorized context consumers.

The following questions arise from this concept: how to best realize context space, context sources, and context consumers in our system? How to timely discover context sources and a type of context information they provide, assuming that context consumers are mobile? How to enable these context consumers to retrieve the needed context information from the discovered context sources? Should these context consumers further distribute this obtained context knowledge to other potentially interested consumers in advance of their arrival at the same location, and (if so) how? Sections 2.3.1 and 2.3.3 give answers to the first question. Section 2.2.1 answers the rest of the questions, describing how we addressed and solved these problems, as well as what results we obtained.

2.2.1 Context discovery vs. context distribution

In ubiquitous computing environments there are a lot of heterogeneous devices that can act as context sources and some of these context sources can generate a huge amount of context data. Additionally, the set of all context sources can generate even larger amount of context data. At the same time users (i.e., context consumers) are frequently mobile and geographically distant. The challenge is to timely discover, collect, and adapt to new context data in an efficient and scalable manner. Context discovery is important because mobile devices can move in and out of communication ranges of other devices and sensors which provide context data. Alternatively, a device can share its context knowledge (which it has discovered, acquired, and modeled) with other geographically distant devices (which have done the same) in order to learn about potential new contexts, in advance of arriving at a new location. Knowing context in advance of arriving to a new location is powerful, because it can potentially reduce the delay or energy required by a device whose application(s) need to adapt to a new environment. If this context information is distributed in advance, then potentially a greater fraction of the context queries can be answered locally. However, there is a trade-off between how far the context information should propagate and how useful this information is in advance (for adaptation by the applications running on a device). Hence, the more context parameters from the environment we distribute the greater probability that some of this information will be used; but potentially a lower and lower fraction of this information is actually useful. In order to understand this trade-off between the distribution of context data over a set of devices and the costs of this distribution versus its time-dependent value – we examined the battery power consumed by context discovery vs. context distribution performed by Bluetooth and WLAN (within a local area) [4][5].

Using Bluetooth, we discovered devices using Bluetooth’s discovery protocol, collected their context information, created an XML file containing this information, and distributed this file to all discovered devices, such that every device obtained the same context information. As context information we were specifically interested in the list of services provided by a device. By propagating the complete list of all the discovered services, we can quickly generate a list of all services that all devices which are currently or soon could be in range have available. Next we performed the same discovery, collect, and distribute functions, but using WLAN. Note that in our WLAN context distribution method, each device has a timer with a random timeout value that needs to be long enough to allow a server device (in a role of context discoverer) to receive files from other client devices, but short enough to keep all devices synchronized. When the timeout occurs, the device will check if it has received a discovery message, thus becoming a client, and sending reply messages and a file containing context information to the server; otherwise it will act as a server, multicasting the discovery message itself and starting another timer in order to wait for clients’ reply
messages and files, thus acting as a server. Therefore, there is no risk for collisions. Additionally, all the files received by the clients are aggregated by the server and sent over multicast again to the clients, such that all clients share the same information.

In each case (i.e., Bluetooth and WLAN context discovery and distribution) we have performed the cycle of operations starting with a fully charged battery and continuing until the device was not able to utilize the selected wireless interface any longer. Finally we compared both approaches to context distribution in terms of battery power consumption.

Based upon the power consumed for the file transmission and the file transfer data rate we estimated how many joules are consumed per transferred user data bit. This is obtained from the following equation:

\[
\text{Energy per transferred user data bit}[J_{\text{bit}}] = \frac{P_{\text{FT}}[W]}{\text{file transfer data rate}[\text{bit/s}]}
\]

\[= \frac{\sum_{i=1}^{n}(U_{\text{FT}_i} \cdot I_{\text{FT}_i} - U_{\text{OFF FT}_i} \cdot I_{\text{OFF FT}_i})}{\sum_{i=1}^{n}\text{file size}_i}}
\]

(1)

where \(U_{\text{FT}_i}\) and \(I_{\text{FT}_i}\) represent the battery voltage and current values obtained during a single file transfer to a device, \(U_{\text{OFF FT}_i}\) and \(I_{\text{OFF FT}_i}\) represent the battery voltage and current values obtained from the device when the Bluetooth interface was turned off and when the device was idle, and \(T_{\text{FT}_i}\) denotes the duration of a single file transfer. Note that in case of HP iPAQ 4150 we obtained 3.9J/MB (i.e., 481.7nJ/bit) as the cost of Bluetooth distribution and 1.56μJ/bit – meaning that 3.2 times more energy was consumed per bit by sending data over WLAN than over Bluetooth. From the file transfer rate we calculated that WLAN is 1.8 times faster than the Bluetooth (i.e., 2*16s in Bluetooth and 18s in WLAN to transfer a 500kB file to two devices). Because we can send data over multicast to multiple users at once, this result tells us that distributing data over WLAN is more power efficient method than using Bluetooth when the number of recipients exceeds three.

We also obtained the values of the energy cost of Bluetooth’s device discovery and service discovery to be 1.18J and 0.16J. Comparing the cost of device discovery (i.e., 1.18J) with the cost to transfer a 1 MB file (i.e., 3.9J) we can observe that the device consumes three times less energy to discover two devices than to transfer a 1 MB file to a single device. This is an important result, showing that Bluetooth file transfer is not an energy efficient method to transfer data (as compared to WLAN). However, it is well suited for discovery of nearby devices.

We showed by now that context discovery should be done by Bluetooth and context distribution using the WLAN multicast. In order to answer the question if it is better to perform context discovery or context distribution, we computed how many joules are consumed by a client to receive a single user bit over the WLAN multicast:

\[
\text{Energy per received user data bit}[J_{\text{bit}}] = \frac{E_{\text{RMF}}[J]}{\text{received file size}[\text{bit}]}
\]

\[= \frac{\sum_{i=1}^{n}(U_{\text{RMF}_i} \cdot I_{\text{RMF}_i} - U_{\text{OFF RMF}_i} \cdot I_{\text{OFF RMF}_i}) \cdot T_{\text{RMF}_i}}{\sum_{i=1}^{n}\text{received file size}_i}\]

(2)

Note that \(E_{\text{RMF}}\) in the equation (2) is the total energy consumed by a device to receive a merged file. Comparing this result of 1.33μJ/bit with the average energy consumed by Bluetooth to discover two devices along with their services (i.e., 1.5J), we can observe that a device would spend significantly less energy to discover 2 other devices and their services (approx. 2.7kB of data) then to receive a file of the same size over WLAN multicast (i.e., 3.9J).
28.7mJ). Note that to consume 1.5J, a device could receive the file of 140kB over WLAN multicast. Moreover, the energy to discover context would increase with the number of nearby devices. Therefore, it is more energy efficient to distribute (once discovered) context knowledge to other devices in advance, rather than having all devices learn this information themselves. Moreover, multicast should be used for distribution of (discovered) context to interested context consumers.

2.3 Context management

A context-aware system consisting of sensors and middleware supporting (context-aware) applications on top of it, as depicted in Figure 4. The middleware creates a knowledge environment responsible for discovering new sources of context information, aggregating information from different sensors, composing existing knowledge into new concepts (i.e. synthesis), and storing context information. This middleware also provides communication and dissemination of context to the applications. Note that applications can themselves provide context information and hence they can also act as a context source while consuming context information from the knowledge environment.

Context management encompasses all of the activities starting with sensing the context, context modeling, context synthesis, and ending with context distribution and querying. We envision that context management takes place in a distributed fashion, but have not yet studied how this occurs or how it can be controlled.

![Figure 4: Context-aware system](image)

2.3.1 Context sensing

Context sensing is the process of collecting context information from the environment through some automated means (i.e., via sensors). Sensors are hardware or software entities that provide raw data from the device or the environment to the system. Sensors are usually objects of everyday use that are equipped with some form of computational capacity and have simple sensing and communication facilities. Some of the approaches to sensing context include: sensing the location, time, people, and nearby objects; orientation; network bandwidth; and other low-level types of physical context such as: light level, vibration,
proximity of humans, sound, temperature, pressure, and the concentration of gases (such as carbon monoxide, carbon dioxide, oxygen, etc.).

We adopt here the approach of context plug-ins [22] designed in scope of MUSIC (Self-adapting Applications for Mobile Users In Ubiquitous Computing Environments) project [23], which act as wrappers around hardware and software sensors and are able to sense the raw context information by delegating the events to the underlying machinery or attaching a thread to the source code. These wrappers are plug-in components to the context-aware middleware. They provide the mechanism to activate or deactivate context sensing. These context plug-ins are identified by the middleware using metadata (i.e., the type of context information, the entity this context information belongs to, and other quality of context parameters their sensors provide, such as freshness, accuracy, resource consumption, etc.). The middleware installs context plug-ins by detecting their identities and registering their provided metadata locally on the device it is running and on a network infrastructure server, so that the information about which sensors provide a particular context information can be found (first on the device itself, otherwise in a distributed system) when a context query arrives (as shown in Figure 5). Thus, an application can obtain the desired context information by sending a context query to context middleware, which will resolve the query, activate appropriate context plug-ins to retrieve the raw context data from sensors, aggregate and optionally process (i.e., filter) the obtained data, and return the result to the application.

Figure 5: Context plug-ins – installation of context plug-ins into context middleware and registration of metadata provided by these plug-ins to a network infrastructure server (on left) and activation of context plug-ins upon arrival of context query (on right)

Note that in this section we assume that the context plug-ins reside on the same device as the context middleware and an application. Section 4.6.2 illustrates how an application could retrieve context information from the remote context plug-ins (i.e., that are available on other devices in the network), when the desired context cannot be provided by the local context plug-ins (i.e., available on the device).
2.3.2 Context modeling

Context modeling is a technique used to represent and model context information. As the model substantially reduces the amount of context data, the model can be used to exchange context information within a context-aware system, as well as between different systems. In the latter case applications of one system can be notified of context changes that were sensed by another system.

Context modeling techniques are classified by the scheme of data structures they use to exchange contextual information in the system. The existing techniques were described and evaluated by Strang and Linnhoff-Popien in [24], according to requirements of ubiquitous computing systems for context modeling. Thus techniques can be classified in terms of the following:

- **Distributed composition**: a ubiquitous computing system is derived from a distributed computing system; therefore a context model should be composed and administered in a distributed manner, being able to cope with changes in time, network topologies, and source.

- **Partial validation**: a context model allow for partial validation of contextual knowledge independently of contextual interrelationships, which can make any modeling error-prone.

- **Richness and quality of information**: as the quality and the richness of information gathered from sensors vary over time, a context model should inherently support quality and richness indication.

- **Incompleteness and ambiguity**: the set of contextual information gathered from sensors is usually incomplete and/or ambiguous. This should be covered by the model, for instance by interpolation of incomplete data on the instance level.

- **Level of formality**: it is desirable to describe contextual facts and interrelationships in a precise and traceable manner, in order to share the same interpretation of the data exchanged and the meaning "behind" it (so called shared understanding).

- **Applicability to existing environments**: it is important that a context model is applicable within the existing infrastructures of ubiquitous computing environments e.g., a service framework such as Web services.

The following context modeling techniques are examined in more detail: **key-value pairs**, **markup scheme models**, **graphical models**, **object-oriented models**, **logic based models**, and **ontology based models**.

2.3.2.1 Key value pairs

Key-value pairs are the simplest data structure for modeling context, proposed by Schilit et al. [25]. A key describes the context parameter and a value contains the value assigned to this context parameter (e.g., activity=running). When context information is described with a list of simple attributes in a key-value manner, the employed context management framework operates based upon applying an exact matching algorithm on these attributes. Therefore, key-value pairs lack a capability for sophisticated structuring of data which is used in efficient context retrieval algorithms.

2.3.2.2 Markup scheme models

Markup scheme models are represented by hierarchical data structures consisting of markup tags with elements and attributes, such as XML. Typical representatives of this approach are profiles that are usually serialized using Standard Generic Markup Language.
extensions of Composite Capabilities/Preferences Profile (CC/PP) [27], and UserAgent Profile (UAProf) [28]. SGML is a meta-language in which one can define markup languages for documents. CC/PP and UAProf are standards developed by the World Wide Web Consortium (W3C) and Open Mobile Alliance (OMA) respectively, aiming at providing a structured format to describe device capabilities and user preferences for the purpose of adapting contents to a device. CC/PP is a general framework that defines the structure of a vocabulary (that describes device capabilities and user preferences). UAProf is a specific vocabulary based on CC/PP.

Markup scheme models are very popular technique to model context, due to attractive characteristics of XML (it is readable to both human and machine, flexible enough to describe any data structure, has a strict syntax, and is widely deployed on various tools, platforms, and applications). However, in order to exchange information, applications need to "understand" the information written inside XML tags. Although XML provides a suitable means for formatting information, it does not add semantics (i.e., the meaning) to the information, which is a crucial requirement for sharing and use of context information among applications in different domains and context-aware systems.

2.3.2.3 Graphical models

Graphical models have the graphics oriented context model. The well known representative is the Unified Modeling Language (UML) with UML diagrams as a graphical component. Because the UML structure is generic enough, UML can be used for modeling context. This approach is described in [29], where contextual aspects are modeled as UML extensions.

Another example of graphical models is a contextual extension of Object-Role Modeling (ORM) approach [30] (shown in Figure 6). ORM simplifies the design process by using natural language as well as intuitive diagrams which can be populated with examples, and by examining the information in terms of simple or elementary facts. By expressing the model in terms of natural concepts, like objects and roles that entity types play, it provides a conceptual approach to modeling. The examples of fact types are: is of type, permitted to use, located at, engaged in, etc. The ORM entity types can be: device (id), device type (code), person (name), location (name), activity (name), etc. Fact types are categorized according to persistence and source as static (facts remain unchanged as long as the entities they describe persist) and as dynamic. Dynamic facts are further distinguished depending on the source of the facts as either profiled, sensed, or derived types. The time aspect of the context is also covered in the historic fact type. The special type of relationship between facts is a fact dependency, the dependsOn relation, where a change in one fact leads automatically to a change in another fact.

The advantage of graphical models is that they can be easily transformed into entity-relationship (ER) models that are used in (conceptual) design of relational database to describe what data should be in the database (i.e., entities) and what relationships are between these data items (i.e., associations or interactions). Graphical models are mostly used for human structuring purposes, because in order to be manipulated by the context management framework, they need to be transformed into object-oriented models. Note that it is possible to derive some source code from the graphical context models, but an implementation effort needs to be made to complete desired code functionality.
2.3.2.4 Object-oriented models

Object-oriented models exploit the main benefits of object oriented approach, which are encapsulation and reusability, for context modeling. Details about context processing are encapsulated on an object level. Access to contextual information is provided only through specified interfaces.

Examples of object-oriented models are cues [31] from ESPRIT project TEA (Technology for Enabling Awareness) [32] and Active Object Model from the GUIDE project [33]. Cues (shown in Figure 7) are abstractions from physical and logical sensors. Physical sensors are electronic hardware components that measure physical parameters in the environment. All information gathered from sensor-based mobile devices (i.e. PDAs, mobile phones, wearable computers) are considered as logical sensors. Each sensor is regarded as a time dependent function that returns a scalar, a vector, or a symbolic value. A set (finite or infinite) of possible values for each sensor is defined. Each cue is based on a single sensor, but different cues can be based on the same sensor.
information included various pictures of the castle and nearby cafe, including a summary of the castle, a description of the castle's architecture, and a menu offered by the nearby cafe. This approach has been driven by the requirement of being able to manage a great deal of personal and environmental contextual information, while maintaining scalability. All the details of data collection and fusion have been encapsulated in active objects and thus hidden to other system components.

Although the object-oriented models offer efficient means for adding new types of classes and objects through well defined interfaces, the invisibility of object contents as a consequence of encapsulation is a drawback to the requirement for formality.

2.3.2.5 Logic Based models

A logic based context model is represented by a formal system defined by facts, expressions, and rules. Context is added to the system as a set of facts, under which a set of rules are applied to infer a new set of facts or expressions (this process is known as reasoning or inference).

One of the first concepts came from McCarty's research group at Stanford University [34] that introduced formalization means to describe contexts in which situations change. They proposed the use of simple axioms with the added lifted rules which relate the truth in one concept to the truth in another concept, as a part of the model itself. The basic relation in the concept is: \( \text{ist}(c,p) \), which asserts that the proposition \( p \) is true in the context \( c \).

Giunchiglia's approach [35] deals more with context reasoning as it considers a context to be a specific subset of the complete state of an individual entity to reason about a particular goal (part of the world which encodes an individual's subjective perspective about it).

Akman and Surav [36] have extended the situation theory proposed by Barwise and Parry [37] who have tried to cover the model-theoretic semantics of natural language in a formal logic system. Akman and Surav extended this system to model the context with situation types which are ordinary situations and thus first-class objects of situation theory. The world in situation theory is viewed as a collection of objects, properties, and relations. Infons (‘unit’ facts) are discrete items of information and situations are first-class objects which describe parts of the real world. Contexts are represented as facts and conditions that capture if the if-then relations hold within the context, as illustrated in Figure 8.

\[
S_1 = [\delta | \delta \models \text{bird}, \hat{a}, 1 \gg] \\
S_2 = [\delta | \delta \models \text{flies}, \hat{a}, 1 \gg] \\
B \models \text{present, air, 1} \gg \land \text{penguin, 0} \gg \land \ldots \\
C = S_1 \Rightarrow S_2 | B
\]

**Figure 8: Context modeling using situation theory [24]**

A similar concept is proposed by Gray and Salber [38], who used first predicate logic to represent contextual propositions and relations.

Logic based models have a very high level of formality and can be composed in a distributed manner. However, there are no full-logic reasoners yet available on mobile devices, as they are computationally demanding.
2.3.2.6  Ontology Based models

An ontology is an explicit and formal specification of conceptualization, meaning that it presents a formal description of concepts and relationships among them in some area of interest. Therefore, an ontology is a terminology that provides a shared understanding of domain, which can be communicated across people and communication systems.

Ontologies are very suitable to describe concepts and interrelations in a data structure understandable to computers. Ontologies add meaning to the information (i.e., semantics), which is important for exchange of information among different communication systems, and enable reasoning support. Reasoning allows one to automatically check for inconsistencies of the ontology and information, unintended relationships between classes, and classify instances into classes. This is very important when designing large ontologies, when multiple authors are involved, and when integrating and sharing ontologies from different sources. The knowledge is expressed in one of ontology languages using an arbitrary number of sub-concepts and facts, which are then put as an input to a reasoning process and evaluated. These characteristics of ontologies make them suitable for representing and modeling context information.

Many context management frameworks today choose ontologies as a context modeling technique. Unfortunately, neither of these frameworks reuses the existing ontologies or extends them for their own purposes, because they are hard to interpret by anyone except the ontology creator(s). Instead, every context management framework starts from the scratch, by defining its own ontology. Therefore, we see a need for (1) creation of graphical tools that could be used even by non-ontology experts for loading, interpreting, and extending existing ontologies; and (2) standardization of context ontology that would define basic context parameters which could be reused and further extended for a particular domain.

When choosing an ontology language to design an ontology, one should be aware of a trade-off between sufficient expressive power (what language can say) and efficient reasoning support (whether a language is computable in real-time). The richer the language is, the more inefficient the reasoning support becomes. W3C has approved the Web Ontology Language (OWL) [39] and Resource Description Framework (RDF) [40] as standards for ontology languages. OWL was built on top of RDF/RDF Schema (RDFS), because of RDF(S)'s insufficient expressivity, which is limited to binary predicates, subclass hierarchy, and property hierarchy with domain and range definitions of these properties. OWL has been designed by adding an additional vocabulary to RDF(S), but preserving the good (previously mentioned) characteristics of RDF(S). OWL is derived from DAML+OIL Web Ontology Language [41], a joint initiative from US and European research groups to define a richer ontology language. Having the trade-off between expressivity and efficient reasoning support in mind, OWL has been designed in three increasingly expressive sublanguages: OWL Lite, OWL Description Language (DL), and OWL Full. OWL Lite is easiest to implement, but it has restricted expressivity, whereas OWL Full offers users maximum expressivity using all the OWL language primitives and keeping its full upward compatibility with RDF, both syntactically and semantically. However, becoming so powerful, this made OWL Full undecidable, without any hope of complete (or efficient) reasoning support. Alternatively, OWL DL includes all the OWL language constructs, but they can be used only under certain circumstances. Thus it offers desirable computational properties for reasoning systems. However, this comes with the price that OWL DL looses full compatibility with RDF: an RDF document needs to be extended in some ways and restricted in others to become a legal OWL document. But, every legal OWL DL document is still a legal RDF document.

Note that ontology-based models are similar to object-oriented models in relating domain classes and instances to classes and objects, which together with high formality level, makes
them very suitable for context modeling. However, their drawback lies in computational load needed for ontology reasoning; thus, this needs to be taken into consideration when choosing an ontology language for designing context modeling schema.

2.3.2.7 Evaluation

The evaluation of the described context modeling techniques are summarized and presented in Figure 9.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Distributed Composition</th>
<th>Partial Verification</th>
<th>Richness &amp; Quality of Information</th>
<th>Incompleteness &amp; Ambiguity</th>
<th>Level of Formality</th>
<th>Applicability to existing frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-Value Models</td>
<td>-</td>
<td>-</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td>Markup Scheme Models</td>
<td>+</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>Graphical Models</td>
<td></td>
<td></td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Object Oriented Models</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Logic Based Models</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>Ontology Based Models</td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>+</td>
</tr>
</tbody>
</table>

*Figure 9: Comparison of context modeling techniques [24]*

Looking at the analysis shown in Figure 9 it can be seen that:

- Key-value pairs fail on all categories, except applicability to existing ubiquitous environments;
- Markup scheme models are particularly suitable for partial verification, because they contain scheme definition (e.g. CC/PP schema) and there are a set of validation tools that can check types and ranges. Markup scheme models can be applied to existing markup-centric infrastructure, such as Web services;
- Graphical models are valuable for applicability requirement, since they can derive code or entity-relationship model from the model. It is possible to do partial validation of a context model. The level of formality is usually low for any graphical model. A graphical model serves more to ease human understanding;
- Object oriented context models are strong in distributed composition requirement: classes (of context information) and objects (used to update information) can be created and used in a distributed environment. A fairly good level of formality is reached through the use of well defined interfaces to access the object's content. They are applicable to existing object oriented ubiquitous runtime environments;
- Logic based models can be distributed, but it is difficult to make partial validation. These models have a very high level of formality, but they lack partial validation support. Applicability to existing ubiquitous runtime environments is a major issue, because there are no full logic reasoners available on ubiquitous computing devices;
- Ontologies have many similarities to object oriented models; but with concepts and facts instead of classes and objects. Therefore they are also very strong in terms of
meeting the distributed composition requirement. Partial validation is possible and a lot of validation tools do exist. Formality level of all ontology models is high. However, the reasoning support poses requirements on computing devices, which often cannot be fulfilled in ubiquitous computing systems.

From the evaluation of those techniques, it has been seen that most of requirements are met using ontologies, followed by object-oriented models.

2.3.2.8 DL-Lite ontology based on Manchester OWL syntax

Given the results of this evaluation, but keeping in mind that the users of our system will mostly use it from their mobile devices, we would like to reduce the computational requirements needed for reasoning. Therefore, we decided to adopt a light version of an ontology model to represent context. We have used the model developed by Warsaw University of Technology for the purpose of the MIDAS (Middleware Platform for Developing and Deploying Advanced Mobile Services) project [42].

This ontology was developed using DL-Lite ontology language [43], which is a subset of OWL-DL optimized for fast reasoning on top of relational databases. This language supports the basic terms of classes and properties, and handles statements about subsumption, disjointness, role-typing, participation constraints, nonparticipation constraints, and functionality restrictions. This ontology was used on a Java enabled mobile device. The limitations in the description logic that made these improvements possible were not limiting when modeling a domain [3]. This DL-Lite context ontology was encoded using Manchester OWL syntax, because it is much easier to parse than the usual OWL syntax (it requires only two linear scans of the ontology file and does not require construction of a tree structure during parsing) and because its representation is in plain text which is a half of a size of the equivalent OWL representation (based on XML).

For representing the ontology on mobile devices researchers at Warsaw University of Technology developed a dedicated Lightweight Ontology library [44], which implements the Jena [45] API in a form suitable for mobile devices. This library parses the ontology file and creates an in-memory representation of the ontology (supporting all the structures present in OWL-DL) based on hash tables. Its simplicity suits resource constrained devices (such as J2ME mobile phones and personal digital assistants (PDAs)).

2.3.3 Context synthesis

Applications use high-level context information, which has been abstracted from the context information obtained from context sources. This high-level context information is inferred from the existing information using application-specific inference rules. This reasoning process is called context synthesis. The problem with context synthesis using existing rule-based reasoning is the long delay experienced by the end-user (or their application) waiting for the result of a context query [46], especially when large data sets and rule sets are used [47].

To reduce the waiting time, we propose to use context operators for context synthesis [2][3]. Operators for context synthesizing are domain-specific functions over context data. By performing operations over existing context information using these domain-specific actions, new context information that previously did not exist in the system can be produced. Operators could be used on a higher level to synthesize information for a certain user, device, network, place, or other object, as illustrated earlier in Figure 2. The result is reduced waiting time as described in the next paragraph.
In rule-based reasoning, rules are applied to the existing data sets in order to infer new context information, which over time increases with an increasing amount of stored data and an increasing number of rules. We use a different approach to provide a user with the desired high-level context information. Firstly, we perform a procedure to find a suitable operator to perform the context synthesis function and secondly we invoke this operator to obtain the synthesized context and return it to a user's application as a result for the context query. In order to find the correct operator, we perform type matching of the user's supplied inputs and a desired output type (set in the context query) against the operator's input and output types. Note that during this matchmaking procedure the system does not know in advance which implementations of the operators exist and how they are realized. As this procedure requires only a subclass ontology matching from the whole ontology-based reasoning support, we avoid performing computationally expensive reasoning needed to infer a new context type. Combining this with the use of Beanshell [48] scripts written in Java code to perform the context synthesis functions over the existing data, results in reduced waiting time for the result to a context query. Note that we have managed to perform this context synthesis procedure on a Java-enabled mobile device, such as the Nokia N800 (for performance results please refer to [3]).

Note that the Beanshell is an open source java script engine [48]. The reasons for choosing Beanshell are: 1) it is a small, free, embeddable Java source interpreter (~150K jar file) with object scripting language features, and is written in Java, 2) it has transparent access to all Java objects and APIs, 3) it can work in security constrained environments without a classloader or bytecode generation for most features, and 4) it runs in four modes: command line, console, applet, and remote session server. In our implementation the operator scripts are part of the context service process and they can be programmatically added and removed by the middleware. The advantage of this approach is that these java scripts do not need to be compiled; only the ones that will be invoked need to be loaded into an interpreter. This saves some compilation time because many of the operators will never be invoked.

To discuss about the performance of interpreting time (that could be degraded in case of repeated calls to the same script), it is important to explain how the Beanshell interpreter works [49]. The first time a script is read or sourced into an interpreter, Beanshell uses the parser to parse the script and stores it representation internally as an Abstract Syntax Tree (AST). Note that this parser only parses the structure of the language – it does not interpret names, or executes methods or commands. The AST consists of Java objects representations of all of the language structures and objects. When executing a script, Beanshell executes each element of AST and tells it to perform an intended operation (e.g., variable assignment, for-loop, etc.). Note that the execution of ASTs avoids re-parsing of the text of the method and its performance is limited only by the speed of application calls, the speed of Java Reflection API (if types are used), and the efficiency of the implementation of structures in Beanshell. When parsing a Beanshell script line by line, the ASTs are executed and thrown away. However, when invoking a Beanshell method, it is parsed only once - when it is declared in the Beanshell script. It is then stored in a namespace like any other variable. Successive invocations of the method execute the ASTs repeatedly, but do not reparse the original text. Therefore, when repeatedly invoking the same scripted method, the script will not be reparsed, resulting in faster execution. So, by wrapping the specialized operators' code in the Beanshell script method and executing this method repeatedly, we avoid the performance problems that we would have – if we would interpret this script repeatedly.

This process of retrieving high-level context information is initiated by a user application by sending a context query. After performing an operation by the operator, the output of the
operation is sent to the application as a result of the context query. This result is called a synthesized context, since it is generated by context synthesis.

Note that in the MUSIC project reasoners are seen as specialized context sensors which process the existing context data acquired by sensors in order to compute high-level context information. These reasoners are plugged into the middleware (see Section 2.3.1) in the same manner as sensors, registering the type of (high-level) context they provide. Therefore, in MUSIC, context synthesis is performed by exact match of the provided context types by sensors and reasoners against the context type required by an application, and obtaining the information from the matching reasoners. Additionally, these reasoners need to be installed and running on the device as well as registered in the network to be discovered and used by interested applications. The advantage of this approach is in flexibility, reusability, and transparency of the reasoners to the middleware and the applications. The disadvantage of this approach is that for every context query there has to be an exact match of the reasoner that is available and registered locally or in the network to provide the desired high-level context information. Note that these reasoners are potentially computationally demanding which is of concern when they are running on mobile devices along with applications, context middleware, and sensors. Currently, our context operators are provided in a file structure to applications running on a device and only the relevant operators (that are determined by an operator matching procedure) are loaded into the interpreter and executed as functions operating on the sensed context data, upon arrival of a context query.

2.3.3.1 Operator model

This subsection provides a formal specification of operators for context synthesis. Let $\text{Op}$ be a set of context operators:

$$\text{Op} = \{ \text{op}_1, \text{op}_2, \ldots, \text{op}_n \}, n \in \mathbb{N}$$

An operator $\text{op}_i \in \text{Op}$ is represented with a bundle of the operator's description and implementation:

$$\text{op}_i = \{ \text{desc}(\text{op}_i), \text{impl}(\text{op}_i) \}$$

An operator's description $\text{desc}(\text{op}_i)$ is defined as:

$$\text{desc}(\text{op}_i) = \{ \text{name}_i, \text{In}_i, \text{Out}_i, \text{Uses}_i \}$$

where:

- $\text{name}_i$ is the name of $\text{op}_i$
- $\text{In}_i$ is a list of types of inputs that will be provided to $\text{op}_i$
- $\text{Out}_i$ is the type of output produced by $\text{op}_i$
- $\text{Uses}_i$ is a list of other (simpler) operators $\text{op}_j$ used in its execution

An operator's implementation $\text{impl}(\text{op}_i)$ specifies an operator's implementation as an implementation of the operation $F_i$, which takes list of input arguments whose types are specified in $\text{In}_i$, produces as a return value of the type specified by $\text{Out}_i$, and invokes implementations of used operators (i.e., $\text{impl}(\text{Uses}_i)$) in its execution.

An example implementation of the operation $F_i$ is the program $F(\text{In})$, shown in Listing 1. The program takes a list of inputs, here represented by variables $\text{In}$. At the beginning, the program checks whether the list of used operators is empty, then performs the specified operation on the list of inputs. If the list of operators is not empty, the program will invoke each operator's function and pass as arguments the newly obtained inputs. This program is abstraction of an operator’s implementation to illustrate the process of context synthesis. In our implementation, this program is realized as a java script. For simplicity, operations
(methods) performed by an operator \( op_i \) and other operators in the program have the same name (i.e., \( F \)).

```java
program F(In)
begin
  if Uses is empty
    out=perform operation on In
  else
    for each op from Uses
      In_New=perform operation on In
      out=op.F(In_New)
  return out
end
```

**Listing 1: An implementation of an operator's function \( F \)**

As defined in this model, operators consist of a description and an implementation. They are described by an ontology, similar to the representation of context. The operator’s description specifies the name of this operator, the types of the required input arguments, the returned output type, and the list of other operators used in performing the operator’s function. Operators are implemented as Java scripts that perform an action as specified in the operator’s ontology. As with the context model, operators are created for a specific domain and can be used by a set of applications in that domain. In order to provide context synthesis functions for applications in another domain, a new set of operators needs to be provided to the middleware, along with their ontology schema.

### 2.3.3.2 Operators ontology

Consider the following context queries:

1. "Find all shops within 500m from me."
2. "Find all streets within 500m from Kista Centrum."
3. "Find all towns within 20km from Stockholm."
4. "Find all post offices within 50m from my office."
   ....

The number of these and other similar context queries (which are very specific and implement the same functionality, but take different input and produce different output types) is quite extensive. If each context query required its own implementation of an operator, it would significantly increase the database storage required along with the time needed to find the most suitable instance. Furthermore, the correct operator might not be found, unless the exact relation between the requested and the desired operator's input is specified. Relationships between operators and their input and output types are described in the operator's ontology (e.g., the context query asks for streets near the user (specified in terms of a "Range"), while available "InRange" operator implementations return postal codes instead of streets).

To solve this problem, we distinguish between generic and specialized operators. Generic operators are part of an ontology schema, representing an umbrella for all the different implementations of a function they provide. They are also part of an API provided to application developers. On the other hand, specialized operators can be created/modified and inserted into the middleware by application or system developers. Specialized operators are not directly visible to application users and which operator is invoked will be determined by the middleware at runtime.

An example of an operator description file (InRange.man) is presented in Figure 10. This file contains all the specialized operator descriptions. Figure 10 shows only the specialized
operator CyclistsInRange, but there could be others as well (e.g., UsersInRange). The description of the CyclistsInRange (specialized) operator is interpreted in the following way: it has the name "CyclistsInRange" and is derived from a generic operator (i.e. InRange). It requires an input of the type Cyclist and produces an output value of the type Cyclist. The operator uses the result from another (simpler) operator DistanceBetweenXYZLocations to calculate the distance between two locations.

![InRange.man]

**Figure 10: CyclistsInRange description**

As noted earlier, specialized operators are implemented as Beanshell scripts.

Note that this example of CyclistsInRange operator is taken from the MIDAS project and our previous paper [3] in order to illustrate the complete process of context synthesis. In MIDAS, all data (including context information) was shared among nodes in the network without considering the user privacy issues. This context synthesis approach was used by a set of sports applications for providing a near real time ranking service during the live cyclocross race in Gieten, Netherlands. Additionally, cyclists gave their consent to be tracked during the race for their ranking to be calculated and displayed to the spectators' mobile devices. However, as it will be shown later in Chapters 4 and 5, our context-addressed communication system is designed with user privacy requirement in mind. Therefore, this example of CyclistsInRange should not be discussed in this context.

2.3.3.3 Operator space

Figure 11 shows the structure of the Operator space – a repository of operators. The root folder (i.e. operators/) contains all generic operators (which are also folders), containing in turn their specialized operators. Note that specialized operators are bundles of an operator description (an instance of the operator ontology encoded in Manchester OWL format, i.e., a .man file) and an operator implementation (a Beanshell script, i.e., a .bsh file).

As the performance of the ontology syntax parser depends on the number of triples it needs to parse and the specialized operator description is written in the form of triples (as depicted in Figure 10), we proposed to use this kind of file structure to reduce the search effort of finding the relevant specialized operators able to synthesize context for the supplied context query. Instead of parsing all triples of a large ontology file describing specialized operators of all the generic operators, we propose to parse smaller ontology files containing sets of fewer specialized operators that belong to the relevant generic operators. By relevant generic operators we refer to the generic operator requested in a context query and its dependency operators. Additionally, a path to the operator description file is programmatically constructed based on the operator root folder and generic operator name, thus there is no need to search for files or folders in the operators' directory.
2.3.3.4 Operator matching algorithm

The context synthesizing process determines the most appropriate specialized operator to invoke from the available (specialized) operators by using a reasoning process (which takes into account the required output type and supplied input types). The idea behind the operator matching algorithm, illustrated in Figure 12, is to enable different applications (or even different context systems) in the same domain (in our scenario a sport domain) to use the same “functions” to synthesize context information, without being concerned about the implementation of these functions. For example in a sport scenario: a racing application and media application deployed on different devices should be able to remotely query each other (using the same middleware API and generic operators) for results of the race and rankings of all athletes in the competition. The operator matching algorithm, as shown in Figure 12, returns the specialized operator with either exactly the same description as specified by the query or a more generic one (if an exact match does not exist). This figure shows the algorithm itself, initiated by the user’s context query, along with the invocation of the matched (specialized) operator.

An example of a context query is: \texttt{InRange("101", 50, ModelConstants.Cyclist)}, where the response time is bounded to 5sec. This example can be interpreted as follows: retrieve all cyclists in the range of 50 meters from the cyclist with the ID="101" and the result should be returned within 5 seconds. If the result is not computed by that time, then the synthesis process will be interrupted, and a response will be returned to the query initiator containing an empty list of values and a flag indicating that the query was unsuccessful (hereby distinguishing whether the query was unsuccessful due to the interruption of context synthesis execution or the inability to acquire and compute the desired (high-level) context). After receiving the query, the operator matching algorithm retrieves all available specialized operators and processes the supplied data in order to find an exactly matching specialized operator (by checking if output and input types of the operator and the query match). Otherwise it will return a more generalized one, i.e. \texttt{UsersInRange}, which would return \texttt{Users} instead of \texttt{Cyclists} as result. Finally, it invokes the matching operator.

Note that in our context model, context values are assigned to the context entities represented by \texttt{DomainInstance} objects. Each of \texttt{DomainInstance} objects can have a number of property values assigned to it, and can belong to a number of classes. These classes are
represented by objects of the type \textit{DomainClass} and \textit{DomainProperty} (respectively) which correspond to those present in the context model ontology. More importantly, these classes are used as input types in the specialized operator description. Therefore, I decided to pass input arguments of a context query as \textit{DomainInstance} objects into an operator script in order to manipulate this data as context information. In order to achieve this, I developed a means of mechanically mapping the domain classes from the context model to the corresponding java classes, as well as from property names to java class variables.

\textbf{Figure 12:} This figure shows the algorithm itself, initiated by the user's context query, along with the invocation of the matched (specialized) operator.

The operator matching algorithm uses the subclass relationship of the context model terms in order to find the matching input and output types. It performs five steps (see Figure 12):
1. The context synthesizer receives the context query containing the generic operator name, the list of input arguments, and the output type that operator needs to produce. This synthesizer initiates the operator match.

2. The operator matching algorithm first retrieves all the specialized operators of this generic operator (i.e., InRange).

3. For each retrieved specialized operator, it checks if its output type is exactly the same as, or otherwise if it is more generic than the output type set in the context query (i.e., its super class). Note that the goal of this matching is to find the most suitable operator to perform the desired operation, without knowing in advance which specialized operators (i.e., implementations) exist and how are they realized.

4. If the condition from step 2 is met, then the algorithm will check if the list of input types (of a specialized operator) is the same as the list of types of input arguments set in the query. If the input type lists match, then the operator’s bundle containing the operator’s description object and the operator’s implementation is returned. Otherwise, null is returned.

5. The matching specialized operator script is invoked and returns its result to the entity (i.e., the user or an application) which sent the context query. Note that in this step we pass the hash table of context input parameters from the context query to the newly created object of the Cyclist class, which is in this example its domain class for these input properties. Next, we retrieve the domain instance of this Cyclist object and add it to the list of inputs, which are used to invoke the matching operator.

2.3.3.5 Evaluation of context synthesis using context operators

Note that design, implementation, and evaluation of context synthesis using context operators was part of MIDAS project, whose aim was to design a platform for easy development and deployment of mobile applications and services. MIDAS was specifically designed to be used in Mobile Ad hoc Networks (MANETs). The most important goal of the MIDAS platform was to enable applications running on different nodes to share information by inserting data in and retrieving data from shared data space. This shared data space is implemented using a combination of data replication and remote operations—but this fact is transparent to applications. Additionally, in the scope of this project, I have published the paper [3] where I have implemented and evaluated the performance of the context operators in terms of the response time to a context query sent by the application. Because of the specifics of this project goal I have assumed that all the context information is available locally on a mobile device. Therefore, the design of operator space is slightly different in this paper than it is presented in this thesis.

Figure 13 illustrates the Operator space file structure that was specifically designed for the MIDAS platform. This Operator space file structure has three specific operators which are responsible for retrieval of context information: GetContext.bsh, GetClassContext.bsh, and GetInstanceContext.bsh. Note that these specific operators do not have a generic operator representing them, and they are used for distinct purposes. When specific context operators need to retrieve context, they will provide DomainInstance objects (i.e., individuals that are instances of a particular class representing context owners containing a set of properties that represent context parameters) to the GetContext operator to retrieve the missing context values. It is also possible to retrieve context data directly from the repository without context synthesis, via the GetInstanceContext and GetClassContext scripts. GetInstanceContext is used to obtain the domain instance with the supplied datatype properties from the context query. We can also query the repository for other properties of the same instance.
GetClassContext is used when we do not know the instance, but rather use a domain class with the specified property name-value pair to identify this instance.

![Operator space file structure designed for the purpose of MIDAS project](image)

**Figure 13: Operator space file structure designed for the purpose of MIDAS project**

As noted earlier, performance of the context operators has been evaluated in terms of the response time to a context query sent by the application. The response time is divided into the time needed to find the correct operator (i.e., operator matching), the time needed to obtain the needed context information (formatted as ontology data) from its repository, and the time needed by this operator to perform the actual context synthesis (i.e., operator invocation).

I ran all performance tests on a Nokia N800 device with a JamVM virtual machine [50] with a compiler for Java 1.4. The device was chosen by the MIDAS project because it is Linux based, allowing network and low-level programming.

Table 1 presents the response times obtained by sending the same context query, but varying the number of available specific operators (i.e., 1, 2, 5, and 10) when performing the operator matching algorithm, and then calculating the mean value.

<table>
<thead>
<tr>
<th>Table 1: Response times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average response times</td>
</tr>
<tr>
<td>with varying number of 10 specialized operators (i.e., 1, 2, 5, 10)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>1st Query</th>
<th>Standard Deviation</th>
<th>10th Query</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matching algorithm time</td>
<td>2.49 sec</td>
<td>0.009 sec</td>
<td>1.94 sec</td>
<td>0.07 sec</td>
</tr>
<tr>
<td>Loading specialized &amp; root scripts time</td>
<td>1.7 sec</td>
<td>0.087 sec</td>
<td>No average, for the first time only (1.7 sec)</td>
<td>No standard deviation</td>
</tr>
<tr>
<td>Total operator matching time</td>
<td>4.2 sec</td>
<td>0.087 sec</td>
<td>1.94 sec</td>
<td>0.07 sec</td>
</tr>
<tr>
<td>Context retrieval time</td>
<td>0.37 sec</td>
<td>0.006 sec</td>
<td>0.09 sec</td>
<td>0.001 sec</td>
</tr>
<tr>
<td>Loading dependency scripts time</td>
<td>0.15 sec</td>
<td>0.001 sec</td>
<td>0.17 sec</td>
<td>0.015 sec</td>
</tr>
<tr>
<td>Operator invocation time</td>
<td>0.67 sec</td>
<td>0.008 sec</td>
<td>0.36 sec</td>
<td>0.04 sec</td>
</tr>
<tr>
<td>Total query time</td>
<td>5.4 sec</td>
<td>0.045 sec</td>
<td>2.57 sec</td>
<td>0.07 sec</td>
</tr>
</tbody>
</table>
Note that before the java scripts can be invoked, they have to be loaded into the interpreter and the classpath has to point to the folder where these scripts reside. These scripts can also invoke other scripts (from different folders), thus these other scripts need to be invoked in the caller's context (the so called namespace). Therefore, when the first query is sent, the total time needed to find the most appropriate specialized operator (i.e., the total operator matching time) also includes the time needed to set the namespace to point to the generic operator folder (e.g., InRange), as well as load specific operator scripts from this folder and from the root operator folder. For all successive queries this operation is cached. When invoking the specialized operator found by the matching algorithm, some additional time is needed to load the scripts from the dependency operator folder (e.g., DistanceBetween).

As it can be seen from Table 1, the response times for the first query are twice as large as for the other following queries, because the caching speeds up the subsequent operations. The operator matching algorithm takes 2 seconds on average, however for the first query it requires 4 seconds (including the initial time needed for loading the necessary scripts). Context retrieval (of three cyclists' data) was rather quick as was the operator invocation time. The number of concepts required by an application was small. With regards to performance with increasing number of domain instances, please refer to [51]. Note that operator invocation time includes the time needed to invoke CyclistsInRange and DistanceBetween operators. We used SQL prepared statements to retrieve context from an HSQL database. The total time needed to receive the result of context query took on average 2.5 seconds, but 5.4 seconds for the first query.

Context distribution and querying

Context information is distributed from sensors to the applications that have expressed interest in retrieving this information. As mentioned earlier, these applications might be running on different devices than the sensors, therefore the information about provided and required context types has to be propagated in the network infrastructure. An application acquires particular context information by sending a context query to the middleware.

We split context queries into two categories, depending on whether they contain an operator or not: complex and simple context queries. Context queries that contain an operator, whose inputs determine the context information that needs to be obtained, are called complex queries. The other type of context query simply specifies the entity, scope pair of the information it wants to retrieve, without using operators, these are called simple queries. A context query also contains a list of so called context quantifiers, which influence the way context information is retrieved before composing and sending back the result. This was one
of the reasons that we introduced asynchronous context queries - to give applications more control in execution of context queries (e.g., to bound the execution time of context queries and terminate the querying process if the timer value set for this query expired). Therefore, query processing is performed in a separate thread and the control of query execution is passed back to the application (via a callback) when the result of context query is composed. The other reason for introducing asynchronous queries is because context information is retrieved from sensors using event-based mechanism. Therefore, upon a context update, the middleware performs the context quantifier logic, and returns the result to the application using the application provided callback.

An example of a context query is: \texttt{InRange("ID01", 50, ModelConstants.Cyclist)}, \texttt{5sec}, which sets the maximum time limit to a context response. When the timer set expires, the TimerCallback is invoked, in which the querying and synthesizing process is interrupted, and the response is sent back in the callback method. Note that we have implemented the following implementations of context quantifiers: \texttt{TimeLimitQuantifier} (to set the maximum time limit on waiting for context response), \texttt{AccuracyQuantifier} (to set the request for the minimum accuracy of the context information), and \texttt{FreshnessQuantifier} (to set the maximum context information age limit). These quantifiers can be used by an application when composing a context query. Application also needs to provide the implementation of callback function, which will be invoked by context middleware for returning the context result.

Listing 2 shows how to create a complex context query. The \texttt{ContextQuery} constructor takes a generic operator name and a desired output type. We select the generic operator name from the available list of names provided by \texttt{GenericOperator} class and the output type from the \texttt{ModelConstant} class. Note that \texttt{ModelConstants} class is generated offline from the context model containing names of its domain classes and properties as fields in the \texttt{ModelConstant} class. Next, list of input types and values are added to the query, by selecting the context class from the \texttt{ModelConstants} and assigning it a value.

```java
ContextQuery query = new ContextQuery(GenericOperator.InRange,
ModelConstants.Cyclist);
List inputs=new ArrayList();
Input input1=new Input();
input1.setType(ModelConstants.PlayersNumber);
input1.setValue(playerNumber);
inputs.add(input1);
Input input2=new Input();
input2.setType(ModelConstants.Range);
input2.setValue(range);
inputs.add(input2);
query.setInputList(inputs);
```

\textbf{Listing 2: Context query creation}

Similarly, a simple context query can be created using the \texttt{ContextQuery} constructor that takes an entity, scope pair as arguments (e.g., to retrieve Bob's location).

```java
IEntity requiredEntity = Factory.createEntity(ContextEntity.User,"Bob");
IScope requiredScope = Factory.createScope(ModelConstants.Location);
ContextQuery query = new ContextQuery(requiredEntity, requiredScope);
```

Listing 3 shows an example of setting the time context quantifier and an application callback function to the context query and how the context query is invoked.
TimeContextQuantifier quantifier = new TimeContextQuantifier();
quantifier.setMaximumTimeLimit(5000);
List quantifierList = new ArrayList();
quantifierList.add(quantifier);
query.setContextQuantifierList(quantifierList);
ApplicationCallbackImpl callback = new ApplicationCallbackImpl();
try {
    contextHandler.resolveQuery(query, callback);
} catch (Exception e) {
    e.printStackTrace();
}

Listing 3: Setting the time context quantifier and query callback function in the context query

Note that TimeContextQuantifier implements ContextQuantifier interface, which does not specify any particular method, because every quantifier has different application logic, but this interface is used in specification of a context query. Listing 4 illustrates an implementation of the TimerCallback class provided by Context middleware with the method timeIsUp that takes as an argument an instance of ContextSynthesizer. When the specified timeout occurs, the timeIsUp method invokes the synthesizer's done method, which interrupts the synthesis process, composes an empty context result, and returns it to the application in its provided callback function.

```java
public class TimerCallback {
    public void timeIsUp(ContextSynthesizer synthesizer) {
        synthesizer.done();
    }
}

Listing 4: TimerCallback implementation
```

Note that how the context synthesizer synthesizes the context and how the context is actually retrieved from the context plug-ins will be explained in the next section, where we will propose and elaborate our design of context-aware system architecture.

2.4 Context-aware system architecture

Based on the decisions made in Section 2.3 about the design and implementation of context management activities, we propose the following architecture for our context-aware system (see Figure 14). This architecture will be used throughout the remainder of this thesis. The architecture consists of: applications, context middleware, sensors plug-ins, context provider, and context distribution.

Applications run on top of context middleware and use context queries to retrieve the desired context information (of its user or some other context entity) when they need it and however often they need it. Note that context queries can be configured to periodical poll or subscribe to changes of the desired context information; instead of fetching of the desired context information only one time. This could be done by implementing a context quantifier for this purpose and inserting this quantifier into the context query. How to achieve this will be elaborated in the rest of this section.
Context middleware (depicted in Figure 14) performs all the context management operations that were described in Section 2.3 and illustrated in Figure 4. This middleware provides a generic context-aware system. Sensors plug-ins are used as drivers for sensors to be used by context middleware. For detailed information about sensors plug-ins please refer to Section 2.3.1 about context sensing. Note that context provider and context distribution (shown in Figure 14) are designed as external entities to the context middleware, because they are used to retrieve context information from remote context plug-ins (i.e., available on other devices in the network) and to register the context types provided by sensors to the network infrastructure server in order to provide this context information to the interested applications running on remote devices in the network, respectively.

Context middleware consists of the following components: (1) **Context handler** – used as an interface between the application and the rest of the middleware components, this Context handler performs the following functions: a) determines if the context query is simple or complex (i.e., whether it needs context synthesis) and subsequently forwards the query to the context synthesizer or retrieves the information from the context manager, and b) controls the execution of a context query based on the indicated context quantifier in the query; (2) **Context synthesizer**, **Operator space handler**, and **Operator space** – all together used for context synthesis; (3) **Context manager** – used as an interface by the Context synthesizer towards the context plug-ins and Distribution manager for retrieving context information from the local and remote sensors; and (4) **Distribution manager** – used as an interface by the Context manager towards the Context provider and the Context distribution entity.

Figure 15 illustrates the procedure for retrieving low-level context information from sensors provided locally on the device and synthesizing this information into the high-level context. This procedure shows the interaction of context middleware components and we will use it to explain the context middleware components (1), (2), and (3). The context middleware component (4), (i.e., Distribution manager) functionality will be explained in
Figure 16 and Figure 17 that illustrate registration of context sensors metadata to the network and retrieval of missing context from the remote sensors.

Figure 15: Application retrieving high-level context information using context synthesis and retrieval of low-level context data from context plug-ins available on the same device

An application sends a context query to the Context handler (as shown in Figure 15). In this example the Context handler determines that the query is complex (i.e., needs a context synthesis) and that the desired context needs to be retrieved only once. Note that the Context handler controls the execution of a context query based on the selected context quantifier (e.g., it can interrupt the execution of the query if a time quantifier is invoked and the response time to the query exceeds, returning the result to the application containing an empty list of values and a flag indicating that the query was unsuccessful). Alternatively, the Context handler can periodically execute the same query or subscribe to the changes of context, as well as perform the context synthesis on the received update (if an appropriate context quantifier is invoked). After determining that the query is complex (and optionally invoking the quantifier logic), the Context handler extracts from the query the generic operator name, the list of input parameters names and values, the desired output type, and a callback method and forwards them to the Context synthesizer. The Context synthesizer
retrieves the list of specialized operator descriptions from the Operator space handler, which interacts with the Operator space (explained in Section 2.3.3.3). The retrieved specialized operator descriptions contain the list of input and output types of a specialized operator (an example of a specialized operator description can be seen in Figure 10). These specialized operator input and output types are matched by the Context synthesizer against the input and output types provided in the context query, in order to find the most appropriate specialized operator for context synthesis (see Section 2.3.3.4 for a detailed explanation of this matching algorithm). Finally, the Context synthesizer contacts the Operator space handler to invoke the matching specialized operator. Before the invocation of the specialized operator's script, the Operator space handler loads this operator's script and all its dependency scripts into an interpreter, and then invokes the method of the specialized operator.

If the implementation of this operator's method lacks some context information for its execution, it will contact the context synthesizer for the missing context, passing the <entity, scope> pair to denote which entity's context information needs to be retrieved. The context synthesizer will initiate the context retrieval by implementing the IContextListener interface and adding itself to the list of listeners at the context manager. The context manager will detect the available sensors able to provide the requested context information, activate these sensors, and retrieve their context updates via contextChanged events. Next, the context manager will process the incoming events and notify its listeners (including the context synthesizer). The context synthesizer will in turn invoke the same specialized operator again, assigning the obtained context information to the domain instance of the context entity, which is assigned to the list of operator's inputs. After retrieving the result of the operator's execution, the Context synthesizer will propagate this result as a synthesized context to the Context handler in the handler's specified callback method. Note that at this point of execution the Context handler can optionally repeat the context synthesis and/or context retrieval if this is specified by the context quantifier. Finally, the Context handler returns the synthesized context result to the application using the application's provided callback method.

Figure 16 shows an installation procedure of context plug-ins to the context manager, which fires the PROVIDED_CONTEXT_TYPE_ADDED event containing the EntityScopePair object that represents the sensors metadata. This event is retrieved by the distribution manager. The Distribution manager invokes the Context distribution entity to register these sensors metadata to the network. Similarly, context plug-ins can be uninstalled from the context manager, triggering the PROVIDED_CONTEXT_TYPE_REMOVED event containing the sensors metadata to be removed from the list of discoverable sensors in the network. Note that this un-installation is implemented programmatically as part of the shutdown procedure of the context plug-ins.
Figure 16: Installation and un-installation of context plug-ins, triggering registration and deregistration of their provided metadata to the network

Figure 17 presents the case where the Context synthesizer contacts the context manager for the missing context information, and the Context manager did not detect the appropriate context plug-ins on the device. Therefore, the Context manager sends a **REQUIRED_REMOTE_CONTEXT_TYPE_ADDED** event, containing the **EntityScopePair** object. This event is received by the Distribution manager. Upon arrival of this event, the Distribution manager subscribes to the Context provider in order to be notified about the context changes. Note that the subscribed method showed in Figure 17 can be used for immediate notification of current context value (i.e., synchronous context retrieval) and for retrieval of further notifications when the context value changes (i.e., asynchronous context retrieval). The Context provider performs the following functions: (1) locates the sensor nodes which can provide the requested context information, (2) selects the sensors based on the quality of context they are able to provide, (3) subscribes to context updates provided by these selected sensors, and (4) propagates the received context updates back to the Distribution manager. After receiving the notification containing the context update, the distribution manager fires the **contextChanged** event. The Context manager processes the incoming event and notifies the listeners (which includes the Context synthesizer).
Figure 17: Context synthesizer retrieves missing context information from context plug-ins available on remote nodes in the network

It is important to remove ContextListener object from the Context manager when the context information is no longer needed. In the example illustrated in Figure 18, the Context manager detects that it needs to unsubscribe from the remote context plug-ins. Thus, it fires the REQUIRED_REMOTECONTEXT_TYPE_REMOVED event containing the sensor metadata that are used to identify the context plug-ins to whose updates the Context provider should unsubscribe from. Finally, the Context provider performs the unsubscribe operation from the appropriate context plug-ins.

Figure 18: Remote context information is no longer needed, removing context listener and unsubscribing from context plug-ins

2.5 Summary

In this chapter an introduction into an area of context-aware systems was given, along with the historical review and analysis of definitions of context, which resulted in defining our own. A key point in our definition is that context consists of a number of attributes that belong to a certain entity and together characterize the situation of this entity. We distinguish between five types of entities which can be characterized as owners of context information: a
person, device, a network (interface or connection), a place, and an object. Thus, by assigning all context information to a certain entity, we can query about some information of an entity, i.e., user context, device context, network context, place context, and object context.

Additionally, context information should be retrieved from the sensors that are deployed in the environment through some automated means. This information should be modeled and processed in a way to be unambiguously interpreted by applications, while hiding low-level sensing details from the applications. Applications can use this information to adapt their behavior and make appropriate decisions in order to assist a user in his/her daily tasks, thus requiring less input from the user and enabling him/her to be more productive.

Since we envisage applications to run on mobile devices, these applications need to timely discover available sensors that provide context information and acquire the needed context information from these sensors. We have learned from our previous work that it is more energy efficient for such applications to collaborate and share context knowledge (that they have discovered, acquired, and modeled) with other applications running on geographically distant locations (which have done the same) than to discover all the information they need themselves when they arrive at each new location. This enables context-aware applications to adapt their behavior (and make appropriate decisions) in advance of arriving at a new location. We also learned that multicast should be used for this context distribution.

We unify these activities, starting from context sensing, context modeling, processing of information into high-level context needed by applications (i.e., context synthesis), and ending with context distribution and querying, by referring to them as context management activities.

We have adopted the existing means of context sensing that uses context plug-ins and their activation/deactivation mechanism to acquire context information from sensors. These context plug-ins were developed by the University of Cyprus in the scope of MUSIC project. On top of this context sensing mechanism we have designed and implemented the context distribution service that disseminates context information (required by an application) from the remote context plug-ins. The design and implementation of this service will be elaborated in Chapter 4.

However, this approach of context plug-ins in not suitable for retrieving (high-level) context information from reasoners, because for every type of context query there should be an exact match of the reasoner deployed on the device that provides this information. Otherwise, this information cannot be queried by applications. Therefore, we have developed our own approach for context synthesis.

Our approach for context synthesis uses context operators, which provide domain-specific functions over the existing context data to produce new context information that previously did not exist in the system. For the implementation of this work we have leveraged the work done by Warsaw University of Technology who developed (in the scope of MIDAS project) the lightweight ontology library for representing and manipulating ontologies on mobile device. Our context synthesis is activated upon an arrival of a context query that is sent by an application. This context synthesis applies relevant context operators based on the match of input and output types supplied in the query against the input and output types provided by an operator. This context synthesis applies relevant context operators based on the match of input and output types supplied in the query against the input and output types provided by an operator. We showed that our context synthesis can be performed on a mobile device, such as Nokia 7700, with 2.5 seconds of average delay. These 2.5 seconds of delay are suitable for delay-tolerant applications that do not need to synthesize context from highly volatile information, which changes value more frequently than once in 2.5 seconds, and for applications that are not mission-critical, thus relying on reliable information (which might be
delayed due to the complexity of an operator's function that is used to synthesize required high-level context).

Note that from these 2.5 seconds, 2 seconds were spent to perform operator matching. Context retrieval as rather quick as was the operator invocation. However, context was retrieved locally from the mobile device repository. Therefore, an open issue is whether the actual bottleneck of the context synthesis is in the performance of the operator matching or in the communication part (i.e., how long does it take to retrieve the missing context information from the remote sensor). Note that this will be investigated as part of the evaluation of our context distribution service. Additionally, we plan (as part of the future work) to improve the performance of operator matching algorithm by caching decisions made by this algorithm for a certain context query. Other open issues that need to be investigated are how to deal with cases when context changes rapidly and how to deal with imperfect or incomplete context data.
CHAPTER 3

CONTEXT-AWARE COMMUNICATION

This chapter focuses on the communication model between communicating parties, introduces context information into this model, and describes how this information can be used to deliver a call/message/content to the receiver using the receiver's preferred communication means and preferred device(s) in their current context. It also sets requirements for the receiver's and network infrastructure that need to be fulfilled in order to support such a context-aware communication delivery service. In this chapter we elaborate reasons why we have chosen to design context-aware communication on the application level of OSI layer using Session Initiation Protocol (SIP) and SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE). We base our decisions on the review of the existing application-level communication protocols in terms of providing a support for asynchronous communication, application-level addressing scheme, and mobility. At the end of this chapter, we introduce the concept of context-addressed messaging and communication.

3.1 Communication model

The communication parties in a communication model can be: caller and callee (when we consider call logic), sender and receiver (in case of messaging), or sender and zero or more receivers (in broadcast/multicast communications). A network infrastructure consisting of communication links and nodes facilitates communication between caller and callee (or sender and receiver). To send a message or to establish a call (session) across the network, the Caller/Sender always initiates the communication and the Callee/Receiver decides to accept the invitation to a session or to accept the message itself or not (as shown in Figure 19). In broadcast/multicast communication the Sender sends a message to a multicast group, to which interested Receivers have previously joined. Both the Caller/Sender and Callee/Receivers have network identifiers, enabling them to establish calls and/or send and receive packets over the network infrastructure. However, note that multicast does not perform network-level identification of hosts that are members of the multicast group. Instead, any host can join the multicast group by responding to an Internet Group Management Protocol (IGMP) query from the router. However, because IGMP operates above the network layer (i.e., IP), the receiver has to have an IP address as a network identifier, to be able to join a multicast group.

In terms of the network infrastructure, today's communication networks are moving from circuit-switched towards packet-switched communications. Today voice, messaging, and
content delivery services are increasingly implemented in packet-switched networks such as Internet. Therefore, we will use the terms Sender and Receiver terms for our communicating entities from now on in the thesis. Additionally, it should be noted that messaging and content delivery are concerned with dissemination of content and not the establishment of a communication session [53]. Van Jacobsson has described dissemination as the third generation of Internet. Van Jacobsson describes three generations as [53]:

- "Generation 1: the phone system – focus on the wires"
- "Generation 2: the Internet – focus on the machines connected to the wires"
- "Generation 3? the dissemination – focus on the data flowing between the machines connected to the wires"

The problem with data dissemination to machines connected to the Internet lies in the heterogeneity of devices, the media formats supported by different devices, and the protocols used for transport of these media formats. People traditionally tried to solve these heterogeneity problems on the lower layers of OSI stack. However, the lower layers of the OSI stack are well defined and standardized, and they are unlikely to be modified.

Note that there are several other widely-used application-level communication protocols, such as the Hypertext Transfer Protocol (HTTP) [54] used for web communication and the Simple Mail Transport Protocol (SMTP) [55] used for e-mail communication. However, when compared to SIP and SIMPLE, HTTP does not provide a support for asynchronous notification. HTTP only provides limited support for mobility via 301 (Moved Permanently – a permanent redirect), 302 (Found – a temporary redirect), 303 (See Other – a temporary redirect), 305 (Use Proxy – redirect to a proxy), and 307 (Temporary Redirect) [56]. SMTP has a good application-level addressing scheme and supports mobility based upon e-mail forwarding -- when the destination information in the e-mail "To" field is incorrect, but the receiver's SMTP server knows the correct destination (i.e., there is a local forwarding address). Depending on the implementation the receiver's SMTP server it either indicates the correct forward path in a 251 or 551 reply to the sender or silently forwards the message to the correct destination without notifying the sender. However, SMTP does not support events and when using relay SMTP servers that store and forward the message (as in Internet e-mail delivery) high latency may occur.

The IETF's of Session Initiation Protocol (SIP) [57] enables the application-level signaling and initiation of multi-user sessions regardless of media content that is transported among participants during the session, where participants can be mobile, can use multiple devices for communication during the session, and can use the same session while switching between different applications (e.g., Internet telephony, multimedia conferencing, and gaming). Being an application-level message-oriented signaling protocol for establishing, maintaining, and terminating sessions between two or more SIP endpoints. It is independent of the underlying transport protocol, allowing multiple sessions to be created between nodes attached to different access networks. It uses SIP URIs to uniquely address users or devices. This enables personal mobility, where a user is reachable regardless of the SIP device he/she is using. Moreover, sessions are not coupled with applications, which allows for session mobility between applications on different devices (i.e., thus a session can begin on one device and move to another device – without having to initiate a new session). SIP has also been extended with notification of presence state using SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) [58], enabling interested subscribers of this information (i.e., watchers) to receive updates of the presence information of a desired entity (i.e., presentity). However, while SIP enables media sessions to be set up and can allow simple messages to be sent in a body of a SIP message, Real-time Transport Protocol (RTP) is used to carry media that cannot be sent in a SIP body. Moreover none of these protocols address
the problems of device heterogeneity or guarantee that a device will be able to decode and render the media format that it receives (only that if a session is being set up using SDP in a SIP message body, that the recipient can indicate that they only support certain CODECs). SIP became a de facto standard for application-level communication, with SIP/SIMPLE providing a number of advantages for mobile devices. Therefore, in this thesis we focus on the design of application-level communication models on top of SIP and SIMPLE.

In the rest of this chapter we will identify the requirements for and design our context-aware communication model. This model will be used as a foundation of our context-addressed communication dispatch system.

3.2 Introducing context information into the communication model

Each communications entity has its own reality formed by its senses and its own experiences of the environment, consisting of its physical location, surrounding people, and objects. Any computationally accessible information sensed about this environment contributes to a so called context space. Some examples of this context information are: the location, ambient temperature, ambient humidity, ambient noise, etc. A message or a call is a means to communicate from the sender to the receiver, potentially over a shared communication medium. Note that the sender and receiver have different context spaces. Figure 20 shows a Sender and a Receiver with their own context space. Each context space is characterized by multiple context parameters. Changing the value of one of these parameters or introducing/deleting a new parameter and its associated value changes the state of a context space.

![Figure 20: Introducing a context space around communicating entities.](image)

Furthermore, a Sender and a Receiver can employ several devices during the course of their communication (see Figure 21). If these parties participate in a communication session, then it is important to preserve a user's reachability across all the devices which this user might use. In practice we need to consider a number of different forms of mobility (for example session mobility, user mobility, service mobility, etc.). Note that these issues are inherently solved by SIP.
Communicating entities can employ different devices in different context spaces (due to the device's specific processing and communication capabilities). As shown in Figure 22, a Sender (currently in context space $i$) uses a smart phone to call/send a message to the Receiver. Receiver$_i$ employs a cell phone and a tablet PC while in context space $j$; and a tablet PC, a smart phone, and a laptop while in the Context space $k$. Therefore, as Receiver$_i$ moves from context space $j$ to context space $k$, a new set of devices becomes available for communication; thus there is a question of should the call or message be automatically redirected to one or more of these devices. This raises the related questions of how does the user indicate which of these devices they want to use and when do they do this. Additionally, should the sender be informed of this change in device or context? Note that these questions will be answered in the sections 5.4 and 5.6 – where we describe the implementation of context-based session initiation and context-aware session adaptation mechanisms in our system.

Figure 22: Communicating entities use their preferred devices in each Context space

Devices can have multiple network interfaces and communication capabilities with regard to voice, messaging, and content delivery. Thus, a receiver might have different preferences regarding the communication means (i.e., how it wants to be contacted) and on which device...
(if there are multiple devices available) in different context spaces. For instance, the receiver can specify that it wants to receive calls and SMS messages via their cell phone while at home (i.e., context space $i$), e-mails on their laptop while in the office and working (context space $j$), as well as SMS messages and RSS feeds about interesting content via their smart phone, while moving between their home and office via some form of transportation (context space $k$). Somehow we need to detect changes in the receiver's context space and adapt the communication to accommodate the user's (pre-specified) preferences for each context space.

To achieve this, a receiver has to have a means to express these context-aware preferences and they must use a trusted entity to be responsible for receiving and adapting this communication (remembering that the communication was and will be initiated by a sender) according to the receiver's preferences in the current context space. Additionally, if the receiver provides information to the sender in terms of the formats of the content it will accept, the sender may wish to adapt what it sends to the preferences of the receiver – subject to the preferences of the sender.

We can now extend our model to have one sender and multiple receivers (i.e., sending a message to multiple recipients). Note that each of $n$ receivers may have multiple devices available in each of their context spaces, as well as different preferences for the communication means and how to be contacted in each of context spaces.

A network infrastructure capable of supporting such a model has to fulfill the following requirements:

- Supports 1:N communication model of a sender and N receivers.
- Support for voice, messaging, and content delivery
- Support for user, terminal, and session mobility enabling a user to utilize communication services while on the move and after switching to another available device in a new context space. This change in the locus of a receiver's communication may or may not be seamless, but the goal is generally that the disruption in communication should be minimized.
- Dynamic adaptation of the communication services should be based upon the receiver's preferences in their current context space (the actual adaptation may be limited by the ability of both the sender and receiver(s) to adapt).

Assuming that the sender has an appropriate application that it can use to send a message to multiple recipients, the receiver's infrastructure has to support the following requirements:

- The receiver's preferences need not be replicated on all devices available in a particular context space, but could reside at the receiver's trusted entity, which is preferably a stationary node (accessible over the Internet).
- This trusted entity should be updated when there is a relevant change in the context information of the receiver's current context space.
- Upon a context update, the relevant preferences should trigger communication adaptation. Depending upon these preferences, this could either affect a new incoming call/message or if a communication session is already established, then this context update could cause the trusted entity to trigger redirection of the session to a new device or even to a new communication means. The context update could also trigger applications to adapt (e.g. by getting information about the available networks via triggers, a device can initiate a handover to a new network, as described in [59]).

Figure 23 presents the resulting receiver's infrastructure which is capable of supporting the above requirements. We introduced a new context provider node (providing context information about the user, devices, network interfaces, and communication links), and a trusted entity node in the receiver infrastructure. This trusted entity node contains in turn one
or more communication dispatchers that are responsible for establishing, maintaining, and terminating sessions with a receiver, as well as delivering call/message/content using the receiver's preferred device and communication means. A context provider obtains context information from the receiver's devices in the receiver's current context space and provides this context information to the receiver's trusted entity. The trusted entity receives context updates from the context provider, manages the receiver's context dependent preferences regarding its preferred device and communication means, and finally invokes an appropriate communication dispatcher to adapt the communication to the receiver.

This model enables delivery of call/message/content to the receiver using the receiver's preferred communication means and preferred device(s). The communication dispatcher can adapt incoming communication based upon parameters explicitly set by a receiver. The adaptation can include filtering/rejecting unwanted messages/call/content. Today preferences are used to decide about the communication means and device which will be used for communication with the receiver, before the session is actually established. Existing preference formats such as CC/PP [27] and the UAProf (User Agent Profile) [28] for content adaptation, as well as CPL (Call Processing Language) [60] scripts for call logic are explicit, static, and have to be communicated to the system before communication is established. According to this model, a receiver would be able to specify whether he/she wants to receive an incoming call/message/content in a certain context space and if so how he/she wants it be delivered (e.g., as an instant message, RSS feed, a VoIP call, or via file sharing) and on which device (if multiple devices are available and applicable). I have previously extended CPL to support context parameters [1], but there is currently no way to modify these preferences based upon the user's current context, in order to trigger dynamic adaptation of the receiver's ongoing communication session (i.e., to transfer the existing session from one device to another (preferred) device and/or modify the session to utilize another (preferred) communication means). This limitation is due to the fact that CPL processing is only invoked at the start of a call. Note that this dynamic adaptation based upon context during a session is a very important requirement for our model (as illustrated in Figure 23), but is not supported by existing technology. The context will be inferred by the system from the context data acquired by the user's device(s). This context information can be used in
order to trigger adaptation for both new and existing communication based upon the receiver's preferences and current context.

The other requirement of this model is learning about the user's (as of yet unspecified) preferences in a specific context space by trying to deliver incoming call/message/content to the receiver's currently employed device and observing the receiver's feedback. Unfortunately, the possibility of a receiver's feedback is not even considered by existing technologies.

When we look at the model from the sender's perspective, a call/message has to be addressed and routed to a specific network address or communication identifier (e.g., IP address, phone number, e-mail address, etc.). As we have enabled the receiver to adapt their communication based upon their context, we would like the caller/sender to be able to specify call/message destinations based on the target users' context rather than simply their current network address. We will refer to this as context-addressed messaging or context-addressed session initiation (the first is for messages and the later for sessions). Note that there are two important classes of possible targets: (1) those who we know of and for which we want to add context as an additional qualifier to our decision of sending a message or initiating a session with and (2) those whom we do not know of – but whose context matches a target context that we specified in the destination address of a message or session initiation. In the next chapter we describe the concept of context-addressed messaging and describe how it might be realized.

3.3 Summary

In this chapter we presented our context-aware communication model. This model assumes that users use different devices in different situations (due to the device's specific processing and communication capabilities). Therefore, the preferences of users (here in the role of receivers) regarding the communication means (i.e., how they want to be contacted) and on which device (if they are multiple devices available) differ in different contexts. Our proposed context-aware communication model models the communication between a sender and a multiple receivers and enables the delivery of call/message/content to receivers using their preferred communication means and the preferred device in their current context. From this model we identified the requirements that the receiver's infrastructure need to fulfill in order to support such a model.

As the most important requirements we extracted the following:

- the need for dynamic adaptation of the communication based on the receiver's preferences in their current context (that also needs to take into account the sender's and the receiver's device capabilities);
- enable users to upload their context-dependent preferences at any time, even during the communication session;
- learning about the user's (as of yet unspecified) preferences in a specific context space by trying to deliver incoming call/message/content to the receiver's currently employed device and observing user's feedback;
- the need for trusted entity in the receiver's infrastructure that would know the receiver's preferences and would be updated when there is a relevant change in context information of a receiver;
- upon a context update, the relevant preferences should will trigger adaptation of both new and existing communication (i.e., affecting a new incoming call/message or if session is already established, causing redirection of the session to a new device or to a new communication means).
Finally, we presented the resulting receiver's infrastructure which is able of supporting the identified requirements.

Open issues that were not discussed in this chapter and that need to be addressed in the future work are:

- How to enable the user to easily express his/her context-dependent preferences regarding preferred communication means, device, and interested content? Should some of these preferences (such as the user's hobbies and free time activities) be imported and/or inferred from existing social networks, e.g., Facebook, MySpace, etc.?

- The user should be aware of context terms that are specified in the context model schema when he/she writes these preferences (in order to be context-dependent). The question that arises is how to disseminate the context model schema to users? Should this schema be part of the system delivery and can it be changed?

As we have enabled the receiver with this model to adapt their communication based upon their context, we would like to enable the caller/sender to specify call/message destinations based on the target users' context that will be routed to the correct recipients (i.e., support context-addressed messaging and context-addressed session initiation). In the next chapter we elaborate in more detail the requirements and design of system infrastructure for context-addressed messaging.
CHAPTER 4

CONTEXT-ADDRESSED MESSAGING

The aim of this chapter is to identify requirements for and design a context-addressed messaging system infrastructure, with mechanisms that support addressing and routing of context-addressed messages from sender to relevant recipients. In this chapter we also propose a novel context-addressed message format that uses an arbitrarily complex high-level context as target destination of this message. We also analyze types of application-level communication and investigate whether they can be used to deliver context-addressed messages. It is shown in this chapter how the result of this investigation has driven the design of our context-addressed messaging mechanism. This chapter also describes and analyzes the relevant related work in this area, while comparing each reviewed system with our design of context-addressed messaging system infrastructure. Finally, we illustrate how to implement context-addressed messaging on top of SIP network infrastructure.

4.1 Introduction

Context-addressed messaging refers to addressing and delivering a message to a receiver based on the receiver's context rather than based on the network address of a specific device. Context-based addressing enables context-aware message delivery. Context-based addressing begins by deriving which destination should be the target of the communication based on the context of the receiver as specified in a context destination – while considering the context of all of the potential recipients. As noted in the previous chapter, the set of potential recipients could be known in advance and the context address is used to select a subset of them or the set of potential recipients might not be known in advance (hence Context-aware message delivery must determine whether and/or how the message should be delivered to a recipient in his/her current context. This process also includes adaptation of the message to the recipient's preferred format and delivery of the message to the recipient's preferred device in the recipient's current context.

4.2 Requirements

The requirements for context-addressed messaging are: relevant message delivery to the recipient(s), timeliness, privacy, and scalability. Relevant message delivery refers to delivery of messages that are of interest to the user in his/her current context and that are delivered to him/her according to the recipient's preferences (i.e., using the preferred communication means and on the preferred terminal(s) in the current context). This requirement assumes that message can potentially be delivered to at least one recipient, that the potential recipient(s) has/have previously expressed interest in receiving messages on a particular topic, and that because the senders and recipient(s) are decoupled in space and time that messages will be delivered in an asynchronous way to the recipients. Timeliness is important for context-aware message delivery because the delivery process has to be fast enough for messages to reach recipients while the contents of the message are still relevant. User privacy is a crucial factor in context-addressed messaging, because users may be sensitive to revealing their personal information to others. Therefore, in order to gain better user acceptance, it is very important to respect each user's privacy when designing such a system. Consequently, because context information is often based on the user's private information, context-addressed messages should not be examined or modified by network infrastructure nodes when traversing physical links owned by ISPs or phone companies on their way to their destination. Finally, it is important for a system to scale with an increasing number of recipients. This is a
challenging task because a message not only has to match the recipient's current context (that can be highly volatile), but also has to be relevant to the recipient and be delivered according to the recipient's preferences in their current context.

In the following sections we give an overview of types of application level communication and investigate whether existing message delivery modes could be used to deliver a context-addressed message from a sender to the correct recipient(s).

4.3 Application-level communication types

There are two types of application-level communication: synchronous and asynchronous. These methods are defined by what the sender does once it stops sending data to the receiver. In synchronous communication, the Sender blocks its execution until the receiver has processed the message and the sender has received a reply. In asynchronous communication, the sender continues its execution immediately after sending a message.

Next, we characterize communication on the application-layer based on which communication entity initiates the data transfer: push (data transfer is initiated by a sender) and pull (data transfer initiated by a receiver). Thus, sender-push is asynchronous while the receiver-pull can be synchronous or asynchronous (i.e., polling or request/response) [61].

In sender-push communication, the sender needs to know the address of the receiver before sending the message to the receiver. The sender initiates transfer of message, which will subsequently be received by the receiver's device. After receiving an entire message, the receiver's device will inspect it and decide if it will discard the message or deliver this message to the receiver application (depending on the receiver's rules as set in the application program). Examples of sender-push communication are SMTP (Simple Mail Transfer Protocol)-based e-mail delivery system, mobile text messaging (SMS, EMS), and voice calls over the telephone network (both traditional and IP-based).

In receiver-pull communication, the receiver initiates a message transfer by contacting the sender. Although the receiver initiates the message transfer, the sender needs to store the content until the receiver is ready to retrieve it or the sender needs to be able to generate the content on-demand when the receiver requests it. HTTP-based web access and FTP-based file transfer are typical receiver-pull communication examples.

There are also two variations of these patterns of communications, which are both asynchronous: sender-intent-based-receiver-pull (SIRP) and receiver-intent-based-sender-push (RISP) [61]. In SIRP, the sender first expresses its intent to send a message to receiver; then if receiver is interested, it will contact this sender and retrieve the message. MMS (Multimedia Messaging Service) is an example of SIRP communication, where the MMS Center provides a notification to the receiver's (mobile) device, indicating that there is a message waiting to be retrieved. When the receiver receives this notification, the message reference is used in the notification to reject or retrieve the message, either manually or automatically, depending on the operator's configuration and the receiver's user profile. Therefore, the MMS Center can be seen as a sender and the receiver's device as the recipient of the SIRP communication. If the receiver is interested in retrieving this message, then it can request this message. In RISP, it is the receiver that expresses its interest to receive specific content, then the sender sends this content to the receiver. A typical example of the RISP model is a publish/subscribe model, where potential recipients (i.e., subscribers) subscribe to a service to receive notifications when specific content is published by a sender (i.e., publisher). The SIRP model gives a receiver greater freedom (than does RISP) to control which traffic it wants to receive and when it wants it to be delivered. However, in SIRP every receiver needs to contact a sender to retrieve the message, so 1:N communication is not possible, whereas in RISP this is not the case.
To deliver data from the sender to the receivers, different transmission methods could be used depending on whether the sender knows about the set of recipients before sending the message. When the sender does not know the recipients in advance (as in case of RISP), the system could utilize network-layer multicast or broadcast to transmit the message (the later can be used only if recipients are on the same subnet). When the sender knows in advance about the recipients (such as in SIRP, sender-push, and receiver-pull), the system can employ unicast to send a message to each of the receivers.

4.4 Design of context-addressed messaging system infrastructure

In this section, we focus on the design of a context-addressed messaging infrastructure, which utilizes context information for addressing and routing of messages to relevant recipients. In context addressed messaging, we replace a static network identifier with a specific target context as a destination address for this message. In the case of an incoming call, the user's context can be used to decide if the call will be accepted, rejected, proxied, or redirected to a third party. Additionally, because of the requirement for context privacy (specified in Section 4.2) most of the processing and routing functionality needs to be handled within the user's own infrastructure (which is in contrast to the traditional operator's model where the operator sits in between all parties and controls the data and signaling traffic). Note that the user's infrastructure is placed between the user and the rest of the network (i.e., as a proxy). This proxy structure becomes a mandatory requirement when designing such a communication system as it is essential to preserve the user's integrity.

Let us consider the following scenario: Alice is interested in receiving local traffic information when commuting to her home or work by car (i.e., the relevant context is her current location). Alice prefers to receive this information only from people that are at the time of publishing this content also located in the same area. At Sergels torg (a major traffic exchange), Bob notices a growing traffic jam and wants to notify people that are driving towards this area about it. Therefore, he uses a speech to text converter to compose a message about the traffic jam (or he might compose a speech message) and sends it to all people that are driving toward and are located within 5 km of Sergels torg. In order for this message to be delivered to Alice, Alice's current activity needs to be driving, her current location needs to be within Bob's specified range, and she needs to be driving towards Sergels torg.

This scenario illustrates several potential requirements for context-aware message delivery (i.e., whether and/or how the message should be delivered to the recipients). In such a system we envisage a message sender and multiple potential receivers (in this scenario Alice might be the only potential receiver). The sender has to be able to send a message with context restrictions to address only the interested recipients (i.e., those recipients whose current context matches these restrictions). Receivers should be able to express their preferences regarding the content they are interested in receiving, as well as how they would like to receive it (i.e., on which device and using what communication means) in different contexts. Finally, this context specification should match the recipient's context before the message is delivered (to the relevant recipient(s)). The requirements for context-aware message delivery and their mappings to existing message delivery modes are presented in Table 2. Note that these requirements have been covered in Section 4.2 that describes all the requirements for context-addressed messaging. However, in this Section we map the requirements for context-aware message delivery to existing message delivery modes in order to check if the existing delivery modes can be used to deliver context-addressed messages or not.

From this table, it can be seen that none of the existing message delivery modes completely satisfies these requirements: Receiver-pull and SIRP do not support 1:N communication; Sender-push does not allow the receiver to control the message delivery, and
RISP allows a receiver only to express interest in the content he/she would like to receive, but not when it will be delivered; Receiver-pull does not support asynchronous communication and SIRP is actually pull-based after receiving the sender's intent to send content. Of these four alternatives, RISP (i.e., publish/subscribe system) seems to be the best candidate for delivery of context-addressed messages; however, it has to be extended to fully support the specified requirements with regard to the receiver's control of message delivery.

![Table 2: Mapping of context-aware message delivery requirements to existing message delivery modes](image)

<table>
<thead>
<tr>
<th>Context-addressed message requirements</th>
<th>Message delivery modes</th>
<th>Sender-push</th>
<th>Receiver-pull</th>
<th>SIRP</th>
<th>RISP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:N communication</td>
<td></td>
<td>+</td>
<td>X</td>
<td>X</td>
<td>+</td>
</tr>
<tr>
<td>Receiver's control over message delivery</td>
<td></td>
<td>X</td>
<td>+</td>
<td>+</td>
<td>+/-</td>
</tr>
<tr>
<td>Asynchronous model</td>
<td></td>
<td>+</td>
<td>X</td>
<td>+/-</td>
<td>+</td>
</tr>
</tbody>
</table>

Note: X indicates does not fulfill, +/- partially fulfills, and + fully fulfills a requirement

Therefore, we have extended the publish/subscribe mechanism with context-based filtering at the receiver's trusted entity (i.e., the receiver's proxy) in order to deliver only the relevant message/content to the receiver's preferred device(s). Such a modified mechanism for context addressed messaging is shown in Figure 24.

### 4.4.1 Context-addressed messaging mechanism

The steps performed by this context-addressed messaging mechanism include the following:

1. A potential receiver uploads (using some device) his/her context-dependent preferences (regarding the content he/she is interested to receive, along with the preferred device and communication means to receive this content in different contexts) to the receiver's trusted entity.
2. The receiver's trusted entity extracts the receiver's preferences for different topics and the receiver's context parameters used in preferences regarding their desired content, as well as subscribes to these context updates at the receiver's context provider.
3. When any of the receiver's context parameters change value, an update is sent to the receiver's trusted entity. Note that the subscription to a particular context parameter will trigger sending of an immediate context update and further updates will be triggered each time when the context value changes.
4. The received context update triggers a new preference regarding preferred topics at the receiver's trusted entity, which subsequently triggers a subscribe action relevant to this topic.
5. After some time, a sender decides to send a context-addressed message on this topic to his/her trusted entity.
6. The sender's trusted entity adds the sender's anonymous address to the message.
7. The sender's trusted entity publishes the message on the indicated topic.
8. The broker sends notifications to receivers who have subscribed to this topic.
9. The receiver's trusted entity performs context synthesis (as described in Section 2.3.3) in order to compute a receiver's higher-level context and determine if he/she is the correct message recipient. Next, the receiver's trusted proxy performs context-based filtering to determine if this message is relevant for the recipient in their current context.

10. If the message is determined to be relevant, then the receiver's trusted entity uses the receiver's preferences to decide how this message should be delivered (i.e., on which device and using what communication means). If the format required for message delivery is understood by the receiver's preferred device (e.g., IM), the message is forwarded directly to the user's preferred device. Otherwise, the communication dispatcher responsible for adapting this kind of communication is invoked by the receiver's trusted entity to adapt the content to the receiver's most suitable current device and communication means.
11. The communication dispatcher adapts the message according to the receiver's specified (preferred) communication means, establishes the session with the receiver's device, and ensures the delivery of this message to the receiver's device. However, in case of SMS, the message is forwarded to a suitable SMS-C for subsequent delivery.

After a session has been established between the receiver's device and the correct communication dispatcher, new context updates can trigger new sessions and communication adaptation (e.g., in case of insufficient bandwidth or new preferred device). However, these details have not been shown in Figure 24 to keep the figure simple.

In order to explain how context-based filtering is performed, we need to first illustrate what the context-addressed message looks like as well as how receivers specify their preferences. In order to answer these questions, let us return to our scenario of Bob sending a message to all (interested) people driving towards and within 5 kilometers of Sergels torg. The message that Bob composes (as shown in Figure 25) will have a subject, it will be published on a specific traffic topic (written in the destination field), and will also include the following context-based address: "all people driving toward and within 5 kilometers of Sergels torg" as part of the message body.

Bob
To: context-based address
"All people driving toward and within 5 km from Sergels torg interested in traffic info"
Subject: traffic jam at Sergels torg
Body: There is a developing traffic jam at Sergels torg!

Figure 25: The context-addressed message created by Bob

Note that the context parameters and values specified in the address need to match Alice's current context in order for this message to be delivered to her. Therefore, Alice subscribes to the traffic information channel stating preferences to receive only the traffic notifications relevant to her current or near future location [62]. Figure 26 illustrates this Alice's subscription.

Alice
Subscribe to: traffic info
Current context: location=Kungstradgården, activity=driving
Preferences: current location based traffic info

Figure 26: Alice's subscription to the traffic information relevant to her current or near future location

Note that both example messages were written in pseudocode. The actual format of the context-addressed message along with a scheme for context-based address URI is described in the following subsection. Next we elaborate how this address is resolved at the receiver's proxy and finally how the context-based matching is performed in order to decide whether this message will be delivered to an intended recipient (i.e., Alice).
4.4.2 Common Profile for Context-Addressed Messaging (CPCAM): Message format

We define a new MIME content type "Message/CPCAM", a message format for protocols that conforms to a Common Profile for Context-Addressed Messaging (CPCAM). We base the CPCAM specification on Common Presence and Instant Messaging (CPIM) as defined in RFC 3862 [63]. Although our implementation of context-addressed messaging is based on SIP, our intent is that this message format should be general enough to be reused by other applications and protocols that are CPCAM-compliant.

Complying with CPIM standard, the CPCAM message format encapsulates arbitrary MIME message content, together with message and content-related metadata. This content can be signed or encrypted using MIME security multiparts according to an appropriate security scheme. The MIME content headers have to include at least a Content-Type header. The content of the context-addressed message can be any MIME type.

A message/CPCAM object consists of a message headers and message content. Message headers carry information needed for an inner-routing at the receiver's trusted entity, i.e., to decide whether this message is relevant to the recipient and if so, to deliver this message to this recipient. (For details of inner routing see Section 4.4.4). Therefore, message headers should not be modified, reformatted, or reordered in transit from sender to receiver (i.e., it's trusted entity). Message headers have a similar syntax and purpose as in an e-mail message format, see RFC 2822 [64]. However, we define our own URI scheme syntax for addressing recipient(s), using the Augmented Backus-Naur Form (ABNF)¹.

The ABNF of our context-based address URI (i.e., so called CAM-URI) is:

```
CAM-URI = "cam:" [operators]
operators = 1* (namespace "." operator)
operator = operator-name (" [output-type] ", [input-types] ")
namespace = absoluteURI
output-type = namespace "." (operator / context-class)
input-types = input-type * ("," input-type)
input-type = namespace "." (operator / context-class)
operator-name = alpha
context-class = alpha
```

Here the symbol "absoluteURI" represents an encoded absolute URI as defined in RFC 2396 [65], and the symbol "alpha" denotes any character from the basic Latin alphabet, including upper and lower cases (also defined in [65]). Note that CAM URIs always start with the "cam:" prefix. Use of the cam: URI follows closely the usage of the mailto: URI. That is, invocation of a CAM URI will cause the user's context-addressed messaging application to start, with a destination address and message headers fill-in according to the information supplied in the URI.

Our concept of CAM-URI is based on context operators used for context synthesis, defined in Section 2.3.3. The motivation for using context operators to define context addresses is based on our previous work of using operators for context synthesis [2][3]. Operators provide an easy to comprehend means of specifying operations that are based on an input data (set) producing a desired output type, which does not need to exist in a context model schema. Only basic vocabulary terms that are common to a domain need to be defined (i.e., roles of users, places encountered during an event, and some abstract entities, such as groups, teams, etc.), on top of these all other operations can be performed. Operators are

¹ Note that the ABNF is used in Internet technical specifications to define format syntax [66].
extensible, reusable, and allow their users to provide their own implementations of an operator's function. Different implementations of an operator enable flexibility in type matching of sender's provided operator's inputs and a required output against the input and output types produced by different implementations of this operator, because if an exact matching operator's implementation cannot be found, a more generic one could be used instead. Using these operators it is possible to define any target high-level context as an address of a message that can be resolved at each of the potential receiver's trusted entity.

A context-addressed message from our earlier example encoded in CPCAM looks like:

Content-type: Message/CPCAM
From: Bob <sip:bob@example.com>
To: cam:operator1.And(operator2.DrivingTo(context.User,
    context.Location="Sergels torg"), operator3.InRange(context.User,
    context.Location="Sergels torg", context.Range="5000"))
DateTime: 2009-01-24T21:40:00+01:00
Subject: traffic jam at Sergels torg
NS: operator1 <http://www.example.com/models/operators/And/And.man>,
    operator2 <http://www.example.com/models/operators/DrivingTo/DrivingTo.man>,
    operator3 <http://www.example.com/models/operators/InRange/InRange.man>,
    context <http://www.example.com/models/context.man>
Content-type: text/plain; charset=utf-8
Content-ID: <12345667890@example.com>

There is a developing traffic jam at Sergels torg!

Note that the "NS" header field enumerates the namespaces of the generic operators that are used in the CAM URI. This message is encapsulated in a SIP PUBLISH message and sent to the traffic topic's Address of Record (AoR), with the Content-Type header set to "message/CPCAM". Therefore, a complete message looks like:

PUBLISH sip:traffic@example.com SIP/2.0
Via: SIP/2.0/UDP bob.example.com;branch=z9hG4bK652hsge
From: <sip:bob@example.com>;tag=1234wxyz
Call-ID: 81818181@pua.example.com
CSeq: 1 PUBLISH
Max-Forwards: 70
Expires: 3600
Event: presence
Content-Type: message/CPCAM
Content-Length: ...

From: Bob <sip:bob@example.com>
To: cam:operator1.And(operator2.DrivingTo(context.User,
    context.Location="Sergels torg"), operator3.InRange(context.User,
    context.Location="Sergels torg", context.Range="5000"))
DateTime: 2009-01-24T21:40:00+01:00
Subject: traffic jam at Sergels torg
NS: operator1 <http://www.example.com/models/operators/And/And.man>,
    operator2 <http://www.example.com/models/operators/DrivingTo/DrivingTo.man>,
    operator3 <http://www.example.com/models/operators/InRange/InRange.man>,
    context <http://www.example.com/models/context.man>
Content-type: text/plain; charset=utf-8
Content-ID: <12345667890@example.com>
There is a developing traffic jam at Sergels torg!

After reaching Bob's SIP trusted entity (which is implemented as a SIP proxy server), it replaces Bob's actual identity with the pseudonym URI, and inserts its own URI into the Via header of the PUBLISH message, in order to identify replies to this message.

### 4.4.3 Address resolution

After retrieving a SIP NOTIFY containing a context-addressed message in its message body, the receiver's proxy needs to perform operator matching in order to find the appropriate specialized operator(s) that will compute the high-level context of the receiver and determine if this receiver should receive this message. We envisage that providers of a context-addressed messaging service will be responsible for providing generic and specialized operators as part of their service offering to their users. Additional updates of specialized operators could be made available to users through a web site, whose structure reflects the file structure of the Operator space, illustrated in Figure 11 on page 30. Thus, if the receiver's proxy does not have the exact matching specialized operator, it can either replace it with a more generic one (if any), or download the exact one from this web site. The path for downloading specialized operators would be constructed by taking this operator's namespace, removing the generic operator name and ".man" extension, and replacing them with the specialized operator name (with the ".bsh" extension).

The following example illustrates the path for downloading the specialized operator "UsersDrivingTo":

```
http://www.example.com/models/operators/DrivingTo/UsersDrivingTo.bsh
```

### 4.4.4 Context-based filtering

In order to determine if Alice should receive a context-addressed message sent by Bob, the context specified in the context-based address needs to be matched against Alice's current context (i.e., if Alice's current activity is driving and her current location matches Bob's specified location and range). This context-based matching is illustrated in Figure 27.

![Figure 27: Context-based matching in case of context addressed message arrival](image)

After determining that Alice is a suitable message recipient, Alice's proxy needs to check if this message is actually relevant to Alice and if so, it may customize this message delivery according to her preferences in her current context. This includes filtering out unwanted messages, delivering relevant messages via her preferred communication means on her preferred device, and/or learning new previously unspecified preferences. For the last function, we continue to consider the Alice & Bob scenario. In some other situations, such as when planning travel by car or a plane, Alice might want to retrieve traffic information related to specific places of interest (such as different cities that she plans to visit), or she might forget to set her preferences for some topics. In the latter case, the system should try to deliver messages to Alice as soon as they become available, and incrementally learn her
preferences according to her observed behavior — with the assistance of her feedback. Alternatively, Alice could specify the situations in which she does not want to receive traffic information (for example, based on time of the day, during particular activities, based on the sender's role, etc.). This complete process from receiving a context-addressed message until delivery of this message to the receiver's device is *Context-based filtering*, and is illustrated in Figure 28. Note that first action is the same as shown in Figure 27. Figure 28 shows the data used as input to different phases of this process, as well as the feedback due to the newly learned preferences to the updated context, preferences, and communication adaptation.

![Figure 28: Context-based filtering](image)

Note that by utilizing this context-addressed messaging mechanism between the communication parties' trusted entities instead of directly between the sender's and receiver's device, we allowed the context-based filtering to determine if the message is relevant to the receiver before delivering it to his/her device. This context-based filtering in turn enables the routing of messages within the user's infrastructure, thus protecting the receiver's privacy, which was a mandatory requirement (indicated in Section 4.2). Performing this context-based filtering at the receiver's side instead of making routing decisions for context-addressed messages at the broker also increases the scalability of the system, because this filtering is performed for each recipient by his/her trusted entity. To protect the sender's identity and improve his/her trust in the system, the sender's trusted entity could implement an anonymizer proxy functionality [67][68] to send the message using a pseudonym temporally assigned to the sender. Finally, by inserting a context-based address in the header of the message and utilizing context matching (as shown in Figure 28) to find suitable recipients for the context-addressed message (the so called inner routing) we achieve the same functionality as if context-based routing was performed in the network infrastructure (i.e., the outer routing).
4.4.5  **Context-addressed messaging system architecture**

A detailed view of the sender's, network, and the receiver's infrastructure for realizing our proposed concept of context-addressed messaging are depicted in Figure 29 and Figure 30. We can observe that different functions are performed by the trusted entity on the sender's and on the receiver's side.

![Diagram of sender's infrastructure for context-addressed messaging](image)

**Figure 29: Sender's infrastructure for context-addressed messaging**

A sender composes a context-addressed message using the application on his/her mobile device and publishes it to a topic. We assume that this application has a graphical user interface that helps a user to compose such a context-based address. Additionally, access to (generic) operators for constructing a context-based address is provided to an application through the `GenericOperator` class (see Section 2.3.4 on page 34). A published context-addressed message is intercepted by the sender's trusted entity that consists of an anonymizer, message publisher, and the reply message handler. Upon receiving a message, the anonymizer inserts a pseudonym (i.e., a URI pointing to the anonymizer proxy) instead of sender's real communication address into the message in order to hide the sender's identity from the network infrastructure, then sends this message to the message publisher, which publishes the message to the broker. Note that anonymizer maintains multiple pseudonyms per user, which are agreed between the user and the anonymizer beforehand. A pseudonym is temporarily assigned to a user by the anonymizer for communication with untrusted parties. The reply message handler handles reply messages from a receiver.
On the receiver's side, the trusted entity performs the following tasks: (1) manages the receiver's context-dependent preferences regarding the message relevance & delivery; (2) subscribes to the receiver's preferred topic(s) when the receiver's context changes; (3) performs operator matching (as specified in Section 2.3.3.4) when a message arrives and computes the receiver's high level context; (4) uses this computed receiver's context to perform context-based filtering; and (5) invokes the responsible communication dispatcher to adapt the communication according to receiver's preferences in their current context as well as to ensure the message delivery in the receiver's preferred format and on his/her preferred device.

Before elaborating about the network infrastructure, it is important to state what kind of device configurations end users can use, along with software components that need to be deployed on these devices. In Section 2.4 we have designed a context-aware system with applications, middleware, context plug-ins, context providers, and context distribution (see Figure 14). However, we did not say anything about deployment of different configurations of this system that end users can have on their devices. Therefore, Figure 31 illustrates an application and sensor device configuration with appropriate components. By application device we refer to a device that runs applications which consume context produced by sensors. A sensor device is a device that runs sensor applications or is able to communicate with hardware sensors (as illustrated in Figure 31). Note that components on sensor device can optionally be present in an application device, but the purpose of Figure 31 is to distinguish between components used to search for and synthesize desired context (deployed on application device) and components used to register and provide context to interested
applications on remote devices in the network (deployed on sensor device). Details about these components functionality can be read in Section 2.4.

![Diagram of application and sensor device configurations with appropriate components]

**Figure 31: Application and sensor device configurations with appropriate components**

In case of sender's and receiver's infrastructure illustrated in Figure 29 and Figure 30, the sender's device should have an application device configuration, whereas the receiver could employ one or more sensor devices to provide context to its context provider. The receiver's application device configuration should be deployed on its trusted entity.

4.5 **Related work on context-addressed messaging**

In this section we review the related work on context-addressed messaging and compare each of the systems with our design of our context-addressed messaging infrastructure. The systems reviewed in this section can be categorized into the following groups based on their approach/technology used to implement context-addressed messaging: (1) distributed location infrastructure, (2) content-based publish/subscribe mechanisms, (3) variations of multicast (such as Xcast or Geocast – see Section 4.5.3), (4) use of restricted flooding (such as narrowcast) & ontology-based reasoning, (5) use of similarity-based profile matching (i.e., Profile-Cast), and (6) preference rule-based reasoning. We will discuss these related works in terms of the requirements specified in previous section.

4.5.1 **Distributed location-based infrastructure**

4.5.1.1 **Spreitzer and Theimer**

The first mention of context-addressed messaging appeared in Spreitzer and Theimer's 1993 paper [69], in which they proposed a note distribution application to send a message to all persons at a given location or a set of locations, as well as a Ubiquitous Message Delivery (UMD) application for delivering a message at the soonest "acceptable" time via the most "appropriate" terminal near the recipient. An acceptable delivery time is a function of the recipient's context (e.g., the recipient's profile can specify that messages below a specific priority level should not be delivered when the recipient is in a meeting). Similarly, the most appropriate terminal depends on the available devices at the recipient's current location, as well as the terminal's characteristics.

The architecture of their location infrastructure is built around User Agents that manage the user's personal information, and a partially decentralized Location Query Service to facilitate location queries, as shown in Figure 32.
A User Agent is a program running on one or more computers in the network, which communicates via remote procedure calls (RPC) with other programs. There is one User Agent for each user in the system. Its responsibilities are to collect all personal information about the user from sources and to provide applications with policy-based controlled access to this information. This information includes knowledge about the user's environment and context as well as his/her preferences with respect to current circumstances. This information is propagated to various applications. These User Agents enable applications to query for a particular user's location. However, in order to find out which users are at a particular location, applications need to use a Location Query Service (LQS) to execute location queries. This LQS gives User Agents control over revealing each user's identity information. The LQS is built around the concept of located objects, which are represented by a tuple consisting of a location, an RPC handle, and an association list describing the object's type (users – represented by User Agents or terminals – represented by Terminal Agents) and other information that an object wants to make available. A query is a predicate over a location and an association list, resulting in the set of tuples that satisfy the predicate. A key idea of the LQS is that located objects can be anonymous by putting an anonymous RPC handle and no identity in the association list. Thus, clients issuing the query would use this anonymous RPC handle to ask the object (i.e., a User Agent) for its identity. The object, depending on its policy, can respond truthfully, falsely, or not at all. The LQS is organized into regions, with a centralized server, called a Location Broker, running in each region. Located objects can register a full description of themselves with their regional Location Broker if they do not wish their identity and locations to be secret. Otherwise they register themselves in an anonymous fashion.

The User Agent keeps track of which terminal the user is currently using as well as what "public" terminals and other people are near the user's current location (the latter is achieved by registering callbacks with the LQS for the user's current location). Some applications, such
as UMD directly interact with User Agents, passing a message to User Agents to forward to the recipients; each agent then delivers this message to its user via a Terminal Agent. There is one Terminal Agent for each terminal. This agent manages information and controls access to the terminal. It exports the terminal characteristics to the Location Broker in an association list. Non-mobile terminals register with the Location Broker, so that they can be found by location. A mobile terminal is dedicated to a particular user and communicates directly with this user's User Agent. When the message is submitted to the User Agent for delivery to the recipient, the user agent checks the user's current situation to determine if this context allows the delivery of this message and if a suitable terminal is available. Otherwise it waits until the user's context changes, then tries again.

Their User agent corresponds to our context provider's and receiver's trusted entity's functionality for managing receiver's context, context-based filtering of received messages, and selecting the most appropriate terminal for message delivery. Their User agent also provides policy controlled access to the receiver's location information. Their Location Broker corresponds to our Broker component in our proposed network infrastructure, as it registers all User and Terminal Agents with their locations and identity information. Their Location Broker allows anonymous registrations of objects, in order to hide the real identities of users and their terminals. In our design we implement anonymizer functionality via pseudonyms (these correspond to their anonymous RPC handles). Note that in their system (shown in Figure 32) applications can directly access the target User Agents, but there is no interaction of applications with the source User Agents, unlike in our design, where applications' messages need to first access the sender's trusted entity, then these messages are propagated to the broker in the network, and finally the messages arrive at the recipient's trusted entity, which determines whether and how the message will be delivered to the recipient. In our design a sender can use any high-level context to specify an address of the message and publish this message to a particular topic (without first finding out which potential recipients have this target context). The correct message recipients will be determined by context-based filtering procedure performed at each receiver's trusted entity, after they have received the message notification. Unfortunately, these functions are not possible to achieve with their system design. Moreover, their system was designed to manage only location information, whereas our system handles all the context parameters specified in the context model schema and uses context operators for synthesizing high-level context.

Although their system design as depicted in Figure 32 scales well, it has some privacy risks, such as: (1) allowing eavesdropping on intermediary links between the location sensing systems and the User Agent, between the user's device and the User Agent, between the User Agent and the Terminal Agent, as well as between the User Agent and Location Broker (if this communication is not secured); (2) any querier can obtain the location information which is directly published to the LQS, even if the Location Broker implements policy-based access control - the broker might poorly implement this access control; (3) traffic analysis of LQS queries or results can reveal the identity of otherwise anonymous queriers or objects in the LQS; or (4) the various location sensing systems and the Location Query Service might deliberately give this information to other parties. Therefore, it is important that the user's device establishes a trust relationship with its User and Terminal Agent, as well as between the User and Terminal Agent (i.e., between the device, the proxy, and context provider in our design) and all communication between entities should be encrypted. Unfortunately, this establishment of trust was not discussed in [69]. Additionally, Spreitzer and Theimer did not elaborate on the heuristics they use to specify an "acceptable" time or the "appropriate" terminal(s). Instead of unknown heuristics these are explicitly encoded as context-dependent preferences in our system. Additionally, we did not find any performance results regarding
UMD, thus are unable to evaluate if this distribution application actually ensures timely message delivery.

Because our infrastructure will be implemented using SIP and SIMPLE, the problem of user and terminal mobility are inherently solved (see [57]); this mitigates the need for the User Agent (i.e., the proxy in our design) to keep track of the user's currently used terminal as our proxy relies on SIP user agents registering with their registrar. Note that one can argue that the functionality is in fact similar, but the advantage of using SIP is that there is a large base of hardware and software that already support this operation – thus we do not have to introduce this functionality ourselves.

4.5.1.2 Dey et al.

In 2002, Dey et al. [70] developed a "Context-Aware Mailing List" application to deliver e-mail messages to members of a research group who are currently in the building. This application uses a location widget from their Context Toolkit [70] to obtain a list of people at a particular location, adds these people to a mailing list, and sends them an e-mail message. This example of dynamically composing a mailing list illustrates the basic concept of context-based addressing. However, their application design poses a number of privacy problems, such as the problems caused by querying for the location of people without any access policy, thus any querier is able to get this information. Additionally, because there is no filtering at the recipient's side of e-mails sent to this mailing list based on the topic or the sender's identifier, and there is no prioritization of messages based on the recipient's current context, thus a malicious sender could send spam e-mail messages\(^2\). The timeliness of this approach is good, as it depends directly on the presence detection time when using Dallas Semiconductor iButton [71] readers (the read time between 60 and 240\(\mu\)s plus the time before a user touches the iButton to a reader) or radio frequency-based Pinpoint [72] 3D-iD indoor RF based positioning system (used in real-time positioning). The scalability of the system mainly depends on the density of deployed readers (and the user's pattern of using them) and the number of people in the building.

4.5.2 Content-based publish/subscribe

In 2000, Carzaniga et al. [73] proposed a model for content-based addressing and routing, and implemented it on top of an event notification service, which enables receivers to express their interest in receiving messages by specifying predicates to select particular content, independently of the sender of the message. Senders generate messages without specifying a specific destination. Receivers receive these messages based upon their own interests, which are matched against the content of messages travelling in the network. This approach is different from the traditional multicast, because there is no predefined group identity and the intended destinations are implicit (i.e. defined in the **content** of messages). Thus, there may be zero, one, or multiple receivers of a certain message (because of the receiver(s) expressed interest in the specific content). The event notification service used by this content-based addressing and routing could be seen as a generalization of a multicast service. However, their approach to content-based routing does not give any guarantees regarding reliability, security, or performance. The trade-off between the expressivity (in their data model and subscription language) and scalability is hard to balance.

In general, in order to deliver a message to N recipients, one can choose to utilize multicasting or a publish/subscribe paradigm. A key issue is the suitability of each of these approaches for context-addressed messaging. With multicasting there are two opposite strategies [73]: either to define many specific multicast groups or a few generic ones in order

\(^2\) Note that the lack of filtering at the recipient and the problems of SPAM are endemic to e-mail – and remain an issue even today.
to define context addresses. In the former case, receivers can join the groups based upon their interest with high accuracy, but senders may need to send the message to multiple groups, whenever the message is relevant to multiple specific multicast addresses. If receivers' interests change frequently, this would lead to highly dynamic restructuring of groups, and we need to verify if the multicast infrastructure is able to efficiently deliver the messages. To quantify this, we will calculate the time needed to rebuild a multicast distribution tree and relate it to receiver's rate of rejoining different multicast groups.

For calculation of time to build a distribution tree for multicast routing in the network, we will use the Protocol Independent Multicast (PIM) protocol [74], because it is currently used by most IP routers. Considering PIM Sparse Mode (PIM-SM) as a multicast routing protocol [75], we express the time to build a multicast distribution tree from a source to a receiver as:

$$T_{n,x}(\text{shared tree})=(n+x)\times t$$

(3)

where \(x\) is a number of hops from receiver to the router acting as a Rendezvous Point (RP), \(n\) is a number of hops between the source and the receiver on the path that includes the RP, and \(t\) is the average transmission time per hop in LAN (for this discussion we will assume that \(t=4\mu s\)). Please refer to Appendix for details about the protocol and how we obtained this equation and the average transmission time per hop.

Optionally, when the amount of data to be sent to receivers exceeds a threshold, then routers can switch to a source tree. In this case we express the time to build a distribution tree as:

$$T_{n,x}(\text{switch to source tree})=(2n'+x-3)\times t$$

(4)

where \(n'\) represents a number of hops from the receiver to a source directly (omitting the RP).

If we assume that \(n'\) equals \(n-1\), then:

$$T_{n,x}(\text{switch to source tree})=(2n+x-5)\times t$$

(5)

For \(n=5\) and \(x=3\), \(T_{n,x}(\text{shared tree})=32\mu s\) and \(T_{n,x}(\text{switch to source tree})=32\mu s\). For \(n=100\) and \(x=50\), \(T_{n,x}(\text{shared tree})=600\mu s\) and \(T_{n,x}(\text{switch to source tree})=980\mu s\). Note that \(T_{n,x}\) (shared tree) is the time needed by a receiver to join a multicast group and build a multicast distribution tree. From this time we can calculate the maximum rate of changing receiver's interest:

$$r_{\text{receiverInterest}}=1/T_{n,x}$$

(6)

which is in case of \(n=5\) and \(x=3\) equal to 31250 joins/sec and 1667 joins/sec for \(n=100\) and \(x=50\); when switching to the source tree the rate is 31250 joins/sec for \(n=5\) and \(x=3\) and 1020 joins/sec for \(n=100\) and \(x=50\). From this result, we can see that the maximum rate of changing receiver's interest decreases with an increasing number of hops between the source and the receiver. Additionally, if there are 100 receivers joining a multicast group, IP multicast is able to efficiently process up to approximately 17 re-joins/sec (in case of shared tree) and 10 re-joins/sec (in case of source tree), in case \(n=100\) and \(x=50\).

From these results we can conclude that packet transmission delays do not significantly influence the efficient delivery of multicast messages in case of frequent receiver's re-joins to multicast groups. In order to verify these numbers, we need to also take into account the time needed by a router for maintaining a multicast tree state or any router processing time (i.e., encapsulating/decapsulating packets, routing table lookups, etc.). We expect that with an increasing network size, there will be an increasing number of entries in the routing table and more incoming packets, so the processing time of the router will increase.

Alternatively, multicast could be used with a small number of generic groups, then senders would send information to one or more of a smaller number of groups, which would be
processed very efficiently by the multicast infrastructure; however, in this approach the receivers would need to receive and filter out a potentially large volume of irrelevant information.

A publish/subscribe approach allows receivers to subscribe for interesting information and to be notified when this information is published by senders. The broker/mediator routes information from publishers to interested subscribers. There are two types of publish/subscribe systems, providing two different strategies: (1) a topic-based approach, which allows a user to subscribe to predefined topics of interest, thus filtering out uninteresting information and (2) a content-based approach, which enables greater expressivity in defining subscriptions over the contents of the event by specifying filters using a subscription-based language.

The topic-based approach is a scalable approach, because interaction of publishers and subscribers is decoupled (publishers and subscribers do not know about each other, they do not need to be active at the same time, and subscriber is asynchronously notified about an event via a callback, thus this is not in the main flow control of publisher or subscriber) and a broker performs a simple process of matching a publication event with subscriptions; however, the publisher has to publish the event to all the specific topics which may be relevant to the event; regarding the privacy issues, although publishers and subscribers are anonymous to each other, the broker can learn their identities; additionally, if the sent and received subscriptions and publish messages are not encrypted, any intermediary component can inspect these messages, thus gaining some knowledge about the recipients.

Although a content-based approach enables greater expressivity in defining subscriptions over the contents of the event by specifying filters using a subscription-based language, this approach adds complexity when matching a publication event with a subscription, because it requires sophisticated protocols for event matching that have higher runtime overhead. As subscription filters have to be applied to each event sequentially, if there are many event publishers and event subscribers in the system, this decreases the system performance and scalability; however, note that this problem could be solved if the broker is able to compile a matching finite state machine (based upon all the subscriptions) that provides all of the matching and the destination selection logic, as described in [76]. Additionally, the content-based approach has the same privacy issues as the topic-based alternative.

Therefore, our approach for context-addressed messaging is realized as a topic-based publish/subscribe system, but we perform the context-based filtering at the receiver's trusted entity, based on the receiver's preferences in the receiver's current context, thus relieving the receiver from the burden of performing this filtering itself. Additionally, because the filtering is performed by the receiver's trusted entity – it is only this address that the sender learns; therefore if this entity also does filtering for many other receivers it becomes difficult to determine the interests of a specific user.

4.5.3 Variations of multicast
4.5.3.1 Xcast

In 2008, H. Lee, et al. proposed to use an explicit multi-unicast (Xcast) mechanism for context-aware messaging service [77]. Xcast [78] builds a multicast tree from source to multiple receivers without clients' explicitly requesting JOIN operations. Xcast efficiently solves the problem of multicast supporting a very large number of small multicast sessions. Instead of using multicast group addresses, Xcast explicitly encodes a list of destination addresses in the data packets. This eliminates per-session signaling and the per-session state information of traditional IP multicast schemes, as well as avoiding multicast address allocation, this allows Xcast to support very large numbers of small multicast groups.
However, it increases the header processing, because of number of routing table lookups for each of the addresses and a requirement to reconstruct the packet header for each next hop. Therefore it is not suitable for multicast sessions with a large number of members. Additionally, in Xcast the source node keeps track of the destinations of the multicast channel it wants to send packets to.

This method of context-addressed messaging requires some node in the network (i.e., the context server) to map context to a list of last-hop routers of the target nodes as proposed in [77], before the message is sent to the network. When context occurs, context server sends a ReportContext message to the multicast source (i.e., context-addressed messaging server (CAMS)), containing the context information and the addresses of last-hop routers. The Xcast router groups these routers' addresses by outgoing interfaces and for each interface, the data message is forwarded only to the last-hop routers pertaining to this specific outgoing interface. When there is only one destination left, the Xcast packet can be converted into a normal unicast packet, which can be unicasted along the remainder of the route. This is called X2U (Xcast to Unicast) conversion. Such a proposed scheme builds a multicast tree using top-down approach, because the CAMS knows the last hop routers of all the destinations. Thus, due to the top-down approach, the multicast tree is built when the first message is forwarded, which reduces latency and control packets. Therefore, the following data messages do not need to include the addresses of the last-hop routers due to the multicast tree information. Moreover, the use of last-hop routers reduces the number of individual destinations in the first data message's header. On receipt of each data message, the last-hop router broadcasts the packet on the subnet. The evaluation in [77] shows that the proposed mechanism has significantly lower overhead and latency than the traditional unicast and multicast mechanisms. However, this approach still lacks support for node mobility and it is not an Internet standard; because of the later it is not widely deployed. It also poses major privacy issues due to the context server which maps context values to last hop routers of the target individual nodes. This context information should be part of the user's infrastructure, and not stored in the network, in order to avoid it being misused. Additionally, control of messages, such as preference and context-based filtering of messages is not possible on the receiver's side.

Note that the similar idea to Xcast is RLS lists in SIP (see Section 4.6.2 for details about RLS lists), which allow one to subscribe to a list of users by explicitly encoding the list of their SIP URIs in the resource list and sending them notifications! As with the Xcast, users in the resource list do not need to explicitly join the multicast group. However, they have to give their consent to the creator of the resource list that they agree to be added to this list.

4.5.3.2 Geocast

In 2002, Y.-B. Ko and N. H. Vaidya designed Geocast [79], a variation of multicast that enables a sender to send a message to a group of people within a particular geographic area. Their approach, targeted at mobile ad hoc networks, defines a set of nodes within a specified area (i.e., a geocast region) as a geocast group. This geocast ( multicast) group is defined as the smallest rectangle covering the geocast region. There is also a forwarding zone, in which nodes forward the received packets to their neighbors in order for packets to reach their recipients in the geocast group. To increase the probability of a data packet reaching its recipients, the forwarding zone should include the geocast group. The authors propose three geocast algorithms ( variations of flooding algorithms which differ in how the forwarding zone is defined): a static zone scheme, an adaptive zone scheme, and an adaptive distance scheme. Their evaluation of these algorithms shows that the proposed adaptive algorithms have a lower message delivery overhead than geocast flooding and the accuracy of geocast delivery is comparable to geocast flooding. Regarding timeliness, 90% of packets were
successfully delivered to their geocast group members (in a network of 30 nodes with a pause time equals to zero) independently of the speed (ranging from 5m/s to 25 m/s) in the adaptive zone scheme and the adaptive distance scheme based upon the simulation.

The drawback of this approach is the lack of receiver control to receive only the messages of interest, because membership in the multicast group is implicit. As soon as nodes arrive in a particular geographic area, they automatically become members of a geocast group.

4.5.4 Similarity-based profile matching

In 2008, W.-J. Hsu, D. Dutta, and A. Helmy proposed a new service paradigm called Profile-Cast [80] for delivering messages to a group of users that have similar mobility behavior. This behavior is inferred from long term location traces, and is stored as the user's mobility profile. This profile contains a matrix of time slots indicating when the user was in particular locations. Each user maintains his/her own profile and exchanges it with others when nodes encounter each other, in order to determine whether a message should be forwarded to this newly encountered node. The profile matrix is transformed into an eigen-behavior vector in order to reduce the size of matrix that will be exchanged with other nodes. This vector describes the user's mobility in decreasing order of importance, with the relative weights computed as the ratio of the corresponding singular values. After two nodes exchange their profiles, a similarity index is calculated as the weighted sum of inner products of the eigen-behavior vectors. If the similarity index is larger than a threshold, then they exchange messages. Note that such a profile could contain information about user's interest(s), social affiliation, etc. rather than their mobility. This approach is similar to our context-based session initiation in that matching of one user's interests against other users' interests or current context initiates communication among them. Moreover, the information that is being exchanged (i.e., preferred locations with their weights) with other nodes can be directly manipulated by the user, thus the user can choose to provide only the desired subset of their context to others, in order to preserve the user's privacy. The benefit of this approach is that there is no explicit group membership to be maintained, thus reducing signaling overhead.

This approach is well suited for delay tolerant networks because it provides a way of navigating messages through the mobile society without relying on established infrastructure or registry, reaching the targeted groups defined by their underlying properties (i.e., the chosen profile). Thus, their message forwarding protocol limits the scope of message delivery in delay tolerant networks to a specific behavioral group, thus avoiding the high overhead of epidemic routing (i.e., it eliminates more than half of the transmissions with a little reduction in delivery success rate) and outperforming random-walk based protocols in terms of delivery delay. Performance-wise it shows a significant (45%) overhead reduction compared to flooding and 30% shorter delay as compared to a random transmission protocol [80]. However, the authors only represented mobility behavior, and no other context information or interests are included. In contrast in our approach interests not only change with time, but also based upon the user's current context. Therefore, it would be interesting to investigate the use of multidimensional matrixes to model context-dependent preferences. Moreover, their approach does not allow a receiver to explicitly express preferences regarding which messages he/she is interested in receiving. Thus, a receiver has no control over which messages he/she wants to receive nor in what format or on which device these messages should be received. In contrast, our preference matching is triggered by the sender's context update which selects a new preference that initiates a group query (sent to a list of sender's contacts belonging to the same social relationship group independently of their location) in order to find the potential receivers whose current interest or context matches the sender's. Note that our approach here relates to the context-based session initiation (regardless of the
communication means used by sender and receiver to communicate). This approach can also
be used to initiate communication between senders and receivers that are not collocated.
Whereas their approach depends on two nodes meeting each other in order to exchange
profiles and match their preferences.

Note also that our context-addressed messaging is initiated by a sender, not by the
matching of preferences. However, in our system potential receivers need to first express
interest in receiving messages that will be published to a particular topic. This is achieved
automatically by the system if the receivers have previously uploaded their context-
dependent preferences to their trusted entities. Therefore, upon a particular context update,
these trusted entities will perform subscriptions to relevant topics. Finally, when the
notification containing the message reaches the receiver's trusted entity, it matches the
receiver's context against the context specified in the address of this message, and it checks
the receiver's preferences if the message is relevant for the receiver in its current context. If
the receiver's trusted entity determines by this procedure that the message should be delivered
to the recipient, it sends this message to the receiver's preferred device, adapted to the
receiver's preferred format in the current context.

4.5.5 Restricted flooding (narrowcast) and ontology-based reasoning

In 2008, Domaszewicz, et al. [81] proposed a one-way, unreliable (best-effort),
connectionless, group-oriented communication service based on addressing people in a
certain context, where a sender specifies a context-based address by defining ontology
assertions from terms taken from a context modeling schema. This address can in turn be
interpreted as a new ontology class, which does not exist in the original ontology. If the node
is an instance of this class, it becomes a receiver of this context-addressed message.
Therefore, receivers of these messages are passive (i.e., they do not contribute to the selection
of messages as in publish/subscribe systems).

It is important to note that this approach targets mobile ad hoc networks. Therefore, this
approach is based on a context-based routing protocol [82] which routes context-addressed
messages to their destinations. This protocol uses restricting flooding (narrowcasting) to
deliver these messages. Each node has a permanent class membership, called a profile, or if
the class membership changes during runtime then it is called a context. However, this
protocol currently supports only the static profile case. The protocol consists of (1) concept
maintenance that proactively spreads nodes' profiles through the network via Hello messages
and forms concept-based routing tables, as well as (2) concept-based message forwarding. If
there is too much profile information to be put in a message, then taxonomy-based
compression is performed (by replacing the existing concepts with a more generic one). The
concept-based message forwarding selects the forwarding nodes based on information kept in
the concept routing table. The nodes whose instances are subclasses of the address concept
are chosen to be sub-class forwarders. If no sub-class forwarders can be found, then super-
class forwarders are used. This protocol has lower overhead than flooding. However, it
assumes that the class membership of a node never changes, which is incompatible with our
more volatile context information. The performance of this protocol also depends on the
number of nodes that are selected as sub-class or super-class forwarders, which increases as
the number of nodes in the network increases, thus increasing the message routing time.
Moreover, their method is unreliable and receivers have no control over message retrieval.
The time to resolve the message's context-based address mainly depends on the size of the
ontology schema as well as on the number of concept constructs (operators) and the number
of concepts comprising this context-based address. For example, for a small emergency
ontology, the address resolving time of two concepts and one operator is 150 ms on a PC with
a 2.66GHz Celeron processor and 1GB RAM, whereas for the well-known pizza ontology it
is significantly larger (i.e., 4000 ms). The maximum resolving time for seven concepts and six operators was 700 ms in the smaller ontology. Therefore, this approach is suitable for simple addresses formed by concepts from small ontologies, but because of the long reasoning time needed to resolve complex addresses formed by concepts taken from large ontologies it is unsuitable for these cases. However the question that arises is whether such an approach is needed for simple context-based addresses? Additionally, context-addressed messaging poses major privacy concerns, because node profiles (i.e., context) are spread through the network for maintenance of routes.

In order to quantify this, we define the time to deliver a context-addressed message to correct recipients as the sum of the time needed to route the message using concept-based message forwarding and the time needed to resolve context-based address, i.e., \( T_{\text{delivery}} = T_{\text{routing}} + T_{\text{resolve}}(X) \). Note that in [51] authors concluded that most of the context-based address resolution time is actually consumed for inserting a new ontology class in the ontology schema (i.e., T-Box), as well as removing this class from this T-Box, and significantly less for the classification process (i.e., determining if the node is an instance of this address new class). Therefore, having \( N \) nodes in the network and \( X \) concepts in the T-Box, if users compose \( Y \) context-based addresses, this means that \( Y \) new concepts will be added to the T-Box and the time to resolve a context-based address will increase accordingly, thus, \( T_{\text{resolve}}(X+Y) = k(\text{size}_{T-\Box}) \cdot T_{\text{resolve}}(Y) \), where \( k(\text{size}_{T-\Box}) \) is a function of the size of the T-Box. However, this address resolution time is inversely proportional to the message delivery rate. Moreover, increasing the number of nodes in the network increases the signaling (i.e., the number of Hello messages broadcasted in this network), the number of nodes that are assigned the same concept Z - which will consequently increase the amount of information stored in the routing tables (because for each concept in the T-Box, a list of neighbors to which the message addressed with that concept should be forwarded is stored), and the number of nodes flooded per addressee, which all together will degrade the performance of routing as well as decrease the timeliness of message delivery. Therefore, we can express the context-based routing time as \( T_{\text{routing}} = f(Z) = f(g(N)) \), where \( f(Z) \) represents a function of the number of nodes that are assigned to the same concept and \( g(N) \) denotes a function of a number of nodes in the network. Finally, time to deliver context-addressed message \( T_{\text{delivery}} = T_{\text{routing}} + T_{\text{resolve}}(X) = f(N, X) \), which is a function of the number of nodes in the network and the number of concepts in the T-Box. As message delivery rate is \( R = 1/T_{\text{delivery}} \), we derive that \( R = 1/f(N, X) \). Therefore, we can conclude that the message delivery rate decreases with an increase in the number of nodes in the network and the number of concepts in the T-Box.

### 4.5.6 Preference rule-based reasoning

In 2002, N. Miller, et al. [83] developed the Context-Aware Message Delivery Service using a Contextual Information Service (CIS) in the scope of the Aura project [84]. This service accepts messages from senders in a number of input formats and chooses the message delivery mechanism based on each user's context-dependent preferences. Currently available message delivery mechanisms are: e-mail, SMS, and instant messaging; however, other channels can be easily added, such as fax, voice mail, etc. Context-dependent preferences regarding message delivery modes are specified by the user using the MyCampus Semantic Web module [85] that enables users to specify these preferences using any combination of relevant contextual information. This module also enables controlled access to the user's context information under different context conditions. Contextual attributes are defined in different domains of OWL-based ontologies (such as calendar activity ontology, location ontology, delivery channel ontology, etc.). Message delivery preferences are saved in the same format and loaded as decision rules into the Semantic Web module. The arrival of a
message is modeled as a new fact, this activates one or more rules that map contextual attributes (needed to process the incoming message) onto context queries of the CIS module and other context sources. However, this context-aware message delivery service does not take into account the receiver's preferences regarding their preferred terminal to receive messages in the current context or the receiver's negative preferences that explicitly state which messages the receiver does not want to receive. It also does not perform context-based addressing nor address the issue of preserving the sender's anonymity. This approach provides timely delivery of volatile context information to applications. For example, answering a context query from multiple context providers, such as 600 access points, took on average 16ms.

From the system's point of view, the design of CIS assumes that web services provide contextual information, and that this information is accessed via an SQL-like interface. However, as indicated in [83], this design is not sufficiently powerful to deal with complex queries, as the diversity of the information providers creates some unique challenges. In our previous work [2][3], we proposed a context synthesizer based design that solves these problems using context operators. Miller et al. aim to develop a variety of special-purpose editing tools to enable users to specify their preferences with regard to predefined sets of ontologies. Their examples of user preferences include message filtering preferences, privacy preferences, food preferences, etc. Their editor is based on XSLT stylesheets, which are independent of the domain ontologies and can be refined to support more specific instantiations of high-level functions, such as "creating", "deleting", "extracting", "updating" a rule, or "adding/deleting concepts", "adding/deleting properties", etc.

4.6 Implementing context-addressed messaging on top of SIP network infrastructure

Because of reasons identified in Section 3.1, the proposed architecture for context-addressed messaging utilizes a SIP network, requiring the following network elements: a SIP server that supports registration, proxy operations, a presence service, and a resource list service; an XCAP server for management of resource lists; and user agents (including presence user agents) representing all SIP endpoints that are managed by a user. We will describe each of these network elements and operations further below.

Registration allows a user to indicate one or more locations (i.e., by uploading SIP URIs) to be used by proxy servers for routing requests. For this purpose a SIP Registrar receives registration requests and associates the user's location (called address-of-record (AoR)) with the one or more hosts. This binding is stored in the Registrar's database and can subsequently be used by proxies in the same domain.

Proxy servers (referred to in this thesis as proxies) are SIP routers that receive a SIP message from a user agent or from another proxy and forward it toward its destination. Routing the message means relaying it to either a destination user agent or to another proxy on the path to such a user agent. Proxies and Registrars are logical entities that can reside in the same physical node.

A Presence service is a system that provides presence information about a user (i.e., a presence entity or presentivity) to interested parties (called watchers) [86]. Presence information is characterized by a set of attributes that characterize the availability and willingness of a presentivity to communicate across a set of devices. Examples of presence information are status, capabilities, communication address, etc. On each device, a presentivity uses a Presence User Agent to provide presence information to a presence service.
A Presence Server is a functional entity that receives SIP SUBSCRIBE requests for the presence information of a presentity, responds to these requests, and generates notifications of changes in the presence state on behalf of a Presence User Agent.

In this presence framework the presence protocol is any protocol capable of enabling the exchange of presence information in close to real time, between the different entities defined by the model. Typically, the presence service is implemented as an application on top of SIP's event notification framework (i.e., using SIMPLE [87]). SIMPLE provides a means of distributing information in both synchronous and asynchronous modes. To achieve this, SIMPLE uses three messages: SUBSCRIBE, PUBLISH, and NOTIFY, where a specifically designed SUBSCRIBE message denotes a context distribution mode. More specifically, setting the expiration time to zero seconds initiates the synchronous fetch of context information (i.e., a request/response mode). This will result in an immediate notification containing the current context state, canceling an outstanding subscription and all further notifications of changes in this context. Otherwise, asynchronous notifications will take place (i.e., the SUBSCRIBE triggers subscription-based mechanism), notifying the watcher(s) about every change in the context state until either the subscription validity time expires or the watcher(s) unsubscribe to this context. Note that the unsubscribe action is performed in the same way as the synchronous fetch of context information and that each SUBSCRIBE message generates an immediate response containing the current state.

We have extended this presence service to provide information about the user's context, regardless of where this context information is produced. This context distribution service allows distribution of a user's context information in both a synchronous and asynchronous way. This service involves three entities: a context entity or contextity, a context server, and a watcher. We implemented context distribution user agents to provide context information of a contextity by gathering information from multiple sensors. A watcher is represented by a context provider that provides context information of a user or a device to an application that initiated context request. The context server is a presence server that supports a context information model in the body of SIP messages and was extended with resource list URIs to support subscriptions to multiple sensors as well as sending aggregated notifications (from multiple sensors) to a watcher (i.e., a context provider).

Because some context information changes more frequently than the other information and this information needs to be available on multiple nodes, Carlos Angeles Piña examined in his thesis [88] different application requirements for retrieving context information in terms of latency, frequency of updates, and network traffic. Combining these requirements with the results of system scalability and latency evaluation (performed by varying the number of users interested in retrieving the information, the rate of context updates, and the user's mobility), he gave recommendations for application developers about when it is better to use synchronous or asynchronous mode, in order to provide the relevant information to applications at the proper time. In our system, we use both context distribution modes with a context provider (application) that based on subscriptions received for context from a SIP proxy, initiates synchronous and asynchronous requests to the appropriate sensors.

Note that in the presence service SIP PUBLISH and NOTIFY messages carry the presence information in the body of a message formatted in Presence Information Data Format (PIDF) [89] and the extension of the PIDF for conveying richer presence information called Rich Presence Information Data (RPID) [90]. However, because our context-addressed messaging uses context operators and these operators require ontology-based context modeling, we transfer context data in Manchester OWL and not in PIDF or RPID format. Therefore, we replace the content type of "application/pidf+xml" with the "text/plain" content type. Additionally, all SUBSCRIBE/NOTIFY transactions contain a SIP Event header field
(identifying the type of the event the subscription or notification is related to) assigned to the value presence, thus identifying the "presence" event package.

A resource list service is a service associated with a group of users, or more generally, with a list of resources. It is defined and associated with a URI, called a resource list URI. When a SIP request is sent to this service URI, the server providing the service reads the list, and performs some kind of operation against each resource on the list. An example of a resource list service is a presence list service that allows a client to generate a single SUBSCRIBE request for presence information of a list of resources\(^3\). In order to process subscriptions for resource list URIs, a resource list server (RLS) is needed. Resource lists are stored in the document in the XML format, encoded in UTF-8. XML Configuration Access Protocol (XCAP) is used for managing these documents.

XCAP is an HTTP based protocol for accessing remote configuration data [91]. XCAP allows a client to read, modify, add, or delete parts of data stored in XML format. These operations are supported using HTTP 1.1. An XCAP server acts as a repository for collections of XML documents. These documents can be stored by different applications. Within each application, documents can be stored by different users. To access these documents or parts of these documents XCAP defines an algorithm for constructing a URI that can be used to reference this component. Components refer to any element or attribute within a document. HTTP resources representing these components are also called XCAP resources. Reading an XCAP resource is accomplished with HTTP GET method, creating or modifying an XCAP resource is done with HTTP PUT, while removing a resource is performed with HTTP DELETE.

Thus an XCAP server can be used to maintain a list of AoRs of available device sensors that provide the same type of context information about a particular entity to which applications can subscribe to. However, we extend the XCAP operations specified in RFC 4825 [91] to support group management and multicast, by allowing sensors to (explicitly) join and leave the group. After joining the group, sensors become members of a multicast group to which other clients (i.e., applications and context providers) can send any type of SIP messages, i.e., not limiting to SUBSCRIBE messages. To support this we designed our own authorization policies for access control and management of resource lists, as well as specifying naming conventions for resource lists, sensor device URIs, and context provider URIs.

Figure 33 illustrates a SIP network infrastructure for context-addressed messaging. This diagram shows Alice's and Bob's infrastructure represented with separate Internet domains: alice.example.com and bob.example.com. Their infrastructure consists of the following entities running on different hosts: a context-addressed messaging application, a context provider user agent (UA), one or more context distribution UAs, a SIP server (for implementation of trusted entity), and an XCAP server. Additionally, a broker infrastructure is presented by a SIP server, with both presence and registrar functions.

---

\(^3\) An example using this approach can be found in [92]
Figure 33: Alice's and Bob's SIP network infrastructure for context-addressed messaging

A context distribution user agent (Context distribution UA) runs on a sensor device that represents a host on which one or more sensors run. This context distribution UA registers these sensors metadata (i.e., the type of context information they provide) in the SIP network in order for them to be discoverable by applications running on remote nodes. Registration of a sensor's metadata is performed by adding this sensor node's AoR as an entry into a resource list (representing a particular type of context information that a sensor produces). This resource list is stored at the XCAP server. If there is a change of this XCAP document, then the XCAP server notifies the Resource List Server (RLS). By collecting the sensors that provide the same type of context information into a group, we are able to track the availability of these sensors, and to provide an event notification service to their context provider, thus providing both events concerning the sensor membership in a group and their context updates. By maintaining a group of available sensors, a context provider can be notified about changes to the sensor availability of this group, and it can select a subset of sensors to subscribe to based on the quality of information these sensors provide with regard to the quality of information requested by an application. The quality of the context might be specified in terms of precision, probability of correctness, trust-worthiness, freshness, resolution, and or availability of application-specific logic. In this thesis we will address the precision (i.e., granularity) of context information as a context quality parameter that can be requested by an application when subscribing for particular context updates or by context sensor plugins when registering this sensor metadata.

A context provider user agent performs resource location (i.e., discovers sensor nodes that are able to provide a particular type of context information) upon arrival of a context subscription request. This is implemented by retrieving the existing resource list from the XCAP server containing AoRs of sensors providing the desired context information type.

An application user agent sends context-addressed messages, uploads receiver's context-dependent preferences, and receives messages published on the receiver's topic of interest under the two conditions: (1) if the context address specified in these messages matches the receiver's current context and (2) if these messages are relevant to the receiver as determined by context-based filtering process.
4.6.1 Context-addressed messaging operations

Figure 34 depicts a diagram demonstrating actions that are performed by Bob when sending a context-addressed message and actions to be performed at Alice's side in order to receive this message.

For simplicity and better readability these actions are each described in the following:

Step 1: The receiver Alice registers with her SIP Registrar using her AoR (i.e., SIP URI) and uploads her context-dependent preferences using an application running on her device. These two actions are implemented by sending a SIP REGISTER message to the SIP proxy, as shown in Figure 34. Note that in this diagram the SIP entity sips.alice.example.com/proxy acts as both Alice's SIP Registrar and Alice's SIP proxy server.

Step 2: Preferences are stored in Alice's SIP proxy server's database for CPL scripts.

Step 3: Upon receiving Alice's preferences, her proxy extracts context parameters upon which these preferences are conditioned and sends a SIP SUBSCRIBE message to Alice's context provider indicating sip:alice.context@example.com as a destination URI in order to obtain Alice's context updates. Note that instead of "context" there should be a particular context parameter name. An immediate SIP NOTIFY message is sent back containing either Alice's current context or a pending response if the context is not available.

Steps 4&5: Alice's context provider user agent gets the resource list document for the desired context from the XCAP server by issuing a HTTP GET request.

Figure 34: SIP operations for context-addressed messaging
Step 6: After obtaining the resource list, the context provider selects the sensors it wants to obtain context from (based on the quality of context parameter) and sends a SIP SUBSCRIBE containing a new resource list with the selected sensors URIs. Providing this new resource list in-line is desirable because it results in lower signaling overhead as the context provider does not need to use a HTTP PUT operation to upload this list at the XCAP server before issuing a SUBSCRIBE message with this list's URI.

Step 7: Since the SUBSCRIBE message indicates the resource list URI as its destination, this request will be intercepted by the RLS. The RLS sends individual SUBSCRIBE messages to all entries in the resource list on behalf of Alice's proxy. These are actually the URIs of context distribution user agents representing sensors that need to be activated, in order to start publishing their context values.

Steps 8 & 9: Upon receiving a SUBSCRIBE message, a context distribution user agent activates its sensor(s) and starts receiving context data

Steps 10 &11: The context distribution user agent sends a NOTIFY message containing a context data update to the RLS. The RLS in turn waits for updates from multiple context distribution user agents, aggregates them into a separate NOTIFY message, and sends it to the context provider user agent.

Steps 13, 14&15: The context provider user agent notifies Alice's proxy, which retrieves the appropriate incoming notification script for this context update, if any, and executes the specified action. The action tells Alice's proxy to which topic(s) it should subscribe to at the broker on behalf of her to receive notifications about the messages published on these topic(s).

Steps 16, 17 & 18: At some point in time, Bob sends a context-addressed message using the SIP PUBLISH message to the topic URI using an application user agent. This message first reaches Bob's proxy server, which replaces Bob's URI with a pseudonym URI in the From header field, inserts its own address in the Via field of the SIP header to ensure that all reply messages will propagate over the proxy on the way to the sender, and forwards PUBLISH message to the broker. The broker in turn notifies Alice's proxy about it.

Step 19: After performing context synthesis and context-based filtering, if this results in message delivery, Alice's proxy sends a SIP MESSAGE to her application user agent containing the message adapted in the appropriate format and on Alice's preferred device in the current context.

4.6.2 Context distribution operations

As mentioned earlier, we provide public and private resource lists as part our context distribution operations. Public resource lists are used to provide a list of all available sensors providing the same context type, while private resource lists are created by a context provider to select and subscribe to a subset of sensors obtained from the public resource list based upon some criteria [6].

A context type in our context model is represented as a triple <DomainInstance, PropertyName, and PropertyValue>. In order to simplify context querying, we have mapped a DomainInstance to an entity and PropertyName to a scope, where entities refer to concrete entities in the real world (e.g., User, Room, Device) and the scope groups property names belonging to the same context domain (e.g., the scope Position groups context properties like: Longitude, Latitude, and Accuracy). Note that these entity and scope terms were taken from the MUSIC context model described in [93]. This method of context modeling is suitable for composing AoRs of resource lists as follows: sip:<entity>.<scope>@example.com, thus allowing easy querying for some entity's context information. Such an AoR represents all
available sensor plugins that are able to provide the requested \textit{<entity, scope>} pair. By sending a \textit{SUBSCRIBE} method to this AoR, one can receive in a single \textit{NOTIFY} message context updates aggregated from all the sensor plugins indicated in this resource list. Note that the term \textit{value} [93] can also be used to specify \textit{granularity} of the requested or provided context value (such as city or address for the location scope).

4.6.2.1 Registration of context sensor’s metadata

The context distribution user agents representing sensor devices register at their startup to the SIP Registrar with a unique username, where a username is composed of user and device name (e.g., alice_PC, alice_laptop, alice_nokia_N800, etc.). After this registration, a context distribution user agent fetches a resource list associated with the context type its sensor provides, adds this sensor device URI to the list, and uploads the modified list to the XCAP server (see Figure 35). The XCAP server in turn updates the RLS with the modified resource list document. Note that each time the XCAP document is modified, the entity tag (ETag) value of the XCAP document changes that enables the version-history of the document. These changes are propagated to all the watchers (i.e., here context distribution user agents) containing the previous ETag, the new ETag, the change that is made on the document, and the patch which when applied, enables a watcher to transform the former (original) document into a modified one. Thus, there is no need for fetching this modified document from the XCAP server.

An example resource list after adding this sensor device URI is shown below:

```
<?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 name="sip:alice.location@example.com">
    <entry uri="sip:alice_nokia_N800@example.com;entity=alice,
scope=location, value=ssid">
      <display name="Alice's SSID sensor"/>
    </entry>
  </list>
</resource-lists>
```

![Figure 35: Registration of context sensors metadata for context distribution](image)

Note that this functionality enables a sensor to (explicitly) join the group that provides a particular type of context information. If the device does not re-register after the initial registration expires, it is considered as being deregistered from this SIP domain.
4.6.2.2 Resource location

Resource location is performed by retrieving an existing resource list document associated with the required AoR from the XCAP server in a synchronous or asynchronous manner [6]. In synchronous resource location (shown in Figure 36) the context provider UA performs an HTTP GET operation to retrieve this resource list. If the resource list document exists, then it is returned in the HTTP response; otherwise, a zero length body is returned in the response.

![Figure 36: Synchronous resource location](image)

Asynchronous resource location is depicted in Figure 37. Here, the context provider user agent creates an empty resource list document, associates it with the requested AoR, and uploads it to the XCAP server using the HTTP PUT method. Finally, the context provider user agent subscribes to the RLS to be notified about any changes in this document (i.e., when a new sensor device that can provide the requested context type becomes available). Note that the issued SUBSCRIBE message differs from the SUBSCRIBE message used in presence information events in that it belongs to the xcap-diff SIP event package [94]. This event package enables clients to subscribe to changes in an XML document and receive notifications whenever a change in this document occurs, by specifying a specific resource that changed and how it changed. The version-history of document comparisons are based on the strong entity tag (ETag) values of XCAP documents which are also indicated with the xcap-diff format [95]. This event package works with the XCAP diff documents that indicate a change in the XCAP document, including previous and new ETags. These documents are transferred in a body of NOTIFY messages representing a partial or full state of an XCAP document.

In our resource location implementation subscribing to changes to resource lists means that whenever a resource list document assigned the requested AoR changes because of the addition or removal of an entry for a sensor device URI, then all the watchers (i.e., context provider user agents that have subscribed to this resource list) will be notified with a subset of this XCAP document (called an XCAP-diff document). Note that this functionality implements the event notification service that provides the events concerning the changes in the sensor availability of this group.
4.6.2.3 Event notification of context information

After retrieving a list of sensors that are able to provide the requested context information, a context provider user agent selects the sensors that it wants to obtain information from, inserts them in a new resource list, and sends a SUBSCRIBE message containing this new list to the RLS. This SUBSCRIBE message has to conform to RFC 5367 [96], which defines how to create a list of a set of resources, put this list in the body of a message, and subscribe to it using a single SIP SUBSCRIBE request. Such a SUBSCRIBE message has to include the "recipient-list-subscribe" option-tag in a Require header field to ensure that a server can process the recipient list body used in a SUBSCRIBE request. Additionally, this SUBSCRIBE message has to include an "application/rlmi+xml" MIME type in the Accept header in addition to the other types supported by this client (including any types required by the event package being used).

This SUBSCRIBE message is received by the RLS, which forwards this request to individual context distribution user agents (as shown in Figure 38) [6]. These context distribution user agents in turn activate sensors to start publishing context data, and to send these context updates in NOTIFY messages back to the RLS. The RLS waits for a short (predefined) time period for context updates from all the context distribution user agents in the resource list, aggregates them in a separate NOTIFY message, and sends this to the context provider user agent.

Figure 38: Event notification of context information
4.6.2.4  Sensor deregistration

We distinguish between two cases of sensor deregistration: (1) when the sensor gracefully turns off and (2) when the context distribution user agent of this sensor fails, thus the node is (ungracefully) disconnected from the network [6]. In the former case (shown in Figure 39), the graceful deregistration procedure should trigger the deletion of this sensor device URI from the resource list stored at the XCAP server, which will subsequently update the RLS server with the modified resource list. The RLS server in turn notifies the clients that have subscribed to this resource list URI that there is an updated resource list. Next time this sensor turns on again, it should verify the state of subscriptions and delete subscriptions that have expired. The context distribution user agent also needs to internally stop its notifications. A context provider user agent that has previously subscribed to this resource list AoR (as depicted in Figure 37) will be notified about the updated resource list, enabling this user agent to modify its private resource list (if it has created one) at the XCAP server (see Figure 39).

![Figure 39: Sensor deregistration when sensor plugin gracefully turns off](image)

In the case of failure, it is not possible to do the housekeeping operations associated with a sensor device URI. Therefore, unless the sensor re-registers, the RLS server will first discover the sensor’s absence when it subscribes to its device URI and receives a response that this URI is no longer available or fails to get a reply (as shown in Figure 40). Next, it will delete this sensor’s entry from the RLS list at the XCAP server and notify context provider user agents subscribed to this RLS URI that this sensor is no longer available.

![Figure 40: Sensor deregistration when context distribution UA fails](image)

83
4.6.2.5 Authorization mechanisms for access control and management of resource lists

As mentioned earlier, we distinguish between public and private resource lists. A public resource list is used for registering and discovering of sensor devices that are able to provide the desired context information. A private resource list is used for selection of and subscription to a subset of available sensor devices from a previously retrieved public resource list. In this subsection we define the authorization mechanisms for access control and management of these resource lists.

By default, XCAP server allows all clients to read, write, or modify their own XML files (i.e., residing in their own directory). However, only trusted clients, explicitly provisioned by the server are able to modify global documents. These rules are defined within a default XCAP authorization policy.

Each XML file on a server (i.e., XCAP resource) is associated with an application [91]. Therefore, application specific conventions are defined to specify how an application should use its XCAP resources. More specifically, these conventions include an XML schema that defines the structure and constraints of the data, well known URIs to bootstrap access to the data, etc. All of these application specific conventions are defined by an application usage. Application usages are identified using the Application Unique ID (auid), a name that uniquely identifies an application usage within the namespace of application usages.

Internet Assigned Numbers Authority (IANA) defines the following XCAP application usages: XCAP caps (auid=xcap-caps), PIDF manipulation (auid=pidf-manipulation), resource lists (auid=resource-lists), RLS services (auid=rls-services), and presence rules (auid=pres-rules) [97]. XCAP caps, as defined in RFC 4825 [91], lists the capabilities of the XCAP server. This usage defines a single document that allows clients to learn the capabilities of the server. PIDF manipulation, as specified in RFC 4827 [98], defines how XCAP is used to manipulate the contents of PIDF-based presence documents. These presence documents are used as inputs for building the overall presence state for the presentity. Resource lists, specified in RFC 4826 [99], defines access to a resource list, identified by URIs, to which operations, such as subscriptions, can be applied. RLS services application usage, defined in RFC 4826, is a SIP application whereby a server receives SIP SUBSCRIBE requests for resource, and generates subscriptions towards a resource list. Presence rules, defined by Open Mobile Alliance (OMA) in Presence XML Document Management (XDM) Specification [100], is an application that uses Presence Authorization Rules documents to control which clients are authorized to subscribe to a presentity's presence information and what content of notifications will be sent to each watcher.

Note that we will use resource-lists application usage for description of public resource lists and RLS-services application usage for description of private resource list. The later is used because RLS-services application usage defines a document that contains a service URI as a resource list identifier which can be used in subscriptions to its resource list.

The application usages can specify a different authorization policy that applies to XML documents associated with their application usage. Alternatively, if application usages do not wish to define their own authorization policy, they can simply state that the default policy is used. The authorization policy defined by an application usage is used by the XCAP server during its operation.

We have specified in the resource-lists' application usage an authorization policy that allows context distribution UAs to modify and delete their own entries in public resource list documents. Note that RLS should have an authority to modify and delete any entry in the public resource list; however in our case RLS was collocated with the XCAP server, thus
having direct access to database tables with XCAP documents – therefore, there was no need to add these special privileges in the authorization policy. In private resource lists (contained in the RLS-services document), we use the same authorization mechanism as in the default policy that allows clients (in our case, application user agents) to read, write, or modify their own private resource lists (i.e., residing in their own directory).

Note that XCAP documents are stored at the server in a mandatory hierarchy. The root of this hierarchy is called an XCAP root (see Figure 41). It identifies the root of the tree within the domain where all XCAP documents are stored. The domain used by XCAP root should be the domain of the service provider. Since we are using SIP, this domain will be equal to the domain part used in the provider's AoR. Additionally, XCAP root is represented with an HTTP URI, called an XCAP root URI. Next in the tree is the auid. As mentioned earlier, we will have two auids: RLS-services and resource-lists for private and public resource lists, respectively. The former application will have data that is set by users, and the later will have global data that applies to all users. Therefore, beneath RLS-services auid is "users" sub-tree whereas beneath resource-lists auid is a "global" sub-tree. Consequently, the "users" folder holds the documents that are applicable to specific users and the "global" folder holds documents applicable to all users. Within the "users" there are zero or more sub-trees, each of which identifies documents that apply to a specific user. Each user known to the server is associated with the username, called an XCAP User Identifier (XUI). For SIP applications, it is recommended that XUI is the AoR of the user. Therefore, beneath "users" there are zero or more XUIs. Underneath each XUI can be anything, but the path eventually leads to the user-specific documents.

To distinguish between private resource lists from different applications in the same user's RLS-services document, we compose a resource list name by combining XUI with the application name, such as: "sip:alice@example.com;application=CAM". This (private) resource list will contain entries of sensor devices AoRs that this application has selected to subscribe to.

![Figure 41: Hierarchy for storing public and private resource lists](image-url)
The described authorization mechanisms in this Section represent an enabler for SIP multicast by allowing context distribution user agents to explicitly join or leave a multicast group (defined by a resource list URI represented by an \(<\text{entity}, \text{scope}>\) pair). Additionally, these authorization mechanisms allow application user agents to examine available sensors belonging to this group in order to select the ones that they wish to subscribe to, as well as to add them to application user agents own groups which they formed for this purpose (i.e., \textit{private} resource lists). Finally, application user agents can send a SUBSCRIBE message to context distribution user agents belonging to a public or a private resource list using the resource list URI as a destination address in this message. Although, we use the above described multicast functionality for context distribution purpose, its use does not need to be limited to it. To this extent, in Section 5.4 we describe how this multicast functionality can be used to send a SIP message to a group of user's contacts that have a particular social relationship with a user. This group of user's contacts corresponds to a public resource list. Alternatively, an application user agent can specify a private resource list containing a subset of the user's contacts from the public resource list and send a message via SIP multicast to this private resource list URI.

\section{Incoming notifications (context triggers)}

Incoming notifications are used for implementation of a context trigger. A context trigger initiates an action upon a context update. To implement this, a user needs to upload his/her context-dependent preferences to his/her proxy, which activates a user's preference upon a particular context update. This preference specifies an action to be performed upon a context update. More details about context triggers are given in Section 5.3.

In context-addressed messaging we use context triggers to subscribe for preferred topics upon a particular context update. These topics are specified in a user's context-dependent preferences, which are uploaded to the SIP proxy server's database as incoming notification scripts (i.e., context triggers).

In the next chapter we describe our design of context switch and context trigger as well as illustrate how context-aware session control can be implemented using only these two constructs.

\section{Summary}

In this chapter we identified the following requirements for context-addressed messaging:

- delivery of relevant messages to the user in his/her current context according to the user's preferences;
- timeliness of message delivery in order to reach this message recipients while the contents of the message are still relevant;
- support for the user's privacy when designing the system in order to prevent context-addressed messages from being examined or modified by network infrastructure nodes when traversing physical links owned by ISPs or phone companies;
- achieve the system scalability, because the system needs to scale with the increasing number of recipients.

Next, we analyzed the types of application-level communication and investigated whether they can be used to deliver context-addressed messages. The conclusion was that none of the existing message delivery modes completely satisfied the requirements. Therefore, we decided to extend the publish/subscribe mechanism with context-based filtering at the receiver's trusted entity in order to realize the mechanism for context-addressed messaging.
This context-based filtering represents a procedure of determining if the message is relevant for the recipient in their current context and/or deciding how this message should be delivered (i.e., on which device and using what communication means). This also includes delivery of the message using the recipient's preferred communication means and the preferred device, as well as learning of new previously unspecified preferences. Note that the context-based filtering enables the routing of messages within the user's infrastructure (so called inner routing), thus protecting the user's privacy. Performing this filtering at the receiver's side instead of making routing decisions for context-addressed messages also increases the scalability of the system, because this filtering is performed for each recipient at his/her trusted entity.

In this chapter we also designed a novel format for context-addressed messaging that uses context operators to define context addresses. In order to resolve the received context-addressed message, a receiver's trusted entity needs to perform operator matching in order to find the correct operator to compute the high-level context of the receiver and determine if the receiver should receive this message. Next, we describe our system architecture for realizing our context-addressed messaging approach with a detailed view of the sender's, network, and the receiver's infrastructure. Note that in the sender's infrastructure we also introduced the anonymizer functionality in order to protect the sender's identity and improve his/her trust in the system.

In the second part of this chapter we reviewed the relevant related work in context-addressed messaging according to the outlined requirements and compared it to our system design. We have categorized the systems that were reviewed into the following groups based on their approach/technology used to implement context-addressed messaging: (1) distributed location infrastructure, (2) content-based publish/subscribe mechanisms, (3) variations of multicast (such as Xcast or Geocast), (4) use of restricted flooding (such as narrowcast) & ontology-based reasoning, (5) use of similarity-based matching (i.e., Profile-Cast), and (6) preference rule-based reasoning.

We concluded that the systems developed using the approaches (1) and (2) had some privacy issues because of risks of revealing users location information to unintended parties (in the former case) and because the broker is able to learn sender and receivers identities and could gain some knowledge about recipients by inspecting sent and received subscriptions and publish messages, if these messages are not encrypted (in the latter case). Therefore, our design decision was to keep processing of context information within the user's infrastructure instead of storing it in the network, in order to avoid it from being misused.

The major drawback of the systems belonging to the groups (3), (4), and (6) is the inability of specifying receiver's preferences regarding the interested content or message delivery and performing context-based filtering of messages, which are both important for implementing relevant message delivery. However, we learned from the Profile-Cast approach that matrixes are a good way of representing user's preferences that change with time and that we can easily compute the similarity index between preferences of two users in order to find out if their preferences match. In contrast to their preferences, our preferences not only change with time, but also based upon the user's current context. Therefore, an open issue for the future work is to investigate the possibility to use multidimensional matrixes to model context-dependent preferences.

Systems developed using the approach (5) are not suitable for routing of arbitrary complex context-addressed messages because they cannot fulfill the timeliness requirement and also have privacy concerns.
Finally, we described how to implement context-addressed messaging using SIP network infrastructure. The main part of our future work will be to implement and evaluate the proposed system in terms of latency and scalability. Other open issues are:

- To investigate how to specify topics for publishing context-addressed messages and who should decide which topics will exist.
- To investigate under which conditions the proxy should anonymize the sender's actual address? Should we also allow the responder of the message to be anonymous? How should the reply messages access be configured in this case?
- To allow learning of users preferences, we should investigate a way to allow messages for which Alice was not subscribed, but that could potentially be interested to her to receive, to be delivered (if we use a publish/subscribe system)? The question that arises from this is: should we invent some new mechanism for subscribing to undefined topic (something similar to the use of wildcards?), but once user feedback is obtained then the new preference can either cause the trusted proxy to create this new topic and subscribe to it, or unsubscribe to this topic and create a negative preference instead? Finally, should we allow the user specify negative preferences? How should the user provide his/her feedback to the system and how to incorporate this feedback into the learning process?
- Context-based filtering is performed on the receiver's trusted entity. It should be investigated where the preference learning should take place – in particular, how will the observed behavior be logged, by which component, how often will it be analyzed and by which methods/tools? Can the user specify when it should not be logged?
CHAPTER 5

CONTEXT-AWARE SESSION CONTROL

In this chapter we describe how we can trigger communication between people based on match of their preferences and current context. We also illustrate on several examples how context information can be used to adapt, modify, and manage user's communication sessions. A key solution to our approach, as it will be described in this chapter, is to enable users to specify their context-dependent preferences regarding the type of communication and content they are interesting in receiving. These context-dependent preferences are activated upon a particular context update, triggering a specific action (i.e., a session control or subscription to the relevant content). This defines one of our two main constructs for implementing context-aware communication services, called context trigger. The other construct is a context switch, which is activated by an incoming communication event and which uses receiver's context information to select from a set of context-dependent actions an action that specifies how to process this event. We implemented a context-switch by extending syntax of Call Processing Language (CPL) scripts and built a context-aware VoIP prototype in order to demonstrate how easy it is to add new context parameters and how complex decision making criteria can be built using our solution. Next, we illustrate how all context-aware communication services can be implemented using only these two types of constructs.

As an example of an action initiated by a context trigger, we designed a group query, to be sent to a group of user's contacts (that have the same relationship with the user) in order to find the ones whose interest or context matches this user's interest. This group query also carries information about the user's interest and can contain some of the user's private context. The response to this query contains a matching result obtained from a member of this group, which if positive, can trigger the initiation of a communication session between the user and this group member. Finally, we design a system infrastructure for context-aware session control, which is able to support context switch, context triggers, handling of user's context-dependent preferences, and group queries.

5.1 Introduction

Different types of users have different preferences regarding the type of the communication and content they are interested in receiving. These preferences may vary with time and the context of the user. This context includes the user's location, activity, or other context parameter(s). An example of user's interest in communication includes finding people (from among a user's contacts that have the same relationship with the user, such as friends, family, colleagues) with the same interest or current context as the user and initiating the communication session with them. If during a session some of the context suddenly changes (e.g., a significant decrease in bandwidth or a match of the user's interests), new preferences (regarding device and communication means) will trigger a specific action (session initiation, adaptation, or termination). Similarly, change in the receiver's context (e.g., change of location from "work" to "home") could change new preferences regarding the content that he/she is interested to receive, which would trigger an action of subscribing to a different type of topic (e.g., "sports" instead of "stocks"). Therefore, a receiver's proxy will subscribe to this specific content as triggered by a change in the receiver's current context.

In this chapter we propose a way how to specify these context dependent preferences and use them to trigger a specific action (i.e., session control or a subscription to the relevant content). Similarly, in case of an incoming communication event, we demonstrate how
context can assist in decision making about the appropriate context-dependent action on behalf of a user. To achieve this we define two types of constructs: a **context switch** and a **context trigger** (shown in Figure 42).

![Figure 42: Context-based and context-triggered communication](image)

A context switch represents a set of actions the receiver takes upon an incoming communication event from a sender (e.g., a call invite or a message arrival). The receiver's current context **causes the selection of an action** from the specified (context-dependent) actions (i.e., whether to accept/reject the call, or forward it to a voicemail). This idea is based on our previous work [1][101], where we extended CPL (Call Processing Language) scripts with contextual parameters to permit context-based call decision-making based on a context ontology. Moreover, context access policy rules could also be modeled with such a context switch element, to enable handling of an incoming or outgoing context query. With policy rules modeled in this way a user could share some of its context in the granted scope or simply deny access to it based on its current context. Context parameters could be the receiver's location, activity, task, and the social relationship with the sender – i.e., whether they are friends, family, colleagues, or strangers. These context parameters would be implicitly inferred by the system. In earlier work we proposed a mechanism for inference of user social relationships from the logging of his/her mobile phone data [7]. We also envisage employing other context parameters concerning the receiver's currently used device, such as the device model, communication capabilities, available bandwidth, the remaining battery power, etc.

If during a session some of this context suddenly changes (e.g., a significant decrease in bandwidth or a match of the user's interests against other users interests or current context) and the new user's preference (regarding device and communication means) gets activated upon this context update, these will **trigger** a specific action (such as session initiation, adaptation, or termination). **Context trigger** is, therefore, used to **initiate an action** based on the context update and preference set in this updated context.

We unify the proposed modes of utilizing context information to manage the receiver's session by referring to them as **context-aware session control**. This **context-aware session control** can be applied for multiple purposes, such as for call and message delivery, or even...
for remote context query (i.e., when a sender wants to retrieve some context information provided by the receiver). In the rest of this chapter we will show how we implemented context switch and context trigger and how to use these constructs to realize the complete context-aware session control.

5.2 Context switch

A context switch supports the services whose decisions are based on the context information of an end user. This section shows how can this context information enhance the functionalities of existing SIP call control services by offering a user the possibility to decide whether to accept an incoming call based on his/her current context.

In CPL, switches represent choices a CPL script can make based on either attributes of the original call request or other items independent of a call. The existing switches are: address switch, string switch, time switch, priority switch, and language switch, and different screening services can be created based on any of the above switches or combinations. All switches have a list of conditions that can match a variable. When the CPL script is executed, the conditions are checked in the order they are presented in the script. The output of the first matching node is taken. The information affecting the choice is carried in the SIP message.

Based upon considering several different scenarios we identified the need to extend CPL with decisions based upon the following context parameters: user's location (e.g. home, office, car, hotel), task (e.g. lunch, in a meeting, relaxing, on vacation, business trip), and activity (e.g. discussing, presenting, listening). To implement these extensions, we have defined a context-switch and its corresponding output context node to support services whose decisions are based on the context information of an end user [1][101].

The syntax of the node "context-switch" and the "context" node are shown below:

<table>
<thead>
<tr>
<th>Node:</th>
<th>context-switch</th>
<th>context switch node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs:</td>
<td>user_context</td>
<td>specific user context parameters to match</td>
</tr>
<tr>
<td>Parameters:</td>
<td>owner</td>
<td>context owner name</td>
</tr>
<tr>
<td>Output:</td>
<td>user_context</td>
<td>context node</td>
</tr>
<tr>
<td>Parameters:</td>
<td>location</td>
<td>location of a context owner</td>
</tr>
<tr>
<td></td>
<td>task</td>
<td>task status</td>
</tr>
<tr>
<td></td>
<td>activity</td>
<td>activity status</td>
</tr>
</tbody>
</table>

Node "context-switch" has one parameter "owner" that identifies a context owner with his/her URI (i.e., a person to whom these parameters relate to). Node "user_context" is the output of the "context-switch" node. It specifies different context attributes, such as: "location", "task", and "activity" of a context owner. Values of context parameters are specified in the user's ontology document as follows: the location ("office", "home", "car", "vacation", or "business trip"), task ("in a meeting", "at lunch", "relaxing", "working", or "talking on the phone"), and activity ("presenting", "discussing", "listening", "driving", "biking", or "free time" - when no task is assigned to the user).

When the context-switch node is invoked, it will match the context values in the CPL script with the receiver's current context values and return the decision of how to process an incoming/outgoing call (accept, reject, redirect, voicemail, etc.).

The definition of CPL extensions for context is specified in the file "context.dtd" [101] and described in Section 5.3.1. An example of CPL script based on this extended CPL is shown in Listing 5. Jim's SIP proxy server will reject the incoming call if he is in the meeting room called Grimeton, in a meeting, and if he is presenting.
5.2.1 Context-aware VoIP prototype

I have utilized a scalable and reliable open source SIP platform, called SIP Express Router (SER) [102], to upload and execute CPL scripts. It can act as a SIP registrar, proxy, or redirect server. I have extended its functionality to support context-based CPL scripts.

Note that CPL scripts can reside on a SIP proxy server, an application server, or intelligent agent. In my case, I have uploaded CPL scripts to the SIP proxy server, SER (as shown in Figure 43). When the SIP INVITE message comes (initiating incoming/outgoing call), SER executes the appropriate part of the user's CPL script that refers to an incoming/outgoing call and manages the call routing logic (accept and route the call to callee, reject the call, forward it to the voicemail, send an e-mail to, redirect, or proxy to some third party). CPL scripts can be uploaded using SIP's REGISTER method or with the aid of graphical programs, such as CPLEd [103].

![Call processing logic diagram]

A CPL script is parsed after uploading to SER. It is stored in an external MySQL database and is loaded and executed upon receiving incoming/outgoing call requests delivered by SIP INVITE messages. The CPL script then processes these calls.
I have implemented a context-aware VoIP prototype in order to make call processing dependent on a user's context, so as to make it easier to specify a suitable action to be taken. When the user wants to upload a context-based CPL script (see Figure 45), he/she has to first upload the ontology to the match component, which first parses the ontology, extracts the user's context parameter values, and stores them into the external MySQL database (that is also used by SER for storing users and CPL scripts). Second, the match component matches context values with the corresponding values in available CPL scripts to determine which script describes rules for the current user's context. Before they are uploaded to SER, these CPL scripts are stored in a CPL repository, while ontologies reside in a context repository. Upon receiving a call or SIP INVITE message from a SIP User Agent (SIP UA), SER loads the user's current CPL script from the database and executes it. If the CPL script contains a context switch, it will match values set in script rules with the corresponding context values, and if they match, take appropriate actions. The wrapper component is used by SER to retrieve context values from the database.

The prototype that I implemented in the lab consists of four components: a client application, match component, wrapper, and extensions to the CPL-C module [104] of SER.
5.2.1.1  Client application

A simple client application is used for uploading ontologies and CPL scripts (as shown in Figure 46). CPL scripts that are not context-based can be uploaded directly, without the need to first upload the context ontology. The application was designed to be used from different machines and different locations, hence the preferable implementation is as an applet.
Note that this applet was built as a proof of concept only. The alternative solution is to have two clients (applets), one for uploading context (ontology) and another for uploading scripts. The applet opens the file chooser dialog (see Figure 47) to browse for a file to open (i.e. in this case ontology).

**Figure 47: File chooser dialog**

5.2.1.2 Match component

The match component is responsible for parsing the selected ontology to get context values, determine the appropriate CPL script, and upload that script via SIP (or HTTP(S)) protocol to SER. Both choices are available, but we mainly focused on SIP in this prototype. SER will, upon receiving the script, store it in the database under the supplied user's credentials.
5.2.1.3 Wrapper

The wrapper was created to pass context values between client application, match component, and SER. The context parameters are stored in the database when the ontology is parsed, and retrieved by the wrapper program when the script is executed.

5.2.1.4 CPL-C module extensions

I had to modify the CPL-C module of the SER source code to support adding of a context-switch and context node. The syntax of the context-switch and context node is given in Section 5.3.1. More details about extending CPL-C module are described in [101].

5.2.2 Evaluation of context-based CPL scripts

To evaluate the SER response time when executing CPL scripts with increasing complexity, we made a series of measurements. We wanted to compare the difference in time when executing standard CPL switches that read SIP header fields against our context-switch that retrieves context parameters via an ontology. We tried to answer the following questions: what is the added delay and what is the cost of adding ontologies.

We started these measurements by executing a CPL script with one address switch (that makes call processing decisions based upon the value of (origin or destination) address fields in the message), and then progressively added an additional switch, up to 5 in total. Next we did the same sort of tests when executing context-dependent CPL scripts. The measurements are summarized in Figure 48.

![Figure 48: Comparison of (standard and context-dependent) CPL scripts response times](image)

We can see from the figure that adding additional standard CPL switches didn't increase the response time – it remained almost constant, with a total increase of 0.15 (in worst case 0.33) milliseconds, which is 4.6% (or at most 10%). When adding the context-switch, we can see a linear increase of response time with the number of context switches.

Adding context switches to a CPL script increases the response time from 0.4 up to 2.3 ms, a 5%-24% response time increase. The total response time increase for 5 context switches is 46.60%. The difference between the first and the second context switch happened to be smaller than the increases in other cases (as shown in Figure 48), because response time of the first context switch includes the time needed for opening a database connection, whose reference is reused by other context-switch nodes in the same CPL script.
Figure 49 shows the comparison between different types of CPL scripts and their response times: first when we have a CPL script with 2 address switches, the second script with 2 context switches, and the third with 1 address switch and 1 context switch. The results show that a combination of only context-switches is the most expensive, while the combination of only address or other standard switches is the least expensive.

![Figure 49: Comparison of different types of CPL scripts and their response times](image)

Regarding scalability, some measurements were performed in [101] with 100 users sending simultaneously INVITE messages. In case of context-based CPL script, 1485 INVITE messages were successfully processed from around 1650 messages in total that SER has received, which corresponds to a 90% acceptance rate. The SER's total processing time (from the moment he received first message until he sent the last provisional response) was 12.3s, however some of the requests were not answered. In case of the conventional CPL script, 1979 INVITE messages were successfully processed from around 2000 messages received, which corresponds to 98.9% acceptance rate. The SER's total processing time was 15.1s. The SER's average response time was 7.4ms for both the conventional and context-based CPL script. Note that this happens because the rate at which the SER processes the requests is less than the rate of sending these requests, so that the queue becomes completely filled and some packets get lost. However, in order to determine the SER's peak and average processing rate we need to perform more measurements in the non-saturated zone with varying rates of sending consecutive requests.

5.3 Context trigger

As described earlier, we want to trigger communication between users based on match of one user's interests against other users interests or current context. To implement this, a user needs to upload his/her context-dependent preferences at his/her trusted proxy, which activates a user's preference upon a particular context update. This preference in turn initiates a group query to other users in order to find those whose current interest or context matches the user's interest indicated in the query. Context trigger can be used not only for communication initiation, but also for adaptation of the existing communication based on a device context update (e.g., in case of high availability of bandwidth or recharged battery on a device). In contrast to the "context-switch" node that makes decisions about an incoming event based on a user's current context, the "context-trigger" node initiates a communication action upon a user's or device's context update. This communication action creates, adapts, or terminates a user's communication session based on this user's and his/her device's contextual parameters.
Based on extensibility and simplicity of CPL scripts, we chose to implement a context trigger as a new type of node in CPL, which is activated by arrival of a SIP NOTIFY message. This node performs a specified communication action upon a particular context update. Because there are two defined top-level actions in CPL ("incoming" and "outgoing") and these are performed upon a call setup event, we added an additional one, "notification" to be performed upon sending and receiving notification events (i.e., corresponding to an outgoing and incoming notification action, respectively). An incoming notification action is performed when a notification arrives whose destination is the owner of the script. We use this action in implementation of a context trigger, as depicted in Listing 6 on page 99. An outgoing notification action is performed by the owner of the script before sending a notification. This action is left open for future work; it could be used for making decisions when to allow or reject sending of notifications.

Syntax of the "context-trigger" node and "context" node is shown below:

<table>
<thead>
<tr>
<th>Node:</th>
<th>context-trigger</th>
<th>context trigger node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs:</td>
<td>context</td>
<td>context parameters to match</td>
</tr>
<tr>
<td>Parameters:</td>
<td>entity</td>
<td>context entity type: &quot;user&quot; or &quot;device&quot;</td>
</tr>
<tr>
<td></td>
<td>uri</td>
<td>SIP URI identifying a user or a device</td>
</tr>
<tr>
<td>Output:</td>
<td>context</td>
<td>context node</td>
</tr>
<tr>
<td>Parameters:</td>
<td>location</td>
<td>location of a user</td>
</tr>
<tr>
<td></td>
<td>task</td>
<td>task status of a user</td>
</tr>
<tr>
<td></td>
<td>activity</td>
<td>activity status of a user</td>
</tr>
<tr>
<td></td>
<td>bandwidth</td>
<td>available bandwidth on a device</td>
</tr>
<tr>
<td></td>
<td>battery</td>
<td>the device's remaining battery power</td>
</tr>
</tbody>
</table>

The node "context-trigger" has two parameters: "entity" and "uri" that identify a context entity to whom context parameters relate to. The parameter "entity" indicates a context entity type (that can be a user or a device), while the parameter "uri" indicates a SIP URI identifying this context entity. Node "context" is the output of the "context-trigger" node. It specifies the same attributes used by the output of the "context-switch" node (i.e., "location", "task", "activity" of a user), but it adds two additional ones: "bandwidth" and "battery" indicating the available bandwidth and the remaining battery on a device. The "bandwidth" parameter takes a numeric value expressed in kbps, while the "battery" parameter takes a numeric value expressed as a percentage of the total battery capacity. These context parameters are defined in the context model schema, so that we can model the parameters into higher level concepts, and use these concepts in CPL scripts for triggering of communication actions.

An example of a communication action that can be specified in the context trigger to be performed upon the context update is a group query. This group query is designed to be sent to a group of user's contacts, identified by a resource list URI, whose context or interest needs to be matched against the interest and/or context of a query initiator.

Syntax of the node "group-query" communication action is defined as follows:

<table>
<thead>
<tr>
<th>Node:</th>
<th>group-query</th>
<th>group query node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs:</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Next node:</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Parameters:</td>
<td>to</td>
<td>resource list URI identifying a list of receivers</td>
</tr>
<tr>
<td></td>
<td>activity</td>
<td>activity to match</td>
</tr>
<tr>
<td></td>
<td>interest</td>
<td>interest to match</td>
</tr>
<tr>
<td></td>
<td>location</td>
<td>location to match</td>
</tr>
</tbody>
</table>
The group query action is used in an example of the context trigger depicted in Listing 6. This action can be interpreted as: send a query to my friends whose current activity is set to biking (or having an interest to go for it) that are currently in the same city as I am. It is worth noting that the location attribute value "my_current_city" used in this example enables a user to implicitly give his permission for inserting some of its private current context into the query, in order to match the receiver's value. This location attribute value will signal the SIP proxy to retrieve the current location with the city scope from the context provider UA and insert it into the SIP MESSAGE. In the same manner, a string "my_current_" can be concatenated to a scope of any of the desired context attributes from the "group-query" syntax.

Listing 6: CPL-based context trigger

As mentioned earlier, the implementation of a group query uses a resource list URI to send a SIP MESSAGE to a list of user's contacts belonging to the same social relationship group. Therefore, for each social relationship group of its contacts, a user has to maintain a resource list at its XCAP server. This can also be performed on behalf of the user by his/her proxy, if a list of a user's contacts with their URIs and social relationship with the user is available as a file to this proxy. User's social relationship groups can be explicitly defined by a user, imported from existing social networking web sites (such as Facebook, MySpace, LinkedIn), or implicitly inferred from the user's communication logs as proposed in our previous work [7].

Other examples of a communication action are: an initiation of a video session, subscribing to a topic of interest, or switching a call to another preferred device.

5.3.1 XML DTD for proposed CPL extensions

We define four types of extensions: (1) a notification sub-level action to indicate an action performed when a notification message is received or sent out, (2) a new type of switch, called context-switch, that makes call based decisions based on the user's current context, (3) a new type of node, called Trigger with the context-trigger type of this node; this Trigger node is invoked by the notification message; and (4) two new operations are defined that can be invoked within a Trigger node: group-query and subscribe. The definition of CPL extensions for context-switch and context-trigger is specified in the file context.dtd.

Note that Wu, Schulzrinne, Lennox, and Rosenberg have proposed in their Internet draft from 2001 the presence extensions of CPL [105] in which they specified (among other extensions) the notification as a sub-level action and the subscribe operation. A similar example of presence-related extensions of CPL was proposed by D. Jiang in his Master of Science thesis [106] in 2003. Both of these works define a new type of switch for handling...
presence information along with four types of operations needed to subscribe to a presentity, approve subscriptions, send notifications to watchers, and accept these notifications. Services that apply to the specified actions taken by this switch can be classified into screening services, forwarding services, and automatic call service. The first two services are used to process an incoming event, whereas the last one originates a new outgoing event.

However, note that NOTIFY messages do not require user's interaction or control (as it is the case with INVITE and SUBSCRIBE messages where CPL scripts specify a user's policies for call-decision making (in case of incoming INVITE) or a user's control access policies to the their presence information (in case of incoming SUBSCRIBE)). Additionally, in our system NOTIFY messages are used as a means to deliver context updates. We could have defined a new language for implementing context trigger functionality. However, as CPL was created to describe and control Internet telephony services and context trigger could influence the call processing, we decided to extend CPL to provide context trigger support. Therefore, we defined notification as a top-level action that will be handled by a new type of node – Trigger. We use the same definition of subscribe operation as was defined in [105]. Additionally, we add a new type of operation, called group-query that has not previously been proposed or specified.

The part of file that contains our proposed CPL extensions is shown below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Adding a new top level action for notification -->
<!ENTITY % TopLevelActions 'outgoing?,incoming?,.notification?' >

<!-- Adding a new node category, called Trigger. Note that a node can be empty, implying default action. -->
<!ENTITY % Node '(%Location;|%Switch;|%Trigger;|%SignallingAction;|%OtherAction;|%Sub;)?' >

<!-- Switch nodes -->
<!ENTITY % Switch 'address-switch|string-switch|language-switch|time-switch|priority-switch|context-switch' -->

<!-- Context-switch makes choices based on user's current context information. -->
<!ELEMENT context-switch (user_context*, (not-present, user_context*)?, otherwise?) >

<!ATTLIST context-switch
     owner CDATA #REQUIRED>

<!ELEMENT context (%Node;) >

<!ATTLIST user_context
     location CDATA #IMPLIED
     task CDATA #IMPLIED
     activity CDATA #IMPLIED

> <!-- at least one and at most three of those attributes must appear -->
<!-- Trigger node makes choices based on user's or device's updated context information. -->

<!ENTITY % Trigger 'context-trigger' (context*, (not-present, context*)?, otherwise? ) >

<!ATTLIST context-trigger
     entity CDATA #REQUIRED
     uri CDATA #REQUIRED >
5.4 Context-based session initiation

We will examine the use of context-based session initiation in the scenario that was previously described in the introduction of this thesis in Sections 1.1.1.1 and 1.1.1.2. This scenario is also illustrated in Figure 50. We will go briefly through the scenario but paying more attention into what information needs to be available on which component and when, and how different system components of the sender's and the receiver's infrastructure interact in order to achieve the context-based session initiation functionality.

Alice is currently available, i.e., she has no current activity or task assigned, and in her preferences she has indicated an interest in biking with her friends during her free time who are located in the same city during that time. Alice has previously created and uploaded her context-dependent preferences to her trusted proxy using an application running on her
device. Much earlier she established a trusted relationship with this proxy. Upon receiving Alice's preferences, her proxy extracts the context parameters, upon which her preferences are conditioned. Later when Alice's context update matches one of context conditions in her preference set, this will trigger her trusted proxy to send a context query to her friends containing Alice's interest for biking and her current city scope, in order to find those whose current interest, location area, and/or activity match Alice's interests. An assumption here is that Alice's and her friends' trusted entities share the same context model schema, either stored as a file or accessed via an URL on the Web, in order to be able to query each other's context. Additionally, Alice's trusted entity has to check if Alice has allowed in the corresponding policy revealing her current interests and location to friends, and if so, in what granularity.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Alice's context provider</th>
<th>Alice's trusted proxy</th>
<th>Bob's trusted proxy</th>
<th>Bob's context provider</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context-dependent preferences</td>
<td>Exports context parameters from preferences</td>
<td>Current preference (interest for biking)</td>
<td>Check policy for revealing context (interest for biking, location, friends)</td>
<td>Context query (Alice's interest for biking &amp; location)</td>
<td>Context trigger</td>
</tr>
<tr>
<td>Context update (free time)</td>
<td>Query (location)</td>
<td>Alice's location</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context trigger</td>
<td>Call initiation with Bob?</td>
<td>Accept</td>
<td>Call Alice</td>
<td>Accept</td>
<td>Accept</td>
</tr>
<tr>
<td>Context switch</td>
<td>Call Bob</td>
<td>Check incoming call action (biking context, headset, friends)</td>
<td>Query (headset)</td>
<td>Session established</td>
<td></td>
</tr>
<tr>
<td>Context update (headset available)</td>
<td>Call</td>
<td>Accept</td>
<td>Accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context response (successful match)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 50: Communication initiated by a preference match*
We will assume that the context query first reaches Bob's trusted entity. Bob's trusted entity then extracts Alice's interests and location information from the received query, queries Bob's context provider for his location, interest, & activity information, and finally it matches the retrieved Bob's context against Alice's supplied data. Since Bob is located in the same city as Alice and he is currently biking, this will yield a positive match. The result of match is sent back to Alice's trusted entity as the response to the context query. An alternative to this approach would be to query for Alice's friends context information and perform the matching of contexts at the Alice's trusted proxy. However, the advantage of this approach is twofold: (1) by sending Alice's interests and location information in a context query to Alice's friends trusted proxies, these proxies would perform the matching separately, thus improving the system scalability and (2) by sending back only the matching result to Alice's proxy, these proxies would not reveal Alice's friends sensitive context information, thus protecting these users' privacy.

The arrival of successful match of Alice's interests and Bob's current context will trigger Alice's trusted entity to present an option to Alice to initiate a call to Bob. If Alice accepts this and this incoming call reaches Bob, then Bob's current context will select the call logic action controlling whether and how to accept the call. This time when he went biking, Bob took his Bluetooth enabled headset with him to be able to receive calls from his family and friends while biking with his phone in his backpack. The presence of this headset and the friendship relationship of Bob and Alice will result in accepting the call from Alice. Otherwise, this call might be redirected to Bob's voicemail.

From the sequence diagram in Figure 50 one can easily observe two context triggers initiated by a change in Alice's preferences upon the context update and a context switch that selects a preferred call action in Bob's current context. Note that using these two constructs the session between Alice and Bob has been established based on match of Alice's interest and Bob's current context. We will show in Section 5.6 the extension of this scenario that demonstrates context-aware session adaptation based on the same constructs.

In the next Section we will design a SIP network infrastructure needed to implement context-aware session control. This network infrastructure will be used to demonstrate interaction between the sender and receiver's components in the rest of Alice and Bob scenario.

### 5.5 SIP network infrastructure for context-aware session control

The context-based session initiation performs the following actions: (1) uploading the sender's context-dependent preferences to the trusted proxy, (2) subscribing to/querying for context information extracted from these preferences, (3) activating a new preference in the current context that will trigger sending of a query to a group of people having the same social relationship with the sender, this query will contain the newly activated preference and optionally some sender's private context, (4) upon receiving the query, matching of the sender's and receiver's interests and contexts at each receiver's proxy, (5) upon receiving matching results from receivers an initiation of communication session with those with successful matching results.

A proposed architecture for context-aware session initiation utilizes a SIP network, thus it requires: a SIP server that supports registration, proxy operations, presence, Resource List Server (RLS), and an XCAP server that is needed by the SIP server, context providers, and device sensors. The SIP server is used for all (1)-(5) operations. The XCAP server is needed for maintaining a list of each sender's social relationship group members (e.g., family members, friends, and colleagues) as well as a list of sensor devices AoRs providing the same
type of context information about a particular entity. These members (i.e., their trusted proxies) are queried by a sender's trusted proxy with a goal of finding those who's context & interests match the sender's in order to initiate a communication session with them.

Figure 51 illustrates an architecture diagram for implementation of context-aware communication initiation in the SIP network. This diagram shows Alice's and Bob's infrastructure represented with separate Internet domains: alice.example.com and bob.example.com. Their infrastructure consists of the following entities running on different hosts: an application, a context provider, a SIP server, and an XCAP server. Note that this network infrastructure highly resembles the SIP network infrastructure for context-addressed messaging (shown in Figure 33), with a difference that this infrastructure also contains a broker in the network, which is used for subscribing to topics and delivering notifications about message published on these topics.

**Figure 51: Alice's and Bob's SIP infrastructure architecture for context-aware communication initiation**

The following message sequence charts demonstrate actions performed by the context-aware communication initiation. Alice registers with the SIP Registrar using her unique address of record (i.e., SIP URI) and uploads her context-dependent preferences using an application running on her device. These two actions are implemented by sending a SIP REGISTER message to the SIP proxy, as shown in Figure 52. Note that in this and the following message sequence charts the SIP entity *sips.alice.example.com/proxy* has a role of the SIP Registrar and Proxy.

**Figure 52: The action of uploading context-dependent preferences to trusted proxy**
Upon receiving Alice's preferences, the SIP proxy extracts context parameters upon which these preferences are conditioned and sends a SIP SUBSCRIBE message to Alice's context provider indicating sip:alice.activity@example.com as a destination URI in order to obtain Alice's activity updates. An immediate SIP NOTIFY message is sent back containing Alice's current activity.

Figure 53: The action of subscribing to context parameters upon which preferences are conditioned

Figure 54 illustrates how context provider obtains context information from two sensors.

Figure 54: Context provider retrieves information from two sensor devices

105
Note that in Figure 54 sensors that provide the same type of context information (i.e., Alice's activity) are grouped in the same resource list, which is assigned a SIP URI as sip:alice.activity@example.com. Alice's context provider subscribes for notifications created from the sensor1 and sensor2 respectively, by sending a SIP SUBSCRIBE that contains the resource list of sensor devices to be activated, in order to start publishing their context values. This SUBSCRIBE message is sent by the RLS to individual sensor UAs. The resource activation follows the procedures defined in [96], regarding Specific Event Notification, using the SIP SUBSCRIBE/NOTIFY functionality.

After receiving a NOTIFY message containing Alice's activity status equal to "free time", a new preference for biking is activated at Alice's SIP proxy, as shown in Figure 55. This is implemented by finding a CPL script at the Alice's proxy, which in case of activity update equal to "free time" initiates sending of a query to a group of Alice's friends, containing Alice's interest for biking and her current location (as Alice's interest indicates to find friends in the same city who are currently biking or who have the same current interest). Detailed description of context trigger and implementation of this CPL script is given in Section 5.3.

In order to obtain Alice's current location, Alice's proxy sends a SUBSCRIBE message with expiration field equal to zero to her context provider, indicating sip:alice.location@example.com as the destination URI. We use the RLS to send a query to a group of SIP resources, as well as to aggregate their responses (i.e., containing the matching results). Alice's SIP proxy invokes an HTTP GET action at the XCAP server in Alice's domain to retrieve the resource list associated with Alice's friends' context providers – see Figure 55. Note that Alice is responsible to create and maintain resource lists of her social contacts. In case this resource list has not been created prior to this GET invocation, HTTP response will contain zero length body description. After obtaining Alice's current location, her proxy inserts it into a group query implemented by a SIP MESSAGE, along with the Alice's specified interest, indicating sip:friends.context@example.com as the destination (resource list) URI. This message is received by the RLS in Alice's domain. Use of RLS is
Currently standardized and supported within SIMPLE; therefore we needed to extend a SIP MESSAGE to support resource list URIs, thus enabling SIP multicast.

Alice's RLS will forward the retrieved MESSAGE to each of Alice's friends' context providers. The first message is reached by Bob's trusted proxy, which extracts Alice's context data and sends a SIP SUBSCRIBE message to Bob's context provider (see Figure 56). After retrieving Bob's context update, Bob's trusted proxy will match Bob's context against Alice's interest and location, and send this result back to Alice's RLS in another MESSAGE.

Figure 56: The action of matching Alice's and Bob's context and interests and returning the matching result

RLS will wait for some (predefined) time for messages from all context providers, aggregate their matching results, and send them in the MESSAGE to sip:alice@example.com. This message will be intercepted by Alice's proxy, which will send to Alice a MESSAGE containing friends with matching interest or context and suggest her to call them (see Figure 57).

In this example, the match was found with Alice's friend Bob and Alice accepted to call him by replying to her proxy with another MESSAGE. Next, Alice's proxy sends first an INVITE to Alice's application and when Alice picks up the call, the proxy sends an INVITE to Bob (on behalf of Alice), which is intercepted by his proxy. This proxy sends SIP 100 Trying response back to Alice's proxy while it executes Bob's context-enabled CPL script. This CPL script specifies Bob's preferences for call logic which are context dependant, e.g., if Bob is in the meeting, redirect an incoming call for Bob to his voicemail. In this example, Bob's script allowed accepting an incoming call from Alice, this triggered forwarding of INVITE message to Bob. After Bob accepts the call and picks up the phone (completing the 3-way handshake), an end-to-end communication session is established between Alice and Bob.
5.6 Context-aware session adaptation

The scenario illustrating context-aware session adaptation was described in Section 1.1.1.4. In this Section we will briefly recapitulate this scenario and demonstrate in message sequence charts interactions between the system components in order to implement this functionality.

When a context change happens, such as: a change of Alice's location from the restaurant to the office and higher bandwidth becomes available - her preferred device changes from a mobile device to the desktop computer and her preferred communication means switches from audio to video calls. Alice's and Bob's context provider monitor their context and upon the change of Alice's context her context provider will send her a MESSAGE suggesting her to switch to a desktop device & start a video session (see Figure 58).

Alice will reply with a SIP REFER message to her proxy, indicating that she wants to be called on her desktop instead of her current device (this is realized by putting her new contact URI into the Refer-To field followed by method=INVITE), and setting Refer-Sub field to false in order to suppress an implicit subscription between the Refer-Issuer and the Refer-Recipient and the resultant dialog (as specified in RFC 4488 [107]). This will cause Alice's proxy to send an INVITE to her desktop and after establishing the session with this new device, terminate the session with her old device. After this call migration, Alice's proxy will send a REINVITE to Alice to establish a video session proposing an H.264 codec. If Alice accepts this SDP description and after sending her an acknowledgment, her proxy will propose the same codec to Bob in the REINVITE message. When Bob accepts this and after receiving an acknowledgement, a video session is established between Alice and Bob and context-aware session adaptation is completed. The call flow for such context-aware session adaptation is shown in Figure 58.
adaptation is shown in Figure 58. Note that SIP ACK is sent to Alice after a video session with Bob has been accepted and acknowledged as a signal that she can start a video call.

Alternatively, Alice could prefer to switch to a messaging mode instead of switching to a video call after changing the context. In that case the Messaging Session Relay Protocol (MSRP) would be used to establish a message stream in the same manner as audio or video session would be established via SIP; however, it would be using Session Description Protocol (SDP) description for MSRP media, as specified in RFC 4975 [108]. MSRP messages are transmitted as series of related instant messages in the context of a session. The difference in user experience between a telephone call and instant messaging is in that when an INVITE request arrives to an endpoint, it alerts a user with a ringing tone, waiting for a user input (i.e., to answer a call) before responding to it with 200 OK. However, in instant messaging an initial message will be displayed to a user as it arrives to an endpoint without waiting for this user to join the conversation, thus no "SIP 180 Ringing" is necessary in this 3-way handshake. After a messaging session is established, MSRP SEND requests are used to deliver messages (the complete messages or in chunks when messages are larger than 2048 bytes), while REPORT requests report on a status of a previously sent message. This is very important in case when a series of SEND requests contain chunks of a single message, in order for sender to know if the whole message has been successfully delivered. Figure 59 shows a call flow for instant messaging session between Alice and Bob.
5.7 Context-addressed messaging

This Section will demonstrate that even context-addressed messaging can be implemented using context trigger and context switch construct.

In another scenario that was initially described in Section 1.1.1.3 and is depicted in Figure 60, let us suppose that Alice's proxy did not find anyone with a current interest or activity in biking. Thus, her proxy subscribes to be notified when someone with this interest appears. This subscription should be for as long as this Alice's current preference for biking is active. Note that earlier, Alice uploaded her preferences to the broker in which she required the location of the sender to be sent in messages published on the biking topic.

Let us suppose that after some time Alice's friend Ted decides to go biking and sends a context-addressed message with invitation to all his nearby friends, who are currently biking in the same city. Ted's proxy will query his context provider for Ted's location and after adding this location information and Ted's anonymous address to the message, this proxy will publish the message on the biking topic. After receiving this message from the broker, Alice's proxy will perform context-based filtering, as illustrated in Figure 28. Since Alice's context matches the message's context address, and her preferences match Ted's preferences, this message will be delivered to Alice. After some time, Alice will reply to Ted that she will join him in biking, and this reply will traverse Ted's trusted proxy, which will allow this message reply (based on message topic identifier and sender address) to reach Ted's device.
In this sequence diagram, there is a context trigger initiated by Alice's preference update (i.e., interest in biking) upon the change of her current context (i.e., free time), which triggers Alice's trusted proxy to subscribe for notifications from nearby friends with a matching interest; and a context switch that upon retrieving the notifications applies context-based filtering based on Alice's current context as well as her preferences in the current context, in order to deliver the relevant messages in an appropriate way to Alice.

5.8 Summary

The contribution of this chapter is threefold. First, we propose a way to trigger communication between users based on the match of one user's interest against the other users' interest or current context, while respecting these users' privacy. Second, we illustrate how context information can be used to control adapt, modify, and manage the user's existing communication sessions according to his/her preferences in the user's current context. Third, we show how context information can assist, in case of an incoming communication event, in decision making about an appropriate context-dependent action on behalf of a user.

To achieve the first and the second functionality, a user needs to upload his/her context-dependent preferences at his/her trusted proxy, which activates a user's preference upon a particular context update. This preference in turn initiates an action, which in case of
communication initiated by a preference match, is a group query that is sent to other users in order to find those whose current interest or context matches the user's interest indicated in the query. Next, a user can choose to initiate a communication session with the other user that has the matching interest or current context. We designed a context trigger to implement the described functionality. A context trigger is, therefore, used to initiate an action (i.e., session initiation, adaptation, and termination) based on the context update and preference set in this updated context.

To implement the third functionality, we designed a context switch. A context switch represents a set of actions that a receiver takes upon an incoming communication event from a sender (e.g., a call or a message arrival). The receiver's current context is used to select an action from the specified (context-dependent) actions (i.e., whether to accept/reject the call, or forward it to a voicemail).

We implemented a context-switch by extending syntax of Call Processing Language (CPL) scripts with context parameters and built a context-aware VoIP prototype in order to demonstrate how easy it is to add new context parameters and how complex decision making criteria can be built using our solution. We utilized a scalable and reliable open source SIP platform, called SIP Express Router (SER), to upload and execute CPL scripts. It can act as a SIP registrar, proxy, or redirect server. We extended SER's functionality to support context-based CPL scripts. In this prototype the user's context is described with context parameters contained in an ontology file. Therefore, this ontology file needs to be uploaded before the user decides to upload a context-based CPL script. After it has been uploaded, this ontology file is parsed, and context parameter values extracted from this file are stored in the external MySQL database (that is also used by SER for storing users and CPL scripts). Next, these context parameter values are matched with the corresponding context values in the available context-based CPL scripts in order to determine which script describes rules for the current user's context. The relevant CPL script is then uploaded to the SER. SER will, upon receiving an incoming call or a SIP INVITE message from a SIP User Agent (SIP UA), load the user's current CPL script from the database and executes it. If the CPL script contains a context switch, it will match values set in script rules with the corresponding context values, and if they match, take appropriate actions.

We also evaluated the SER's response time when executing CPL scripts with increasing complexity (that is expressed in the number of switches in a CPL script). We wanted to compare the difference in time when executing standard CPL switches that read SIP header fields against our context switch that retrieves context parameters via an ontology. We tried to answer the following questions: what is the added delay and what is the cost of adding ontologies. We showed that adding context switches to a CPL script (up to 5 in total) increases the response time from 0.4 up to 2.3 ms, which corresponds to a 5%-24% response time increase due to the added reasoning and storage of context values in the database. Regarding scalability, some measurements were performed in [101] with 100 users sending simultaneously INVITE messages. In case of context-based CPL script, 1485 INVITE messages were successfully processed from around 1650 messages in total that SER has received, which corresponds to a 90% acceptance rate. The SER's total processing time (from the moment he received first message until he sent the last provisional response) was 12.3s, however some of the requests were not answered. In case of the conventional CPL script, 1979 INVITE messages were successfully processed from around 2000 messages received, which corresponds to 98.9% acceptance rate. The SER's total processing time was 15.1s. The SER's average response time was 7.4ms for both the conventional and context-based CPL script. Note that this happens because the rate at which the SER processes the requests is less than the rate of sending these requests, so that the queue becomes completely filled and some
packets get lost. However, in order to determine the SER's peak and average processing rate we need to perform more measurements in the non-saturated zone with varying rates of sending consecutive requests.

In our system NOTIFY messages are used as a means to deliver context updates. As CPL scripts are used to describe and control Internet Telephony Services and context trigger could influence the call processing, we decided to extend CPL to provide support for context triggers and group queries.

Next, we have examined the use of context-aware session control and context-addressed messaging on several examples, and have found out that both of these context-aware communication services can be implemented using these two types of constructs: context switch and context trigger.

Finally, we designed a system for context-aware session control on top of SIP network infrastructure and demonstrated using message sequence charts actions performed by the context-aware communication initiation and the context-aware session adaptation.

We plan to implement and evaluate the proposed system as part of our future work. The following open issues have been identified after this chapter:

- User's context-dependent preferences should be mapped by the system to our internal format of extended CPL scripts with context parameters and context trigger node before they are uploaded to the SIP proxy server. One way to solve this is to develop a graphical tool to enable a user to easy specify his/her preferences, similarly to CPLEd.
- We have not specified how a user could state which preferences are more important than others in order to make a better delivery decision. One possibility is to investigate use of multidimensional matrices to capture user's context-dependent preferences with weights representing their order of importance.
CHAPTER 6

CONCLUSIONS

This chapter describes conclusions of this thesis with respect to the problem statement, discusses open issues, and provides an outlook of what will be the next steps of the work presented in this thesis.

6.1 Conclusions

In this thesis we have presented context-addressed communication dispatch system that can be used for context-addressed messaging (i.e., to send messages to other people based on their context rather than their network address) and context-aware session control (i.e., to initiate, adapt, and terminate user's communication sessions based on this user's current context). Context-addressed messages are routed from the sender to the correct recipient(s) and delivered to their preferred devices, using their preferred communication means in these recipient(s) current context. This system also enables initiation of communication session among users based on their preferences and current context, taking the relation between these users into account. Therefore, it has access to the user's social relationship model as part of this user's context knowledge. Additionally, this system enhances the session initiation decision making process with the context information in order to route the incoming call to the callee's preferred device based on his/her current context. Based on the user-specified context-dependent preferences regarding the communication and content, the system can adapt, modify, and manage user's communication sessions or subscribe to a user's desired content upon the context update. This system enables a user to modify his/her preferences at any time during a communication session. The communication adaptation that is based on context is implemented in our system using two constructs: context switch and context trigger. Context switch selects a communication action from the set of context-dependent actions upon an incoming communication event based on the receiver's current context, whereas context trigger initiates a communication action based on the context update and preference that is set in this updated context. We designed this system on top of SIP and SIMPLE network infrastructure, by extending CPL (Call Processing Language), a language for describing and control of Internet Telephony Services, and demonstrated its use on several use case scenarios. We also implemented context switch by extending syntax of Call Processing Language (CPL) scripts with context parameters and built a context-aware VoIP prototype in order to demonstrate how easy it is to add new context parameters and how complex decision making criteria can be built using our solution. We evaluated the cost of adding context switches to a CPL script (up to 5 in total) is a 5%-24% response time increase due to the added reasoning and storage of context values in the database.

To be able to provide all context-aware communication to users, this system implements context management functionalities, thus it is able to timely discover and acquire raw context data from sensors, model this data as context information to be unambiguously interpreted by applications and system components, process this information into high-level context (i.e., synthesize context), and use this knowledge to enable context-addressed communication. It also understands user's context-dependent preferences, upon which change this system automatically selects and switches to user's optimal communication means and device in his/her current context. To perform context synthesis we have introduced a novel approach of context operators. Due to these operators benefits: simplicity that is achieved using the functional approach to context data, the operators reusability and flexibility (because they can be added or removed at any time during system runtime without changing the context
middleware source code), we used the same operators concept in forming context-based addresses. Thus, we have defined our own format for composing context-addressed messages, called Common Profile for Context-Addressed Messaging (CPCAM). The routing of these messages to the correct recipients is performed by matching of context specified in the address against the receiver's (high-level) context. The receiver's high-level context is computed by finding and invoking the appropriate implementations of operators specified in the message address on the receiver's context data. After determining if the receiver is the correct message recipient, the system evaluates, according to this receiver's preferences, if this message is relevant for this user in his/her current context, and if so, delivers this message to the user's preferred device using his/her preferred communication means in his/her current context.

We implemented our approach for context synthesis using context operators and evaluated its performance on the Nokia 7700 in terms of response time to context query sent by the application. We obtained a 2.5 seconds average delay, where 2 seconds were spent to perform operator matching. Note that these 2 seconds of delay are not suitable for applications that require to context synthesized from very volatile information whose value changes more frequently than once in two seconds or for mission critical applications that need to have reliable information (e.g., if some person's life is in dangerous). However, in our case, this context operator approach was used to develop a set of sports applications that were used during a live race at the Super Prestige Cyclocross in Gieten, Netherlands. These applications demonstrated the use of context synthesis to dynamically compose gaps and groups of cyclists in order to provide a nearly real-time virtual ranking service [52]. For this service, where the position of cyclists in a group was presented to the spectators every 4 seconds, the spectators have reported that this delay did not affect their "near real-time experience".

To provide context information from available sensors to the context-aware framework, we needed to discover sensors providing the desired type of context information and obtain this information. Before implementing this, we wanted to investigate whether it is more energy efficient for a mobile device to discover available sensors each time it arrives at a new location or to distribute the context knowledge that the device has already discovered and acquired itself to another device prior to coming at a new location. To achieve this, we examined the battery power consumed by context discovery vs. context distribution performed by Bluetooth and WLAN. The key result of this work was that it is more energy efficient to distribute context knowledge to other devices, than having each device learns this information itself. Moreover, multicast should be used for distribution of (discovered) context to interested context consumers.

Based on this decision, we have designed and implemented SIP-based multicast by allowing sensors to explicitly join and leave the multicast group that can be used for context distribution, group management, and group queries. By grouping the sensors providing the same type of context information we are able to provide event-notification service about the context changes and the sensors membership in the group. Additionally, group queries are sent to a group of user's contacts that have the same social relationship with the user in order to find the members of this group whose interest or context matches the user's interest and initiate communication with these matching group members. These group queries that are triggered by a context update represent at the same time our approach to implement context-based session initiation.

We believe that a proposed context-aware communication framework could enhance users' communication, by making it more personal and aware of user's surroundings, thus providing more chances for communication interaction with people that are in the same context and have the same interest as this user. This enhancement of users' communication is also
expected to be achieved by delivering only relevant messages and calls to the user, as well as discovering, selecting and switching to an optimal communication means and device to adapt the existing session with the user in his/her current context.

6.2 Open issues and future work

We are currently implementing this context-addressed communication dispatch system. After completing its implementation, we plan to perform a performance evaluation with regard to latency and scalability.

In the rest of this Section we describe the rest of open issues that could be part of future work (most of them were already identified at the end of each chapter):

- How should the user express his/her preferences? One way to solve this is to develop a graphical tool to enable a user to easily represent context-dependent preferences regarding preferred communication means, device, and interested content. Another option is that some of these preferences (such as the user's hobbies and free time activities) be imported and/or inferred from existing social networks, e.g., Facebook, MySpace, etc.

- These context-dependent preferences should be mapped to our internal format of extended CPL scripts with context parameters and context trigger node before they are uploaded to the SIP proxy server. We have not specified how a user could state which preferences are more important than others in order to make a better delivery decision. One possibility is to investigate use of multidimensional matrices to capture user's context-dependent preferences with weights representing their order of importance.

- The user should also be aware of context terms that are specified in the context model schema when he/she writes these preferences (in order to be context-dependent). The question that arises is how to disseminate the context model schema to users? Should this schema be part of the system delivery and can it be changed?

- We did not define in this thesis how topics for publish/subscribe system should (or even could) be specified and who decides which topics will exist.

- The trusted proxy has the functionality of an anonymizer, as it replaces the sender's actual address in a context-addressed message with a pseudonym. The question that remains to be answered is: should the proxy always anonymize the sender's actual address? When and when not?

- Another open question is what if the responder of the message also chooses to be anonymous? How should the reply messages access be configured in this case?

- To allow learning of users preferences, we should investigate a way to allow messages for which Alice was not subscribed, but that could potentially be interested to her to receive, to be delivered (if we use a publish/subscribe system)? The question that arises from this is: should we invent some new mechanism for subscribing to undefined topic (something similar to the use of wildcards?), but once user feedback is obtained then the new preference can either cause the trusted proxy to create this new topic and subscribe to it, or unsubscribe to this topic and create a negative preference instead? Finally, should we allow the user specify negative preferences? How should the user provide his/her feedback to the system and how to incorporate this feedback into the learning process?

- Context-based filtering is performed on the receiver's trusted entity. One should investigate where the preference learning should take place – in particular, how will
the observed behavior be logged, by which component, how often will it be analyzed and by which methods/tools? Can the user specify when it should not be logged?

- Similarly, the logging a user's daily communication data (for inference of user's social relationships) could be considered as a privacy issue, because a user might not want to log communication that is originating from or is destined to some of this user's private numbers. Therefore, the system should enable a user to specify the conditions when it should not log the user's communication. It should be investigated how to enable the user to specify such conditions.

- The learning process is usually related to the usability of the system, because the system needs some time for the learning curve before it can be used. In case of the user's social relationships inference, some of the user's social relationships could be extracted from existing sources (i.e., the user's email application, social networks, and instant messaging programs) and inserted into the system as an a priori knowledge in order to be used before the first social relationships are inferred. One should study, during the course of learning, the usability of the system as a function of the amount of a priori knowledge inserted into the system.

- Investigate what are the risks of a system unexpected, emergent behavior and how to deal with it.

- We defined context management as a set of activities starting from context sensing, context modeling, context synthesis, and ending with context distribution and querying. It should be studied how to perform and control these context management activities in a distributed manner.

- We evaluated the context synthesis approach using context operators in case the context information was available at the repository on a mobile device. We need to investigate if the real bottleneck of the context synthesis is in the operator matching procedure or the retrieval of context information from the remote sensors.

- The performance of context synthesis should also be improved by caching decisions made by the operator matching algorithm for a specific context query.

- Other issues, such as how to deal with context uncertainty and highly-volatile context data should also be investigated.
REFERENCES


[8] A. Devlic, "Context-addressed communication dispatch", a poster presented at Wireless@KTH, as part of the KTH Research Assessment Exercise, 26 June 2008.


[56] HTTP/1.1 Status Code Definitions, Section 10 of RFC 2616, Available at: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html, last accessed in April 2009.


[103] CPL Editor (CPLEd) 0.4.0, an open source tool for editing CPL scripts, http://mac.softpedia.com/get/Communications/CPL-Editor.shtml, last accessed in April 2009.


## List of Acronyms, Abbreviations, and Standards

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABNF</td>
<td>Augmented Backus-Naur Form</td>
</tr>
<tr>
<td>AoR</td>
<td>Address of Record</td>
</tr>
<tr>
<td>CC/PP</td>
<td>Composite Capabilities/Preference Profiles</td>
</tr>
<tr>
<td>CIS</td>
<td>Contextual Information Service</td>
</tr>
<tr>
<td>CPCAM</td>
<td>Common Profile for Context-Addressed Messaging</td>
</tr>
<tr>
<td>CPIM</td>
<td>Common Profile for Instant Messaging</td>
</tr>
<tr>
<td>CPL</td>
<td>Call Processing Language</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hyper Text Transfer Protocol</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>LQS</td>
<td>Location Query Service</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>MIME</td>
<td>Multipurpose Internet Mail Extensions</td>
</tr>
<tr>
<td>MMS</td>
<td>Multimedia Message Service</td>
</tr>
<tr>
<td>MSRP</td>
<td>Messaging Session Relay Protocol</td>
</tr>
<tr>
<td>PIDF</td>
<td>Presence Information Data Format</td>
</tr>
<tr>
<td>PIM-SM</td>
<td>Protocol Independent Multicast – Sparse Mode</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comment</td>
</tr>
<tr>
<td>RISP</td>
<td>Receiver Intent-based Sender Push</td>
</tr>
<tr>
<td>RLS</td>
<td>Resource List Server</td>
</tr>
<tr>
<td>RP</td>
<td>Rendezvous Point</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RPF</td>
<td>Reverse Path Forwarding</td>
</tr>
<tr>
<td>RPID</td>
<td>Rich Presence Information Data</td>
</tr>
<tr>
<td>RSS</td>
<td>Really Simple Syndication</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-time Transport Protocol</td>
</tr>
<tr>
<td>SDP</td>
<td>Session Description Protocol</td>
</tr>
<tr>
<td>SIMPLE</td>
<td>Session initiation protocol for Instant Messaging and Presence Leverage Extensions</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>SIRP</td>
<td>Sender Intent-based Receiver Pull</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>UAProf</td>
<td>User Agent Profile</td>
</tr>
<tr>
<td>UMD</td>
<td>Ubiquitous Message Delivery</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>XCAP</td>
<td>eXtensible markup language Configuration Access Protocol</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>eXtensible Stylesheet Language Transformations</td>
</tr>
<tr>
<td>XUI</td>
<td>XCAP User Identifier</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over IP</td>
</tr>
</tbody>
</table>
APPENDIX. PIM-SM: CALCULATION OF THE TIME TO REBUILD A MULTICAST DISTRIBUTION TREE

For calculation of time to build a distribution tree for multicast routing in the network, we will use the widely used Protocol Independent Multicast (PIM) protocol. IP routers use Reverse Path Forwarding (RPF) techniques to build a path from receiver to a source on which the multicast content will traverse through the network. PIM uses two types of distribution trees: "shared trees" and "source trees". A shared tree is built to be shared by multiple sources. Alternatively, a separate tree can be build for each source (called a source tree). Source trees use the most optimal paths (and least latency) for multicast traffic, whereas shared trees consume much lower router memory resources. Services and applications that use multicast can use either of them or a combination of both.

Considering PIM Sparse Mode (PIM-SM) as a multicast routing protocol (see Figure 61), we express the time to build a multicast distribution tree from a source to a receiver as: $T_{n,x}(\text{shared tree})=(n+x)*t$, where $x$ is a number of hops from receiver to the router acting as a Rendezvous Point (RP), $n$ is a number of hops between the source and the receiver on the path that includes the RP, and $t$ is the average transmission time per hop in LAN. We obtained $n+x$ by summing up $x$ join messages from the receiver to RP, 1 multicast packet from the source to its router, $n-x-1$ register messages containing this multicast packet from the source's router to RP, and $x$ forwarded multicast packets to the receiver.

![Figure 61: PIM-SM building a multicast tree](image)

- **a)** When receiver H2 joins the multicast group, join message is propagated to RP,
- **b)** Source sends multicast packet which is received by R1. R1 unicast-encapsulates this packet into the register message and sends it to RP. Upon receiving the register message, RP decapsulates it and forwards this packet into the tree.

Note that in the phase two of PIM-SM, in order to avoid expensive operations of encapsulating and de-capsulating packets, RP will choose to switch to native forwarding of packets. Therefore, RP will send a join message to the source. When this join reaches R1, it will start sending native packets to RP. While in the process of joining the source-specific
tree, the native packets will flow to RP along with the encapsulated register messages. At this point, RP will start discarding the duplicate packets and it will send a register-stop message to the R1 to prevent it from unnecessary encapsulating the packets. However, as this phase is not part of the multicast tree construction, therefore we do not use it in our calculation of shared tree.

For this discussion we will assume that $t$ is equal to $4\mu s$ (taken based on an average per hop delay on a 1Gb/s Ethernet link). Optionally, when data to receivers exceeds a threshold, routers can switch to a source tree. In this case we express the time to build a distribution tree as: $T_{n,x}(\text{switch to source tree}) = (2n' + x - 3) \cdot t$, where $n'$ represents a number of hops from the receiver to a source directly (omitting the RP). We obtained $2n' + x - 3$ by summing up $n'$-2 join messages from the receiver's router to the source's router, $n'$ multicast packets, and $x$-1 prune messages. If we assume that $n'$ equals $n$-1, then $T_{n,x}(\text{switch to source tree}) = (2n + x - 5) \cdot t$. For $n=5$ and $x=3$, $T_{n,x}(\text{shared tree}) = 32\mu s$ and $T_{n,x}(\text{switch to source tree}) = 32\mu s$. For $n=100$ and $x=50$, $T_{n,x}(\text{shared tree}) = 600\mu s$ and $T_{n,x}(\text{switch to source tree}) = 980\mu s$. Note that this is the time needed by a receiver to join a multicast group.

![Diagram](image)

**Figure 62: PIM-SM – Switching to source-based tree:** a) R3 sends an explicit join message towards the source and b) When data arrives from a source at R3, it sends a prune message to the RP

Next, we calculated the time needed by a receiver to leave a multicast group (we call this leave latency). When a router receives an IGMP leave report, this means that at least one host wants to leave a multicast group. Upon receiving this leave report, the router checks if the interface is not configured for IGMP Immediate Leave (i.e., immediate removal of the host from the multicast group). If the host should not be immediately removed, then the router sends a group-specific query in order to learn if there are still hosts interested in the particular group. This query indicates that hosts who are still joined to this group should respond within the maximum response time (set by default to 1sec). To compensate for the packet loss, the router will wait for this maximum response time to expire and if no one responds during that
time, it will repeat this process. After another maximum response timeout expires, the router will learn that there are no more hosts interested in this group and will stop the multicast traffic. Thus, it waits some additional time, approximately 0.5 seconds and finally removes IGMP state for the group. Therefore by default, the leave latency after the router receives the leave report is $2 \times 1\text{sec} + 0.5\text{sec} = 2.5\text{sec}$, before stopping the multicast traffic flow.