Context-addressed communication dispatch

ALISA DEVLIĆ

KTH Information and Communication Technology

Licentiate Thesis in
Communication Systems
Stockholm,Sweden 2009

KTH Information and
Communication Technology

Context-addressed communication dispatch

ALISA DEVLIĆ

Licentiate Thesis in
Communication Systems
Stockholm, Sweden 2009

TRITA–ICT–COS–0902
ISSN 1653–6347
ISRN KTH/COS/R--09/02--SE

KTH Communication Systems
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie licentiatexamen 29 May 2009 klockan 13.00 i sal C2, KTH-Electrum, Kungliga Tekniska Högskolan, Isafjordsgatan 22, Kista.

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

TABLE OF FIGURES

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.

1.1.1.1 *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' 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.

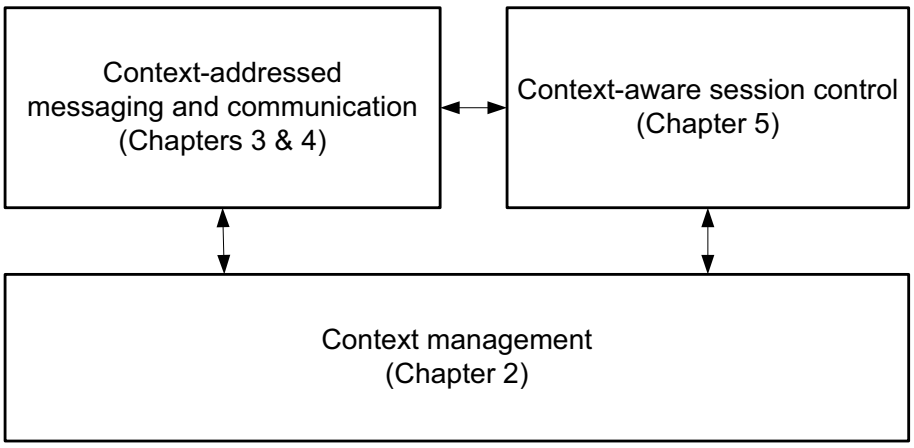


Figure 1: Main components of context-addressed communication dispatch system

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.

[1] A. Devlic, "Extending CPL with context ontology", In Mobile Human Computer Interaction (Mobile HCI 2006) Conference Workshop on Innovative Mobile Applications of Context (IMAC), Espoo/Helsinki, Finland, September 2006.

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.

[2] A. Devlic and E. Klintskog, “Context retrieval and distribution in a mobile distributed environment”, Third Workshop on Context Awareness for Proactive Systems (CAPS 2007), Guildford, UK, June 2007.

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.

[3] A. Devlic, M. Koziuk, and W. Horsman, "Synthesizing context for a sports domain on a mobile device", In Proceedings of the 3rd IEEE European Conference on Smart Sensing and Context (EuroSSC 2008), Zurich, Switzerland, Springer-Verlag, LNCS 5279, October 2008

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.

[4] A. Devlic, A. Graf, P. Barone, A. Mamelli, and A. Karapantelakis, “Evaluation of context distribution methods via Bluetooth and WLAN: Insights gained while examining Battery Power Consumption”, In Proceedings of the Fifth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2008), Dublin, Ireland, July 2008.

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.

[5] A. Devlic, A. Graf, P. Barone, A. Mamelli, and A. Karapantelakis, "Ad hoc context distribution methods using Bluetooth and WLAN", IC@ST magazine, ICST, 1st online edition, November 2008.

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

[6] A. Devlic and G. Panagiotou, "Context Distribution using SIP-based multicast", submitted for publication.

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].

[7] A. Devlic, R. Reichle, M. Wagner, M. Kirsch Pinheiro, Y. Vanrompay, Y. Berbers, and M. Valla, "Context inference of users' social relationships and distributed policy management", In Proceedings of the 6th IEEE Workshop on Context Modeling and Reasoning (CoMoRea) at the 7th IEEE Conference on Pervasive Computing and Communications (PerCom'09), Galveston, Texas, March 2009, pp. 755-762.

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 $user_1$, 85% for $user_2$, and 69% for $user_3$, 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.

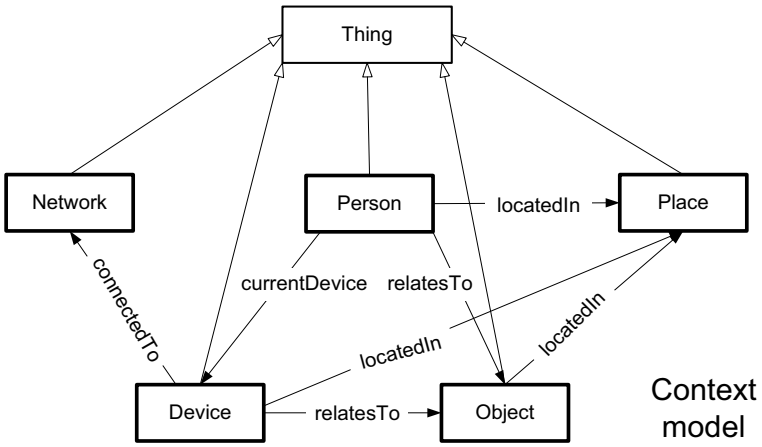


Figure 2: Context entities and their relations

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.

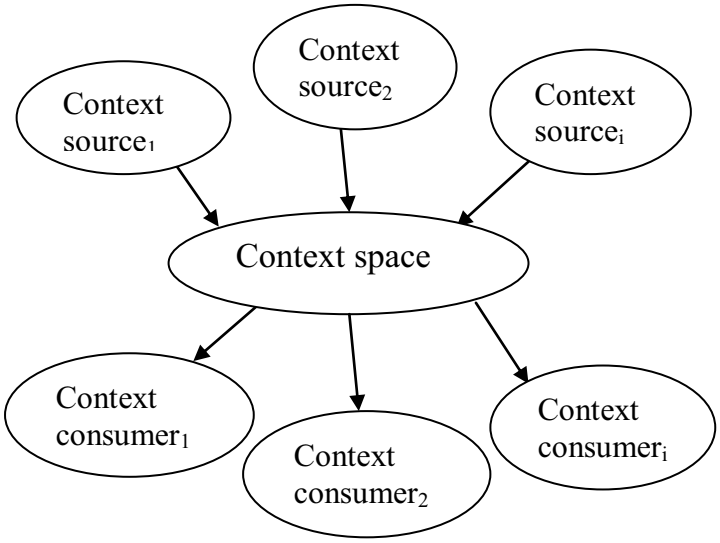


Figure 3: Context-aware system consisting of the following entities: context sources, context space, and context consumers

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:

$$Energy\_per\_transferred\_user\_data\_bit[\frac{J}{bit}] = \frac{P_{FT}[W]}{file\_transfer\_data\_rate\,[bit/s]} = \frac{\sum_{i=1}^{n}\left(U_{FT_i} * I_{FT_i} - U_{offFT_i} * I_{offFT_i}\right)}{\sum_{i=1}^{n} file\_size_i / T_{FT_i}} \quad (1)$$

where U_{FT_i} and I_{FT_i} represent the battery voltage and current values obtained during a single file transfer to a device, U_{offFT_i} and I_{offFT_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_{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 as the cost of WLAN distribution – 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:

$$Energy\_per\_received\_user\_data\_bit[\frac{J}{bit}] = \frac{E_{RMF}[J]}{received\_file\_size[bit]} = \frac{\sum_{i=1}^{n}\left(U_{RMF_i} * I_{RMF_i} - U_{offRMF_i} * I_{offRMF_i}\right) * T_{RMF_i}}{\sum_{i=1}^{n} received_{file_{size\ i}}} = \frac{1.33\mu J}{bit} \quad (2)$$

Note that E_{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.,

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.

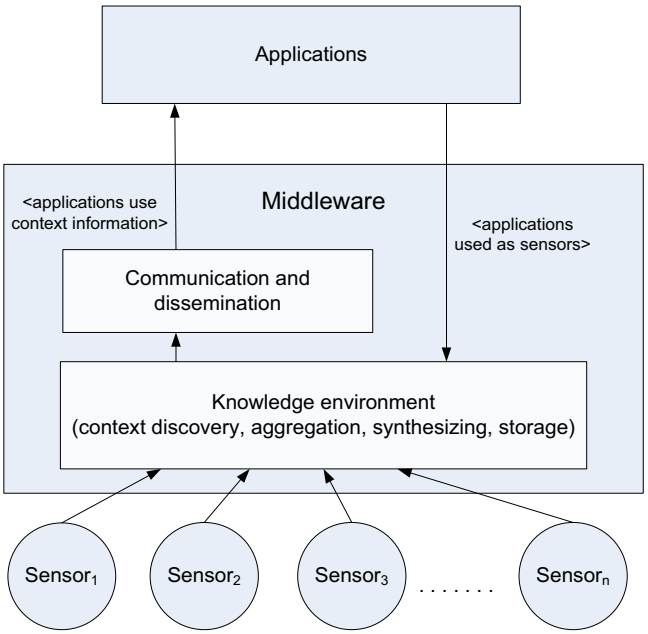


Figure 4: Context-aware system

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.

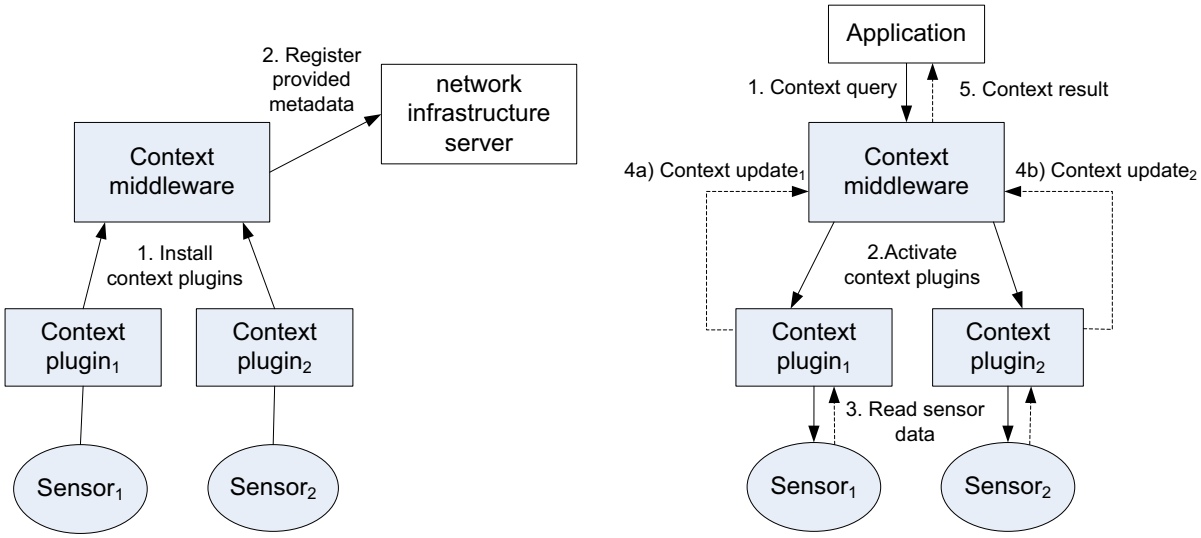


Figure 5: Context plug-ins – installation of context plug-ins into context middleware and registration of metadata provided by these plug-ins to a network infrastructure server (on left) and activation of context plug-ins upon arrival of context query (on right)

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*

(SGML) [26], 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.

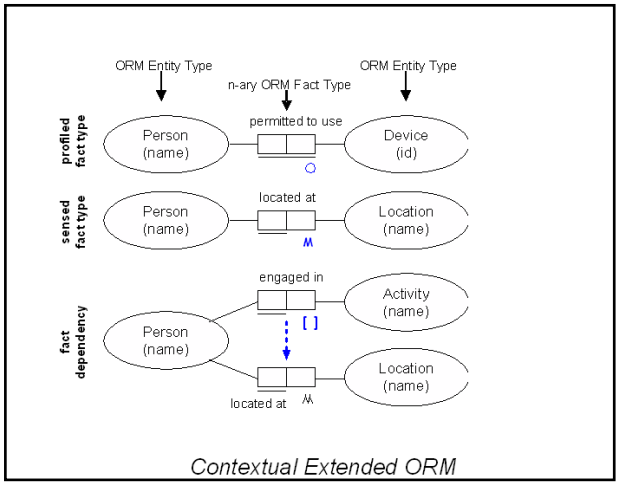


Figure 6: Contextually extended ORM [24]

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.

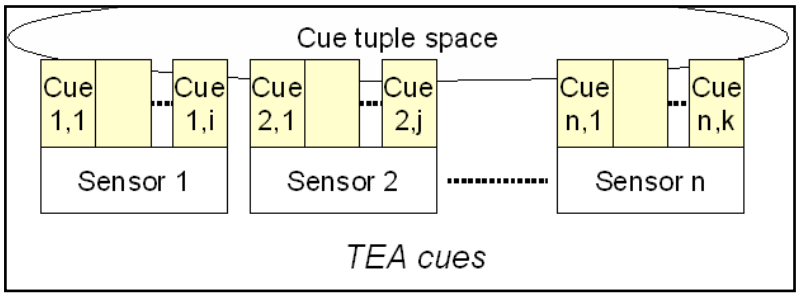


Figure 7: Cues [24]

The context is modeled on top of cues, as a two dimensional vector consisting of a symbolic value describing situation and a number indicating the certainty that the user (or device) is currently in this situation. The finite set of symbolic values is defined. The cues are objects that provide the contextual information through their interfaces, hiding the details of determining the output values.

Another approach is the *Active Object Model* that uses active objects to encapsulate data (fusion of HTML based packets of information) to produce contextual knowledge (dynamically composed required HTML pages). In the example of the castle [33], packets of

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: *ist (c,p)*, which asserts that the proposition *p* is true in the context *c*.

Giunchglia'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.

$$S_1 = [\dot{s} \mid \dot{s} \models\!\ll bird, \dot{a}, 1 \gg]$$
$$S_2 = [\dot{s} \mid \dot{s} \models\!\ll flies, \dot{a}, 1 \gg]$$
$$B \models\!\ll present, air, 1 \gg \wedge \ll penguin, \dot{a}, 0 \gg \wedge \ldots$$
$$C = S_1 \Rightarrow S_2 \mid B$$

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.

Requirement / Approach	Distributed Composition	Partial Verification	Richness & Quality of Information	Incomplete-ness & Ambiguity	Level of Formality	Applicabity to existing frameworks
Key-Value Models	-	-	--	--	--	+
Markup Scheme Models	+	++	-	-	+	++
Graphical Models	--	-	+	-	+	+
Object Oriented Models	++	+	+	+	+	+
Logic Based Models	++	-	-	-	++	-
Ontology Based Models	++	++	+	+	++	+

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 the 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 in 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 **Op** be a set of context operators:

$$\mathbf{Op}=\{op_1, op_2, \dots, op_n\}, n \in N$$

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

$$op_i=\{\mathbf{desc}(op_i), \mathbf{impl}(op_i)\}$$

An operator's description $\mathbf{desc}(op_i)$ is defined as:

$$\mathbf{desc}(op_i)=\{name_i, In_i, Out_i, Uses_i\}$$

where:

- $name_i$ is the name of op_i
- In_i is a list of types of inputs that will be provided to op_i
- Out_i is the type of output produced by op_i
- $Uses_i$ is a list of other (simpler) operators op_i used in its execution

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

An example implementation of the operation F_i is the program F(In), shown in Listing 1. The program takes a list of inputs, here represented by variables *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*).

```
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 "*CylistsInRange*" 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.

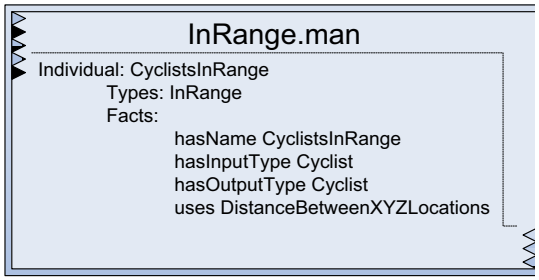


Figure 10: CyclistsInRange description

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 cyclo-cross 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.

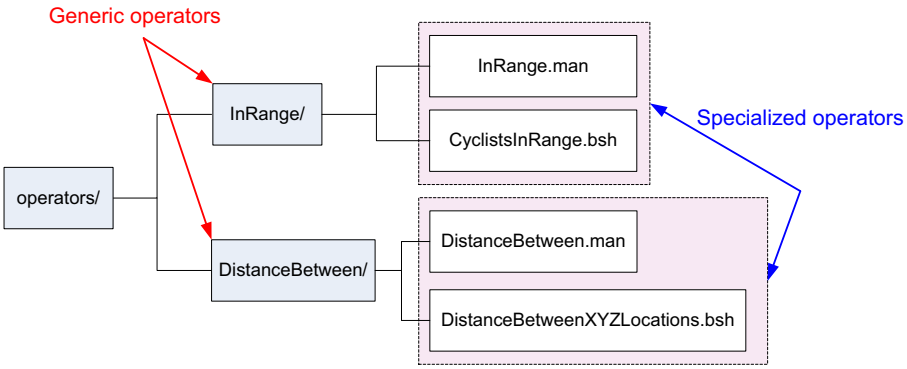


Figure 11: Operator space file structure

Note that in our design all the generic operators are directly sub-classed from the class Operator and specialized operators are individual instances of these generic operators. Therefore, the parser does not need to resolve a deep class hierarchy in order to find the generic operator classes that a specialized operator might be an instance of.

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: *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. *UsersInRange*, which would return *Users* instead of *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 *DomainInstance* objects. Each of *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 *DomainClass* and *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 *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.

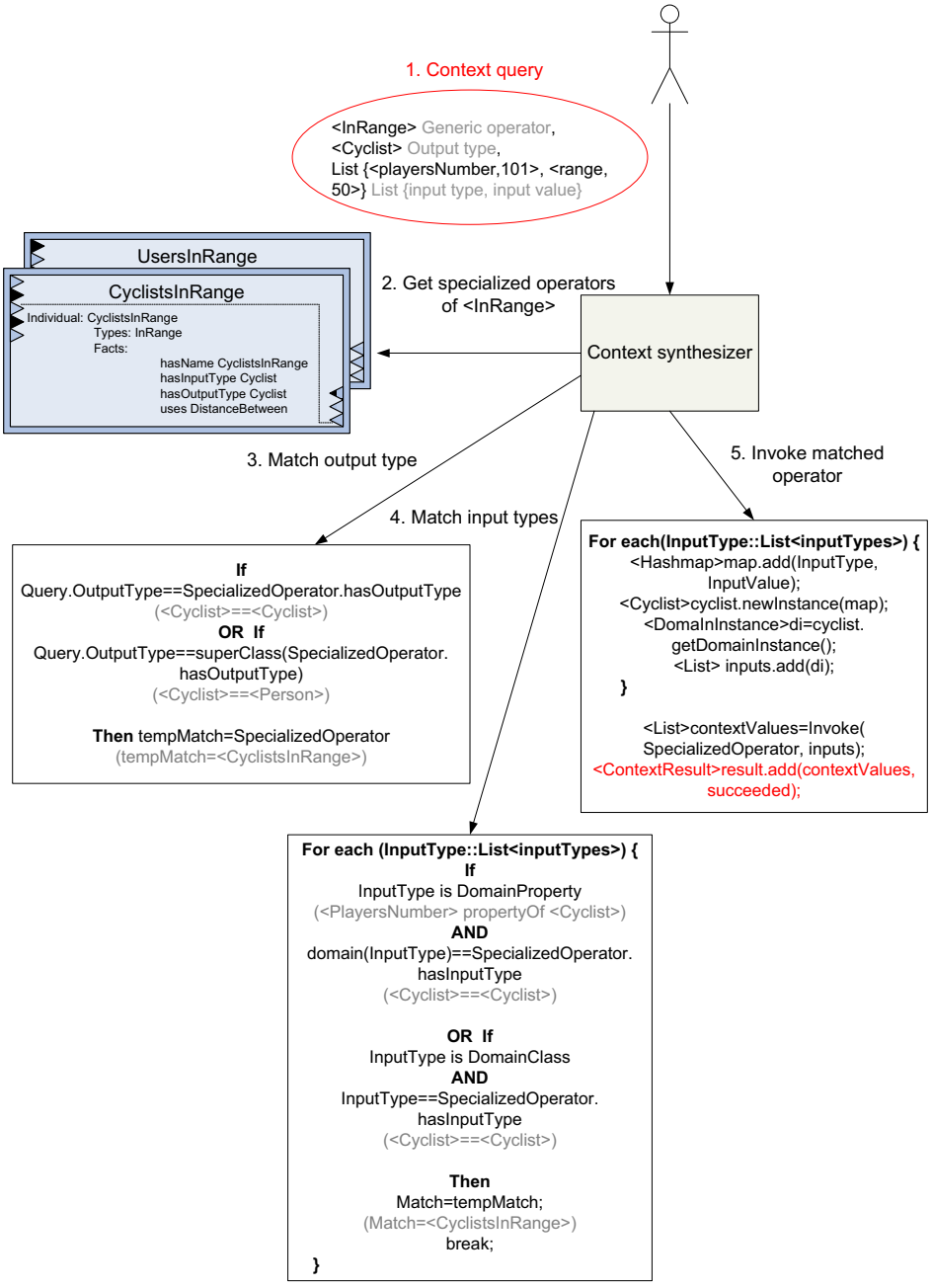


Figure 12: This figure shows the algorithm itself, initiated by the user's context query, along with the invocation of the matched (specialized) operator.

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.

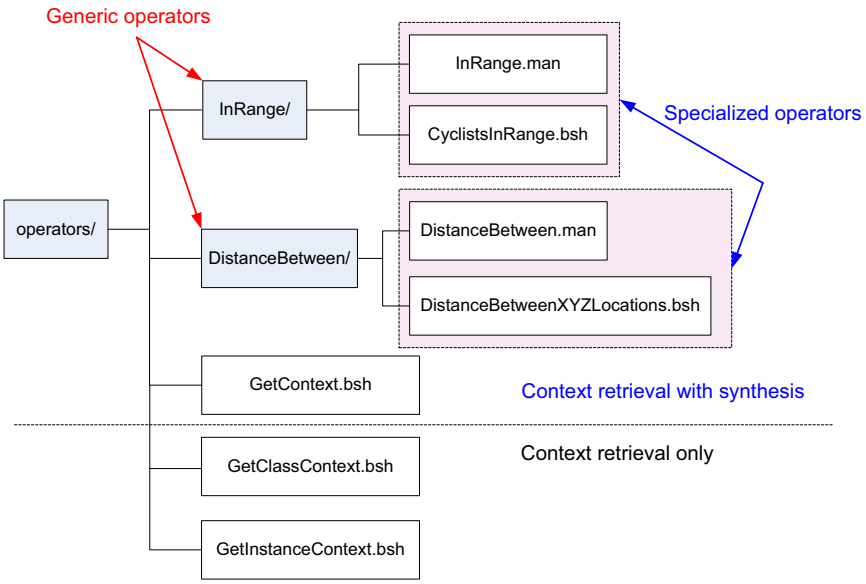


Figure 13: Operator space file structure designed for the purpose of MIDAS project

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 1: Response times

Average response times with varying number of specialized operators (i.e., 1, 2, 5, 10)	Based on 10 **first** queries	Standard deviation (based on 10 **first** queries)	Based on 10 **subsequent** queries	Standard deviation (based on 10 **subsequent** queries)
Matching algorithm time	2.49 sec	0.009 sec	1.94 sec	0.07 sec
Loading specialized & root scripts time	1.7 sec	0.087 sec	No average, for the first time only (1.7 sec)	No standard deviation
Total operator matching time	4.2 sec	0.087 sec	1.94 sec	0.07 sec
Context retrieval time	0.37 sec	0.006 sec	0.09 sec	0.001 sec
Loading dependency scripts time	0.15 sec	0.001 sec	0.17 sec	0.015 sec
Operator invocation time	0.67 sec	0.008 sec	0.36 sec	0.04 sec
Total query time	5.4 sec	0.045 sec	2.57 sec	0.07 sec

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.

Note also that in some other scenario it could happen that after the second query the first query is made again but containing some other operator, this will also require operator matching. However, I plan (as future work) to introduce caching of queries and matched specialized operators in order to reduce the total query time.

A set of sports applications was developed in the scope of MIDAS project that were using my context synthesis solution. A scenario based on a live race at the Super Prestige Cyclocross in Gieten, Netherlands 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]. There were 1000 spectators along the race course. Note that this deployment was intended as a proof of concept to validate MIDAS platform functionalities and was not designed to be an evaluation of the system using a statistically significant number of users. However, the impression of 9 users (monitoring the race on 6 tablet PCs and 3 Nokia N800 devices) was very positive. A few seconds of delay did not affect their "near real-time experience".

2.3.4 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: *InRange("ID01", 50, ModelConstants.Cyclist), 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: *TimeLimitQuantifier* (to set the maximum time limit on waiting for context response), *AccuracyQuantifier* (to set the request for the minimum accuracy of the context information), and *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 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 GenericOperator class and the output type from the ModelConstant class. Note that ModelConstants class is generated offline from the context model containing names of its domain classes and properties as fields in the ModelConstant class. Next, list of input types and values are added to the query, by selecting the context class from the ModelConstants and assigning it a value.

```
   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);
```

Listing 2: Context query creation

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

```
   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.

```
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.

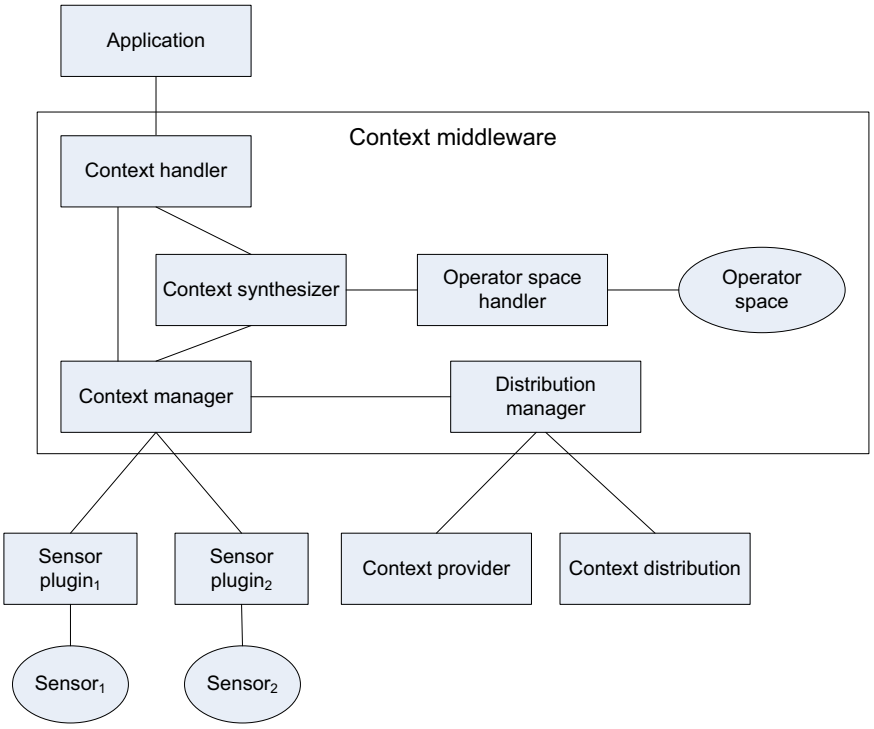


Figure 14: Context middleware in interaction with applications, sensors, context provider, and context distribution

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.

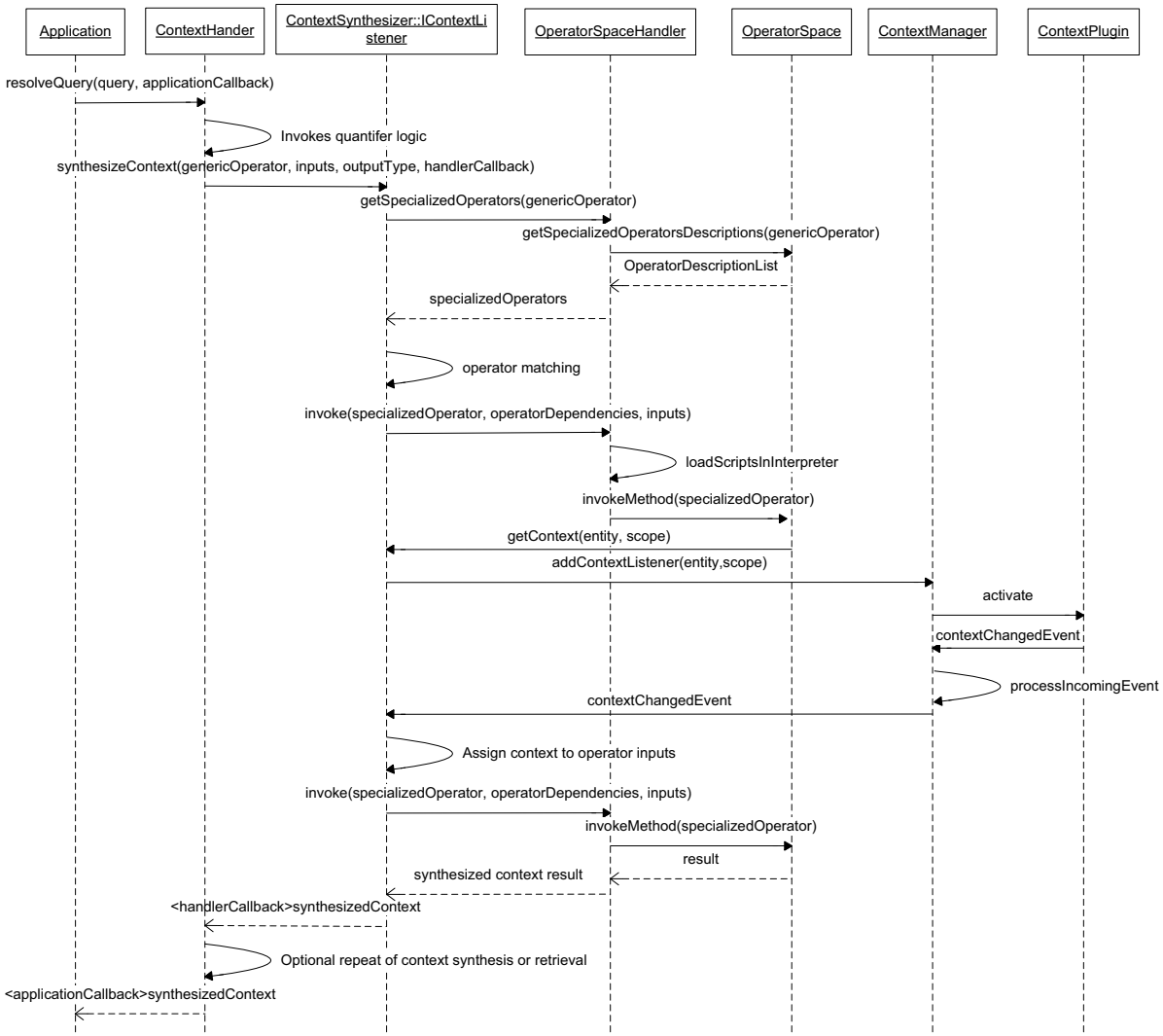


Figure 15: Application retrieving high-level context information using context synthesis and retrieval of low-level context data from context plug-ins available on the same device

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.

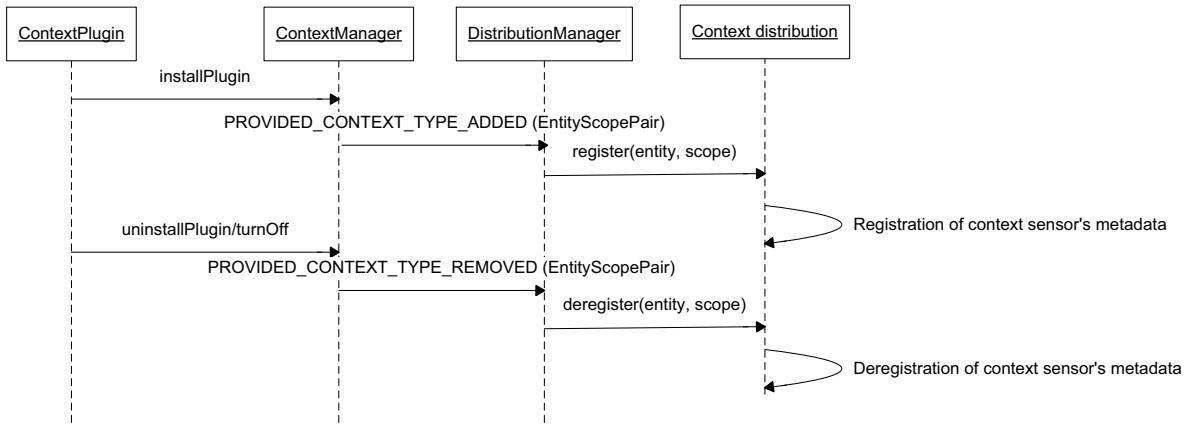


Figure 16: Installation and un-installation of context plug-ins, triggering registration and deregistration of their provided metadata to the network

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).

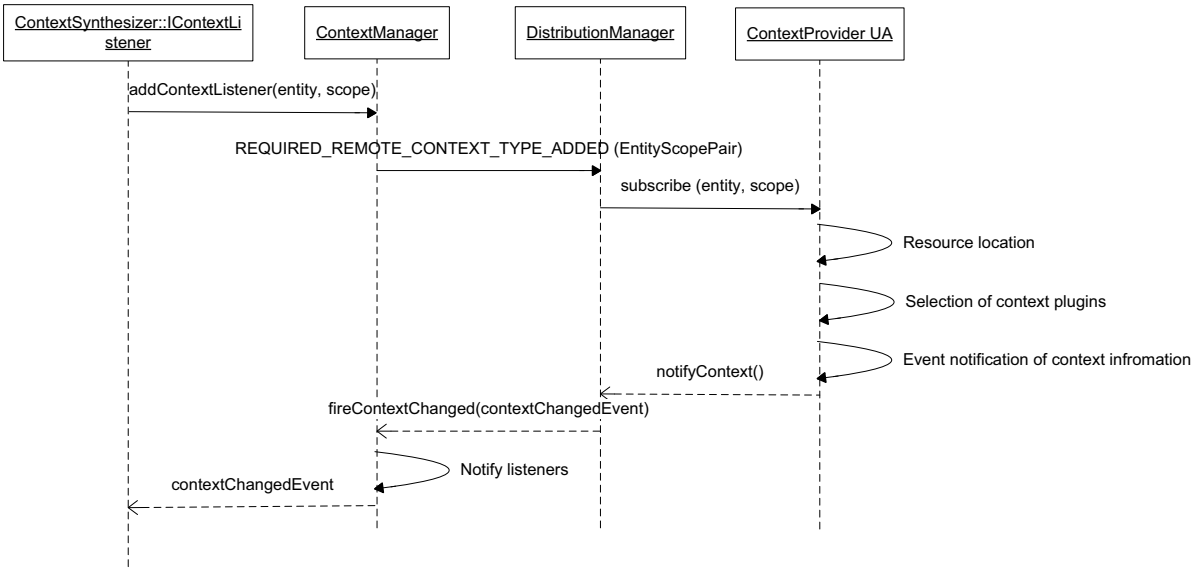


Figure 17: Context synthesizer retrieves missing context information from context plug-ins available on remote nodes in the network

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_REMOTE_CONTEXT_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.

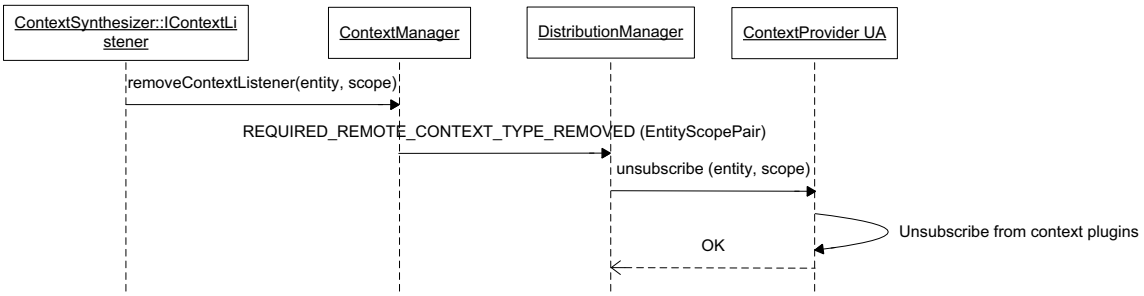


Figure 18: Remote context information is no longer needed, removing context listener and unsubscribing from context plug-ins

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 one. 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.

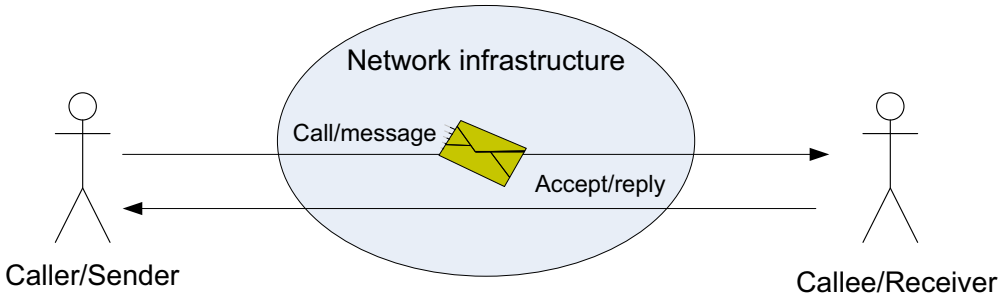


Figure 19: Communication model

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.

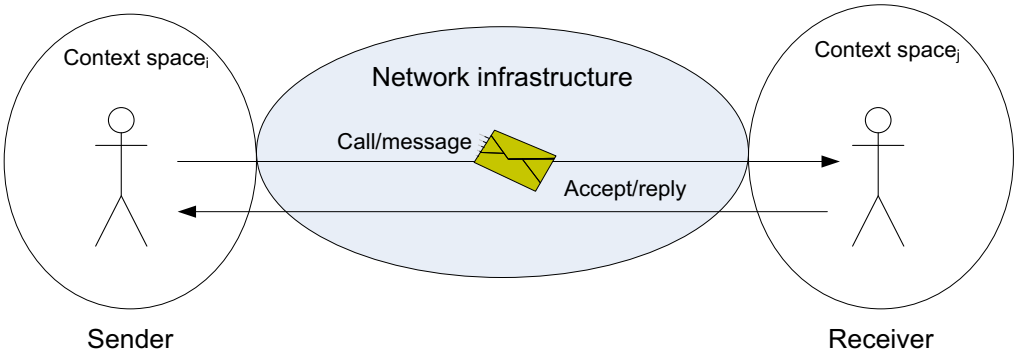


Figure 20: Introducing a context space around communicating entities.

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.

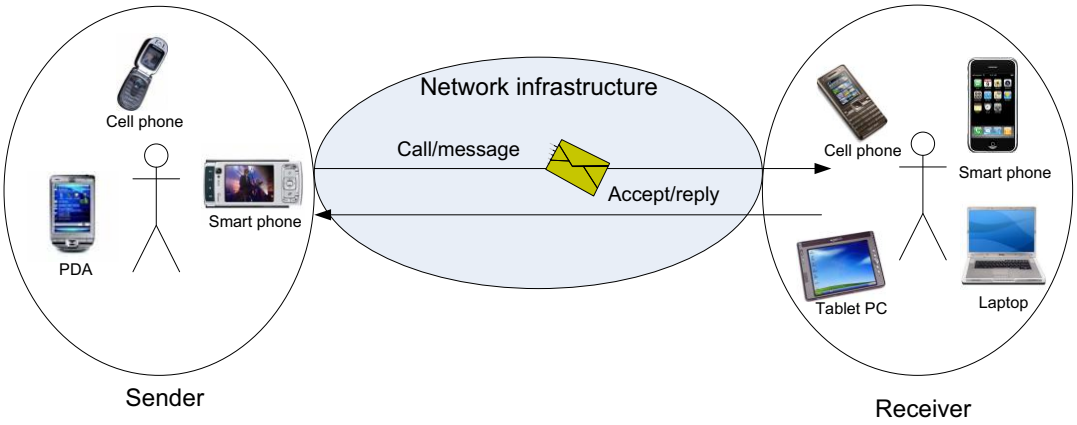


Figure 21: Communicating entities can employ multiple devices for communication

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.

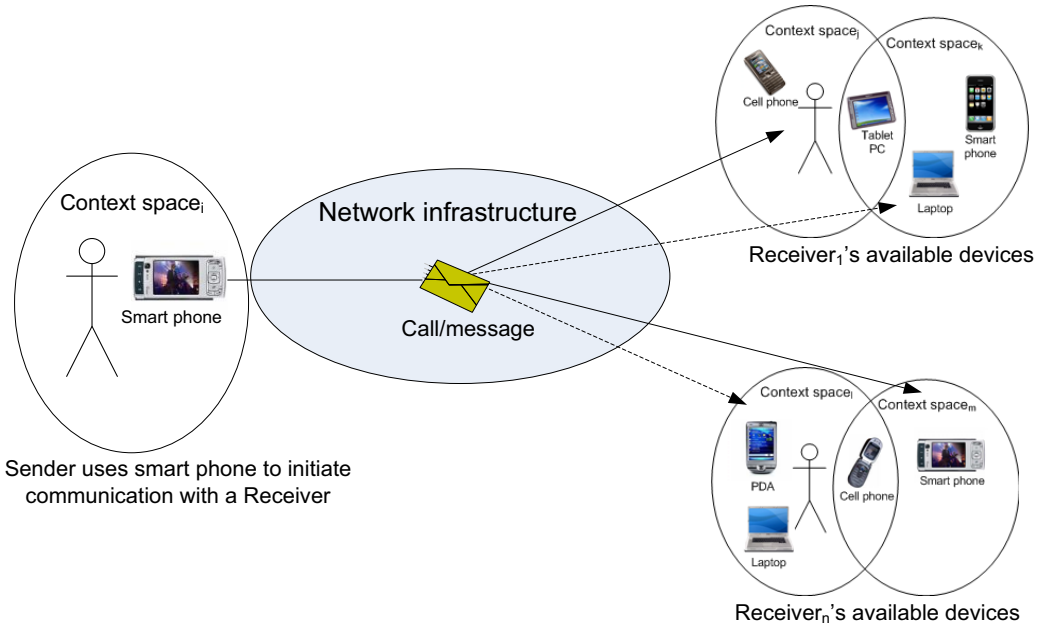


Figure 22: Communicating entities use their preferred devices in each Context space

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.

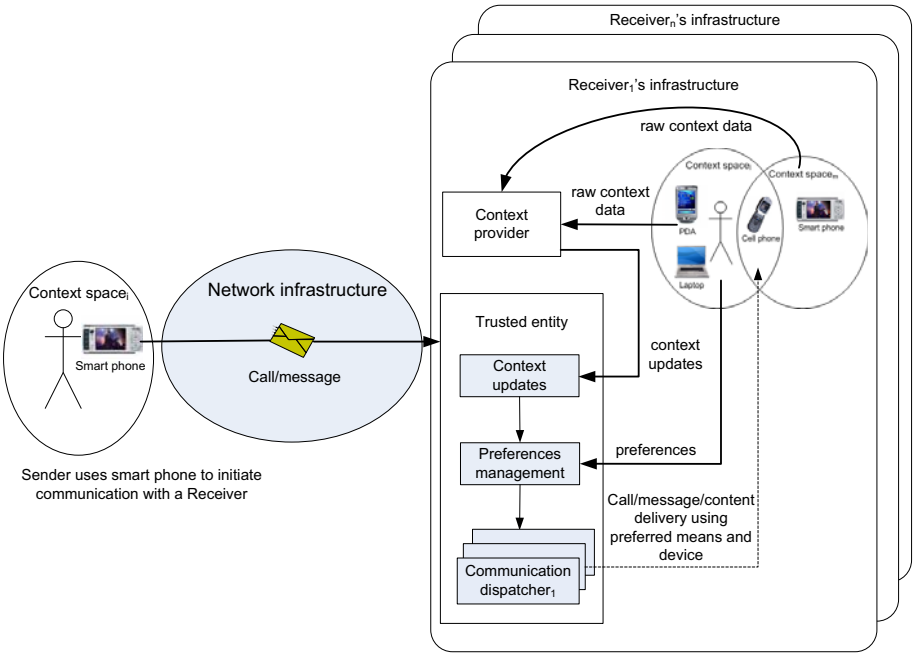


Figure 23: Receiver's infrastructure for context-aware communication.

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

Context-addressed message requirements	Message delivery modes			
	Sender-push	Receiver-pull	SIRP	RISP
1:N communication	+	X	X	+
Receiver's control over message delivery	X	+	+	+/-
Asynchronous model	+	X	+/-	+

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.

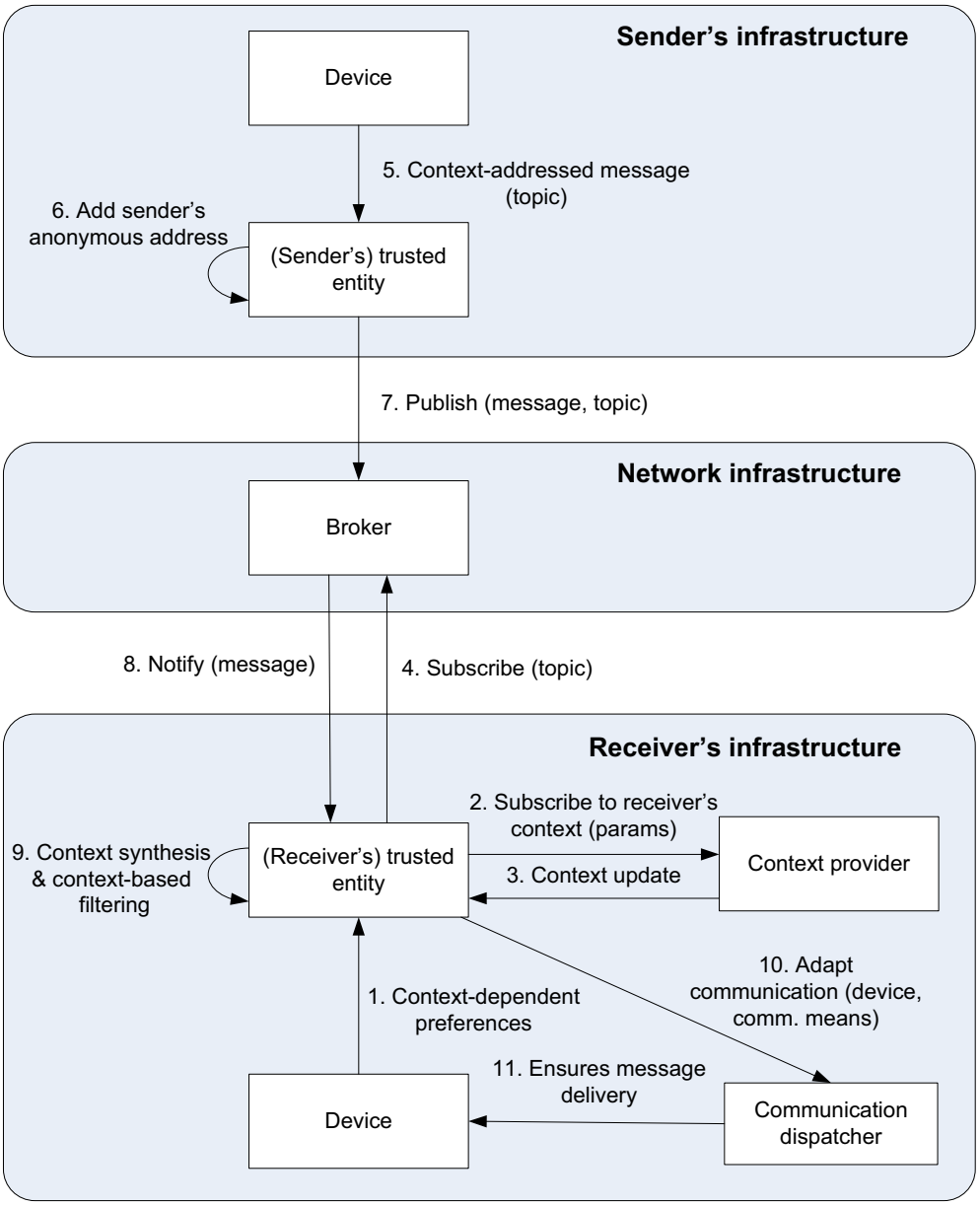


Figure 24: Context-addressed messaging mechanism

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
To: <sip:traffic@example.com>
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.

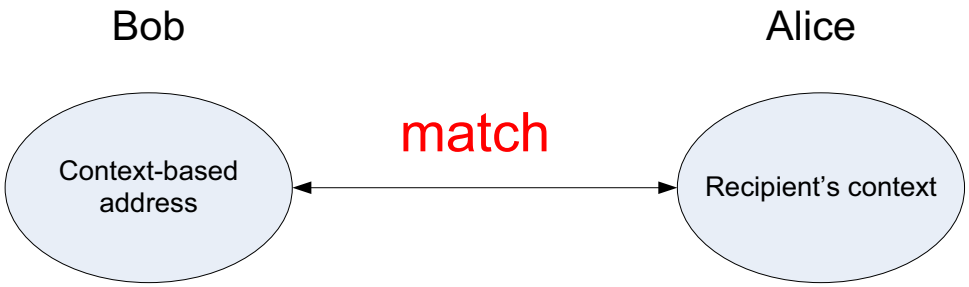


Figure 27: Context-based matching in case of context addressed message arrival

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.

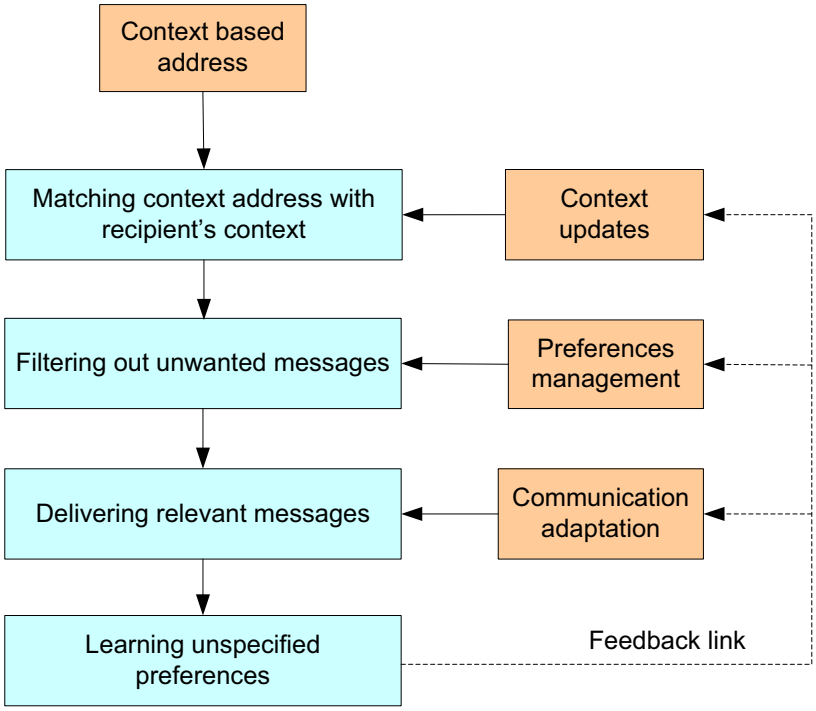


Figure 28: Context-based filtering

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.

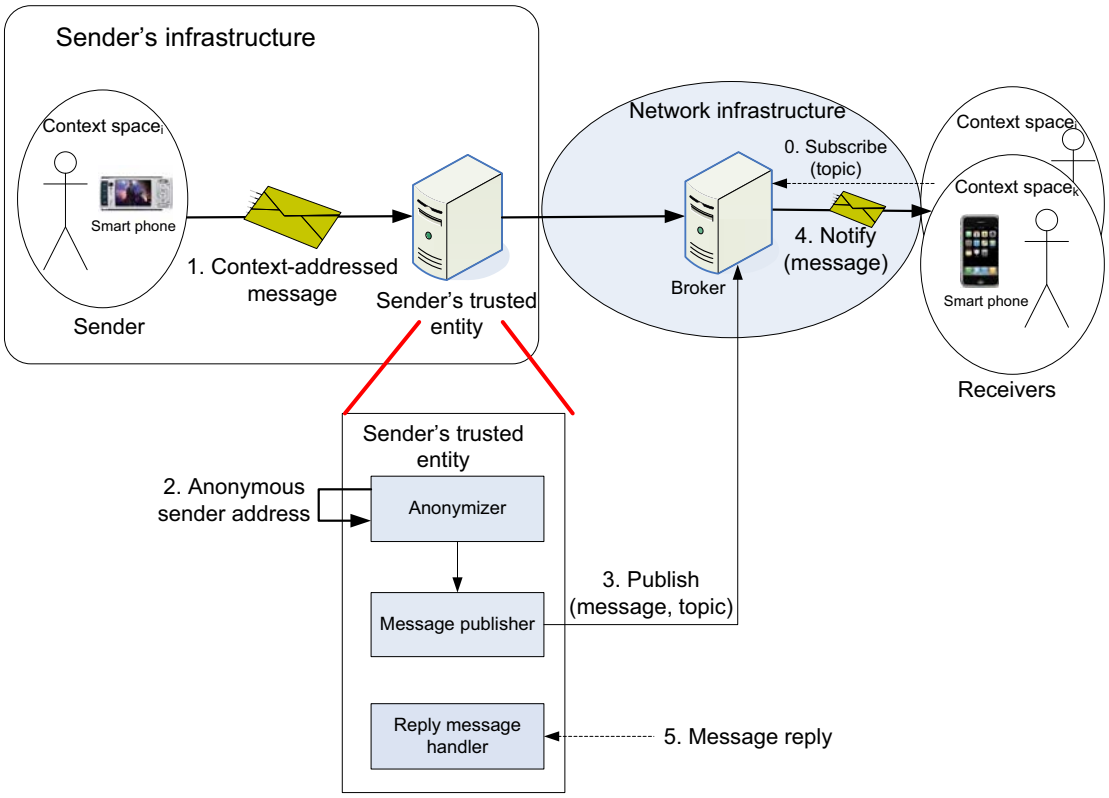


Figure 29: Sender's infrastructure for context-addressed messaging

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.

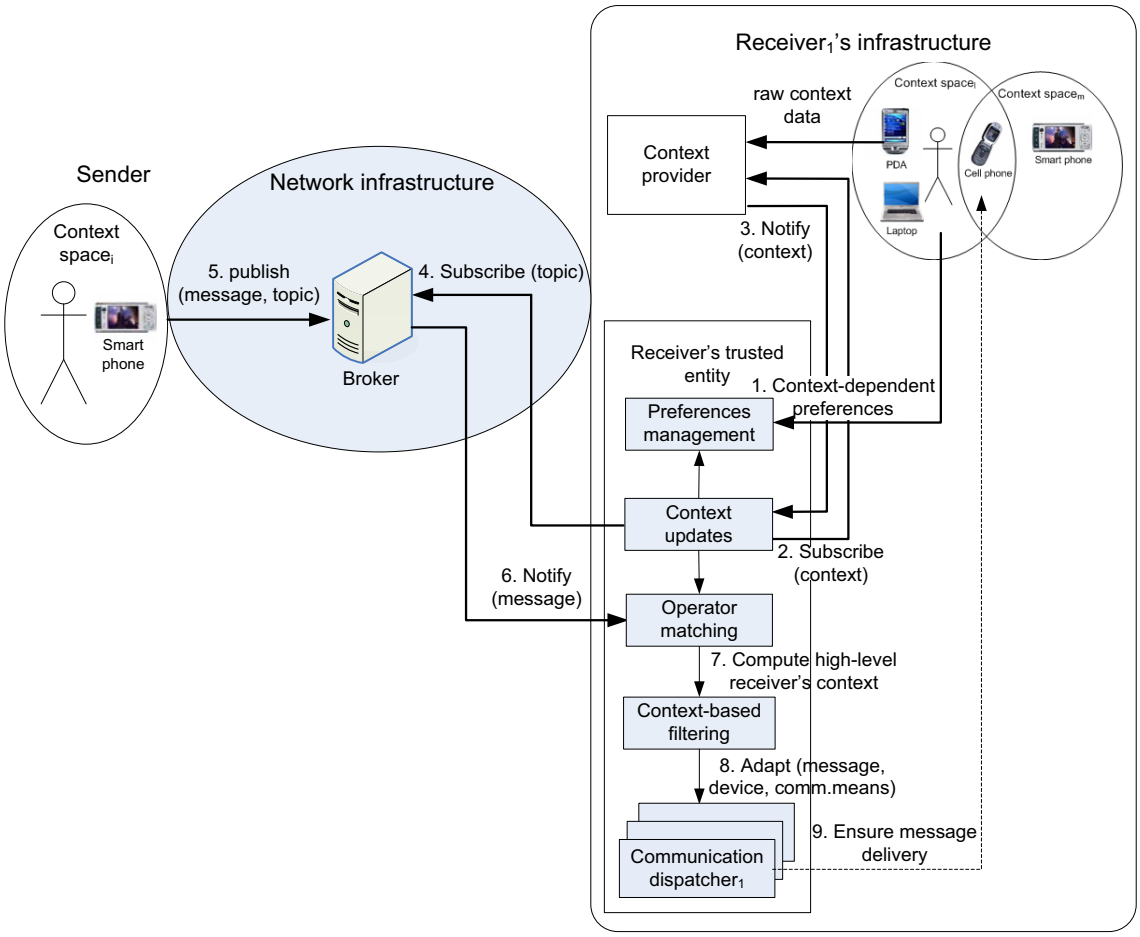


Figure 30: Receiver's infrastructure for context-addressed messaging

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.

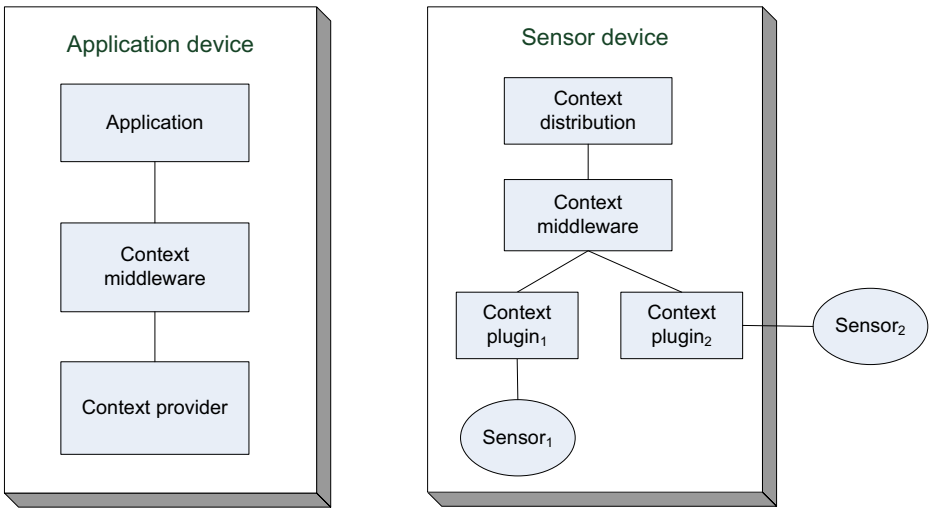


Figure 31: Application and sensor device configurations with appropriate components

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.

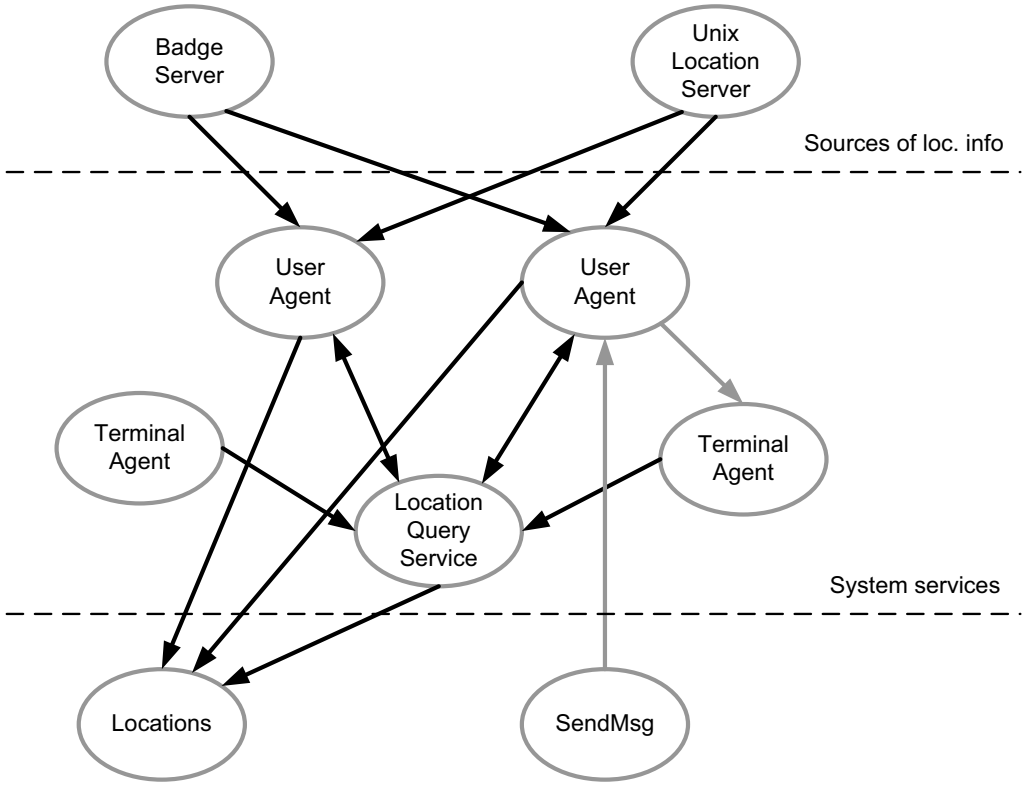


Figure 32: Basic system architecture, with an instance of UMD message delivery illustrated by gray arrows [69]

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 240µs 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:

$$T_{n,x}(\text{shared tree})=(n+x)*t \quad (3)$$

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=4μs). 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:

$$T_{n,x}(\text{switch to source tree})=(2n'+x-3)*t \quad (4)$$

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:

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

For n=5 and x=3, $T_{n,x}$(shared tree)=32μs and $T_{n,x}$(switch to source tree)=32μs. For n=100 and x=50, $T_{n,x}$(shared tree)=600μs and $T_{n,x}$(switch to source tree)=980μs. 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:

$$r_{receiverInterest}=1/T_{n,x} \quad (6)$$

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., $T_{delivery}=T_{routing}+T_{resolve}(X)$. 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, $T_{resolve}(X+Y) = k(size_{T\text{-}Box})*T_{resolve}(Y)$, where $k(size_{T\text{-}Box})$ 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 $T_{routing}=f(Z)=f(g(N))$, 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 $T_{delivery}=T_{routing}+T_{resolve}(X)=F(N, X)$, 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 $R=1/T_{delivery}$, 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 presentity) to interested parties (called watchers) [86]. Presence information is characterized by a set of attributes that characterize the availability and willingness of a presentity to communicate across a set of devices. Examples of presence information are status, capabilities, communication address, etc. On each device, a presentity 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 or 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]

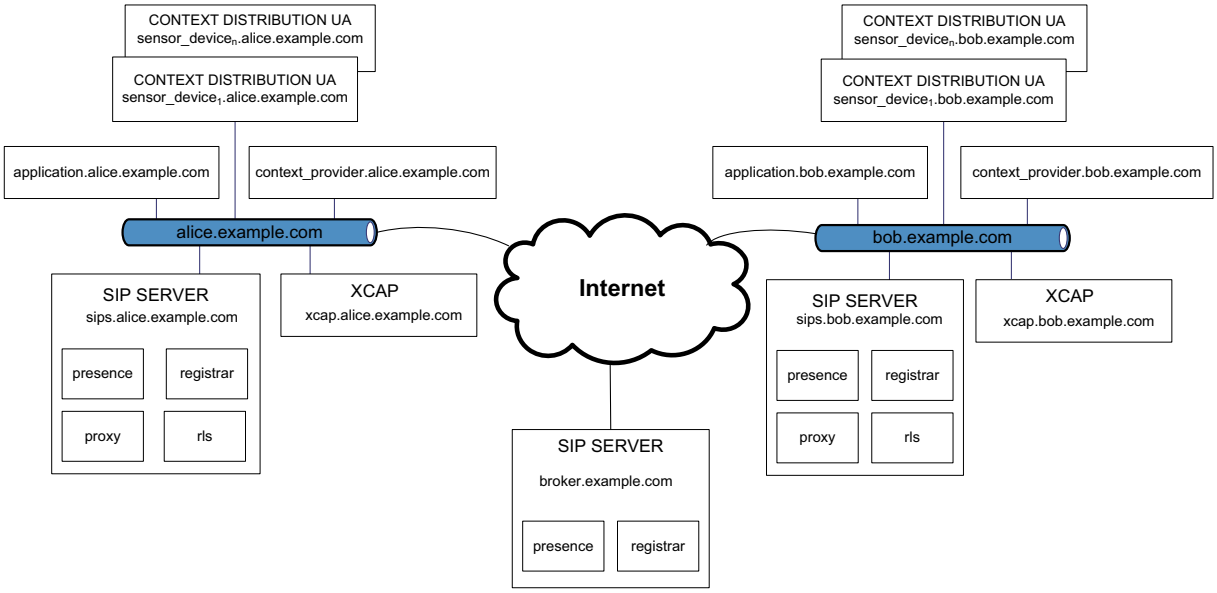


Figure 33: Alice's and Bob's SIP network infrastructure for context-addressed messaging

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 in the receiver's current context 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.

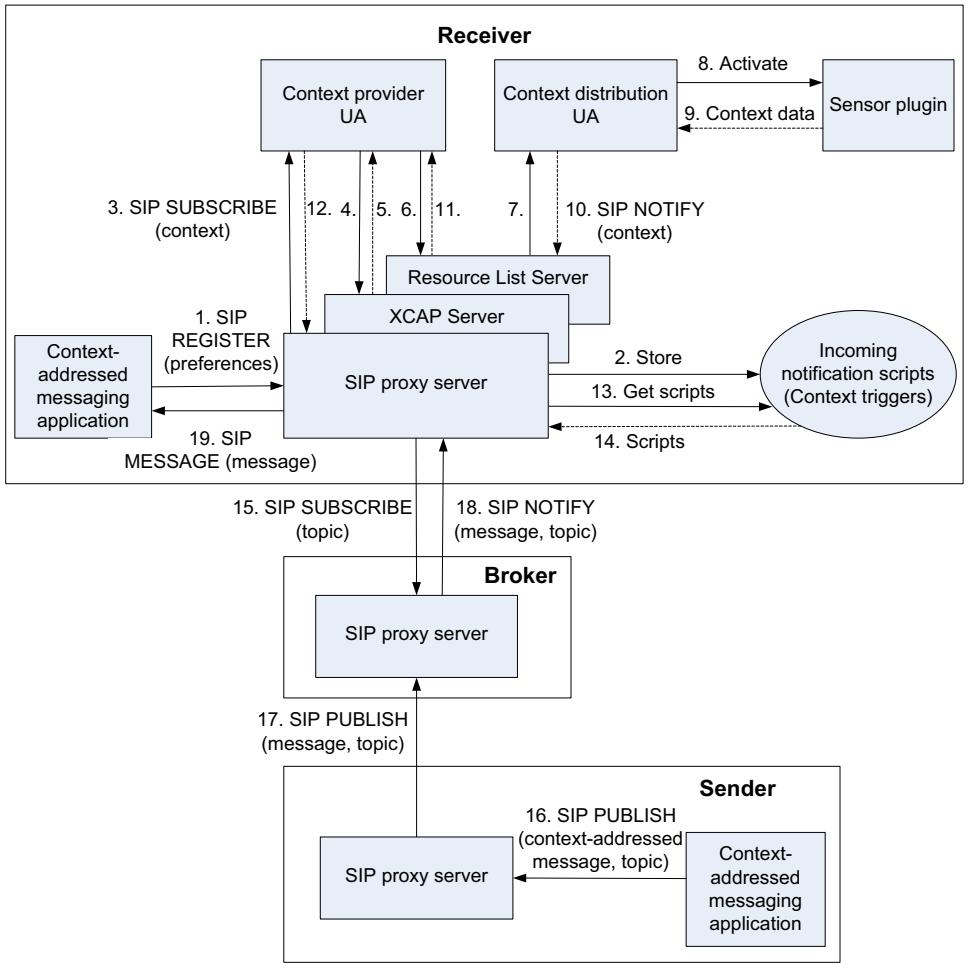


Figure 34: SIP operations for context-addressed messaging

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.

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 <*entity*, *scope*> pair. By sending a SUBSCRIBE method to this AoR, one can receive in a single NOTIFY message context updates aggregated from all the sensor plugins indicated in this resource list. Note that the term *value* [93] can also be used to specify *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>
```

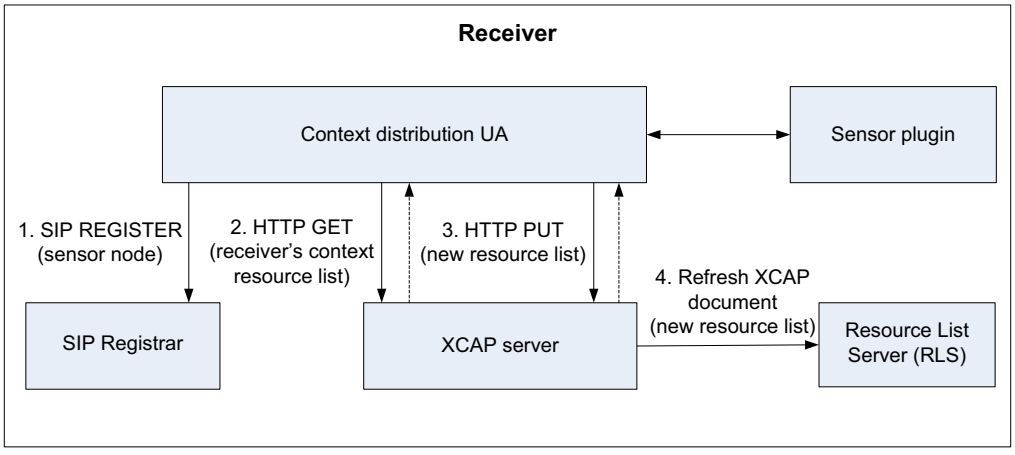


Figure 35: Registration of context sensors metadata for context distribution

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.

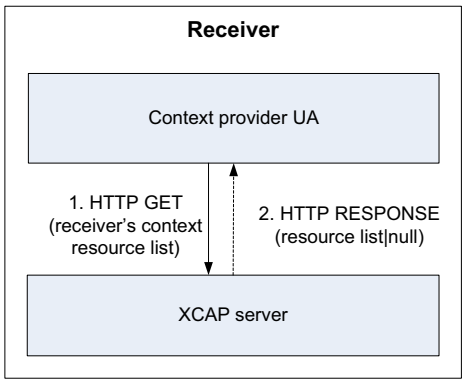


Figure 36: Synchronous resource location

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.

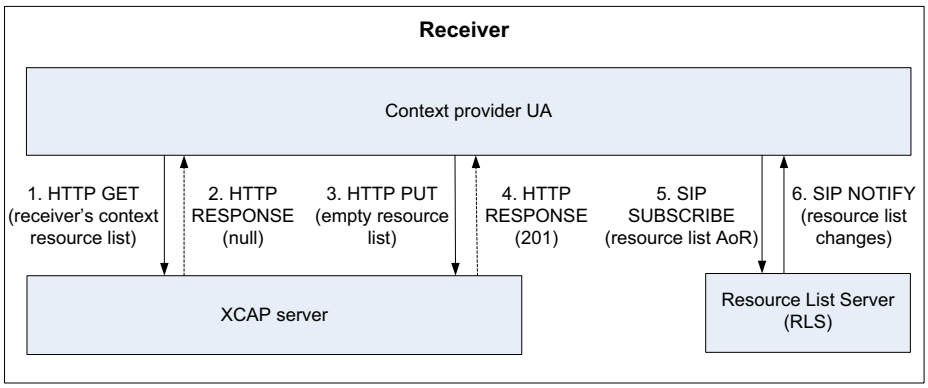


Figure 37: Asynchronous resource location

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.

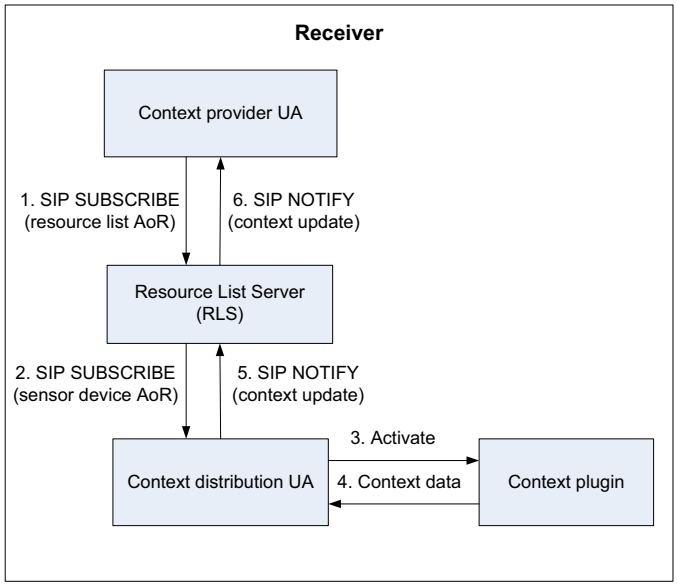


Figure 38: Event notification of context information

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).

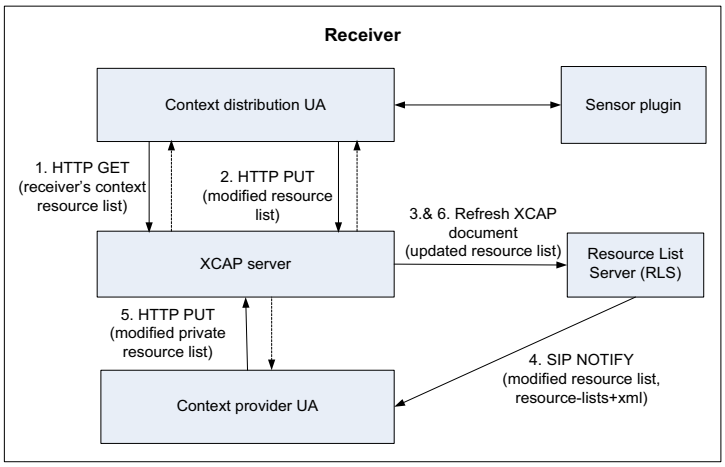


Figure 39: Sensor deregistration when sensor plugin gracefully turns off

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.

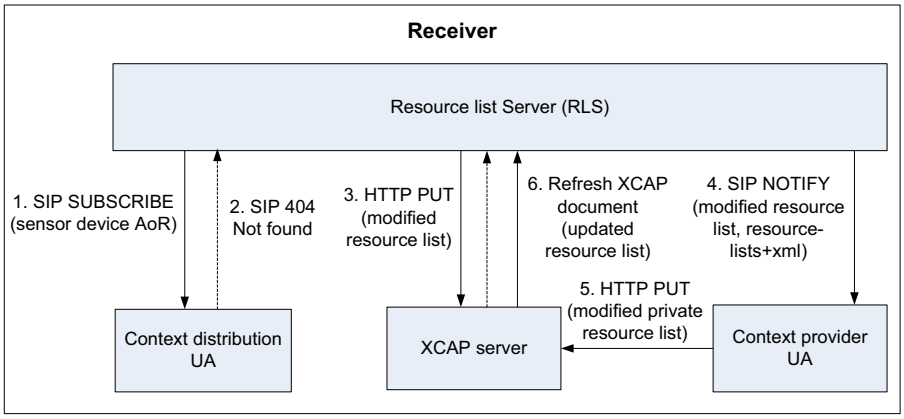


Figure 40: Sensor deregistration when context distribution UA fails

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.

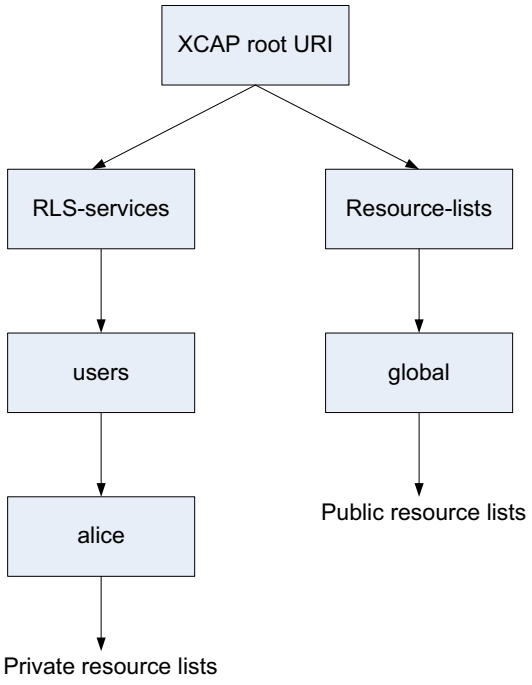


Figure 41: Hierarchy for storing public and private resource lists

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 <entity, 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., *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.

4.6.2.6 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.

4.7 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).

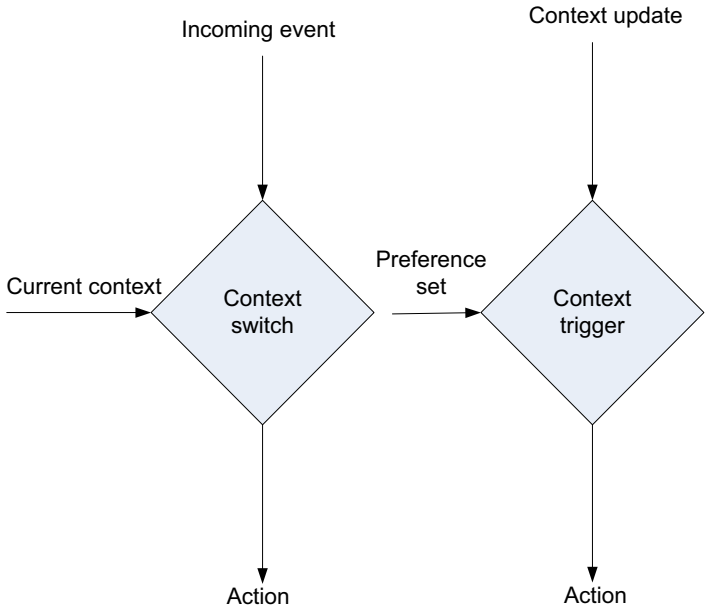


Figure 42: Context-based and context-triggered communication

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:

```
Node:            context-switch      context switch node
Outputs:         user_context        specific user context parameters to match
 Parameters:     owner               context owner name

Output:          user_context        context node
Parameters:      location            location of a context owner
                 task                task status
                 activity            activity status
```

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.

```
<xml version="1.0" encoding="UTF-8">
<cpl xmlns="http://web.it.kth.se/~devlic/context.dtd">
    <incoming>
        <context-switch owner="jim">
            <user_context location="grimeton" task="meeting" activity="presenting">
                <reject status="reject" reason="InMajorMeeting_And_Presenting"/>
            </user_context>
        </context-switch>
    </incoming>
</cpl>
```

Listing 5: CPL-based context switch

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].

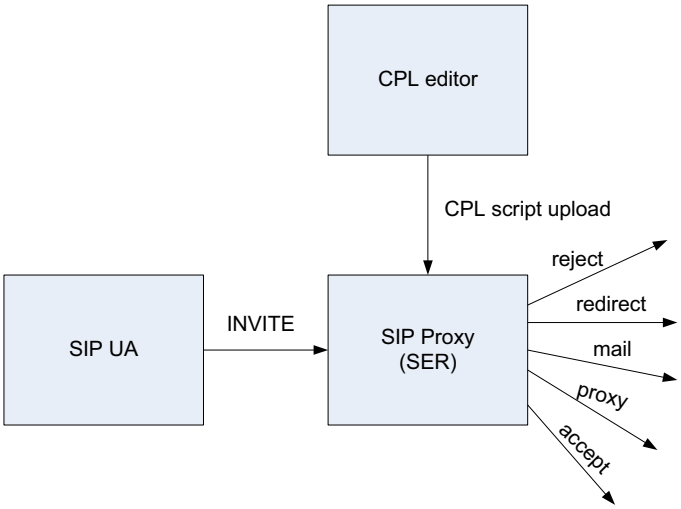


Figure 43: Call processing logic

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.

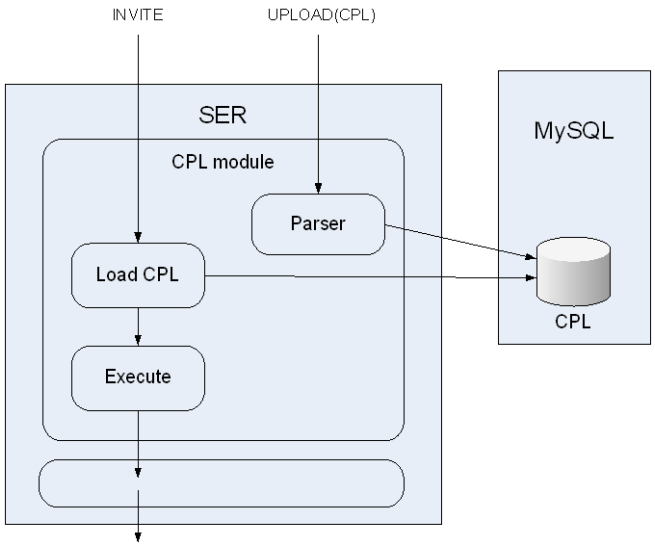


Figure 44: SIP Express Router (SER)'s processing of CPL scripts

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.

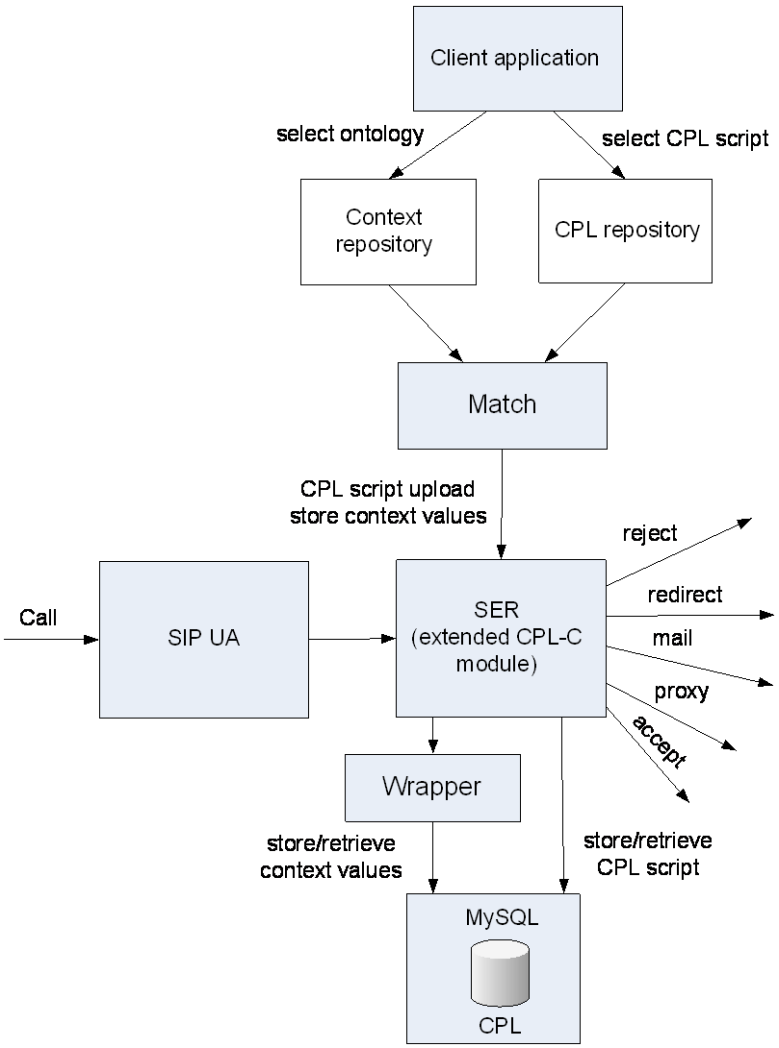


Figure 45: Context-aware VoIP prototype

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.

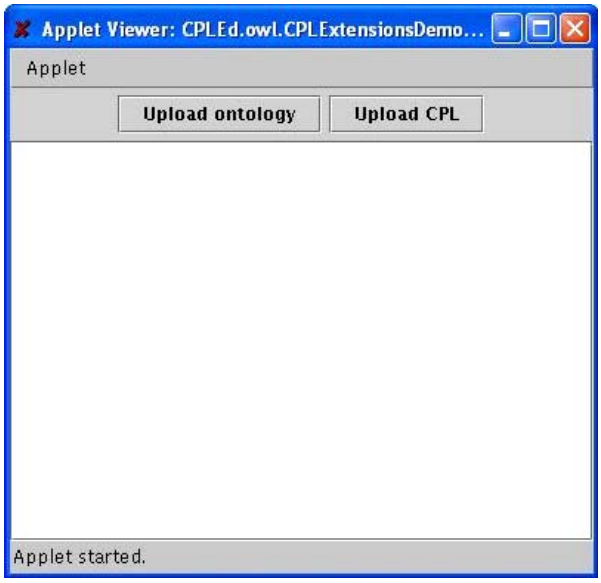


Figure 46: Client application

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).

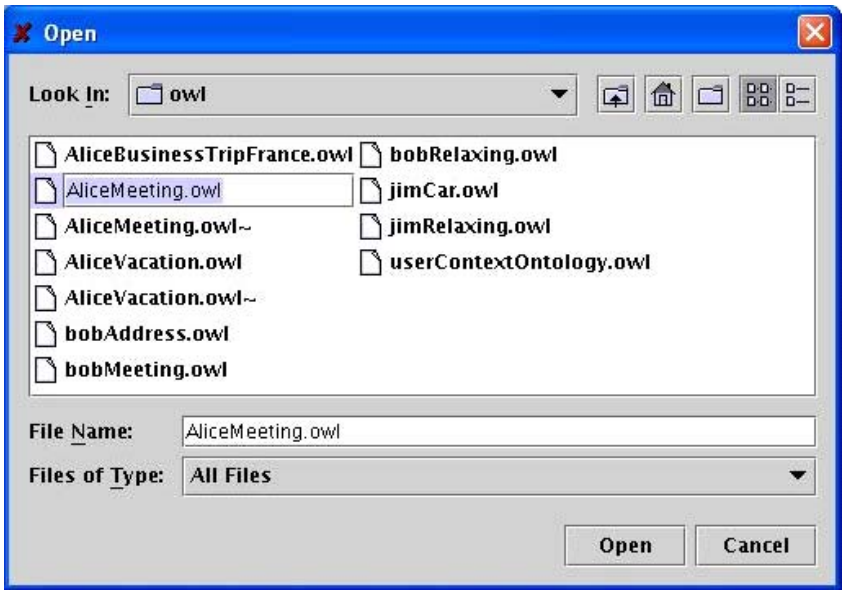


Figure 47: File chooser dialog

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.

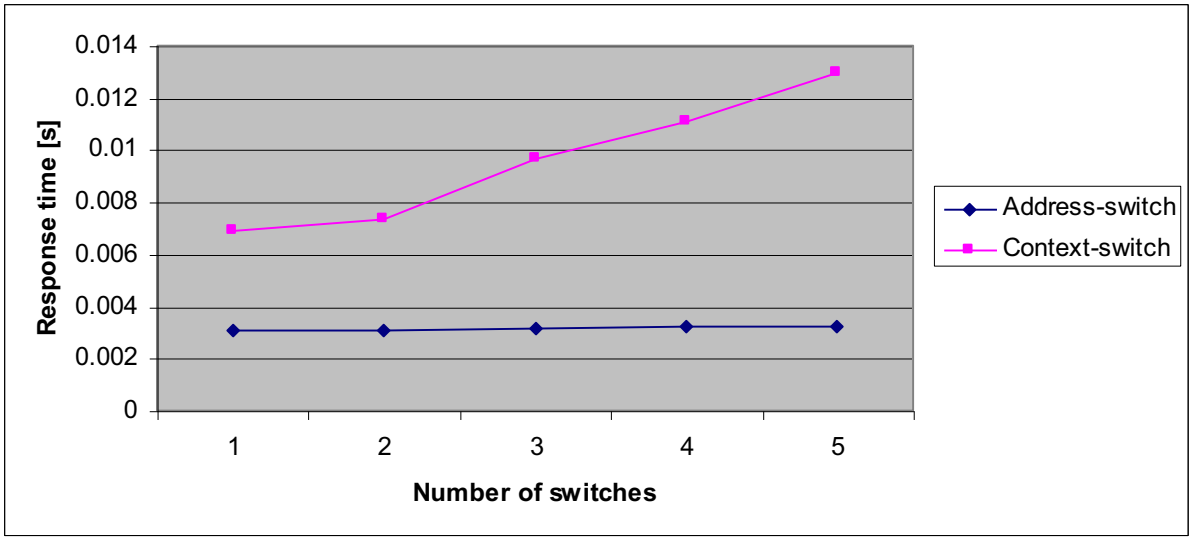


Figure 48: Comparison of (standard and context-dependent) CPL scripts response times

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.

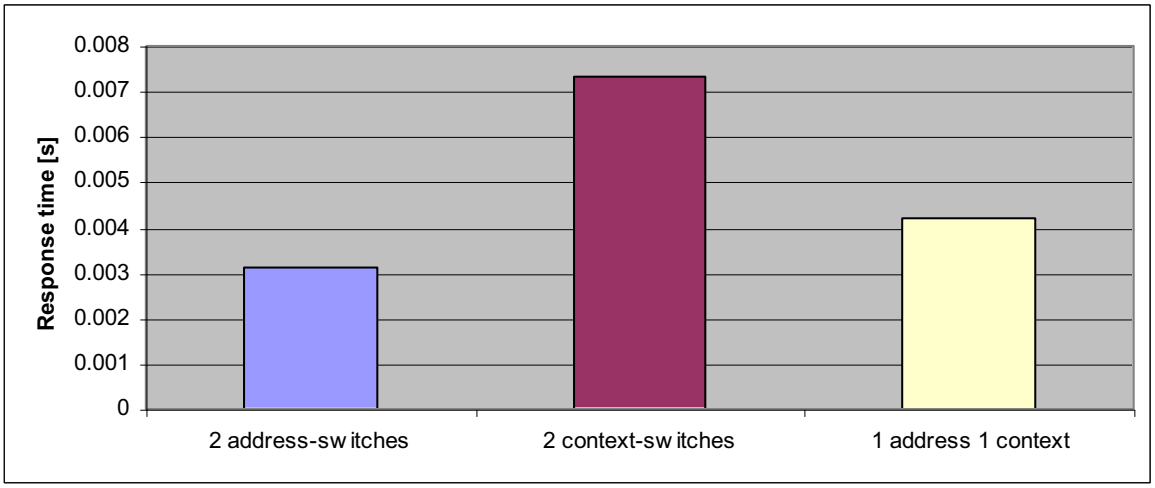


Figure 49: Comparison of different types of CPL scripts and their response times

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:

```
     Node:      context-trigger    context trigger node
  Outputs:      context            context parameters to match
Parameters:     entity             context entity type: "user" or "device"
                uri                SIP URI identifying a user or a device

  Output:       context            context node
  Parameters:   location           location of a user
                task               task status of a user
                activity           activity status of a user
                bandwidth          available bandwidth on a device
                battery            the device's remaining battery power
```

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:

```
Node:         group-query  group query node
Outputs:      None
Next node:    None
Parameters:   to           resource list URI identifying a list of
                           receivers
              activity     activity to match
              interest     interest to match
              location     location to match
```

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.

```
<xml version="1.0" encoding="UTF-8">
<cpl xmlns="http://web.it.kth.se/~devlic/cpl.dtd"
     xmlns:cplContext="http://web.it.kth.se/~devlic/cplContext.dtd"
     xmlns:cplCommunication="http://web.it.kth.se/~devlic/cplCommunication.dtd">
    <cplCommunication:notification>
        <incoming>
            <!--context trigger node-->
            <cplContext:context-trigger entity="person" url="sip:alisa@example.com">
                <cplContext:context activity="free_time">
                    <!--send a group query to friends who are in the same city and are currently biking-->
                    <cplCommunication:group_query to="sip:friends.context@example.com" activity="biking" location="my_current_city"/>
                </cplContext:context>
            </cplContext:context-trigger>
        </incoming>
    </cplCommunication:notification>
</cpl>
```

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 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 >
```

```
<!ELEMENT context (%Node;) >

<!ATTLIST context
          location   CDATA #IMPLIED
             task CDATA #IMPLIED
             activity CDATA #IMPLIED
          bandwidth CDATA #IMPLIED
          battery CDATA #IMPLIED
> <!-- at least one and at most three of those attributes must appear -->
<!-- Signalling action nodes -->
<!ENTITY % SignallingAction 'proxy|redirect|reject|group-query|subscribe' >

<!ELEMENT group-query (match?, no-match?, noanswer?) >

<!-- The default value of timeout is "20" if the <noanswer> output exists. -->
<!ATTLIST group-query
          timeout CDATA #IMPLIED
             to CDATA #REQUIRED
             activity CDATA #IMPLIED
             interest CDATA #IMPLIED
             location CDATA #IMPLIED
> <!-- at least one and at most three of activity, interest and location attributes must
appear -->

<!ELEMENT match ( %Node; ) >
<!ELEMENT no-match ( %Node; ) >

<!ELEMENT subscribe ( approve?,pending?,deny?,noanswer?,default? ) >

<!ATTLIST subscribe
          timeout      CDATA    #IMPLIED
             recurse      (yes|no) "yes"
             ordering     CDATA    "parallel"
>
<!ELEMENT approve ( %Node; ) >
<!ELEMENT  pending ( %Node; ) >
 <!ELEMENT deny ( %Node; ) >
 <!ELEMENT noanswer ( %Node; ) >
 <!ELEMENT default ( %Node; ) >
```

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.

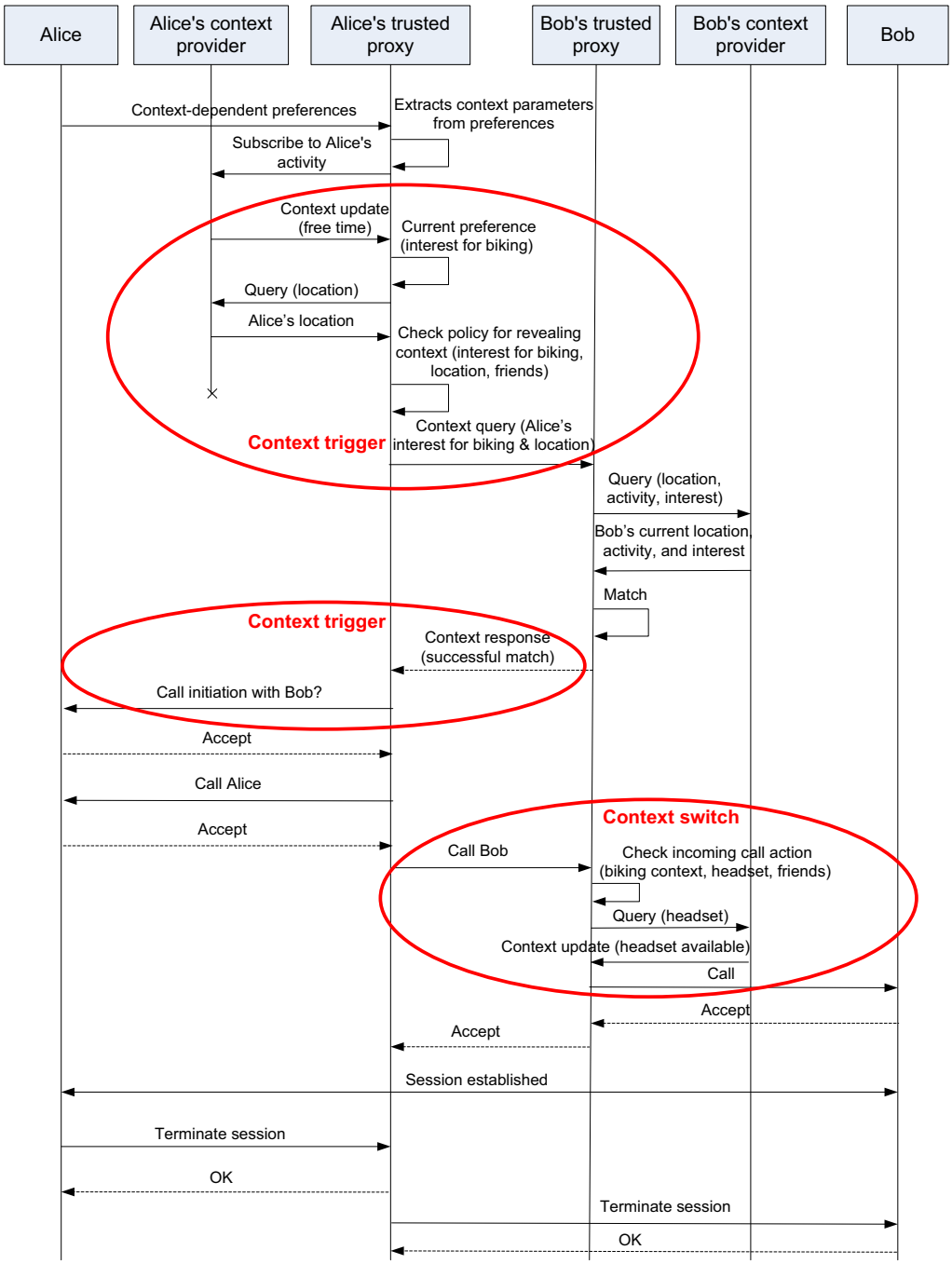


Figure 50: Communication initiated by a preference match

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.

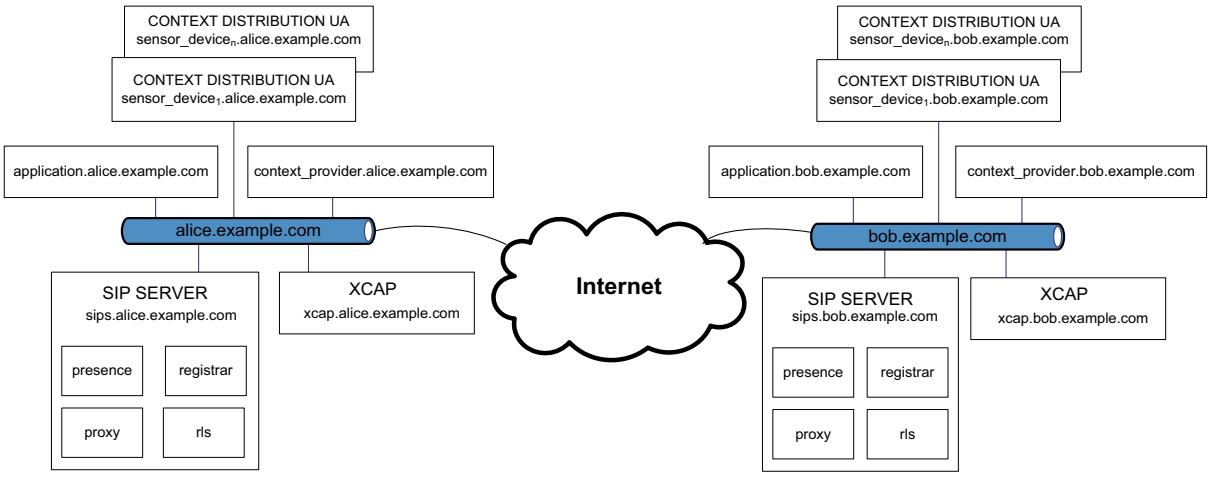


Figure 51: Alice's and Bob's SIP infrastructure architecture for context-aware communication initiation

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.

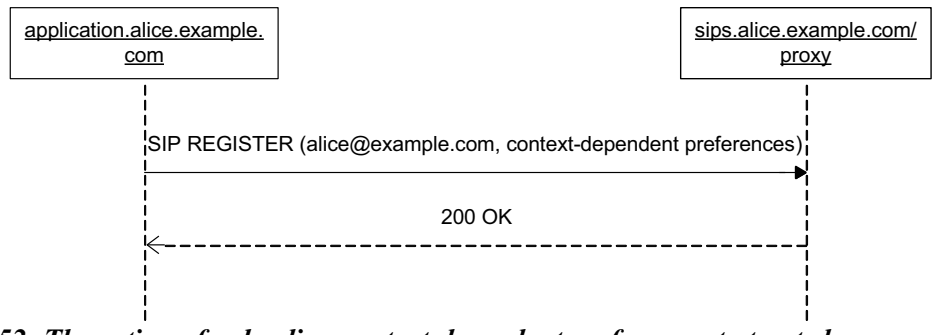


Figure 52: The action of uploading context-dependent preferences to trusted proxy

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.

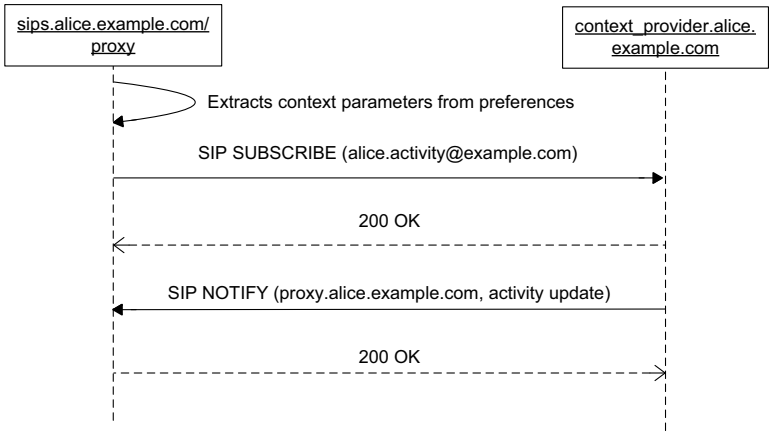


Figure 53: The action of subscribing to context parameters upon which preferences are conditioned

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

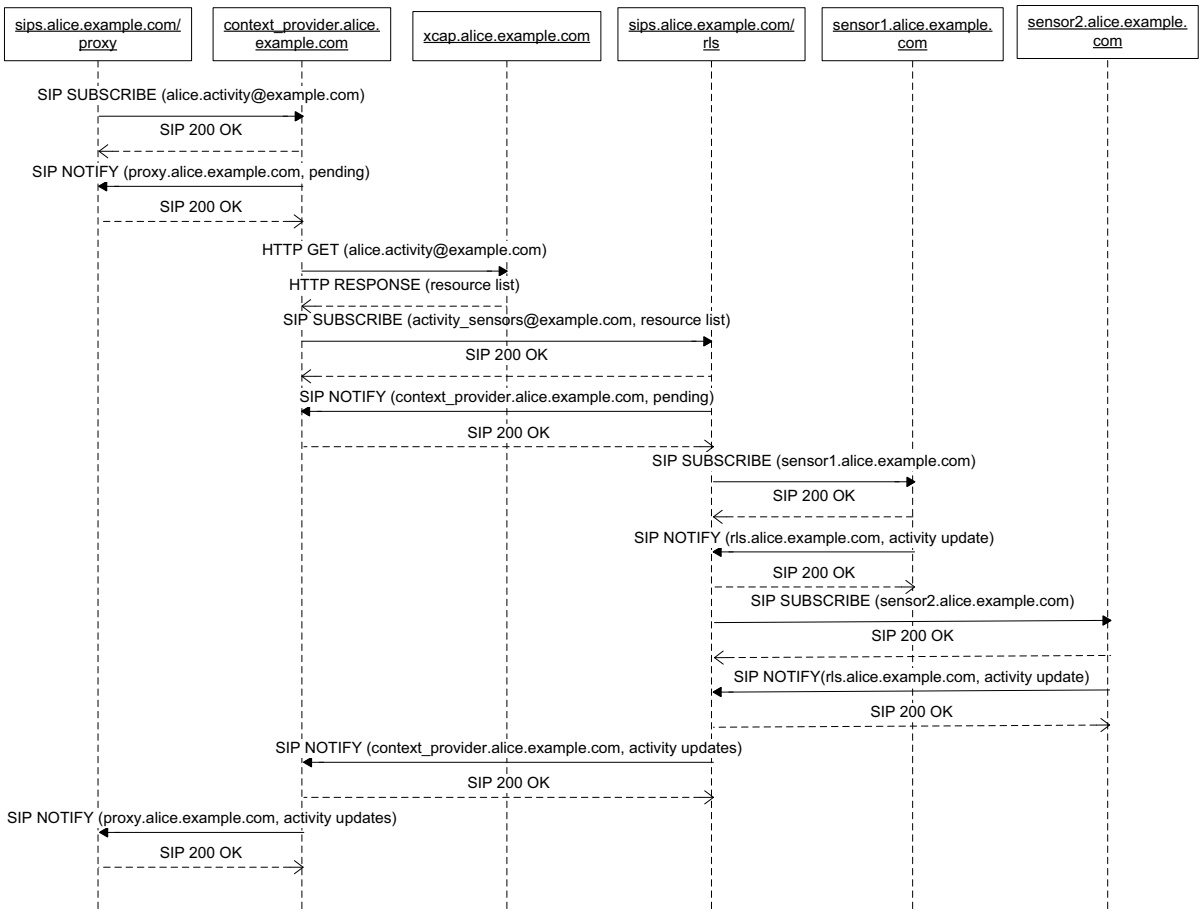


Figure 54: Context provider retrieves information from two sensor devices

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.

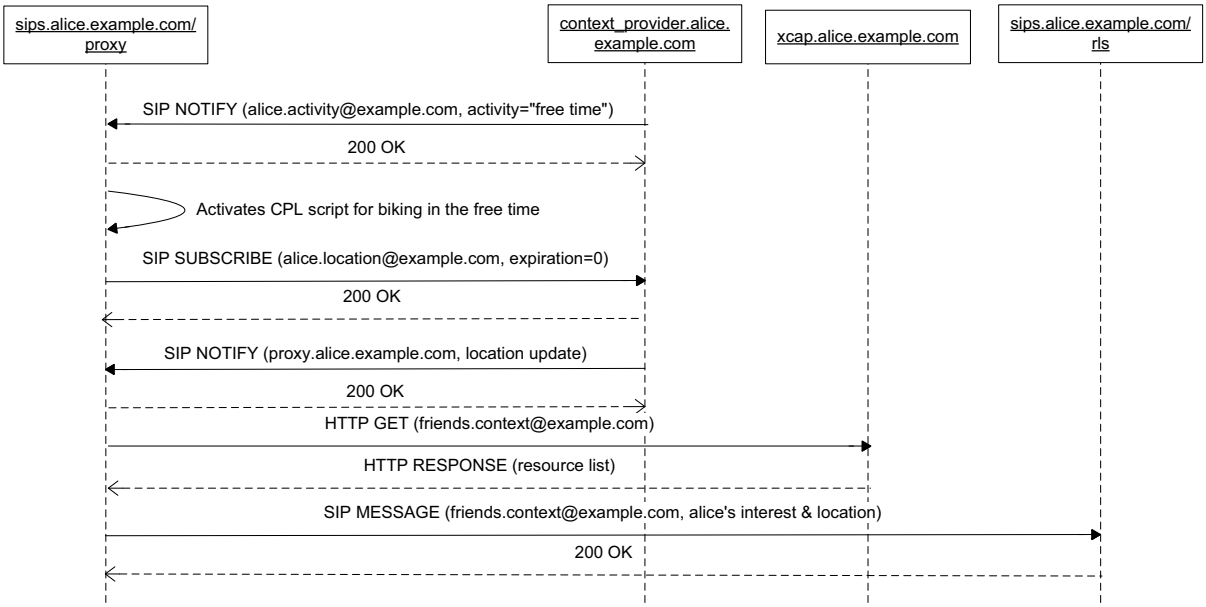


Figure 55: The action of activating a new preference in the current context and sending a query to a group of social contacts

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.

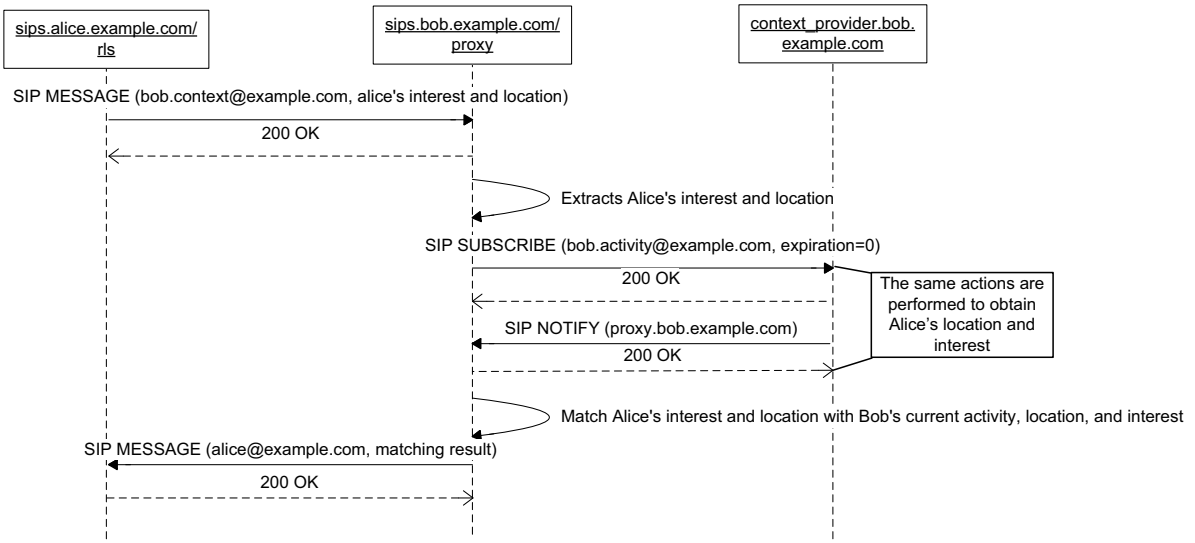


Figure 56: The action of matching Alice's and Bob's context and interests and returning the matching result

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.

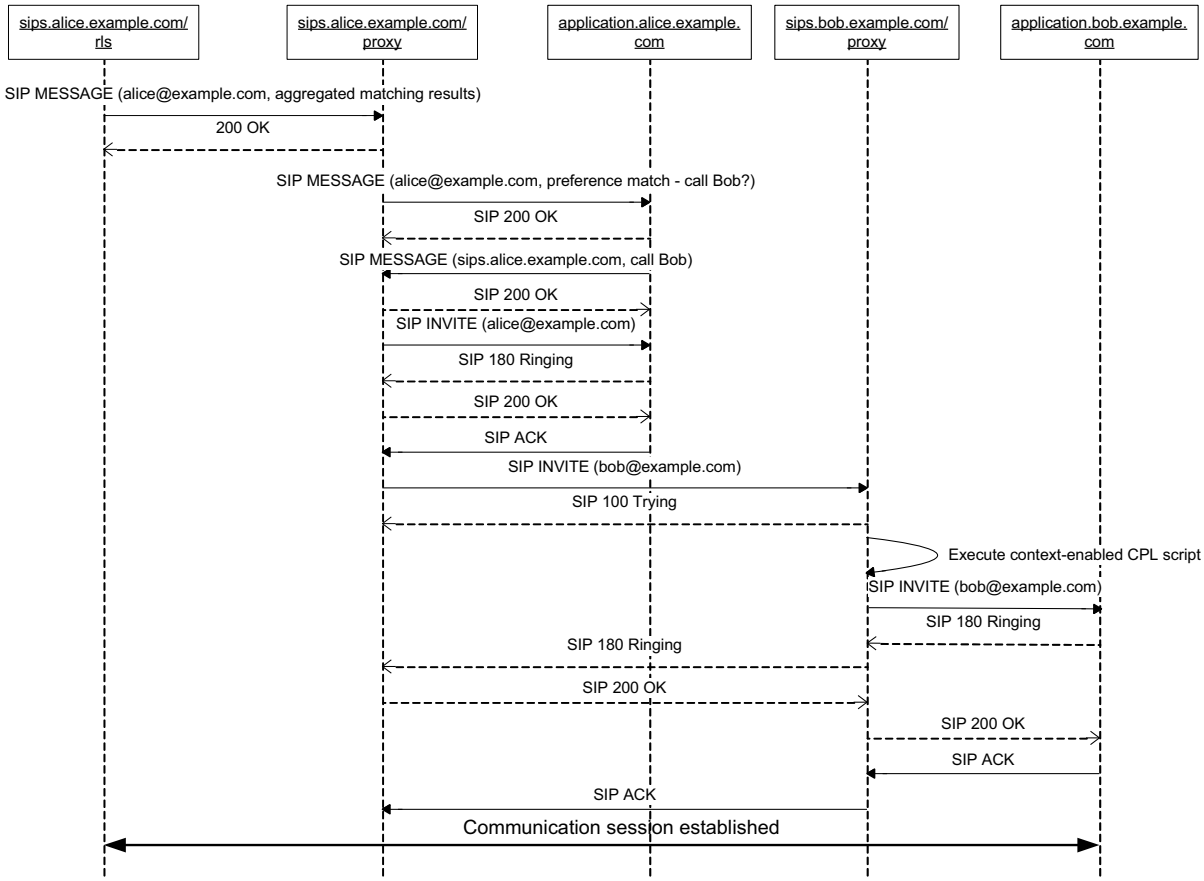


Figure 57: The action of initiation and establishment of communication session with Alice and Bob

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. 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.

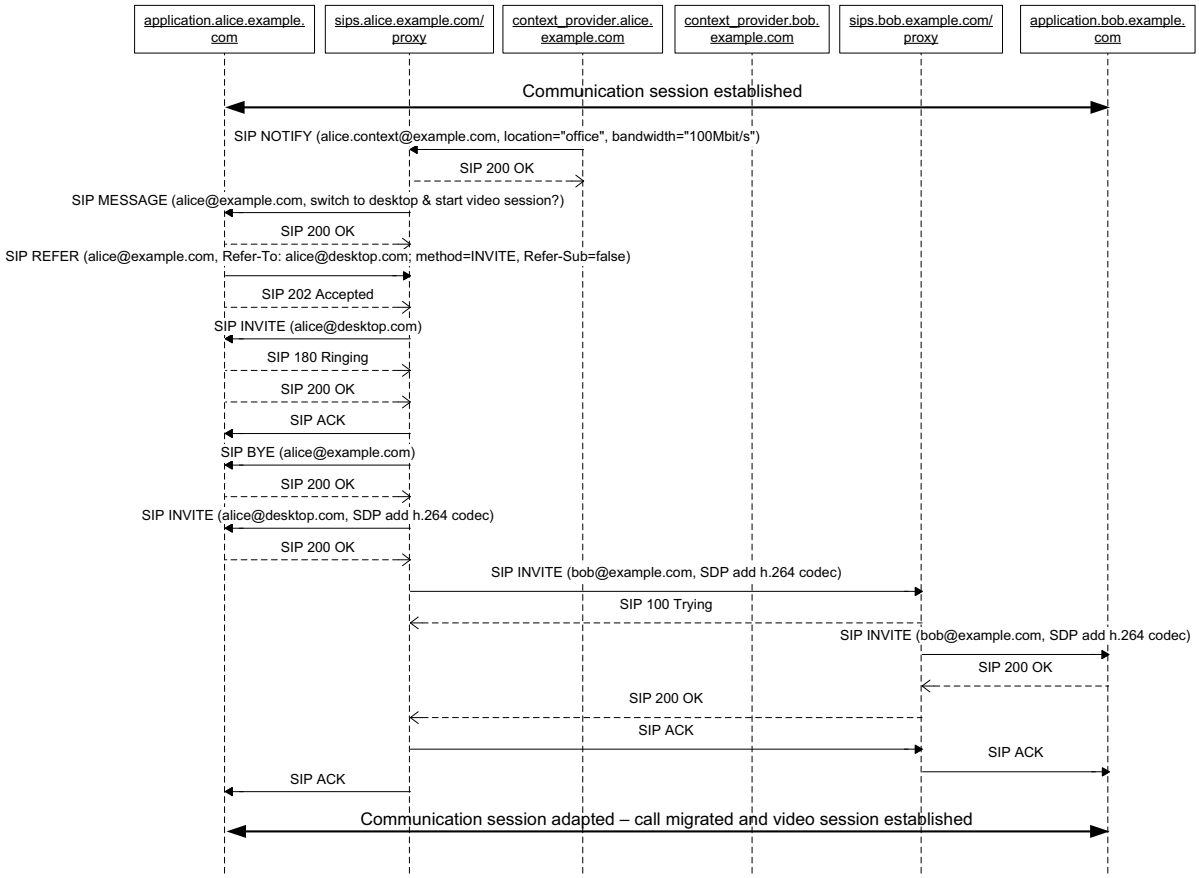


Figure 58: Context-aware session adaptation resulting in a call migration and establishment of video session

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.

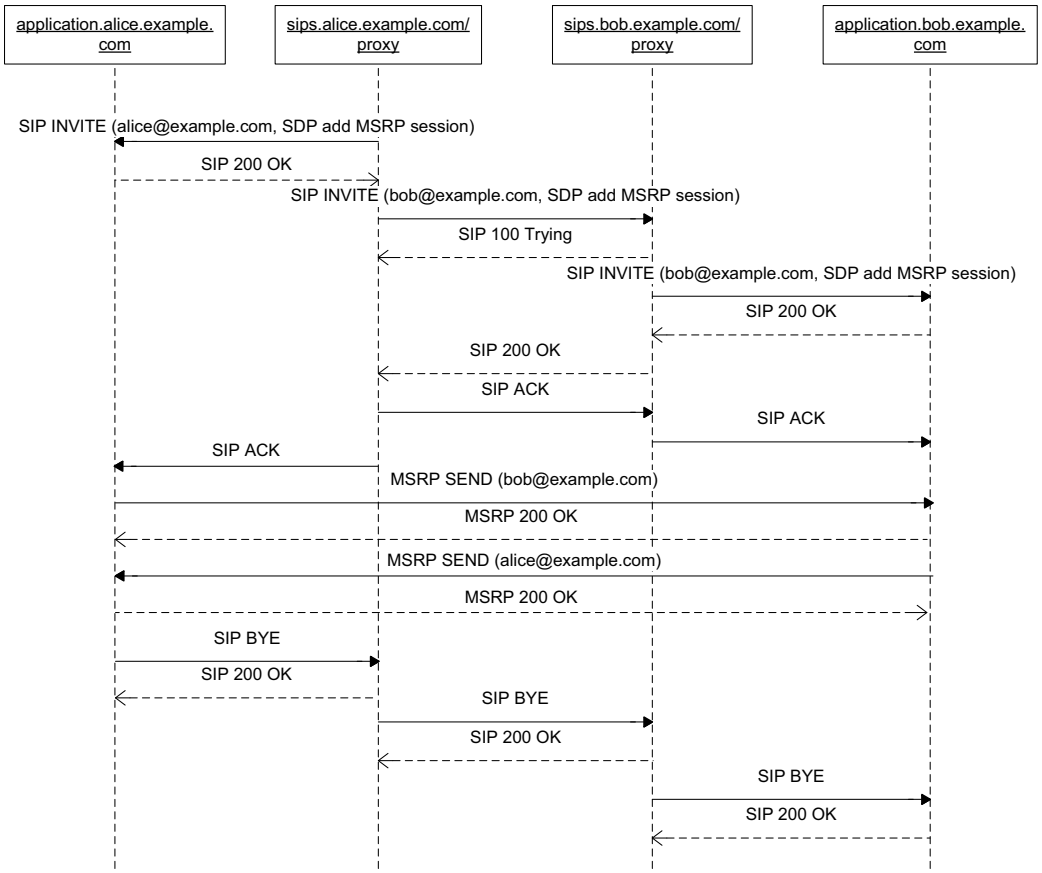


Figure 59: Instant messaging session between Alice and Bob

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 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.

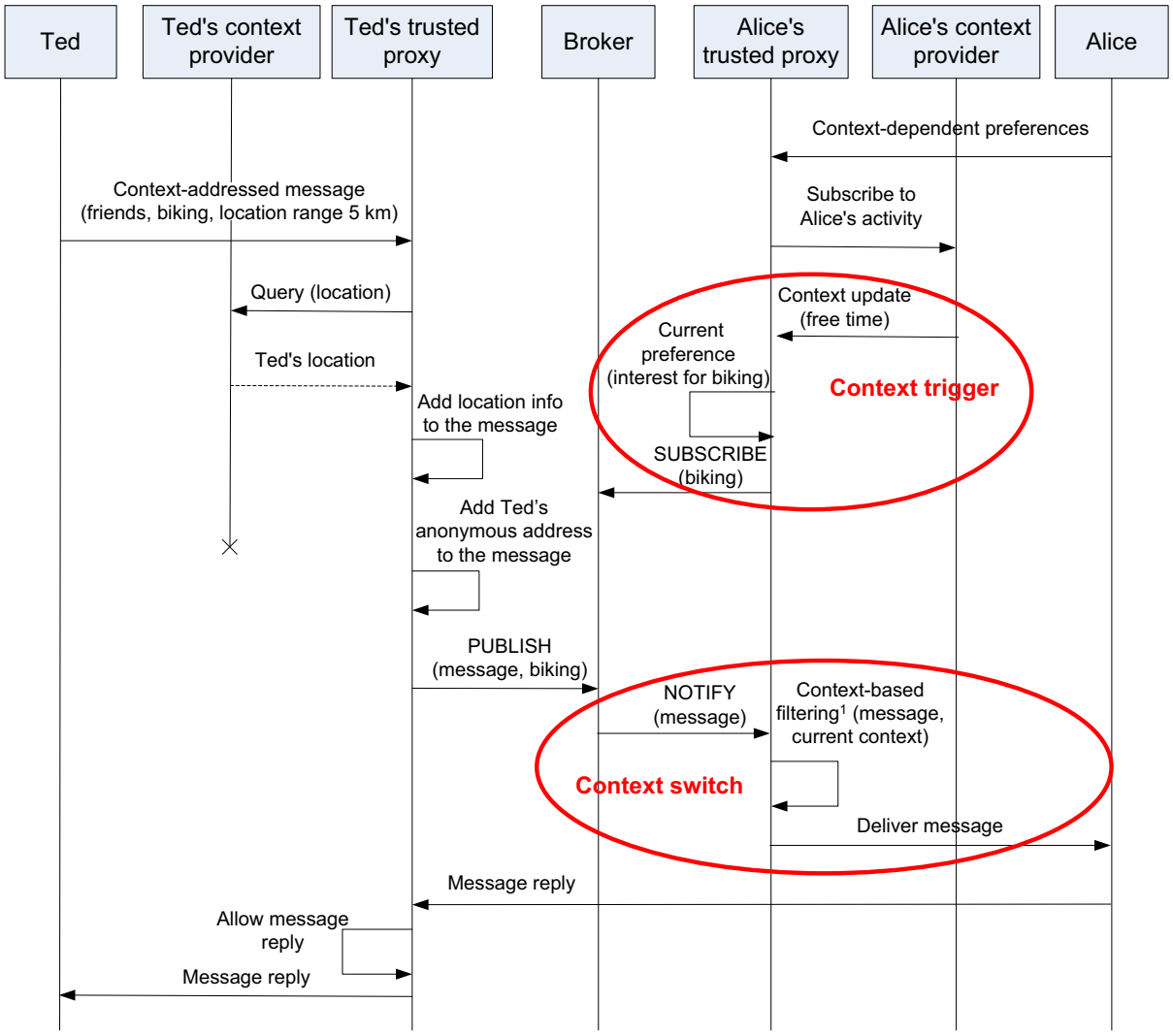


Figure 60: Context addressed messaging scenario

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 this script store it in the database under the supplied user's credentials. Upon arrival of an incoming call or a 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.

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 matrixes 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 easy 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 matrixes 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 it 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

[1] A. Devlic, "Extending CPL with context ontology", In Mobile Human Computer Interaction (Mobile HCI 2006) Conference Workshop on Innovative Mobile Applications of Context (IMAC), Espoo/Helsinki, Finland, September 2006.

[2] A. Devlic and E. Klintskog, "Context retrieval and distribution in a mobile distributed environment", Third Workshop on Context Awareness for Proactive Systems (CAPS 2007), Guildford, UK, June 2007.

[3] A. Devlic, M. Koziuk, and W. Horsman, "Synthesizing context for a sports domain on a mobile device", In Proceedings of the 3rd IEEE European Conference on Smart Sensing and Context (EuroSSC 2008), Zurich, Switzerland, Springer-Verlag, LNCS 5279, October 2008, pp. 206-219.

[4] A. Devlic, A. Graf, P. Barone, A. Mamelli, and A. Karapantelakis, "Evaluation of context distribution methods via Bluetooth and WLAN: Insights gained while examining Battery Power Consumption", In Proceedings of the Fifth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2008), Dublin, Ireland, July 2008.

[5] A. Devlic, A. Graf, P. Barone, A. Mamelli, and A. Karapantelakis, "Ad hoc context distribution methods using Bluetooth and WLAN", IC@ST magazine, ICST, 1st online edition, November 2008.

[6] A. Devlic and G. Panagiotou, "Context Distribution using SIP-based multicast", submitted for publication.

[7] A. Devlic, R. Reichle, M. Wagner, M. Kirsch Pinheiro, Y. Vanrompay, Y. Berbers, and M. Valla, "Context inference of users' social relationships and distributed policy management", In Proceedings of the 6th IEEE Workshop on Context Modeling and Reasoning (CoMoRea) at the 7th IEEE Conference on Pervasive Computing and Communications (PerCom'09), Galveston, Texas, March 2009, pp. 755-762.

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

[9] OpenNETCF Consulting, Smart Device Framework, a software library for Microsoft .NET Compact Framework application developers, http://www.opennetcf.com/CompactFramework/Products/SmartDeviceFramework/tabid/65/Default.aspx, last accessed in April 2009.

[10] Hyperic, SIGAR – System Information Gatherer and Reporter, a cross-platform API to collect system information, http://support.hyperic.com/display/SIGAR/Home, last accessed in March 2009.

[11] 3GPP specification TS 23.228, IP Multimedia Subsystem, stage 2, Release 8, March 2009.

[12] Google Android – An Open Handset Alliance Project, http://code.google.com/android/, last accessed in April 2009.

[13] iPhone OS from Apple, http://developer.apple.com/iphone, last accessed in April 2009.

[14] Open Handset Alliance, http://www.openhandsetalliance.com/, last accessed in March 2009.

[15] Theo G. Kanter, "Adaptive Personal Mobile Communication, Service Architecture and Protocols", Doctoral dissertation, Department of Microelectronics and Information Technology, Royal Institute of Technology (KTH), November 2001.

[16] B. Schilit and M. Theimer, "Disseminating active map information to mobile hosts", IEEE Networks, vol. 8, no. 5, Sept./Oct. 1994, pp. 22–32.

[17] B.N.Schilit, N.Adams, and R.Want, "Context-Aware Computing Applications", In Proceedings of the 1st International Workshop on Mobile Computing Systems and

Applications, Santa Cruz, California, IEEE Computer Society Press, December 1994, pp.85-90.
[18] G.Chen and D.Kotz, "A Survey on Context-Aware Mobile Computing Research", Darmouth Computer Science, Technical report TR2000-381, November 2000.
[19] J. Pascoe, “Adding generic contextual capabilities to wearable computers,” in Proceedings of the Second International Symposium on Wearable Computers, Pittsburgh, PA, IEEE Computer Society Press, 1998, pp. 92–99.
[20] A.K. Dey, G.D. Abowd, and A. Wood, “Cyberdesk: a framework for providing self-integrating context-aware services” *Knowledge-Based Systems*, Elsevier, vol. 11, 1998, pp. 3–13.
[21] A.K.Dey and G.D.Abowd, "Towards a better understanding of context and context-awareness", In Conference on Human Factors in Computing Systems CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness, ACM Press, Amsterdam, Netherlands, April 2000.
[22] N. Paspallis, R. Rouvoy, P. Barone, G. A. Papadopoulos, F. Eliassen, and A. Mamelli, "A Pluggable and Reconfigurable Architecture for a Context-aware Enabling Middleware System", In Proceedings of the 10th International Symposium on Distributed Objects, Middleware, and Applications (DOA'08), Monterrey, Mexico, LNCS 5331, Springer-Verlag, November 2008, pp. 553-570.
[23] EU FP6 IST MUSIC project, http://www.ist-music.eu, last accessed in April 2009.
[24] T. Strang and C. Linnhoff-Popien, "A Context Modeling Survey", In UbiComp 2004 Workshop Proceedings: 1st International Workshop on Advanced Context Modeling, Reasoning and Management, Nottingham, England, September 2004, pp. 33-40.
[25] B.N.Schilit, M.M.Theimer, and B.B.Welch, "Customizing mobile applications", In Proceedings of USENIX Mobile & Location Independent Computing Symposium, Cambridge, Massachusetts, August 1993, pp.129-138.
[26] W3C, Overview of SGML Resources, http://www.w3.org/MarkUp/SGML/, November 1995.
[27] W3C, CC/PP Information Page, http://www.w3.org/Mobile/CCPP/, October 2007.
[28] Open Mobile Alliance, User Agent Profile, OMA-TS-UAProf-V2_0-20060206-A, approved version 2.0, February 2006.
[29] J.Bauer, "Identification and Modeling of Contexts for Different Information Scenarios in Air Traffic", Bachelor of Science Thesis, Technical University of Berlin, March 2003.
[30] K. Henricksen, J. Indulska, and T. McFadden, "Modelling Context Information with ORM", On The Move To Meaningful Internet Systems, Springer Verlag, 2005, vol. 3762, pp. 626-635.
[31] A. Schmidt, M.Beigl, and H.-W. Gellersen, "There is more to context than location", Computer & Graphics Journal, Elsevier, Volume 23, No.6, December 1999, pp 893-902.
[32] TEA project (Technology for Enabling Awareness, ESPRIT), http://www.teco.edu/tea/, completed September 2000.
[33] K.Cheverst, K.Mitchell, and N.Davies, "Design of an Object Model for a Context Sensitive Tourist GUIDE", Computers & Graphics Journal, Elsevier, Volume 23, Number 6, December 1999, pp. 883-89.
[34] J.McCarthy, "Notes on formalizing contexts". In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers, Inc., Chambery, France, 28th August-3rd September 1993, pp. 555-560.

[35] F.Giunchiglia, "Contextual reasoning", Epistemologica-Special Issue on I Linguaggi e le Macchine 16, pp. 345-364. Also IRST-Technical Report 9211-20, IRST, Trento, Italy, 1993.
[36] V.Akman and M.Surav, "The use of situation theory in context modeling", Computational Intelligence, Wiley, Volume 13, Number 3, 1997, pp. 427-438.
[37] J.Barwise and J.Perry, "Situations and Attitudes", MIT Press, 1983.
[38] P.Gray and D.Salber, "Modeling and Using Sensed Context Information in the design of Interactive Applications", In Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction (EHCI 2001), Toronto, Canada, May 2001, LNCS 2254, Springer-Verlag, pp. 317-336.
[39] W3C OWL Working Group, Web Ontology Language (OWL), http://www.w3.org/2004/OWL/, last accessed in March 2009.
[40] W3C RDF Core Working Group, Resource Description Framework (RDF), http://www.w3.org/RDF/, last accessed in March 2009.
[41] The DARPA Agent Markup Language (DAML), http://www.daml.org/, last accessed in March 2009.
[42] EU FP6 IST MIDAS project, http://www.ist-midas.org, completed December 2008, last accessed in February 2009.
[43] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "DL-Lite: Tractable description logics for ontologies", In 20th National Conference on Artificial Intelligence (AAAI 2005), AAAI Press, Pittsburgh, Pennsylvania, USA, July 2005, pp. 602–607.
[44] M. Jabłonowski and P. Boetzel, "Middleware Layer For Semantic Object Tagging", Master of Science Thesis at Warsaw University of Technology, Warsaw, Poland, 2007.
[45] J.J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: Implementing the semantic web recommendations", HP Technical Report, December 2003, available at http://www.hpl.hp.com/techreports/2003/HPL-2003-146.pdf, last accessed in March 2009.
[46] X.H. Wang, D.Q. Zhang, T. Gu, and H.K. Pung, "Ontology based context modeling and reasoning using OWL", In Proceedings of IEEE Workshop on Context Modeling and Reasoning, pp.18-22, Orlando, Florida USA, March 2004.
[47] X. Wang, J.S. Dong, C.Y.Chin, and S. R. Hettiarachchi, "Semantic Space: An Infrastructure for Smart Spaces", IEEE Pervasive computing, vol. 3, 2004, pp. 32-39.
[48] Beanshell - Lightweight scripting for Java, http://www.beanshell.org/, last accessed in April 2009.
[49] Beanshell, Simple Java Scripting version 1.3, The Beanshell User's Manual, http://www.beanshell.org/manual/bshmanual.html, last accessed in April 2009.
[50] JamVM – A Compact Java Virtual Machine (2008), http://jamvm.sourceforge.net/, last accessed in April 2009.
[51] M. Koziuk, J. Domaszewicz, R.O.Schoeneich, M. Jablonowski, and P. Boetzel, "Mobile Context-Addressable Messaging with DL-Lite Domain Model", In Proceedings of the 3rd European Conference on Smart Sensing and Context (EuroSSC 2008), Zurich, Switzerland, 2008. Springer-Verlag, LNCS 5279, pp. 168-181.
[52] MIDAS video showing a scenario based on a live race at the Super Prestige Cyclocross in Gieten, Netherlands, http://www.youtube.com/watch?v=yulUmlVH8Jc, 2007.

[53] Van Jacobson, "If a Clean Slate is the solution what was the problem?", Stanford Clean Slate Seminar, February 27, 2006, http://cleanslate.stanford.edu/seminars/jacobson.pdf, last accessed in March 2009.
[54] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", IETF RFC 2616, June 1999.
[55] J. Klensin, "Simple Mail Transfer Protocol", IETF RFC 2821, April 2001.
[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.
[57] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol", IETF RFC 3261, June 2002.
[58] SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) RFCs, available at http://www.voip-telephony.org/rfc/simple, last accessed in April 2009.
[59] Vivek Gupta, IEEE 802.21 Tutorial, IEEE 802.21 Media Independent handover, July 17 2006, http://www.ieee802.org/21/Tutorials/802%2021-IEEE-Tutorial.ppt, last accessed in March 2009.
[60] J. Lennox, X. Wu, and H. Schulzrinne, "Call Processing Language (CPL): A Language for User Control of Internet Telephony Services", IETF RFC 3880, October 2004.
[61] Z. Duan, K. Gopalan, and Y. Dong, "Push vs. Pull: Implications of Protocol Design on Controling Unwanted Traffic", In Proc. USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI 2005), Cambridge, MA, 2005.
[62] A. Devlic and G. Jezic, "Location-Aware Information Services using User Profile Matching", In Proceedings of the 8th International Conference on Telecommunications (ConTEL05), Institute of Electrical and Electronics Engineers (IEEE), pp.327-335, Zagreb, Croatia, June 2005.
[63] G. Klyne and D. Atkins, "Common Presence and Instant Messaging (CPIM): Message Format", IETF RFC 3862, August 2004.
[64] P. Resnick, "Internet Message Format", IETF RFC 2822, April 2001.
[65] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", IETF RFC 2396, August 1998.
[66] D. Crocker, "Augmented BNF for Syntax Specifications: ABNF", IETF RFC 2234, November 1997.
[67] W. Li, F. Kilander, and C. G. Johansson, "Towards a Person-Centric Context Aware System", In International Conference on Pervasive Computing (Pervasive 2006)'s Workshop on Requirements and Solutions for Pervasive Software Infrastructures (RSPSI), Springer-Verlag, Dublin, Ireland, May 2006.
[68] R. Bulander, M. Decker, G.Schiefer, and B. Kölmel, "Enabling Personalized And Context Sensitive Mobile Advertising While Guaranteeing Data Protection", In Proceedings of EURO mGOV 2005, pp. 445-454. Mobile Government Consortium International LLC, Brighton, UK, 2005.
[69] M. Spreitzer and M. Theimer, "Providing Location Information in a Ubiquitous Computing Environment". ACM SIGOPS Operating Systems Review, vol. 27, pp. 270-283, 1993.
[70] A. K. Dey, G. D. Abowd, and D. Salber, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications". Human-Computer Interaction (HCI) Journal, Special Issue on Context-Aware Computing, Vol. 16, pp. 97-166, 2001.

[71] Dallas Semiconductor. iButton Home Page. http://www.maximic.com/products/ibutton, last accessed in April 2009.
[72] PinPoint. PinPoint 3D-iD introduction. http://www.rft.com/solutions/pinpointrtls/, last accessed in April 2009.
[73] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Content-Based Addressing and Routing: A General Model and its Application", University of Colorado, Department of Computer Science, Technical Report CU-CS-902-00, January 2000.
[74] Protocol Independent Multicast (PIM) Working Group, specifications available at: http://www.ietf.org/html.charters/pim-charter.html, last accessed in April 2009.
[75] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol Independent Multicast – Sparse Mode (PIM-SM): Protocol Specification", IETF RFC 2117, June 1997.
[76] L. A. Hollar, "Hardware Systems for Text Information Retrieval", In Proceedings of the 6th annual international ACM SIGIR conference on Research and development in information retrieval, Bethesda, Maryland, US, June 1983, pp. 3-9.
[77] H. Lee, B. Jeon, S. Park, T. Kwon, and Y. Choi, "An Efficient Multicasting Architecture for Context-Aware Messaging Services in the Future Internet", In Proceedings of the 10th International Conference on Advanced Communication Technology (ICACT 2008), Institute of Electrical and Electronics Engineers (IEEE), Phoenix Park Pyeongchang, Gangwon-do South Korea, February 2008, pp. 630-633.
[78] R. Boivie, N. Feldman, Y. Imai, W. Livens, and D. Ooms, "Explicit Multicast (Xcast) Concepts and Options", IETF RFC 5058, November 2007.
[79] Y.-B. Ko and N.H.Vaidya, "Flooding-Based Geocasting Protocols for Mobile Ad Hoc Networks", Kluwer Academic Publishers, Mobile Networks and Applications, December 2002, volume 7, pp. 471-480.
[80] W.-J. Hsu, D. Dutta, and A. Helmy, "*Profile-Cast*: Behavior-Aware Mobile Networking", Wireless Communications and Networking Conference (WCNC 2008), IEEE, Las Vegas, April 2008, pp. 3033-3038.
[81] J. Domaszewicz, M. Koziuk, and R. O. Schoeneich, "Context-Addressable Messaging with ontology-driven addresses", In the 7th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2008), Monterrey, Mexico, LNCS, Springer-Verlag, November 2008, pp. 1471-1481.
[82] R. O. Schoeneich, J. Domaszewicz, and M. Koziuk, "Concept-Based Routing in Ad-Hoc Networks", In Proceedings of the 10th International Conference on Distributed Computing and Networking (ICDCN 2009), Springer-Verlag, Hyderabad, India, January 3-6, 2009.
[83] N. Miller, G. Judd, U. Hengartner, F. Gandon, P. Steenkiste, I-H. Meng, M-W. Feng, and N. Sadeh, "Context-Aware Computing Using a Shared Contextual Information Service", Pervasive 2004, "Hot Spots", In "Advances in Pervasive Computing" book of the Austrian Computer Society (OCG),Vienna, April 2004.
[84] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, "Project Aura: Towards Distraction-Free Pervasive Computing". IEEE Pervasive Computing, Special Issue on Integrated Pervasive Computing Environments, Vol. 1, Number 2, April-June 2002, pp. 22-31.
[85] F. Gandon and N. Sadeh, "Semantic Web Technologies to Reconcile Privacy and Context Awareness". *Web Semantic Journal*, Elsevier, Vol. 1, Number 3, 2004, pp. 241-260.
[86] J. Rosenberg, "A data model for presence", IETF RFC 4479, July 2006.

[87] SIMPLE RFC's, SIP for Instant Messaging and Presence Leveraging Extensions, available at: http://www.voip-telephony.org/rfc/simple.
[88] C. Angeles Piña, "Distribution of Context Information using the Session Initiation Protocol (SIP)", Master of Science thesis, Royal Institute of Technology (KTH), COS/CCS, Stockholm, Sweden, June 2008.
[89] H. Sugano, S. Fujimoto, G. Klyne, A. Bateman, W. Carr, and J. Peterson, "Presence Information Data Format (PIDF)", IETF RFC 3863, August 2004.
[90] H. Schulzrinne, V. Gurbani, P. Kyzivat, and J. Rosenberg , "RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF)", IETF RFC 4480, July 2006.
[91] J. Rosenberg, "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", IETF RFC 4825, May 2007.
[92] Darwin Valderas Núñez, "Integration of sensor nodes with IMS", Master of Science thesis, Royal Institute of Technology (KTH), COS/CCS, Stockholm, Sweden, October 2008. Available at: http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/081008-DarwinValderas-with-cover.pdf
[93] R. Reichle et al., "A Comprehensive Context Modeling Framework for Pervasive Computing Systems", In Proceedings of the 8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS), IFIP, Oslo, Norway, Springer-Verlag, June 2008, pp.281-295.
[94] J. Urpalainen, "An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Diff Event Package", IETF Internet draft, October 2008.
[95] J. Rosenberg and J. Urpalainen, "An Extensible Markup Language (XML) Document Format for Indicating A Change in XML Configuration Access Protocol (XCAP) Resources", IETF Internet draft, May 2008.
[96] G. Camarillo, A. B. Roach, and O. Levin, "Subscriptions to Request-Contained Resource Lists in the Session Initiation Protocol (SIP)", IETF RFC 5367, October 2008.
[97] IANA, "Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Parameters", August 2007, available at: http://www.iana.org/assignments/xcap-parameters, last accessed in April 2009.
[98] M. Isomaki and E. Leppanen, "An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Usage for Manipulating Presence Document Contents", IETF RFC 4827, May 2007.
[99] J. Rosenberg, "Extensible Markup Language (XML) Formats for Representing Resource Lists", IETF RFC 4826, May 2007.
[100] Open Mobile Alliance, "Presence XDM Specification", OMA-TS-Presence_SIMPLE_XDM-V1_0_1-20061128-A, approved 28 November 2006.
[101] A. Devlic, "CPL extensions", Report for the Practical VoIP course, http://web.it.kth.se/~devlic/CPL extensions.pdf, May 2006.
[102] iptel.org, SIP Express Router (SER), http://www.iptel.org/ser, 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.
[104] CPL-C module of SER, http://www.iptel.org/ser/component/module/cpl_c, last accessed in April 2009.
[105] X. Wu, H. Schulzrinne, J. Lennox, and J. Rosenberg, "CPL Extensions for Presence", IETF Internet draft, expired November 2001.

[106] Dongmei Jiang, "Internet Telephony Services for Presence With SIP and Extended CPL", Master of Computer Science thesis, University of Ottawa, Ontario, Canada, December 2003.
[107] O. Levin, "Suppression of Session Initiation Protocol (SIP) REFER Method Implicit Subscription", IETF RFC 4488, May 2006.
[108] B. Campbell, R. Mahy, and C. Jennings, "The Message Session Relay Protocol", RFC 4975, September 2007.

LIST OF ACRONYMS, ABBREVIATIONS, AND STANDARDS

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

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: $T_{n,x}$(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.

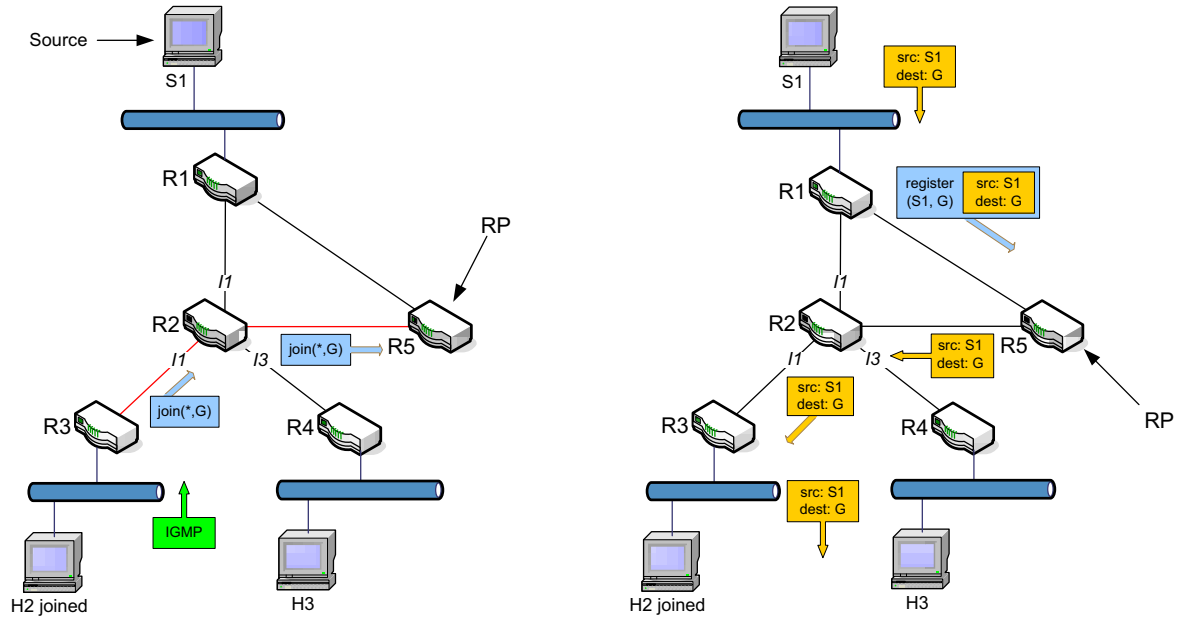
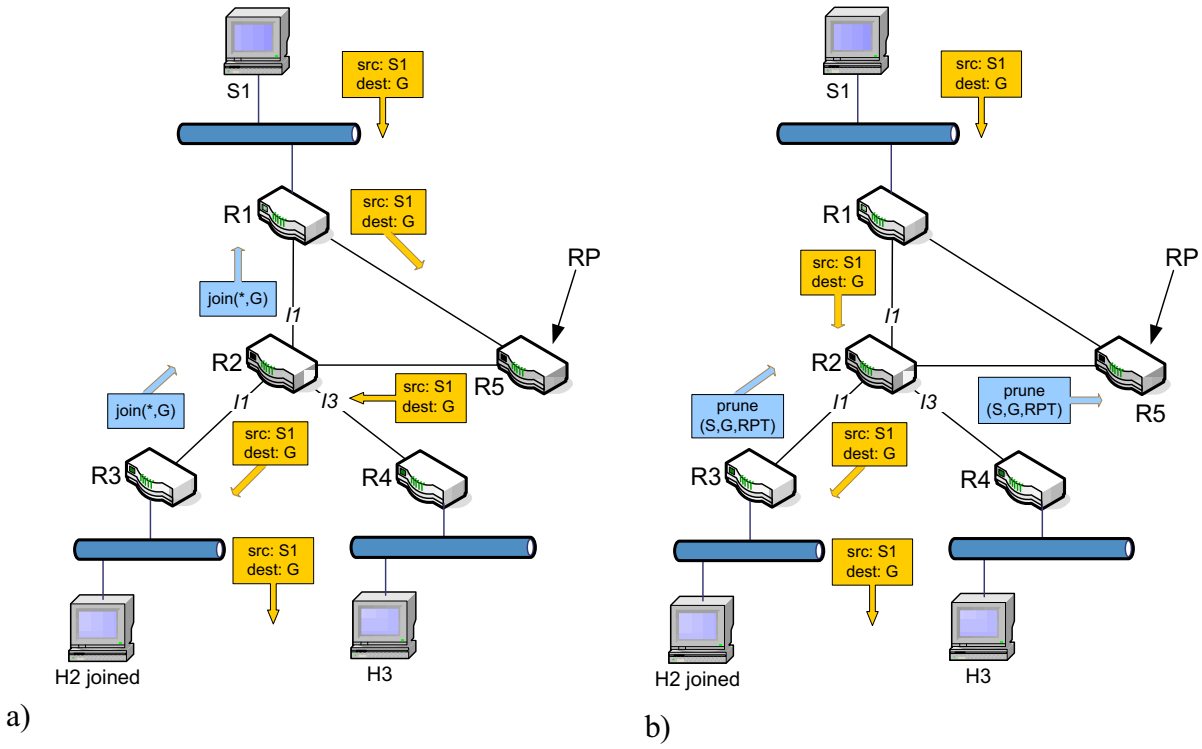


Figure 61: PIM-SM building a multicast tree – a) When receiver H2 joins the multicast group, join message is propagated to RP, b) Source sends multicast packet which is received by R1. R1 unicast-encapsulates this packet into the register message and sends it to RP. Upon receiving the register message, 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 4µs (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: $T_{n,x}$(switch to source tree)=(2n'+x-3)*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 $T_{n,x}$(switch to source tree)=(2n+x-5)*t. For n=5 and x=3, $T_{n,x}$(shared tree)=32µs and $T_{n,x}$(switch to source tree)=32µs. For n=100 and x=50, $T_{n,x}$(shared tree)=600µs and $T_{n,x}$(switch to source tree)=980µs. Note that this is the time needed by a receiver to join a multicast group.



a) b)

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*1sec+0.5sec=2.5sec, before stopping the multicast traffic flow.

PAPER 1

Extending CPL with context ontology

Alisa Devlic
Appear Networks
Kista Science Tower
16451 Kista, Sweden
alisa.devlic@appearnetworks.com

In Proceedings of Mobile Human Computer Interaction (Mobile HCI 2006) Conference Workshop on Innovative Mobile Applications of Context (IMAC), Espoo/Helsinki, Finland, September 2006.

Extending CPL with context ontology

Alisa Devlic
Appear Networks
Kista Science Tower
16451 Kista, Sweden
alisa.devlic@appearnetworks.com

ABSTRACT

Communication has always been an essential part of people's everyday life. Nowadays most of people would like to be reachable on multiple devices at anytime, anyplace. As a consequence, there has been a need to know and exploit a user's availability for communication (so called presence information), so that he/she can control incoming calls and make the decision to accept this call or not based on the user's current context. The appearance and acceptance of Session Initiation Protocol (SIP) as a signalling protocol for next generation networks has opened the door for multiple services, such as Voice over IP (VoIP), chat, games, instant messaging, and other innovative communication services. Built on top of existing data communication networks, this has enabled easy integration of voice and data services. In this paper we present an idea on how to use the context information to enhance the power of existing SIP call control services, to enable users to have greater control over their incoming/outgoing calls. These services are implemented using Call Processing Language (CPL), a language to describe and control Internet Telephony Services. They are extended with context parameters to permit context-based decision making based on context ontology. We want to show how easy is to add new context parameters to the CPL and how complex criteria can be built using our solution.

Keywords

Context-aware, SIP, VoIP, CPL, call processing, ontology

1. INTRODUCTION

The paper describes a motivation and benefits for integrating contextual parameters into call processing capabilities of the existing VoIP system. It also identifies the essential components and concepts needed to realize this solution. By contextual information we mean any information that can characterize a user and his/her current situation, such as: the location, task, activity, time of the day, etc. We have tried to illustrate real life scenarios that would capture a need for context parameters and call processing possibilities an end user would use. From the scenarios we have identified the needed parameters and modelled them in the ontology to represent a user's current context.

In the paper we show how can this context information enhance the functionalities of existing SIP [11] call control services by offering a user the possibility to decide whether to accept an incoming call based on his/her current context. These services are implemented as Call Processing Language (CPL) [10] scripts, and their behavior is described using a set of rules. We have extended the CPL syntax to support rules based on this context information.

We have utilized a scalable and reliable open source SIP platform, called SIP Express Router (SER) [1], to upload and execute CPL scripts. It can act as a SIP registrar, proxy, or redirect server. We have extended its functionality to support our context-based CPL scripts.

The goal of our research work is to show the benefits of extending CPL scripts with context ontology, allowing the easy extensibility and enabling simple CPL to be more powerful.

The paper is organized as follows: first, we give a short introduction of CPL scripts, SER, and call processing capabilities that can be achieved with the existing syntax. Second, we try to illustrate real-life scenarios to indicate the need for context parameters and model them in the ontology. Third, we give a description of our prototype and its components. Finally, we conclude the paper including the plans for the future work.

2. CPL SCRIPTS

CPL scripts are XML-based documents. The Document Type Definition (DTD) is specified in the cpl.dtd file available at [2]. It consists of ancillary information about the script and call processing actions. Ancillary information is information which is necessary for a server to correctly process a script, but which does not directly describe any operations or decisions. A call processing action is a structured tree that describes operations and decisions a telephony signalling server performs upon a call setup event. There are two types of call processing actions: top-level actions and subactions. Top-level actions are actions that are triggered by signalling events that arrive at the server. Two top-level actions are defined: "incoming", the action performed when a call arrives whose destination is the owner of the script,

IMAC Workshop 2006, Espoo, Finland

and "outgoing", the action performed when a call arrives whose originator is the owner of the script. Subactions are actions which can be called from other actions.

The graphical representation of a CPL action is shown in Fig. 1. An action is described by a collection of nodes that describe operations that can be performed or decisions that can be made. A node can have several parameters, which specify the behavior of the node. They usually have outputs, which depend on the result of a decision or action. Nodes are represented as boxes, and outputs as arrows. Nodes are arranged in a tree, starting at a root node. Outputs of nodes are connected to other nodes. When the action of the top-level node is invoked, based on the result of that node a server follows one of the node's outputs, and the subsequent node it points to is invoked. This procedure is repeated until the node with no outputs is reached.

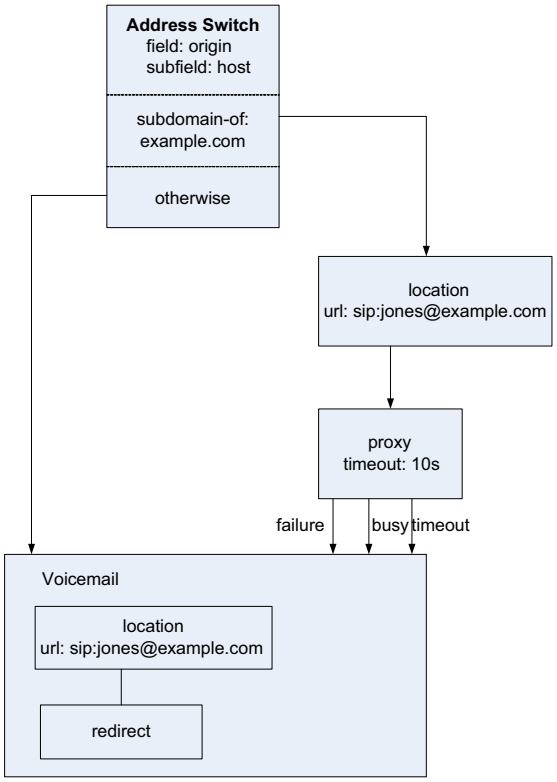


Figure 1: Graphical representation of a CPL script

There are four types of nodes: *switches*, which represent choices a CPL script can make, *location modifiers*, which add or remove locations from the location set, *signalling operations*, which cause signalling events in the underlying protocol, and *non-signalling operations*, which trigger behavior which does not effect the underlying protocol.

CPL scripts can reside on a SIP proxy server, an application server, or intelligent agent. In our case, we have uploaded CPL scripts to the SIP proxy server, SER (Fig. 2). 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 [6].

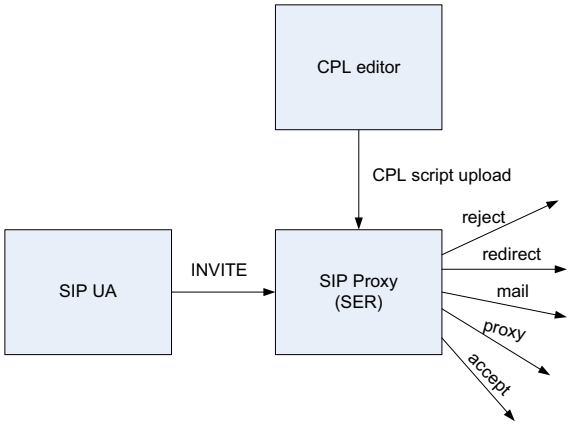


Figure 2: Call processing logic

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.

3. APPLICATION SCENARIOS

We have tried to identify the need for context parameters by specifying scenarios that would be applicable in the real life situations. We have also wanted to illustrate call processing possibilities that an end user would use.

1. Alice works for a company "example.com". When she is **in a meeting** and is **presenting** new solution to the management staff, she wants to forward all incoming calls to her voicemail. On the other hand, if she is **listening** to someone's presentation she may want to receive calls that are labeled with an urgent priority on her mobile phone.

2. When she is on **vacation**, Alice wants to reject all incoming calls from the company.

3. When Alice is in her **car**, she would like for safely reasons to set the policy to disable all outgoing calls from her mobile phone while **driving**.

4. When she is on the **business trip** in France, she prefers to get e-mails instead of receiving expensive roaming calls on her mobile phone. However, if the language settings are set to French and the call is coming from her customer's company "trade.com", these calls should be forwarded to her mobile phone.

From the scenarios above we can extract the following context parameters: the **context owner** (the person to whom the context parameters relate to; i.e. Alice), **location** (office, home, vacation, business trip, car), **task** (in a meeting, at lunch), and **activity** (presenting,listening, driving). We

modelled the context parameters in the Web Ontology Language (OWL) ontology, available at [7]. The reason for it is that we can model the parameters into higher-level concepts, and use these concepts in CPL scripts for decision making.

4. CPL EXTENSIONS FOR CONTEXT

CPL extensions for context were designed to describe call processing services related to context relevant to Internet Telephony. We have chosen to utilize context parameters identified in the previous section: context owner, his/her location, task, and activity. In these extensions, we define a **context-switch** to support the services whose decisions are based on the context information of an end user.

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.

Adding a context switch allows an end user to make decisions based on the current context parameters of a context owner. The context owner can be the user himself/herself or the user can specify context for some other person. However, we will not consider this later case further in this paper. Values of context parameters are specified in the user's ontology document. The user's context determines which script will be uploaded to the SER. When the context-switch node is invoked, it will match the context parameters set by ontology with context values in the CPL script and return the decision of how to process an incoming/outgoing call (accept, reject, redirect, voicemail, etc.).

Node "context-switch" has one parameter "owner", that identifies a context owner. Node "context" is the output of the "context-switch" node. It specifies different context attributes, such as: "location", "task", and "activity" of a context owner. These attributes were identified after analyzing application scenarios for a typical business user. Syntax of the node "context-switch" and the "context" node is shown in the Table 1.

Table 1: Syntax of a context-switch

Node:	context-switch	context-switch node
Outputs:	context	context parameters
Parameters:	owner	context owner

Output:	context	context node
Parameters:	location	location of a context owner
	task	task status
	activity	activity status

The definition of CPL extensions for context is specified in the file "context.dtd" [8]. An example of CPL script based on this extended CPL is shown below. 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.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cpl SYSTEM
'file:C:/Programs/CPLEd/context.dtd'>
<cpl>
  <incoming>
    <context-switch owner="jim">
      <context location="grimeton" task="meeting"
               activity="presenting">
        <reject status="reject"
               reason="InMajorMeeting_And_Presenting"/>
      </context>
    </context-switch>
  </incoming>
</cpl>
```

5. CONTEXT-AWARE VOIP PROTOTYPE

The idea of context-aware VoIP prototype was to make call processing dependent on context parameters, so as to make it easier to specify a suitable action to be taken.

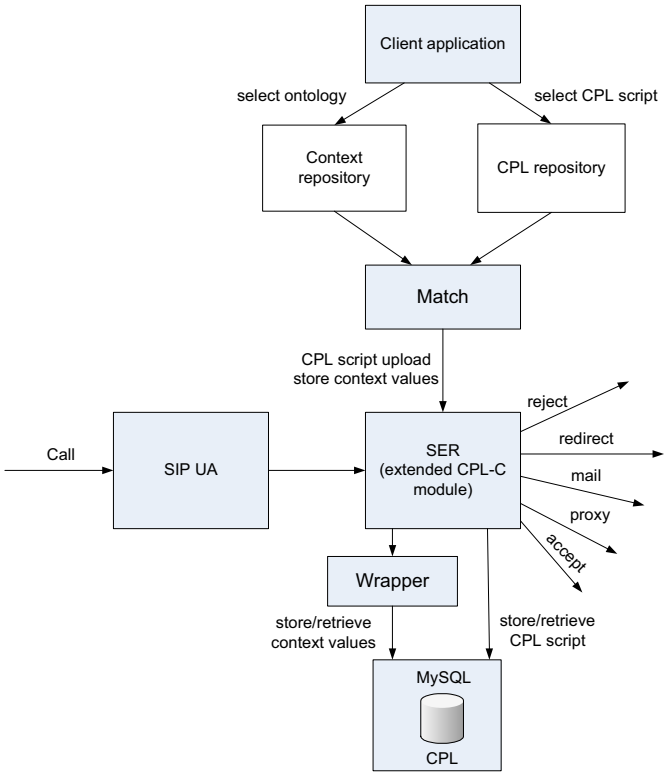


Figure 3: Context-aware VoIP prototype

When the user wants to upload a context-based CPL script (Fig. 3), he/she has to first upload the ontology to the match component, which first parses it, 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 we implemented in the lab consists of four components: a client application, match component, wrapper, and extensions to the CPL-C module [5] of SER.

5.1 Client application

A simple client application is used for uploading ontologies and CPL scripts (Fig. 4). CPL scripts that are not context-based can be uploaded directly, without the need to first upload the context ontology. The aplication was designed to be used from different machines and different locations, hence the preferable implementation is as an applet.

Figure 4: Client application

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 (Fig. 5) to browse for a file to open (i.e. in this case ontology).

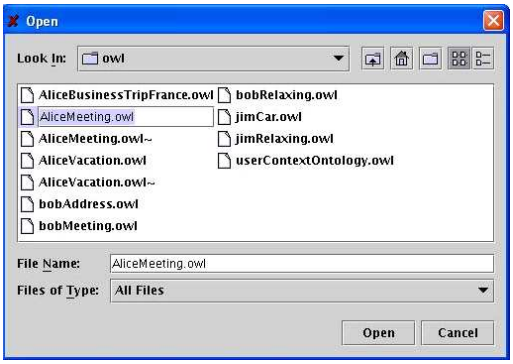


Figure 5: File chooser dialog

5.2 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.

5.3 Wrapper

The wrapper was created to pass context values 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.

5.4 CPL-C module extensions

We had to modify the cpl-c module of the SER source code to support adding of our context-switch and context node. This is explained in more detail in [9].

6. CONCLUSIONS

We described in the paper the solution on how to extend CPL with context parameters to make call decision-making process more powerful. In the initial measurements, we saw that the SER server with CPL module is very scalable - it can register 4000 users in 10 seconds [12]. When we add a context-switch at a time to the script, the response time increases about 5%-24%, what we expected, because of added reasoning process and storage of values in the database [9]. In the future work we plan to: a) build a proxy proxy between SIP UA and SIP proxy to be able to make "advanced" calls based on the call priority, language that caller has set, and a free form string set, b) add sensor services like positioning systems (e.g. Cell-ID, GPS) and Calendar service to set the available context parameters: location, task, and activity in the ontology, c) find the way how to dynamically plug-in new sensor services from the environment and use them as context providers, d) make an ontology and CPL script uploaded automatically by the system, and not explicitly by the user, and e) do more measurements of initiating call requests from multiple users in the time to determine the SER's average and peak call processing capability.

7. BIOGRAPHY

I work as research engineer and industrial PhD student in the company Appear Networks, which is involved in two EU FP6 projects: MIDAS (Middleware Platform for Developing and Deploying Advanced Mobile Services) [3] and SIMS (Semantic Interfaces for Mobile Services) [4]. Our research group is looking the way to design a middleware which will simplify and speed up the task of developing and deploying mobile applications and services, taking into account large number of users, their context information, limited connectivity infrastructure, and short notice communication setup. We also investigate the usage of semantic interfaces for rapid development, dynamic discovery, and composition of mobile services.

This research work was carried out in the Wireless center of the Royal Institute of Technology and I would like to thank my advisor, Prof. Gerald Q. Maguire Jr., for his help and guidance.

8. REFERENCES

[1] CPL XML DTD draft. http://www.iptel.org/ser/, April 2006.

[2] SIP Express Router (SER). http://xml.coverpages.org/CPL-DTD-200201.txt, January 2002.

[3] MIDAS (Middleware Platform for Developing and Deploying Advanced Mobile Services) project. http://www.ist-midas.org, January 2006.

[4] SIMS (Semantic Interfaces for Mobile Services) project. http://www.ist-sims.org/, January 2006.

[5] CPL-C module of SER. http://www.voip-info.org/wiki-SIP+Express+Router, June 2006.

[6] CPLEd, a free graphical CPL editor. http://www.iptel.org/products/cpled/, September 2002.

[7] A. Devlic. Context ontology. http://web.it.kth.se/ devlic/userContextOntology.owl, June 2006.

[8] A. Devlic. CPL extensions for context DTD. http://web.it.kth.se/ devlic/context.dtd, June 2006.

[9] A. Devlic. CPL extensions. Report for the Practical VoIP course, http://web.it.kth.se/ devlic/CPL

[10] J.Lennox, X.Wu, and H.Schulzrinne. Call Processing Language: A Language for User Control of Internet Telephony Services. RFC 3880, http://www.ietf.org/rfc/rfc3880.txt, October 2004.

[11] J.Rosenberg, H.Schulzrinne, G.Camarillo, A.Johnston, J.Peterson, R.Sparks, M.Handley, and E.Schooler. SIP: Session Initiation Protocol. RFC 3261, http://www.ietf.org/rfc/rfc3261.txt, June 2002.

[12] Y. Oukhay. Context-aware services. In *Master of Science Thesis*. Royal Institute of Technology, March 2006.

PAPER 2

Context retrieval and distribution in a mobile distributed environment

Alisa Devlic and Erik Klintskog
Appear Networks
Kista Science Tower
16451 Kista, Sweden
alisa.devlic@appearnetworks.com, erik.klintskog@appearnetworks.com

In Proceedings of Third Workshop on Context Awareness for Proactive Systems (CAPS 2007), Guildford, UK, June 2007.

Context retrieval and distribution in a mobile distributed environment

Alisa Devlic and Erik Klintskog

Appear Networks
Kista Science Tower, 16451 Kista, Sweden
{alisa.devlic,erik.klintskog}@appearnetworks.com
http://www.appearnetworks.com

Abstract. Context-aware services are gaining momentum in mobile computing. To enable rapid development of context-aware services, context information has to be retrieved from the environment, modeled, processed, and distributed to these services.
MIDAS is a European research project concerning 3G and beyond, which aims to define and implement a platform to simplify and speed up the task of developing and deploying mobile applications and services. MIDAS context engine provides mechanisms to retrieve, model, synthesize, and distribute context information in a distributed, mobile environment. This paper presents a way to retrieve and distribute context information using context queries and triggers. A novel approach to perform context synthesis will be presented using operators.

Key words: Distributed context engine, context query, context trigger, context synthesis, context retrieval and distribution

1 Introduction

Mobile applications, unlike desktop applications, need to adapt to environmental changes such as change in the user's location. As the user changes location, the situation and environment around him/her changes (nearby people, places, objects), i.e., different context information becomes available [1]. The challenge is to utilize available sources of context information within a suitable time and make this information visible and usable by applications in a distributed system.

1.1 Motivation

This paper describes an approach to be used by a context engine to retrieve distributed context information concerning users and their surroundings, and to provide this context information to mobile applications.

The context information is retrieved from its source based upon an application's request. *Context information sources* are wrappers that provide semantic markup for raw context information coming from physical and virtual sensors, which can be separated hardware devices or software running on the user's

device. Often, the node where the context source resides is different from the node which runs the end user's application.

Applications use high-level context information, which is abstracted from the information obtained from context information 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 are the long response times which the end-user (or their application) needs to wait for result to his context query [2], especially when large data sets and rule sets are applied [3].

Contexts in mobile environments may change at very high rate. A problem arises if contexts are updated in the database as soon as new context information becomes available, but this information is subsequently not used by context consumers (i.e. applications). Frequent transfer of context information over the network leads to high resource consumption on the devices which host context information sources (in terms of power consumption, bandwidth utilized, etc.).

1.2 Contribution

Distributed context retrieval and distribution raises many issues, some of which have just been elaborated. The proposed solutions for those problems are the following:

- *Retrieval of context information from the remote context information source.* Our approach is 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 also cached in a database until its validity expires.
- *Context queries and context triggers for context retrieval.* 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". Context queries can be simple (i.e., requiring only a database query for the specific context information) or complex (i.e., requiring context synthesis).
- *Context synthesis using operators.* The approach we propose is to use *operators* for context synthesis. Operators are functions that take as input certain context information, and produce as output new context information (e.g., "UsersInRange" takes "UserID=alice" and "Range=500" as inputs, and produces "UserIDBag=bob, ted" as output). Operators are described with a common ontology, similar to the representation of context. They are implemented as programs that perform the operation described in the operator's ontology. Operators can be application- or domain-specific. Therefore, they are meant to be inserted into the context engine by the application or system developer by extending the existing ontology and implementing the operator's function for an application or domain purpose.

2 Context management

Context management encompasses activities starting with acquiring context information, context modeling and reasoning, to providing high-level context information to services relevant to the end user. This paper focuses on providing context information produced by context providers to context consumers.

A distributed approach for retrieving, synthesizing, and disseminating context information for all end users to mobile applications, based upon the MIDAS project, will be presented in this paper. In MIDAS, context consumers are mobile applications that retrieve context information using a context engine. The context engine provides mechanisms to retrieve, model, synthesize, and distribute context information in a distributed, mobile environment.

2.1 Context query

A context query is a request for context information. The context query can use an operator to execute an operation in order to produce the desired context information. Operators can be considered as functions that take an input (or list of inputs), perform an operation, and produce an output. Operators are briefly described in Section 3, which illustrates the process of context synthesis.

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 obtains its context information directly from the repository, i.e. the database, without using operators, these are called simple queries. A context query also contains a so called context quantifier, which influences the way context information is retrieved (specifically whether one waits for *one* or for *all* context values, or waits for context values for a specified *number of milliseconds*), before composing and sending back the result.

The context engine distinguishes between *static* and *dynamic context information*, and issues *local* and *remote context queries* accordingly, as illustrated in Fig. 1. Static context information is the type of context information that doesn't vary with time and/or it is not influenced by other processes (e.g. user profiles). Local context queries are queries for static context information from the context repository on the same node. Dynamic context information changes frequently (e.g. a user's location). Such information is retrieved by sending remote context queries to context information sources (on remote nodes), providing this context information. Context information sources request the raw data from sensors and model this data as context information. This context information is cached locally on the context repository.

The context engine contains a context mapping component that serves as a yellow pages for finding addresses of context information sources that provide different types of context information. The context mapping component issues local and remote context queries. Communication between the context mapping component and the context repository is based on queries/responses.

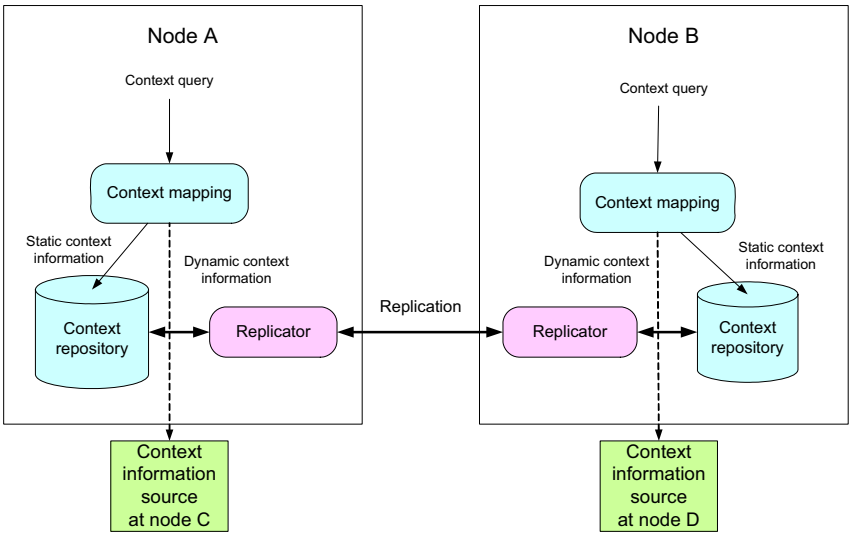


Fig. 1. Local and remote context query

Communication between the context mapping component and context information source(s) is(are) realized by activating context triggers and exchange of queries/responses.

Static context information is stored in the context repository and is replicated across nodes in a distributed system (see Fig. 1). Details about the replication mechanism will not be elaborated upon in this paper.

2.2 Remote retrieval of context information

As an example of the retrieval of location information from the remote context information source, Fig. 2 shows the mapping of context to its source and the querying of a remote source. Note that nodes A and C in Fig. 2 correspond to the nodes A and C in Fig. 1. The context mapping component on the node A maps context parameters to nodes that provide values for those parameters. For the location information, there is a mapping in the mapping component's table to nodes C and D (see Fig. 2). When the context query arrives and the context mapping component is asked for location, the first available node (node C), is selected.

The context mapping component will send the query for location to the node C, i.e. *C.getRemote(Location)*. The communication and routing component from the MIDAS framework will take care of finding the remote node and transferring the appropriate message.

On the node C, the context information source is realized by an application (e.g., GPS application that talks to a GPS sensor and wraps the GPS coordinates to location information). This application needs to register its instance and the

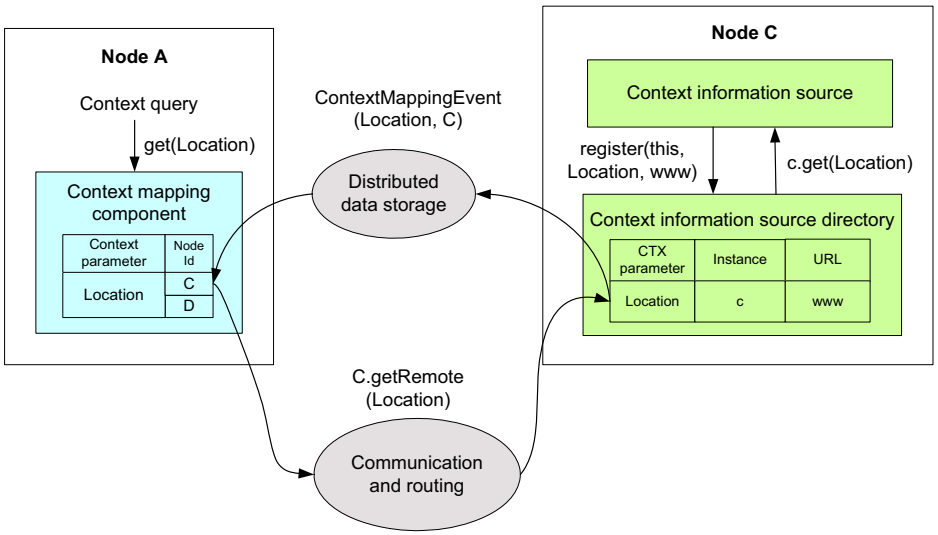


Fig. 2. Retrieval of location information from remote context information source

context parameter it provides to the context information source directory, at the startup. Optionally, this application can be replaced by a web service, in which case it would register its URL, instead of the application instance, to the context information source directory. After this registration is stored in the directory, the context mapping event is fired, containing the pair of context parameter and node identifier (i.e., *ContextMappingEvent(Location, C)*). The context mapping component will store this pair into its table. Thus, by listening to those events the context mapping component updates its context mappings. Distributed data storage acts as a mediator for context mapping events, as it stores all data in the MIDAS framework, and will pass the received event to the node A.

When the query for a location arrives at node C, the context information source directory will be searched for the instance of context information source, which will be invoked to get the current location (i.e., *c.get(Location)*).

2.3 Context trigger

Statefull context queries are implemented using context triggers. Context triggers execute a predefined action when a specified condition is fulfilled, i.e. context information has reached a certain state. A context trigger is specified with the context condition that needs to be fulfilled in order to trigger a context action, the context action itself, and the type of context condition event (" *OnEnter*" - when user enters or " *OnLeave*" - when user leaves the context). A context condition is specified as context parameter name-value pair. A context action specifies an action to be performed when the context condition event is fired, the time when the action should be triggered, the duration of the action, and/or the interval in which the specified action is periodically triggered.

When created, context triggers are inserted directly into context sources. Context sources pool sensors for context information and compare the retrieved value with the specified context condition. When the match is discovered, the context condition event is fired. The context engine listens to those events and executes an appropriate action.

3 Context synthesizing

Context synthesizing is a process of generating new knowledge (in the form of more abstract context), as a result of a reasoning process applied to context information that was already present in the system (e.g., deriving the abstract concept of weather by combining temperature, humidity, and wind speed). This context synthesis requires some rules (which are presented by operators in this paper) to drive the reasoning process forward.

In our distributed context retrieval, context synthesis begins with interpretation of a context query to retrieve a description and implementation of the operator that matches the requested operator's description in the query. The context query is represented by an expression of operators, arguments, and a context quantifier e.g. *(InRange (alice, 500, Users), All)*, which according to the context query definition means "find all users within 500 meters from alice".

The description of the retrieved operator specifies the operation performed by this operator, the required input arguments, and the output returned by this operation. Before the operation is performed, the missing input values are either obtained from the context information source (e.g. users locations) or are explicitly stated by user in the query (e.g. user id, range). 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.

3.1 Operators

This subsection will provide a specification of operators. Let **Op** be a set of operators:

$$\mathbf{Op} = \{op_1, op_2, \ldots, op_n\}, n \in \mathbb{N}$$

An operator $op_i \in \mathbf{Op}$ is represented with a bundle of *operator's description* and *implementation*:

$$op_i = \{\mathbf{desc}(op_i), \mathbf{impl}(op_i)\}$$

For every $op_i \in \mathbf{Op}$, let:

- **In** denote a set of required inputs $\mathbf{In} = \{in_1, \ldots, in_n\}, n \in \mathbb{N}$
- **Out** denote a set of possible outputs $\mathbf{Out} = \{out_1, \ldots, out_m\}, m \in \mathbb{N}$
- **F** denote an operation that takes provided inputs and produces an output $\mathbf{F} : In \rightarrow out_j$, $out_j \in Out$
- **Uses** denote a set of used operators $\mathbf{Uses} = \{op_i, \ldots, op_j\}, i, j \in \mathbb{N}$

An operator's description $\mathbf{desc}(op_i)$ is defined as:

$$\mathbf{desc}(op_i) = \{name_i, F_i, In_i, out_i, Uses_i\}$$

where:

- $name_i$ is the name of op_i
- F_i is the operation provided by op_i
- In_i list of inputs for F_i
- out_i an output produced as a result of F_i
- $Uses_i$ list of other (simpler) operators op_i uses in its execution

An operator's implementation $\mathbf{impl}(op_i)$ is defined as:

$$\mathbf{impl}(op_i) = \mathbf{impl}(F_i(In_i)) = out_i$$

The above definition specifies operator's implementation as an implementation of the operation F_i, which takes In_i as arguments, produces out_i as a result, and invokes implementations of used operators (i.e. $\mathbf{impl}(Uses_i)$) in its execution.

```
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.
```

An example implementation of the operation F_i is the above program F(In). The program takes a list of inputs, here represented by variables *In*. At the beginning, the program checks if the list of used operators is empty, and performs specified operation on the list of inputs. If the list of used operators is not empty, then the program will invoke each operator and pass as arguments the newly obtained inputs. For simplicity operations of op_i and other operators in the program have the same name (i.e. F).

3.2 Operators Ontology

Consider the following context queries:

1. "Find all users in range 500m from Alice."
2. "Find all streets in range 500m from Kista Centrum."
3. "Find all towns in range 20km from Stockholm."
4. "Find all phones in range 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 would employ its own implementation of an operator, it would significantly increase the database storage required along with the time needed to find the right one. Furthermore, the right 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 inputs and outputs are described in the operator's ontology (e.g. the context query asks for streets in the user's range, and available "InRange" operator implementations return postal codes instead of streets).

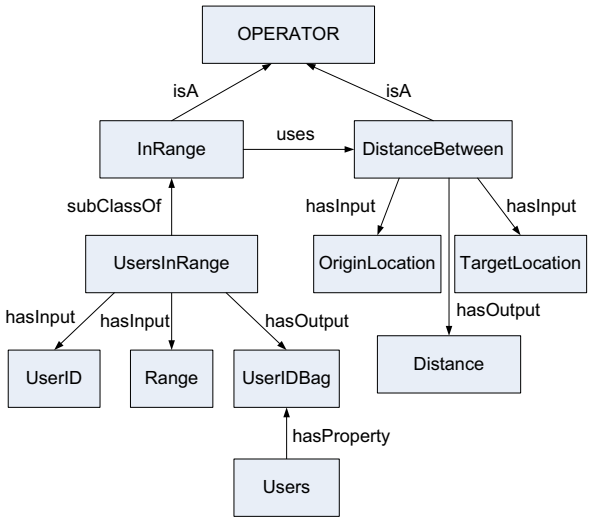


Fig. 3. Ontology of "InRange" operator

Operators often have dependencies on another operator(s): the "InRange" operator utilizes the "DistanceBetween" operator, as shown in Fig. 3. Furthermore, the "DistanceBetween" operator can have multiple implementations, such as: "DistanceBetweenUsers", "DistanceBetweenCities", "DistanceBetweenPhones", etc. The querying process becomes even more complex for solutions without proper semantics and hierarchical relations.

The following example will show how to find an appropriate operator for the context query: "Find all users in range 500m from Alice". Fig. 3 shows a subset of the operators' ontology, defining two operators: "InRange" and "DistanceBetween". "InRange" has a subclass called "UsersInRange" that takes two inputs "UserID" and "Range", and produces "UserIDBag" as an output. "UserIDBag" is a set of "UserID"s, and represents a property of the class Users.

The reasoning process will find the appropriate operator that satisfies input and output requirements set in the query. The subclass of the oper-

ator "InRange", "UsersInRange", with its method *List<User> getUsersInRange(userID, range)* and its dependency operator "DistanceBetween" having the method *Distance getDistanceBetween(originLocation, targetLocation)* are the result of this reasoning process. An execution of *getUsersInRange* and *getDistanceBetween* will yield a synthesized context, which will be returned to the application and cached in the context repository for the time specified by the "InRange" operator.

4 Related work

Authors of Context Toolkit mention in [4] that an automatic path creation can be adapted to be used for refining and transforming raw sensor data into higher-level context data (which we call context synthesis). This automatic path creation relies on operators, special services that transform data from one form into another (e.g. GPS location data to ZIP code data). Operators could be automatically composed based on high-level needs and on available resources.

The issue of matching application requests for context information and available context parameters has been tackled in [5]. If both sets of parameters are described in terms of ontologies, the matching problem lies in mapping the request parameters from an application-domain ontology to candidate matching context descriptions from the context domain ontology. A solution proposed was to synchronize changes in one ontology with another by replacing the unmatched candidate parameter with a semantically related one that has a match in the application-domain ontology. We mitigate this problem by introducing operators in context queries that specify the context information that needs to be retrieved.

In [3] context queries are performed on a context knowledge base using RDF Data Query Language [6]. The limitation of this approach lies in querying over the existing information stored in the context model using triple (<*subject, predicate, object*>) patterns. Our approach allows operations performed on the context data resulting in new information that previously did not exist in the system. This gives applications greater freedom in forming context queries.

Mobilife project [7] has developed Context Management Framework (CMF) for discovery, exchange, and reasoning on context information. Context information produced by Context Providers is discovered and delivered to Context Consumers using Context Brokers. An appropriate Context Provider is found by semantic matchmaking of required and advertised services. A context engine employs a direct mapping of a Context Provider node and context parameter provided. CMF uses a proxy-based design to manage distributed Context Brokers, where a single Context Broker proxy is the first point of contact to any request for a Context Provider. MIDAS context architecture utilizes the approach of Super Nodes (i.e. nodes having more resources: memory, CPU, etc.) which perform context synthesis and host context mapping component, being discovered by a procedure that finds the closest available Super Node to a Context Consumer. CMF uses rule-based and Bayesian model reasoning, whereas a context engine introduces operators for context synthesis.

5 Conclusion

We have presented a model for retrieving and distributing context information in a mobile distributed environment. Based on the observation that different context information has different update and read patterns, we provide two different mechanisms for context distribution. Context information which is static in its update pattern should be replicated among nodes and context which is volatile should be distributed by remote reads. Example of the former include user profiles which rarely changes. Example of the latter could be a users position.

Moreover, we have introduced the use of typed 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, based upon the operators' ontology. This results in a system flexibility, extensibility, and enhances code reuse.

We are currently implementing a prototype of a context engine utilizing the proposed mechanisms of context queries, triggers, and operators. Operators will be described using OWL-DL, implemented as Java scripts using BeanShell [8], and their performance will be compared with Semantic Web rule-based reasoning. Privacy concerns and context scope will be added later in the project.

Acknowledgments. This work is part of the EU IST MIDAS project sponsored by European Commission under contract 027055.

References

1. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In Proceedings of the Workshop on Mobile Computing Systems and Applications. IEEE Computer Society, Santa Cruz, CA (1994) 85–90
2. Wang, X.H., Zhang, D.Q., Gu, T., Pung, H.K.: Ontology based context modeling and reasoning using OWL. In Proceedings of Workshop on Context Modeling and Reasoning. Orlando, Florida USA. IEEE (March 2004) 18–22
3. Wang, X., Dong, J.S., Chin, C.Y., Hettiarachchi, S.R.: Semantic Space: An Infrastructure for Smart Spaces. Pervasive computing, July-September Vol. 3., IEEE (2004)
4. Hong, J.I., Landay, J.A.: An Infrastructure Approach to Context-Aware Computing. Human-Computer Interaction. Vol. 16 (2001) 287–303
5. van Kranenburg, H., Bargh, M.S., Iacob, S., Peddemors, A.: A Context Management Framework for Supporting Distributed Context-Aware Applications. Communications Magazine. Vol. 44. IEEE (September 2006) 67–74
6. Seaborne, A.: RDQL - A Query Language for RDF. W3C Member Submission. HP Labs Bristol (January 2004)
7. EU IST-FP6 Mobilife project: http://www.ist-mobilife.org (2006)
8. BeanShell lightweight scripting for Java, http://www.beanshell.org/

PAPER 3

Evaluation of context distribution methods via Bluetooth and WLAN: Insights gained while examining Battery Power Consumption

Alisa Devlic
Appear Networks Systems AB &
Royal Institute of Technology (KTH)
Stockholm, Sweden
devlic@kth.se

Alan Graf
Ericsson AB
Tellusborgsvägen 83-87
Stockholm, Sweden
alan.graf@ericsson.com

Paolo Barone
HP Innovation Center
Via Grande 4
Milan, Italy
paolo.barone@hp.com

In Proceedings of the Fifth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2008), Dublin, Ireland, July 2008.

Evaluation of context distribution methods via Bluetooth and WLAN: Insights gained while examining Battery Power Consumption

Alisa Devlic
Appear Networks Systems AB &
Royal Institute of Technology (KTH)
Stockholm, Sweden
devlic@kth.se

Alan Graf
Ericsson AB
Tellusborgsvägen 83-87
Stockholm, Sweden
alan.graf@ericsson.com

Paolo Barone
HP Innovation Center
Via Grande 4
Milan, Italy
paolo.barone@hp.com

ABSTRACT

In a traditional context-aware system, most context information is local to a device. However, we may need access to context information from outside the device. Increasingly mobile electronic devices are equipped with Bluetooth and/or WLAN network interfaces. Both of these technologies enable ad hoc discovery & networking. In this paper we evaluate the use of these technologies for context distribution within a local area (i.e., limited to a single hop). Using Bluetooth, we begin by discovering devices using Bluetooth's discovery protocol, collect their context information, create an XML file containing this information, and distribute this file to all discovered devices, such that every device now has the same context information. Next we perform the same discovery, collect, and distribute functions, but using WLAN. In each case 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 compare both approaches to context distribution in terms of battery power consumption. We observe that Bluetooth consumes 2-6 times more energy for transmission of a 1MB file to two devices than to discover these two devices. Furthermore, the transfer of this file is two times slower than WLAN, and we must unicast this file to each device. Multicasting via WLAN proved to be *less energy consuming* than the Bluetooth transmission, if data is to be sent to more than three users. In addition, the energy to discover 2 devices along with their services using Bluetooth consumed 52 times more energy than to receive the same amount of data via a WLAN multicast. Thus, this paper shows that it is *more energy efficient to distribute context knowledge* to other devices, than having each device learn this information itself. Finally, we give equations for calculating the battery power consumption of transmitting data using any protocol that runs over Bluetooth or over WLAN.

MobiQuitous 2008, July 21–25, 2008, Dublin, Ireland.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication; D.4.8 [**Performance**]: Measurements; C.4 [**Performance of systems**]: Measurement techniques

General Terms

Measurement, Performance, Design, Experimentation.

Keywords

Battery power evaluation, context distribution, Bluetooth, WLAN

1. INTRODUCTION

In a context-aware system, devices are frequently mobile and geographically distributed, thus devices need to timely discover, collect, and adapt based upon context information from its surroundings. Here, context is used to describe the situation of an entity [1]. Alternatively, a device can share its context knowledge (which it has discovered and acquired) 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. Advance knowledge of context is powerful, because it can potentially reduce the delay or energy required by a device that needs to adapt to a new environment. If this context information is distributed in advance, then the query can be answered locally. There is a cost related to distribution of context that will never be used by another peer (in terms of communications, storage, and battery power consumption), but as much of this information changes slowly and this information can include other context, such the available projectors, scanners, printers, access points, power outlets, etc. in the same environment, the probability that none of this information is used decreases. Thus there will be a trade-off between how far context information should propagate and how useful this information is in advance (for adaptation by both the device and the user).

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.

Due to the limited power available, hardware sensors such as mobile phones, sport sensors, medical sensors, wireless keyboards, mice, etc. are typically equipped with short range radios, quite often Bluetooth class 2 or 3 radios. While more powerful devices, such as handheld devices, tablet PCs, laptops, etc. are equipped with both Bluetooth and WLAN network interfaces. 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 consider a single hop).

The context distribution methods that we describe 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. We have performed these operations in cycles and measured the battery power consumption starting with a fully charged battery and continuing until the remaining battery power is too low to continue and the application is shut down by the operating system.

In this paper we present our preliminary results and experiences from performing these measurements over Bluetooth and WLAN on two different handheld devices: HP iPAQ 4150 and 6915. In order to evaluate the cost of sending small vs. big chunks of data, we append the context information each time it is collected to a file, which is in turn transmitted to all the discovered devices. As the file grows in size, we are able to collect data for different file sizes; hence we can use this data to estimate the amount of battery power required for any specific file size. Fitting this data to a formula for the cost of transferring data in terms of J/bit enables us to estimate the power consumption for any given protocol that will run over Bluetooth or WLAN. Therefore we can calculate the optimum frequency (with respect to the battery power consumption) to distribute and retrieve relevant context information.

This paper is organized in seven sections. Section 2 presents our proposals for context distribution using Bluetooth and WLAN. Section 3 explains the hardware and software used for measuring battery power consumption. Sections 4 and 5 describe how measurements were performed; discuss the results obtained using two different handheld devices, compare both wireless link technologies, and describe the insights gained by performing these measurements. Section 6 provides a brief overview of the related work. We conclude in section 7 with a recapitulation of the results and future plans.

2. CONTEXT DISTRIBUTION

2.1 Bluetooth Context Distribution

Context distribution using Bluetooth, as illustrated in Figure 1, works as follows: a device with a Bluetooth interface and a context distribution application initiates discovery of nearby devices along with the context information which each can provide and adds this information to a file, then sends this file to all discovered devices – in order to propagate this information.

As context information we are 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. It is the distribution of the *aggregated information* which enables the discovery of devices and services *beyond the single hop limit*. Note that Bluetooth limits this distribution of context to seven or fewer simultaneous devices that are within a range of ten meters.

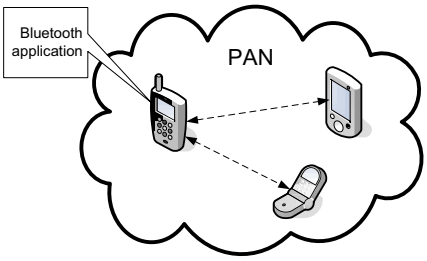


Figure 1. Bluetooth context distribution

Bluetooth service discovery can be done in two ways: by searching for a particular service or by browsing for available services using Bluetooth's Service Discovery Protocol (SDP) [2]. We utilize the latter (i.e., the browsing method), in order to learn about *all* available services offered by discovered devices [3].

File transfer is performed using Bluetooth's file transfer profile [4]. This file transfer profile depends on several underlying profiles and protocols. Two profiles handle discoverability and connection establishment. To transfer files the object exchange (OBEX) protocol is used. OBEX [4] allows a client to initiate a file transfer operation and push, pull, browse, or manipulate objects (files) on the server. The server needs to be available to other devices, accept incoming connections, and allow basic file transfer operations. The OBEX PUT operation is used to transfer objects to the server.

2.2 WLAN Context Distribution

In WLAN context distribution we assume that each device with a WLAN interface has already discovered some context and stored it in a local file. Distribution, as shown in Figure 2, works as follows: a device first enters a listening state, where it starts a timer with a random timeout value (initially selected between 6 to 9 seconds and increasing by 3 seconds each time the size of the merged file increases by an additional 150kB). Note that we have chosen these initial timeout values after experimenting with the protocol. This time is long enough to allow a server device to receive files from two other clients, but short enough to keep all devices synchronized for the entire measurement period.

When the timeout occurs, the device will check if it has received a *discoverPeers* message. If this message has been received, then the device acts as a client, sending a *peerReply* message followed by the current file containing the discovered context information. After sending the reply and file, the client will listen for a multicast of a merged file to arrive from the server. After receiving the merged file, the client returns to listening (in the *discoverPeers* state).

If the device determines after the timeout, that it has not received a *discoverPeers* message, then it will itself multicast this message, thus acting as a server. After multicasting this message, the server starts a timer and waits for *peerReply* messages and files from clients. Note that the client sends *peerReply* message

prior to the file transfer, thus there are two separate receive operations on the server side. When the timer expires, the server checks if *peerReply* messages and files have arrived and if so, it merges the received files into its existing file and multicasts the resulting merged file to all clients. Otherwise, it will multicast the existing file (generated in the previous round). After multicasting the file, the server returns to the *discoverPeers* state. For the next round, a new server will be randomly selected. In this way, context knowledge is shared among devices which are connected to the same wireless access point. However, this protocol could also be used on ad hoc WLAN networks.

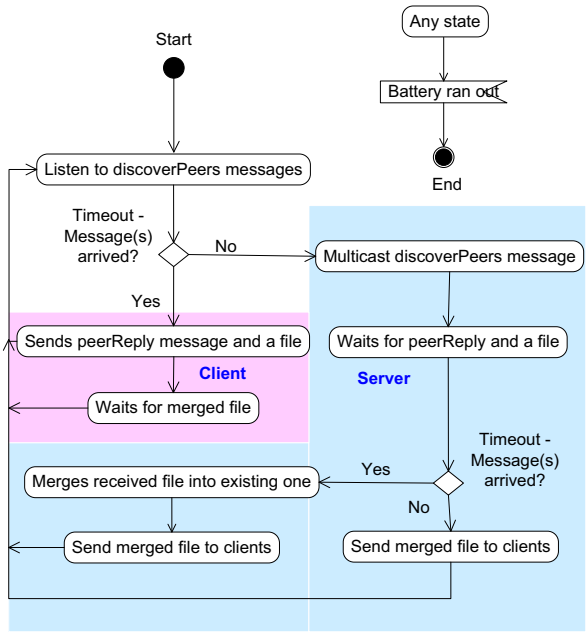


Figure 2. WLAN service discovery and file transfer protocol

3. MEASURING BATTERY POWER CONSUMPTION

3.1 Bluetooth and WLAN

Bluetooth in mobile devices generally uses one of two stacks: Widcomm/Broadcom stack [6] or Microsoft stack [7]. These APIs provide functions such as: device discovery, service discovery, and transfer of files between devices. We use High Point Software's BtAccess library [8], as the devices chosen both use the Widcomm Bluetooth stack. On top of this library we implemented our own context distribution application. To implement our WLAN context distribution we have used the System.Net.Sockets package (a part of the .NET Compact Framework).

3.2 Retrieval of battery power status

Most of today's mobile devices utilize a Smart Battery System [5]. Using Microsoft's coredll.dll library we developed for Microsoft Windows Mobile devices a P/Invoke function to retrieve the device's battery power status. The class SYSTEM_POWER_STATUS_EX2 is passed to the function as the first parameter.

To measure battery power consumption, we require the battery current and voltage values (as specified in this class). We also acquire the (remaining) battery life percentage values to provide information about the amount of battery life remaining as a percentage of the battery's initial full life time (capacity). We have developed a C# application that uses this library to log these values to a file.

3.3 Hardware

Measurements have been performed on two different types of devices (each of which has a separate internal backup battery to maintain data integrity during main battery replacement; additionally each has an integrated 802.11b WLAN interface). Table 1 shows characteristics of these devices. These devices were chosen because of their availability and differences in processing power, versions of the Bluetooth stack, and the battery capacity. Also both have the same type of Bluetooth stack and an integrated WLAN interface.

Table 1. Types of devices used in measurements

Device	Battery	CPU	Bluetooth stack
iPAQ 4150	Lithium ion, 1000mAh, 3.7V	400MHz Intel PXA255	Widcomm Bluetooth v1.4
iPAQ 6915	Lithium ion, 1200mAh, 3.7V	416MHz Intel PXA270	Widcomm Bluetooth v1.7

4. BLUETOOTH MEASUREMENTS

4.1 Measurements Description

During our measurements we have used 3 devices, each equipped with a Bluetooth interface. One device initiates device and service discovery, appends results to a file, and distributes this file to the other two discovered devices (each one acts as a remote device), responding to discovery inquiries, and retrieving the file. An application was developed and deployed that continuously performs and repeats Bluetooth device discovery, service discovery, and file transfer, as well as logging the battery current, voltage, and the remaining battery life (percentage) to a file for each of the activities (i.e., device discovery, service discovery, and file transmission). All phases were time stamped with both a start time and an end time.

In order to determine the battery power consumed for each operation, we have subtracted battery power values obtained in a measurement when the Bluetooth radio was turned off from the battery power values of each particular phase (see equation (1)). These measurements (with the Bluetooth interface turned off) were separately performed on each device. The values subtracted were chosen to match (in time) the battery power values in each phase of the series of Bluetooth operations. Note that device discovery is denoted as DD, service discovery as SD, and file transfer as FT.

$$P_X = \sum_{i=1}^{n} \left(U_{X_i} * I_{X_i} - U_{offX_i} * I_{offX_i} \right),\ X = \{DD, SD, FT\} \quad (1)$$

The overall measurement sequence is illustrated in Figure 3. The application was launched when the device was fully charged and continued until the battery level was too low to continue.

Figure 3. Measurement sequence

We append new discovery data and information about the battery power consumption to the file in each round. Therefore, the file size increases following each file transfer round. The information in the file is encoded in XML. In order to avoid reading and parsing the file when we need to append new information, the application simply seeks to the end of the file minus the length of the string of the last end tag, appends new data, and writes an end tag. This operation is constant in time and it takes less than 1ms to append new data to the file. As we could not measure this very brief operation using our application; we do not separate out the power required for this operation.

In our test environment we have three isolated devices: one master and two slaves. Their names were hard-coded in the application's discovery source code. They were initialized and configured to allow device discovery, service discovery, and file transfer operations initiated by the master device. The two different master devices used to perform measurements utilize two different versions of the Widcomm Bluetooth stack (as noted in Table 1).

4.2 Measurements Results

4.2.1 HP iPAQ 4150 Device

The measurements lasted for 8 hours, 6 minutes, and 14 seconds. While measurements performed on the same device in an idle state with the Bluetooth interface turned **off** lasted for 12 hours, 27 minutes, and 30 seconds, whereas when the Bluetooth was activated, but the device was idle, they took 11 hours, 26 minutes, and 32 seconds.

Figure 4 shows the battery consumption as a function of time calculated as P(t)=U(t)*I(t) from the values obtained from the measurements, where U(t) and I(t) represent the battery voltage and current values. The upper curve shows the battery consumption for all the activities performed during the measurements: discovery of two devices, discovery of services on a single device, and file transfer to a single device. It can be seen that battery power consumption (rate) is roughly constant. The lower curve shows the "corrected" battery power consumption, which is actually the battery power required during each phase of the measurement process reduced by the battery power values aligned in time when the Bluetooth interface was turned off and when the device was idle. This "corrected" battery power consumption for each Bluetooth phase is specified by equation (1). The average battery power consumed in the measurements was 335mW before and 109mW after subtracting the power when the Bluetooth interface was off.

Since the phases were not equally long (i.e., they took different amounts of time), one can not directly compare their battery power consumption; instead, we multiplied the average battery power (i.e., $\overline{P_x}$) consumed in each round with the average duration of each operation (i.e., $\overline{T_x}$) to obtain the average energy consumed from the battery (i.e., $\overline{E_x}$) due to each operation.

$$\overline{E_X} = \overline{P_X} * \overline{T_X} = \frac{\sum_{i=1}^{n}\left[\left(U_{X_i} * I_{X_i} - U_{offX_i} * I_{offX_i}\right) * T_{X_i}\right]}{n}, \quad (2)$$

$$X = \{DD, SD, FT\}$$

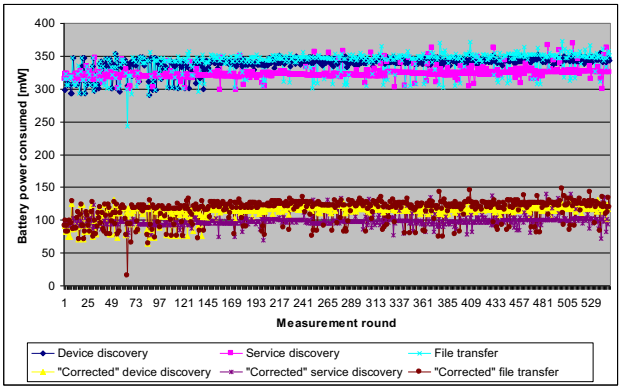


Figure 4. Comparison of battery power consumption for all three activities before (above) and after subtracting the power consumed when the BT interface was off (below).

Details about battery power consumption per activity are summarized in Table 2.

Table 2. Battery power consumed by all Bluetooth activities with a HP IPAQ 4150

Average	Device discovery	Service discovery	File transfer
power consumed: before correction	**339.7mW**	**324mW**	**340.3mW**
after correction	**114.6mW**	**98.6mW**	**114.9mW**
duration	**10.3 sec**	**1.6 sec**	**19.9 sec**
energy consumed	**1.18J**	**0.16J**	**2.35J**

Note that the size of the file that was transferred was 2.7kB at the beginning and 1.29MB at the end of the measurement period. The file size increases linearly in time, filesize(t)=2.34t +0.24, where the file size is expressed in kB and time in seconds.

Based upon the power consumed for the file transmission and the file transfer data rate we can estimate how many joules are consumed per transferred user data bit. The result is 3.9J/MB (i.e. 481.7nJ/bit) as obtained from the following equation:

$$Energy\_per\_transferred\_user\_data\_bit\left[J/bit\right] = \frac{P_{FT}[W]}{file\_transfer\_data\_rate\left[bit/s\right]} = \sum_{i=1}^{n}\left(U_{FT_i} * I_{FT_i} - U_{offFT_i} * I_{offFT_i}\right) \Big/ \sum_{i=1}^{n}\left(file\_size_i / T_{FT_i}\right) \quad (3)$$

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.

Figure 5 shows the file transfer rate vs. file size. A logarithmic increase of the file transfer rate with the file size can be observed. Thus the file transfer rate initially increases with increasing file size up to a certain point; after which the file size does not significantly influence the file transfer rate. The maximum OBEX packet length is 255 bytes; therefore the file transfer always requires multiple OBEX packets. This means that more than one PUT request needs to be sent to and acknowledged by the server. The last PUT request will have the final bits set, thus indicating to the server that client is finished sending packets. There are no timeouts between OBEX packets.

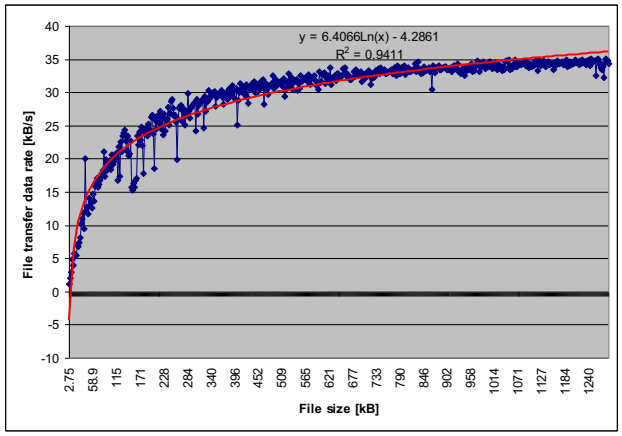


Figure 5. File transfer rate vs. file size

Using a maximum transmission unit of 255 bytes, the data rate at a file size of 520.3kB was 31.5kB/s and 33.9kB/s at a file size of 1MB. This MTU is too small to efficiently send large files, because sending more packets means waiting for more confirmation packets, which limits the transfer rate. We assume that the propagation time of responses is significantly less than the time required for processing packets of the file on the server side.

HP iPAQ 4150 uses a BRF6100 [9] Bluetooth single-chip which integrates Bluetooth baseband, RF, memory (ROM and RAM), and power management to enhance performance, reduce cost, and minimize board space. In its lowest power mode this Bluetooth transceiver requires 25mA in transmit mode and 37mA in receive mode at a supply voltage of 1.8V. Based upon this we can calculate the power, as 45mW in transmit mode and 66.6mW in receive mode. According to [10] the energy required to transmit a single burst of data from an initially powered-down transmitter can be expressed as follows:

$$E_{tx}(N, R_C, P_{amp}) = P_{start} * T_{start} + \frac{N}{R_C * R}(P_{txElec} + P_{amp}) \quad (4)$$

The two terms in the expression represent the energies for startup and transmission, respectively. Where P_{start} and T_{start} represent the power and latency of radio startup, P_{txElec} is the active transmission power, P_{amp} the dissipated amplifier power, N the number of bits before FEC, R the radio bit rate, and R_C the convolutional rate. Assuming that the energy consumed for the startup was significantly lower than the energy consumed for transmission (as the time needed for transmitter startup is in the order of hundreds of milliseconds versus seconds of transmission time and $P_{start} << P_{txElec}$), we assume values for the lowest power consumption where $P_{amp} \approx 0dBm$, $R_C=0.5$, and use a radio transmit data rate as R=721kbps, this results in an energy consumption per transmitted bit of:

$$\frac{E_{tx}}{N(1bit)} \approx \frac{P_{txElec}}{R_C * R} \approx 1.25 * 10^{-7} J/bit = 125nJ/bit \quad (5)$$

When comparing this result with the value calculated from our measurements using (3) (i.e., 481.7nJ), we can see that they are of the same magnitude; with this calculated value being smaller, since this computation assumed the *lowest* power consumption case. Because our measured value includes all the other operations required to actually get the bits to transfer in addition to effectively transferring the user data bits we expect it to be higher. It is important to note that we are computing in equation (5) the energy consumed to send a single bit from a transmitter, while the earlier calculations concerned the total energy to send a single user data bit, not including the coding of the user bits (or the header bits) nor the extra overhead bits which are set for the lower layer protocol (for example framing, addressing, synchronization, polling/response, link layer management, etc.), or the time (and energy) listening to be polled, waiting for an acknowledgement, and the protocol overhead at the higher layers. We note that the ratio between the energy consumed per transferred user data bit and the energy consumed to transfer a single bit from the transmitter is 3.9.

4.2.2 HP iPAQ 6915 Device

For this device the measurements lasted for 9 hours, 2 minutes, and 49 seconds until the remaining battery power was 12% of the full battery capacity[1]. If the measurement would have run until the battery was at zero capacity, the estimated duration of these measurements would be 10 hours, 23 minutes, and 22 seconds. For comparison, the measurements on the same device in an idle state with Bluetooth interface turned off lasted for 23 hours, 1 minute, and 14 seconds, whereas when the Bluetooth was activated, but the device was idle, they took 19 hours, 23 minutes, and 4 seconds.

Figure 6 shows the battery consumption for all activities performed during the measurements. The lower curve shows the battery power consumption after correction (in the same way as described in §4.2). The average battery power consumed during the measurement was 405.5mW before and 231.9mW after correction. The lower curve also shows several power values close to zero - these are errors due to the simple way in which the measurement values were corrected.

[1] The BT device in the iPAQ 6915 could no longer be used once the battery voltage dropped below 3.664V. At this time we are not certain why this is true, but suspect that the OS in purposely turning off the device to save some remaining battery power. Note that in our earlier measurements with HP iPAQ 5550 the operating system turned off the WLAN interface at some point to preserve some operating time without the WLAN.

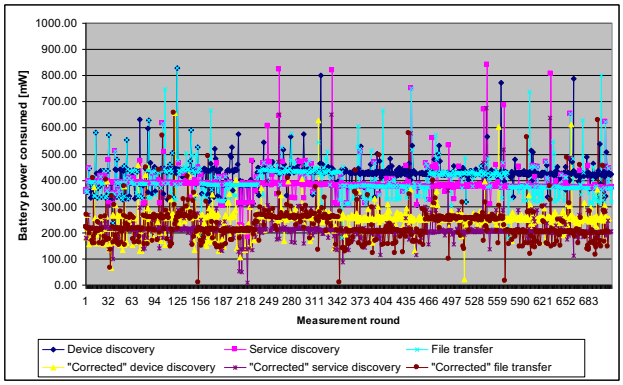


Figure 6. Comparison of battery power consumption for all three activities before and after correction

Details about battery power consumption per activity are summarized in Table 3.

Table 3. Battery power consumed by all Bluetooth activities for a HP iPAQ 6915

Average	Device discovery	Service discovery	File transfer
power consumed: before reduction	**425.7mW**	**388.4mW**	**402.5mW**
after reduction	**251mW**	**215.3mW**	**229.3mW**
duration	**11.7 sec**	**2.1 sec**	**10.9 sec**
energy consumed	**2.93J**	**0.43J**	**2.49J**

Applying the same equations used earlier for the HP iPAQ 4150, we calculate that the energy consumed (until the point when the remaining battery power was 12%) for each Bluetooth phase. As shown in Table 4, these results are 2-3 times higher for the device and service discovery, and almost the same for file transfer. Table 4 summarizes the energy consumption results for both devices: average energy consumed for device (DD) and service discovery (SD), energy consumed per transferred user data bit (FT), as well as the total energy consumed for all rounds.

Table 4. Comparison of energy consumptions

HP iPAQ	DD	SD	FT (per user data bit)	Total energy consumed
4150	1.18J	0.16J	481.7nJ/bit	3383.4J
6915	2.93J	0.43J	401.5nJ/bit	6203.5J (88%) 7334.7J (estimated for 100%)

HP iPAQ 6915 uses a BRF6150 [11] Bluetooth single-chip solution. In its lowest power mode, its Bluetooth transceiver utilizes 25mA in transmit mode and 37mA in receive mode at a supply voltage ranging from 1.8V to 3.6V. Calculating the power, we obtain 67.5mW (using the mean voltage value) in transmit mode and 99.9mW in receive mode. When calculating the energy consumed per transmitted bit, we get:

$$\frac{E_{tx}}{N(1bit)} \approx 1.872 * 10^{-7} J/bit = 187.2nJ/bit \qquad (6)$$

Figure 7 shows the file transfer rate vs. file size, which follows the same trend as in Figure 5, but with significantly higher data rates. It can be seen that the data rate of this device is 2.4 times faster than the rate of the other model. The size of the file that was transferred was 2.7kB at the beginning and 1.48MB at the end of the measurement period. The data rate at a file size 519kB was 75.95kB/s and 82kB/s at a file size of 1MB.

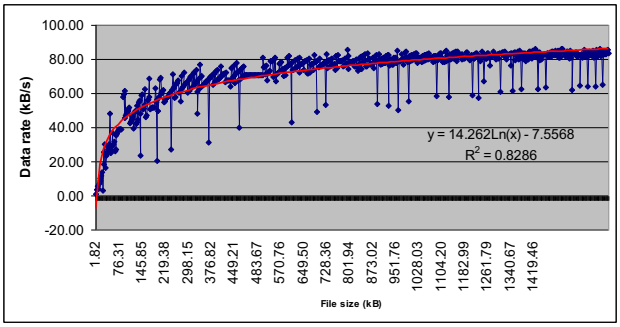


Figure 7. File transfer rate vs. file size

Moreover, the total energy consumed by the iPAQ 6915 for all rounds was 6203.5J. The estimated total energy consumed (given the ability to fully utilize the battery capacity) is estimated to be 7334.7J, while the total energy consumed by 4150 device was 3383J. Thus we observe that iPAQ 6915 consumed twice as much energy from the battery as the iPAQ 4150. Possible reasons for this are that iPAQ 6915 has a faster processor than iPAQ 4150 (Intel PXA270 at 416 Mhz vs. Intel PXA255 at 400Mhz), a newer Bluetooth stack (version 1.7 vs. 1.4), thus **waiting** for Bluetooth input consumes significantly more battery power because the processor is not being put into a low power mode, despite the fact that it is to perform the same operations as the iPAQ 4150. Another reason is that the set of measurements ran (11%) longer on the iPAQ 6915 than on the iPAQ 4150, thus we would already suspect that unless the iPAQ 6915 consumed less energy on average per operation than the iPAQ 4150 – that its total energy consumption would be greater – however it is more than proportionally greater.

Note that in order to estimate the total energy consumed for the full battery capacity, we first calculated the sum of the estimated durations of all phases for the rest of the (estimated) time (that the measurements would run if we were able to continue to operate until there was no battery power left), multiplied by the average battery power of each phase, and add them to the already calculated energies.

5. WLAN MEASUREMENTS

5.1 Measurements Description

In the WLAN measurement we have also used 3 devices, each equipped with a WLAN 802.11b interface, in conjunction with a D-Link DI-524 high speed IEEE 802.11g wireless router, which is 802.11b compatible. As explained in §2.1, after listening for a random period, a device which times out attempts to assume the

role of being a server, while the other (two) devices act as clients.

A randomly selected measurement sequence is shown in Figure 8. The client's and server's activities are also illustrated as a function of time. First we measure the duration of each activity which a device performs, and determine the corresponding battery power consumed for this operation, these results can be correlated and compared with the results from the earlier Bluetooth measurements.

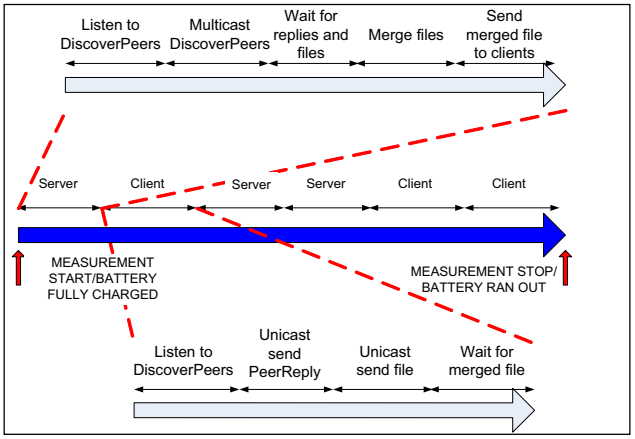


Figure 8. WLAN measurement sequence

In order to determine the battery power consumed for each of the client and server operations, we have subtracted the battery power values obtained when the WLAN interface was turned off. These measurements (with the WLAN interface turned off) were performed separately on the same device and the values were used to correct the power consumed in each of the phases of the WLAN operations.

Bluetooth generated files that contain context information on each client were sent to a server, which merges them into a single file and sends this file back to other clients. A merged file is composed at the beginning of the measurement period and the battery information for each activity a device performs is appended in each round to this file. The reason for doing so is to simulate the increasing file size in order to be comparable to the earlier Bluetooth measurement, while avoiding the exponential increase in file size caused by *appending* a newly generated merged file to the existing one.

5.2 Measurements Results

5.2.1 HP iPAQ 4150

The measurements lasted until the device could no longer operate the WLAN interface – which took 2 hours, 14 minutes, and 42 seconds, i.e. four times shorter than the measurement period of the Bluetooth measurements. After analyzing this data, we eliminate the power consumed in following phases (because they were too short to be considered in the energy consumption calculation): the server's multicast of the *DiscoverPeers* message, the client's sending of the *PeerReply* message, and the client's sending of the Bluetooth generated file.

Figure 9 shows the battery power consumption for activities that a device performs in the server role: listening to *DiscoverPeers* message, waiting to receive peer replies & files from clients, and sending a merged file to discovered clients. Note that the first operation relates to the blocking receiving function on a UDP multicast socket, while the other two activities are receiving and sending a file over a TCP socket, respectively.

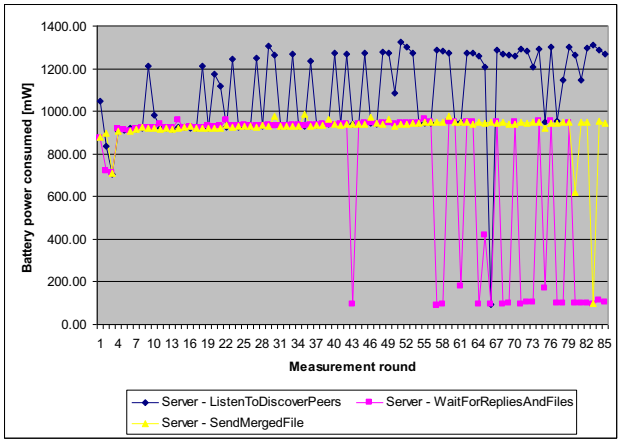


Figure 9. Comparison of battery power consumption for all server activities after subtracting the power consumed when the WLAN interface was off

It can be seen that listening to *DiscoverPeers* is the most power consuming operation, which varies between 912mW and 1325mW. Note that a device in this phase does not receive any *DiscoverPeers* message after the timeout expires, because it acts as a server and multicasts this message. Note also that the device had the automatic power save mode[2] turned on during the measurement period, meaning that the WLAN card enters a *Sleep* state [12] (where a majority of the circuitry is switched off, except for some critical parts) after a certain elapsed period of inactivity. It wakes up after a preset interval to check for the traffic queued for it at the access point. The battery power consumption of a WLAN device in the idle (after it was reduced by the values when WLAN was off) is shown in Figure 10.

Since listening for *DiscoverPeers* activity is realized by the blocking socket receiving function, we assume that the majority of the time the device is in the *Listen* state [12] and that it does not go to the *Sleep* state. In the *Listen* state a device listens for the (multicast) traffic, but does not pass any data to the host. We also assume that instantaneous power consumption, illustrated by periodic peaks, corresponds to short periods when the device was receiving an announcement frame from the access point. The announcement period (DTIM (Delivery Traffic Indication Message) interval) was set to 300 ms.

The waiting for *PeerReplies* & files and sending a merged file have very similar power consumptions in the first half of the measurement period. The reason for this similarity is that they both receive and send data via a TCP socket. The difference is that the waiting for *PeerReplies* and files operation is terminated after the preset timeout value, which increases with the merged file size increase (as explained in §2.2), while the sending of a

[2] The automatic power save mode is by default set by the device manufacturer because it achieves the maximum power saving without degradation of performance.

merged file operation finishes immediately after the data transmission completes. Note that the receive operation is non–blocking, meaning that if there is no data to receive, the device will be idle, and will go to the *Sleep* state. This could explain the occurrence of the instantaneous drops of the battery power to the same low values as when the WLAN was idle (see Figure 10). A drop in battery power consumption also happened during the send merged file operation just before the end of the measurement period. This can be seen in Figure 11, and a possible explanation is that the device briefly disconnected from the access point.

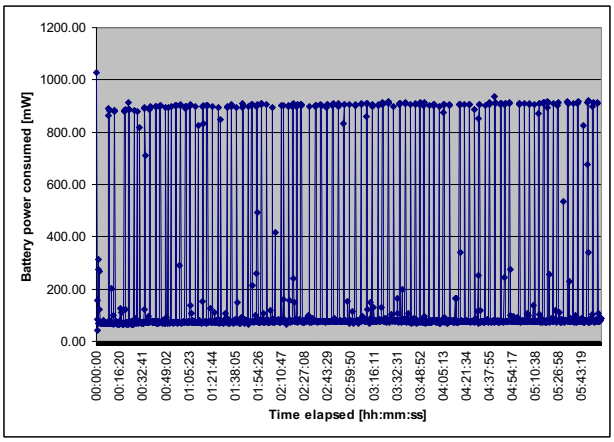


Figure 10. Battery power consumption plot for WLAN idle measurement after correction

Figure 11 shows the battery power consumption for activities that a device performs as a client: listening for the *DiscoverPeers* message and receiving a merged file from the server. In the listening for *DiscoverPeers*, the client actually gets the message, but this retrieval is too short to be captured in the log files because the whole message fits into a single packet of 1024 bytes. Thus, its battery power consumption is the same as in the server role.

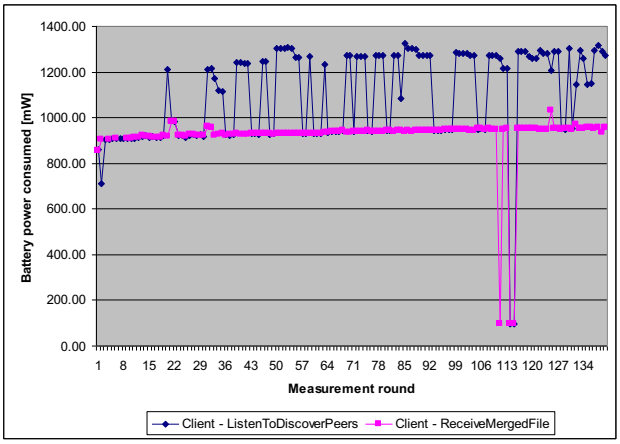


Figure 11. Comparison of battery power consumption for all client activities after correction

It can be noticed that the power consumption of receiving the merged file is the same as the sending of the merged file. Three decreases in battery power consumption can be observed in Fig.11, which can be explained as the client device not receiving the file because of a temporary disconnection from the access point.

Tables 5 and 6 summarize the details of the average power and energy consumption per activity in the server and client role. We can observe that the most energy was consumed during blocking receive operations when listening for a *DiscoverPeers* message, while the least energy was spent to actually send and receive data. In waiting for *PeerReplies* and files operation the device consumed less energy than in the listening activity, due to the non-blocking receiving operation.

Table 5. Battery power consumed by WLAN server activities for a HP iPAQ 4150

Average	Listen to *DiscoverPeers*	Wait for *PeerReplies* and files	Send merged file
power consumed: before reduction	**1312.7mW**	**959.3mW**	**1151.8mW**
after reduction	**1083.4mW**	**728.4mW**	**920.7mW**
duration	**15sec**	**15sec**	**13sec**
energy consumed	**16.67J**	**9.96J**	**11.5J**

Table 6 Battery power consumed by WLAN client activities for a HP iPAQ 4150

Average	Listen to *DiscoverPeers*	Receive a merged file from a server
power consumed: before reduction	**1316.5mW**	**1144.5mW**
after reduction	**1089.8mW**	**919.2mW**
Duration	**14.9 sec**	**4.5 sec**
energy consumed	**16.6J**	**4.1J**

To compare the cost of WLAN to discover two devices with the Bluetooth device discovery, we needed to compare the energy consumptions of the corresponding Bluetooth and WLAN activities. Bluetooth device discovery corresponds to the WLAN's listen for *DiscoverPeers* and receiving *PeerReplies*. However, sending and receiving of a *PeerReply* message takes a very short time and can be ignored in the energy consumption calculation. Thus, we can conclude that Bluetooth device discovery consumed significantly less energy than its WLAN counterpart (1.18J vs. 16.6J). Looking at their average durations, it also took less time to discover two devices via Bluetooth (10.3s) than in WLAN (15s). However, one should note that the duration of the listen for *DiscoverPeers* phase was set programmatically by the random timeout value that increases with the merged file size (see §2.2).

Figure 12 shows the transfer rate vs. file size of multicasting this file to two devices and the results of fitting this to a four degree polynomial function. The size of the merged file was 16.8kB at the beginning and 751.7kB at the end of the measurement period. The data rate at a file size 520kB (per device) was 57.8kB/s and 57.2kB/s at a file size of 743.6kB. These values are lower than expected, since sending of the merged file was performed by writing data to the socket while reading from the file in parallel.

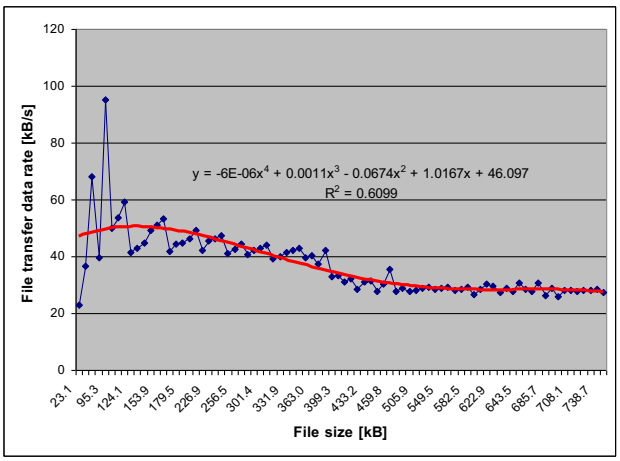


Figure 12. File transfer rate vs. file size

After calculating the energy consumed to transfer a single user data bit using our formula, we got 1.56µJ/bit, meaning that 3.2 times more energy was consumed per bit by sending data over WLAN than over Bluetooth. Additionally, WLAN is 1.8 times faster than the Bluetooth (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 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 will compute how many joules are consumed by a client to receive a single user bit over the WLAN multicast:

$$Energy\_per\_received\_user\_data\_bit\left[J/bit\right] = \frac{E_{RMF}\left[J\right]}{received\_file\_size\left[bit\right]} \quad (7)$$

$$= \sum_{i=1}^{n}\left(U_{RMF_i} * I_{RMF_i} - U_{offRMF_i} * I_{offRMF_i}\right) * T_{RMF_i} \Big/ \sum_{i=1}^{n} received\_file\_size_i = 1.33\mu J/bit$$

Note that E_{RMF} in the equation (7) 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., 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 information** to other devices in advance (using WLAN multicast), rather than having all devices learn this information themselves.

6. RELATED WORK

In the literature there are many context-aware frameworks for enabling mobile devices to adapt their configuration to environmental conditions. All of these frameworks need to employ some context discovery and distribution mechanism in order to provide the right context anywhere, anytime. Some of the most popular frameworks are the Context Toolkit [16], the Service-Oriented Context-Aware Middleware (SOCAM) [17], and the Context Broker Architecture (CoBrA) [18]. However, all of them are based on a centralized discovery mechanism where distributed entities that provide context information (i.e. sensors, context providers) have to register in order to be discovered. A pure peer-to-peer context-aware system such as Hydrogen [19] uses the device's local context, i.e. context acquired by local built-in sensors. Due to its limited capabilities a device cannot sense all the context information itself, Hydrogen provides a mechanism to share sensed context with other nearby devices. Context sharing is based on a peer-to-peer connection over LAN, WLAN, or Bluetooth. However, authors do not mention distributing the "aggregated context", i.e., context originating from two or more devices, which can be exchanged with a newly encountered device in order to learn about context beyond a single hop. In [20], authors designed a ubiquitous-oriented peer-to-peer context sharing model (PCSM) that constructs channels for remote registration of Context Database Agents through a Registration Query. Although this model is well designed for disconnected operations by using lightweight messages, the authors did not investigate the communication and battery power costs of exchanging small vs. big chunks of context data.

A lot of research has studied battery power measurements for mobile devices, in particular for optimization. However, only a small amount of work targeted context. In [14], the authors propose a system for enabling applications running on mobile devices to adapt their behavior in order to reduce their energy consumption, by optimizing the collaboration between applications and the underlying operating system. A similar goal drives the research illustrated in [13], where the authors propose an energy-aware QoS model (e-QoS) providing QoS guarantee in terms of energy consumption of network-centric applications running on mobile devices. This is accomplished by dynamically selecting and adapting application protocols. Finally, the work described in [15] introduces a system for context aware battery management, based on prediction algorithms, which helps a mobile device user to prevent a complete battery discharge.

7. CONCLUSION AND FUTURE WORK

In this paper, we introduced and evaluated context distribution methods in mobile systems environments using Bluetooth and WLAN technologies. The evaluation of the proposed methods was performed by collecting and comparing battery power consumption measurements. Such measurements were performed separately on two different models of Bluetooth and WLAN enabled mobile devices. We have seen that the HP iPAQ 6915 device consumed twice the energy of the HP iPAQ 4150 to perform the same Bluetooth operations (i.e., device discovery, service discovery, and file transfer). The possible reasons for this

are that iPAQ 6915 has a faster processor than iPAQ 4150, and a newer Bluetooth stack, all of which lead to more battery power being consumed to perform the same operations. We found out that Bluetooth consumes 2-6 times more energy to send a file of 1MB size to two devices than to discover them – hence **distributing** this information **via Bluetooth** is more expensive than directly **learning** it! Additionally, the file transfer by the model 6915 was 2.4 times faster, however we do not know whether this is due to the newer BT stack or the faster CPU.

The WLAN measurements were performed (at the time of writing) only on iPAQ 4150. The results showed that the energy consumed per transferred user data bit was 1.56μJ/bit for WLAN vs. 481.7nJ/bit for the Bluetooth file transfer (a ratio of 3.2). Additionally, the WLAN transfer is faster than the Bluetooth, taking a half of the time to transfer the same amount of data between two devices. Therefore, if data is sent to more than three devices at once via **WLAN multicast** this **is more energy efficient than** using **Bluetooth**.

By comparing the energy used to discover two devices and their services (i.e., 1.5J) with the energy that would be consumed to receive the file of the same size (i.e., 28.7mJ), we concluded that it is **more energy efficient to distribute** (once discovered) **context information** to other devices in advance, rather than having all devices need to learn this information themselves.

We also found out that the main reason that WLAN consumed more energy than the Bluetooth was a **long timer value** set programmatically on the WLAN **to discover devices**. The blocking receive operation in the WLAN discovery phase did not let the processor go into its low power mode. Therefore, we plan to shorten this timeout value in the WLAN protocol implementation and modify the measurement application to include the periods of a processor's activity and inactivity, in order to be able to estimate the difference between the energy consumption of a device performing vs. waiting for an operation.

Note that this paper did not explicitly address the issue of the time waiting for link layer acknowledgements. However, measuring the details of the effects of waiting would require additional experiments and might be subject for a future paper.

8. ACKNOWLEDGMENTS

The authors of this paper would like to thank their partners in the MUSIC-IST project and acknowledge the partial financial support given to this research by the European Union (6th Framework Programme, contract number 35166). We would also like to thank Prof. Gerald Q. Maguire Jr., for his fruitful discussions and valuable comments during this research work.

9. ADDITIONAL AUTHORS

Additional authors: Alessandro Mamelli (HP Innovation Center, Milan, Italy, e-mail: alessandro.mamelli@hp.com) and Athanasios Karapantelakis (Ericsson AB and Royal Institute of Technology (KTH), Stockholm, Sweden, e-mail: athkar@kth.se).

10. REFERENCES

[1] Dey, A., 2000., Providing Architectural Support for Building Context-Aware Applications, PhD thesis, College of Computing, Georgia Institute of Technology, AT, USA

[2] Specification of the Bluetooth system, Core Specifications, version 2.1 + EDR, v2.0 + EDR, v1.2, and v1.1

[3] Institute of Electrical and Electronics Engineers, 1999. Short Description of the Standard, IEEE P802.11 working group

[4] Bluetooth Special Interest Group, 2001. Specification of the Bluetooth System: Profiles, Volume 2, Version 1.1

[5] Benchmarq Microelectronics Inc. et al, Smart Battery Data Specification, Smart Battery System Specification report, rev. 1.1, 1998.

[6] Broadcom Inc., Widcomm Bluetooth software, http://www.broadcom.com/, last visited on 16-March-2008.

[7] Microsoft Inc., Bluetooth stack architecture, http://msdn2.microsoft.com/de-de/library/ms890956.aspx, last visited on 16-March-2008.

[8] High Point Software Inc., BtAccess home page, http://www.high-point.com/, last visited on 16-March-2008.

[9] Texas Instruments, BRF6100 Fully Integrated Bluetooth Transceiver, Product Brief, 2002.

[10] Rex, M. and Chandrakasan, A., A Framework for Energy-Scalable Communication in High-Density Wireless Networks, ACM ISLPED'02, Monterey, CA, USA, August 2002.

[11] Texas Instruments, BRF6150 Bluetooth Specification v1.2 single-chip solution, Product Brief, 2004.

[12] Atheros Comm., Power Consumption and Energy Efficiency Comparisons of WLAN products, White paper, 2003.

[13] Lufei, H. and Shi, W., Energy-Aware QoS for Application Sessions across Multiple Protocol Domains in Mobile Computing, Computer Networks: International Journal of Computer and Telecommunications Networking, Vol. 51, No. 11, 3125-3141, August 2007

[14] Flinn, J. and Satyanarayanan, M., Energy-aware adaptation for mobile applications. In Symposium on Operating Systems Principles, pp. 48-63, December 1999.

[15] Ravi, N., Scott, J., Han, L., and Iftode, L., Context-aware Battery Management for Mobile Phones. IEEE International Conference on Pervasive Computing and Communications (PerCom 2008), Hong Kong, China, March 2008.

[16] Dey, A.K. and Abowd, G.D., 2000. The Context Toolkit: Aiding the Development of Context-Aware Applications, In the Workshop on Software Engineering for Wearable and Pervasive Computing, Limerick, Ireland, June 6, 2000.

[17] Gu, T., Pung, H.K. and Zhang, D.Q. A service-oriented middleware for building context-aware mobile services, In Proc. of IEEE Vehicular Technology Conference (VTC), Milan, Italy, August 2004.

[18] Chen, H., An Intelligent Broker Architecture for Pervasive Context-Aware Systems, PhD Thesis, University of Maryland, Baltimore Count, 2004.

[19] Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., and Altmann, J., 2003. Context-Awareness on Mobile Devices – the Hydrogen Approach, In Proc. 36th Annual Hawaii International Conf. on System Sciences, pp.292-302, January 2003.

[20] Ye, J. Li, J., Zhu, Z., Gu, X., and Shi, H., PCSM: A Context Sharing Model in Peer-to-Peer Ubiquitous Computing Environment, In International Conf. on Convergence Information Technology, Gyeongbuk, Korea, IEEE Computer Society, pp. 1868-1873, November 2007.

PAPER 4

Synthesizing context for a sports domain on a mobile device

Alisa Devlic[1,2], Michal Koziuk[3], and Wybe Horsman[4]
[1] Appear Networks, Kista Science Tower,
16451 Kista, Sweden
alisa.devlic@appearnetworks.com
[2] Royal Institute of Technology (KTH), Department of Communication Systems,
Electrum 418, SE-164 40 Kista, Sweden
devlic@kth.se

[3] Institute of Telecommunications, Warsaw University of Technology,
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
mkoziuk@tele.pw.edu.pl

[4] Capgemini NL bv, Papendorpseweg 100
3528BJ Utrecht, The Netherlands
wybe.horsman@capgemini.com

In Proceedings of the 3rd IEEE European Conference on Smart Sensing and Context (EuroSSC 2008), Zurich, Switzerland, Springer-Verlag, LNCS 5279, October 2008, pp. 206-219.

Synthesizing Context for a Sports Domain on a Mobile Device

Alisa Devlic[1,2], Michal Koziuk[3], and Wybe Horsman[4]

[1] Appear Networks, Kista Science Tower,
16451 Kista, Sweden
alisa.devlic@appearnetworks.com
[2] Royal Institute of Technology (KTH), Department of Communication Systems,
Electrum 418, SE-164 40 Kista, Sweden
devlic@kth.se
[3] Institute of Telecommunications, Warsaw University of Technology,
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
mkoziuk@tele.pw.edu.pl
[4] Capgemini NL bv, Papendorpseweg 100
3528BJ Utrecht, The Netherlands
wybe.horsman@capgemini.com

Abstract. In ubiquitous computing environments there are an increasing number and variety of devices that can generate context data. The challenge is to timely acquire, process, and deliver these data to context-aware applications. The role of context synthesis is to generate new knowledge, as a result of a reasoning process applied to context information that is already present in the system. The success of this mechanism mainly depends on the response time that the end-user or an application must wait for the response to a context query. This paper describes and evaluates an approach to context synthesis on a mobile device to be used by a set of applications in a sports domain. A scenario based on a live race at the Super Prestige Cyclocross in Gieten, Netherlands demonstrates the use of context synthesis to dynamically compose gaps and groups of cyclists in order to provide a nearly real-time virtual ranking service.

Keywords: context synthesis, context operators, context modeling, sport scenario.

1 Introduction

Imagine experiencing a sport event from your phone, where you are your own director deciding upon your own point of view by actually moving about the event locale. Rather than simply selecting one of many video streams on your screen, instead you utilize the abstract view of the event (as viewed on your smartphone) to select your own personal viewpoint of the event. Therefore, you want answers to questions, such as: what are the positions of the Rabobank riders, what gaps and groups of riders are there on the track today, who is the virtual leader of the race at this moment, at what time can I expect the leader to pass my current position on the track, etc. Based upon

D. Roggen et al. (Eds.): EuroSSC 2008, LNCS 5279, pp. 206–219, 2008.

the answers to these questions you will move to the position which you decide will give you the best vantage point.

It is November 2007 in Gieten, Netherlands. It is cold and rainy, the perfect conditions for an international cyclo-cross race. The track lies partly in the woods and partly around a pond with steep banks. The track is about 3 km in length and each rider must complete 7 laps. The difference between the top riders and the ones that have a bad day is very big; after 2 laps the slowest rider is lapped by the fastest, but by then most distinguishing features of the riders are covered with mud – thus after 3 or 4 laps it is hard to see who is who. Using devices with the MIDAS middleware and race application preinstalled, attendees are able to see their own location and the location of the individual riders on a map of the track, the leading cyclist in the race, the total and remaining distance in meters, the gap between riders, as well as which riders are riding in the same group (so called gaps and groups analysis). All of this is shown live on your mobile device.

MIDAS (*Middleware Platform for Developing and Deploying Advanced Mobile Services*) [1] is a European research project concerning 3G and beyond, which aims to define and implement a platform to simplify and speed up the task of developing and deploying mobile applications and services. It is specifically designed to be used in MANETs. MIDAS enables applications running on different nodes to share information by inserting data in and retrieving data from a 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. Therefore, for the purpose of this paper, we assume that all context information is available locally on a mobile device.

An application using this middleware calculates and displays gaps and groups of cyclists in near real time. This calculation needs to be performed as the cyclists' relative positions change, resulting in the synthesis of gaps and groups context. Moreover, the presentation on the display needs to be updated to reflect the current composition of groups. Thus, the middleware periodically obtains cyclists' geographic locations and utilizes information about the waypoints in the race. This data can be combined with the cyclists' data (such as name, team name, identification number, etc.), in order to perform context synthesis.

This newly generated context information can be in turn used by multiple applications. Hence, applications requesting customized context may share the cost of producing this synthesized context information. Additionally, each application need not be concerned with how this synthesis is implemented. However, some applications may want to implement their own synthesis functions. We refer to these functions as *context operators*. Context operators enable different applications and even different context-aware systems in the same domain to query each other about the context information which could be synthesized using the functions they implement. For example: a racing application and media application deployed on different devices should be able to remotely query each other (using the same middleware API and context operators) for results of the race and rankings of all athletes in the competition. The output of the operator is sent as a result of a context query. This result is called a *synthesized context*, since it was generated by context synthesis.

Our motivation and the idea for context synthesis using operators was previously presented in [2]. The main advantages of our approach are increased reusability, extensibility, and interoperability, facilitated by context operators and exploiting ontology

based context modeling. This paper describes the realization of this approach via a race application developed on this middleware, and evaluates the context synthesis 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, the time needed to obtain context information (formatted as ontology data) from its repository, and the time needed by this operator to perform the actual context synthesis.

The rest of the paper is organized in seven sections. Section 2 elaborates the MIDAS context modeling approach using ontologies for mobile devices. Section 3 presents our approach for context synthesis using context operators, while Section 4 describes the set of applications developed for sports scenario that illustrate the use of context operators for context synthesis. In Section 5 we give a performance evaluation. Section 6 provides a brief overview of related work. We conclude in Section 7 with a summary of the results and plans for future work.

2 Context Modeling Using Ontologies

In order for MIDAS to be a context aware framework it needs a mechanism for modeling and representing context. This context model must contain information specific to a specific domain of deployment of a MIDAS based service. A context model of a domain describes the people, objects, and relations between them which are typically encountered in a specific situation (or types of situations). Focusing only on a single domain makes it possible to create a very specific model, capable of representing a very fine level of detail, which otherwise could not be captured due to growing complexity of a more general model.

The context model used is an ontology, which is provided to the system in the form of a file. Thus the same middleware can be used in various domains given a new ontology file. We envision that an organizer of an event creates an ontology which represents the domain of this event. This ontology is provided to application developers (who can create applications for this particular domain). Once this ontology is created, other similar events can re-use the ontology, modifying it as required.

The ontology language chosen for the domain model is DL-Lite [3], 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 it handles statements about subsumption, disjointness, role-typing, participation constraints, non-participation constraints, and functionality restrictions. MIDAS implements an architecture for handling DL-Lite ontologies on a Java enabled mobile device [4].

The decision to use DL-Lite as a language for MIDAS ontologies was motivated by the results obtained during an initial experiment [5]. This experiment showed that using OWL-DL [6] with existing off the shelf solutions such as Jena [7] and Pellet [8] could not be applied on mobile devices, given their poor (slow) performance even on desktop machines and very high memory requirements. The use of existing ontology query languages, such as RQL [9] or SPARQL [10] was not analyzed. However, these solutions are usually not designed for mobile devices as their main focus is high expressivity. Thus, their practical usability in a mobile device setting is unlikely. We

chose DL-Lite because of its relative simplicity and optimization for fast performance. The limitations in the description logic that made these improvements possible turn out not to be limiting when modeling a domain.

The syntax chosen for context model ontologies is the Manchester OWL Syntax [11] due to features which make it more suitable for applications on mobile devices than the usual OWL Syntax [12]. The main feature is that it is much easier to parse, as it requires only two linear scans of the ontology file, and does not require construction of a tree structure during parsing. Another feature is that an ontology is approximately half the size of the equivalent OWL representation, and because it is human readable it is possible to edit it by hand if necessary.

For representing the ontology on a MIDAS enabled mobile device we created a dedicated Lightweight Ontology library [13], which implements the Jena [7] 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 HashTables. Its simplicity suits resource constrained devices (such as J2ME mobile phones).

The scenario examined in this paper is a cycling race. Part of the context model ontology is shown in Fig. 1. This example shows only the part of the class hierarchy from the domain model, which contains classes corresponding to roles of users. Other classes (not shown) in the domain model are used to represent places encountered during the event, abstract entities such as a group of cyclists, a gap between two groups, etc.

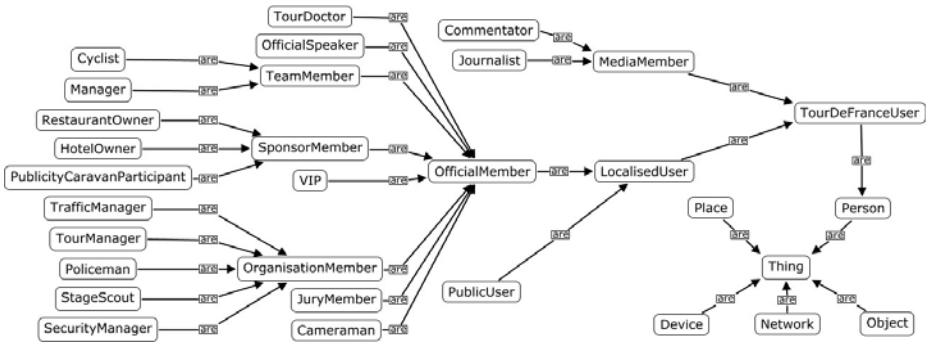


Fig. 1. Classes that describe roles of users involved in the cycling event

We consider five types of entities, which can be characterized as owners of context information: a person, a device, a network, a place, and an object. However, these entities are not independent, but have the following relations (see Fig. 2): a device is connected to a network; a person uses certain device(s); a person, device, and an object may be located at a place; a person and a device are somehow related to some other object(s). All entities are subclasses of the root class "Thing" in the ontology, from which all other terms are derived. Thus, we assign all context information to a certain entity and we can query information about an entity—i.e., user context, device context, network context, place context, and object context.

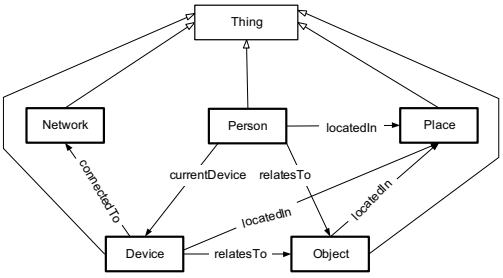


Fig. 2. Context entities and their relations in the context model

The context modeling architecture is implemented by a Context Knowledge Base component in the middleware. The API offered by this component makes it possible to model context by means of objects of the type *DomainInstance*, each of which represent physical entities. A *DomainInstance* can be added or removed as needed. Each *DomainInstance* object 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 *DomainClass* and *DomainProperty* (respectively) which correspond to those present in the domain model ontology.

Context information needs to be stored by the middleware before it can be queried or synthesized. In order to store, retrieve, and manipulate the formatted (higher-level) context information, we 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.

3 Context Synthesis Using Context Operators

Operators for context synthesis are domain-specific functions over the context data. The benefits of these operators are that by performing operations over the existing context information, new context information that previously did not exist in the system can be produced. The output of the operation performed by an operator, a *synthesized context*, is sent to the application as a result of a context query. Operators could be used on a higher level to synthesize information of a certain user, device, network, place, or other object, as illustrated earlier in Fig. 2.

Operators are bundles of both a description and implementation; and described by an ontology, similar to the representation of context. They are implemented as java scripts that perform an action specified in the operator's ontology. 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. 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.

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 developers; which operator is invoked will be determined by the middleware at runtime.

Specialized operators are implemented as scripts using Beanshell [23], an open source java script engine. In our implementation the operator scripts are part of the context service process and they can be programmatically added and removed by the middleware.

Fig. 5 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 java script written in Beanshell, i.e., a .bsh file).

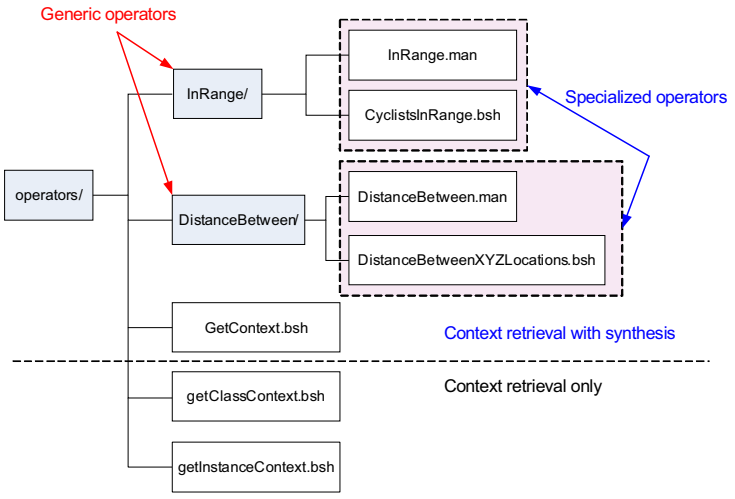


Fig. 3. Operator space file structure

The root folder of the *Operator space* shown in Fig. 3 also contains three specific operator scripts which are responsible for retrieving context data of the specified context owner, from the Context knowledge base: *GetContext.bsh*, *GetClassContext.bsh*, and *GetInstanceContext.bsh*. Note that they do not have a generic operator representing them, and they are used for distinct purposes. As previously noted, when specific context operators need to retrieve context, they will provide *DomainInstance* objects 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 *GetClassContext* and *GetInstanceContext* scripts. *GetInstanceContext* is used to obtain the domain instance with the supplied datatype properties from the context query.

We can also query the Context knowledge base 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.

An example of an operator description file, *InRange.man* is presented in Fig. 4. This file contains all the specialized operator descriptions. Fig. 3 shows only *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 "*CylistsInRange*" 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.

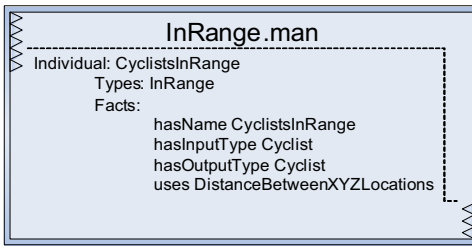


Fig. 4. CyclistsInRange description

In order to generate this description file, the developer needs to programmatically set the type of this specialized operator, the list of input types, the output type, as well as operator dependencies. The middleware will automatically append this operator description to the correct ontology file (if this file does not exist, it will be created in the correct location).

Note that all specialized operator scripts take as inputs *DomainInstance* objects, which are instances of classes specified as input types in their operator description file. Thus these domain instances pass the input arguments from the context query to the operator's method, and can be used to retrieve the missing information from the Context knowledge base (if needed).

3.1 Operator Matching

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 Fig. 5, 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. The operator matching algorithm returns the specialized operator with either exactly the same description as specified by the query or a more generic one.

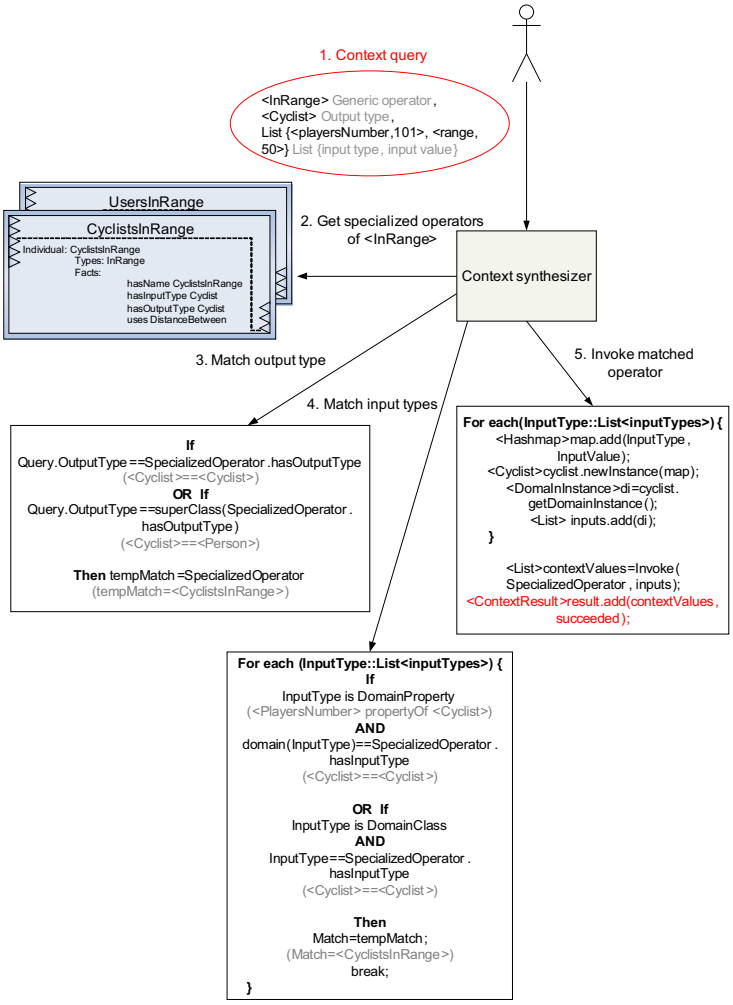


Fig. 5. This figure shows the algorithm itself, initiated by the user's context query, along with the invocation of the matched (specialized) operator

An example of a context query is: *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, 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. 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. *UsersInRange*, which would return *Users* instead of *Cyclists* as result. Finally, it invokes the matching operator.

4 Cyclist Race Application

The cyclist race application set consists of a number of applications responsible for: 1) entering static cyclist data and managing track waypoints, 2) processing and showing a list of the latest rider location data and 3) showing the actual gaps and groups of cyclists to the end user during the race. The last (end-user) application is available with a user interface in three different form factors for display on a small device (Nokia N800), a laptop (HP tablet), and as a side bar next to a web page shown on a laptop. The processing application was at the time of the race not actually deployed on a mobile device; however, in Section 5 we give an evaluation of context synthesis on a Nokia N800.

The geographic locations of the cyclists are obtained from GPS receivers attached to cyclists' arms and this context is synthesized into *gaps and groups* information. The *gaps and groups* information is in turn broadcasted to all interested users equipped with the MIDAS middleware and an end-user application installed on their devices. The frequency of updates is about once every three seconds. A video demonstrating the live race at the Super Prestige Cyclocross using MIDAS middleware and the described application set can be seen at [21].

The gaps between groups of cyclists and the composition of groups are synthesized from the following cyclists' context: the last known cyclists position information, the roadbook waypoints, and the configured maximum distance between two consecutive cyclists of one group. A gap is defined as a distance between locations of two consecutive cyclists that exceeds a predefined threshold. In cycling a distance of about 25 meters is considered a gap. Cyclists between two consecutive gaps compose a group. In order to calculate gaps and groups, the application needs to calculate the distance between the successive waypoints and their distance to the finish (based on Haversine's Formula [14] combined with John P. Snyder's curvature [15]).

Figure 6 illustrates the operators used to calculate gaps and groups information. In order to improve the performance, all real time objects are stored in the memory. To share these objects between applications singleton instances are stored in the operator space running environment; therefore an operator has to be used to interact with these objects. Moreover, the output of one operator is used as an input to the other one.

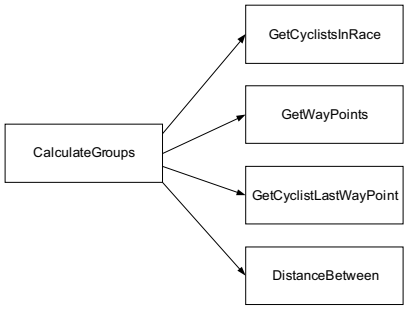


Fig. 6. Set of operators used in the application

To calculate the gaps and groups, the algorithm exploits the fact that every cyclist cycles from one waypoint to the other and sends several GPS measurements while on this path to reach the waypoint. Based on received GPS measurements, the algorithm computes location, distances between cyclists, their order, and if the distance between two cyclists is 25 meters or greater, then there is a gap.

Groups in the race are presented graphically to the user via a user application, as shown in Fig.7. The circle represents the whole track. Each dot represents a group. The progress of the groups is shown relative to Start and Finish. Additionally, the list of groups is presented to the user as a textual table. The first column of the table contains the group name (1 to n), the second column shows the number of cyclists present in the group, and the third one contains the distance to the preceding group. The leading group distance is replaced with a "Leading" indicator. For every cyclist, the first and last name, player number, as well as distance to the finish are shown. Once a group finishes the race, the distance is replaced by a "Finished" indicator, the cyclist icon is replaced by a flag, and the line is printed in green.

Fig. 7. Application GUI with actual data

5 Performance Evaluation

The MIDAS middleware and applications are implemented in Java. We ran all performance tests on a Nokia N800 device with the JamVM virtual machine [22] with a compiler for Java 1.4. This device was chosen by the MIDAS project because it is Linux based, allowing network and low-level programming. We also used a third party library for implementation of java scripts, Beanshell [23].

The performance of context synthesis is evaluated in terms of the response time of operator matching, context retrieval, and context processing (i.e., operator invocation),

Table 1. Response times

Average response times with varying number of specialized operators (i.e., 1, 2, 5, 10)	Based on 10 **first** queries	Standard deviation (based on 10 **first** queries)	Based on 10 **subsequent** queries	Standard deviation (based on 10 **subsequent** queries)
Matching algorithm time	2.49 sec	0.009 sec	1.94 sec	0.07 sec
Loading specialized & root scripts time	1.7 sec	0.087 sec	No average, for the first time only (1.7 sec)	No standard deviation
Total operator matching time	4.2 sec	0.087 sec	1.94 sec	0.07 sec
Context retrieval time	0.37 sec	0.006 sec	0.09 sec	0.001 sec
Loading dependency scripts time	0.15 sec	0.001 sec	0.17 sec	0.015 sec
Operator invocation time	0.67 sec	0.008 sec	0.36 sec	0.04 sec
Total query time	5.4 sec	0.045 sec	2.57 sec	0.07 sec

as well as the overall response time to a query sent by an application. The values shown in Table 1 were 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.

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 [4]. 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.

Note also that in some other scenario it could happen that after the second query the first query is made again but containing some other operator, this will also require operator matching. However, we plan (as future work) to introduce caching of queries and matched specialized operators in order to reduce the total query time.

There were 1000 spectators along the race course. Note that this deployment was intended as a proof of concept to validate middleware functionalities and was not designed to be an evaluation of the system using a statistically significant number of users. However, the impression of 9 users (monitoring the race on 6 tablet PCs and 3 Nokia N800 devices) was very positive. A few seconds of delay did not affect their "near real-time experience". Furthermore, users liked the way that they could select their favorite cyclists in the application, in order to know when he/she will pass their location. Zooming functionality also helped to improve overcome the limitations of the small screen when more cyclists were tracked during the race.

So far we have not examined the cases when context changes rapidly nor we have considered the issues concerning uncertainty in the context. We plan to address these issues in future work.

6 Related Work

Our context synthesizing work was inspired by the Aura Contextual Information Service (CIS) research project [16]. However, our queries are not SQL-like, but instead they are object-oriented, containing context operators which perform synthesizing operations. Context operators can in turn use other simpler operators to execute smaller tasks and to reuse existing functionality.

Modeling context with ontologies is in itself not a novel idea. Surveys of context modeling frameworks clearly indicate that modeling context with ontologies is the most expressive way to do it [17]. Typically, mobile devices being part of a context aware system need to remotely access the ontology model, and the context data. In case of CoBrA [18] the remote facility is an ‘intelligent agent’, called a Context Broker, which acts as a central point of the system maintaining a representation of context common to all the devices in the network. The SOCAM [19] solution also relies on a shared context space located on an external device (an OSGi gateway) which can be accessed by multiple context aware services. MIDAS differs from these architectures in that it is capable of handling ontologies on mobile devices, which makes it possible to provide local access to context modeled with ontologies for every device in the network. This seems especially useful in ad-hoc networks where access to a central server cannot be provided.

J. I. Hong and J. A. Landay [20] emphasize a need for creating a basic infrastructure services and application-specific services, the latter implemented on a case-by-case basis. One such basic infrastructure service is automatic path creation, which transforms raw sensor data to higher-level context data. It automatically composes operators based on high-level needs and what resources are available. Our work extends this idea to enable multiple applications or even different context-aware systems to use the same operators designed for a specific domain without being concerned about their implementation. Moreover, we also enable chaining of operators, where each operator takes some existing context information (as defined by a context model)

as input and provides new context information as an output. All applications can reuse already deployed operators and add their own implementations of the same generic operators.

7 Conclusion and Future Work

We have presented and evaluated the approach for context synthesis using operators on a Nokia N800 device. Operators for context synthesizing are domain-specific functions over the context data. The benefits of these operators are that by performing operations over the existing context information, new context information that previously did not exist in the system can be produced. Moreover, applications can use the same operators to synthesize context information, without being concerned about their implementation. This also enables applications to share the cost of context synthesis by querying about the result of operators invocation.

We have evaluated this operator-based context synthesis approach in terms of response time to context query sent by the application and 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. The main advantages of context operators are the reusability, extensibility, and interoperability, facilitated by ontology-based context modeling. For this purpose MIDAS provides a dedicated Lightweight Ontology library for representing and manipulating ontologies on mobile devices. We also demonstrated the use of context operators in the cyclist race application.

We plan to evaluate the response time of executing the remote operator invocation as well as to use caching decisions made by operator matching algorithm for a certain context query. We will also conduct a user study based on our next deployment. As a next step in context synthesizing we plan to use operators to combine inference algorithms in order to derive about high-level context.

Acknowledgements. The authors of this paper would like to acknowledge the partial financial support given to this research by the EU IST MIDAS project (6th Framework Programme, contract number 027055). We would also like to thank Prof. Gerald Q. Maguire Jr. for his valuable comments to this research work.

References

1. EU FP6 IST MIDAS project (2008), `http://www.ist-midas.org`
2. Devlic, A., Klintskog, E.: Context retrieval and distribution in a mobile distributed environment. In: Third Workshop on Context Awareness for Proactive Systems (CAPS 2007), Guildford, UK (2007)
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable description logics for ontologies. In: 20th National Conference on Artificial Intelligence (AAAI 2005), Pittsburgh, Pennsylvania, USA, pp. 602–607 (2005)
4. Koziuk, M., Domaszewicz, J., Schoeneich, R.O.: Mobile Context-Addressable Messaging with DL-Lite Domain Model. In: The 3rd European Conference on Smart Sensing and Context (EuroSSC 2008), Zurich, Switzerland, October 29-31 (to appear, 2008)

5. Domaszewicz, J., Koziuk, M., Schoeneich, R.O.: Context-Addressable Messaging with ontology-driven addresses. In: The 7th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2008), Monterrey, Mexico (to appear, 2008)
6. Smith, M.K., Welty, C., McGuinness, D.L.: OWL Web Ontology Language Guide. W3C Recommendation (2004), http://www.w3.org/TR/owl-guide/
7. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the semantic web recommendations. Technical Report HPL-2003 (2003), http://citeseer.ist.psu.edu/carroll04jena.html
8. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web 5(2), 51–53 (2007)
9. Magkanaraki, A., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M., Tolle, K.: RQL: A Functional Query Language for RDF. In: Gray, P.M.D., Kerschberg, L., King, P.J.H., Poulovassilis, A. (eds.) The Functional Approach to Data Management: Modelling, Analyzing and Integrating Heterogeneous Data. LNCS. Springer, Heidelberg (2004)
10. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL Query for OWL-DL. In: Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions, Innsbruck, Austria, June 6-7 (2007)
11. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang, H.: The Manchester OWL Syntax. In: OWL: Experiences and Directions 2006, Athens, Georgia, USA (2006)
12. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation (2004), http://www.w3.org/TR/owl-semantics/
13. Jabłonowski, M., Boetzel, P.: Middleware Layer For Semantic Object Tagging. Master Thesis at Warsaw University of Technology, Warsaw, Poland (2007)
14. Sinnott, R.W.: Sky and Telescope. Virtues of the Haversine 68(2), 159 (1984)
15. Snyder, J.P.: Map Projections – A Working Manual., U.S. Geological Survey, Professional Paper 1395, US Government Printing Office, Washington DC (1987)
16. Judd, G., Steenkiste, P.: Providing Contextual Information to Pervasive Computing Applications. In: First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), Fort Worth, Texas, pp. 133–142 (2003)
17. Strang, T., Popien, C.L.: A context modeling survey. In: Workshop on Advanced Context Modeling, Reasoning and Management as part of the 6th International Conference on Ubiquitous Computing (UbiComp 2004), Nottingham, England, pp. 33–40 (2004)
18. Chen, H., et al.: A Context Broker for Building Smart Meeting Rooms. In: Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAAI Spring Symposium, Palo Alto, CA, USA (2004)
19. Gu, T., Wang, X.H., Pung, H.K., Zhang, D.Q.: An Ontology-based Context Model in Intelligent Environments. In: Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, San Diego, California, USA (2004)
20. Hong, J.I., Landay, J.A.: An Infrastructure Approach to Context-Aware Computing. Human-Computer Interaction 16(2, 3, 4), 287–303 (2001)
21. MIDAS video (2007), http://www.youtube.com/watch?v=yulUmlVH8Jc
22. JamVM – A Compact Java Virtual Machine (2008), http://jamvm.sourceforge.net/
23. Beanshell - Lightweight scripting for Java (2008), http://www.beanshell.org/

PAPER 5

Context inference of users' social relationships and distributed policy management

Alisa Devlic[1,2], Roland Reichle[3], Michal Wagner[3], Manuele Kirsch Pinheiro[4,5], Yves Vanrompay[4], Yolande Berbers[4],
Massimo Valla[6]

[1]Appear Networks, Kista, Sweden
[2]Royal Institute of Technology (KTH), Department of Communication Systems, Kista, Sweden
[3]University of Kassel, Kassel, Germany
[4]Katholieke Universiteit Leuven, Department of Computer Science, Leuven, Belgium
[5]Centre de Recherche en Informatique, Université Paris 1–Panthéon Sorbonne, Paris, France
[6]Telecom Italia Lab, Torino, Italy
alisa.devlic@appearnetworks.com, {reichle, wagner}@vs.uni-kassel.de,
{yves.vanrompay, yolande.berbers}@cs.kuleuven.be,
manuele.kirsch-pinheiro@univ-paris1.fr, massimo.valla@telecomitalia.it

In Proceedings of the 6th IEEE Workshop on Context Modeling and Reasoning (CoMoRea) at the 7th IEEE Conference on Pervasive Computing and Communications (PerCom'09), Galveston, Texas, March 2009, pp. 755-762.

Context inference of users' social relationships and distributed policy management

Alisa Devlic[1,2], Roland Reichle[3], Michal Wagner[3], Manuele Kirsch Pinheiro[4,5], Yves Vanrompay[4], Yolande Berbers[4], Massimo Valla[6]

[1]Appear Networks, Kista, Sweden
[2]Royal Institute of Technology (KTH), Department of Communication Systems, Kista, Sweden
[3]University of Kassel, Kassel, Germany
[4]Katholieke Universiteit Leuven, Department of Computer Science, Leuven, Belgium
[5]Centre de Recherche en Informatique, Université Paris 1–Panthéon Sorbonne, Paris, France
[6]Telecom Italia Lab, Torino, Italy

alisa.devlic@appearnetworks.com, {reichle, wagner}@vs.uni-kassel.de, {yves.vanrompay, yolande.berbers}@cs.kuleuven.be, manuele.kirsch-pinheiro@univ-paris1.fr, massimo.valla@telecomitalia.it

***Abstract*— Inference of high-level context is becoming crucial in development of context-aware applications. An example is social context inference – i.e., deriving social relations based upon the user's daily communication with other people. The efficiency of this mechanism mainly depends on the method(s) used to draw inferences based on existing evidence and sample information, such as a training data. Our approach uses rule-based data mining, Bayesian network inference, and user feedback to compute the probabilities of another user being in the specific social relationship with a user whose daily communication is logged by a mobile phone. In addition, a privacy mechanism is required to ensure the user's personal integrity and privacy when sharing this user's sensitive context data. Therefore, the derived social relations are used to define a user's policies for context access control, which grant the restricted context information scope depending on the user's current context. Finally, we propose a distributed architecture capable of managing this context information based upon these context access policies.**

Keywords- Context inference of user social relations, context access control policies, context scope, user privacy.

I. INTRODUCTION

Deriving context information without explicit user input is a key requirement for context-aware applications development. Additionally, some information can only be inferred by analyzing the user's activities over time. An example is social context inference - i.e., deriving social relations from a user's daily communication with other people. The user's communication with others is captured on a mobile device by logging the data about sent and received SMS & MMS messages, call logs, and e-mails to a file (see Figure 1). This file is uploaded once a day to a computer for analysis of the user's communication patterns in context – in order to infer the user's social relations with the people he/she interacts with. This inferencing is based on rule-based data mining, Bayesian network inference, and user feedback to compute the probabilities of another user being in a specific social relationship with a given user. The inferred social relationships are stored in the form of a Friend-Of-A-Friend (FOAF) ontology extended with social relationship terms (i.e., family, friend, colleague, or unknown).

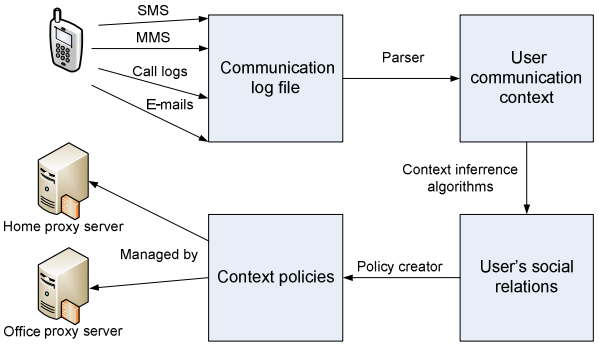


Figure 1. User social relations inference and policy management approach

Because users are sensitive about sharing their context with other users without explicitly giving their permission, a non-intrusive means of indirectly gaining the user's permission to share some context information is needed. To do this, we propose to use the user's social relations as a means to create user specific policies for granting access to their context information. This enables a user to specify different levels of access to his context information based on the relation he/she has with the other user that requests it (e.g. whether this other user is a family member, a friend, a 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. As a result the user only has to explicitly determine the access to be given to a class of users, rather than to each specific user. However, these policy rules might depend upon the situation the user might be in. Therefore, in our policy design we also introduce context conditions to allow a user to define different rules (i.e., allowing distribution of a particular context to a specific scope

978-1-4244-3304-9/09/$25.00 ©2009 IEEE

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.

II. MOTIVATING SCENARIO

The research presented in this paper belongs to a larger initiative, the MUSIC Project [1], aiming to develop a platform and tools for rapid development of context-aware self-adapting applications. Among the scenarios considered, the Instant Social (IS) scenario [2] proposes a mobile content sharing platform that enables users to browse, search for, and share multimedia content scattered over devices in different contexts.

In IS scenario, Paul is visiting a rock festival and wants to share photos with other users at the festival. The IS application running on Paul's mobile device communicates with other instances of itself running on nearby devices, sharing not only media content[1], but also context information about these users (their preferences, location, etc.) and their mobile devices (available memory, CPU utilization, and network bandwidth). Context information is used by the IS platform for adaptation purposes [2]. For example, these instances may utilize local services or services running on other instances in order to save memory or battery power. The proposed implementation is based on the MUSIC adaptation middleware which bases this choice on the result of a utility function that ensures sufficient replication of data for stable operation when members join and leave dynamically, and balances the load over the member devices according to their resource availability [2].

Similar to content sharing, context distribution also raises privacy issues. For most of people, social interactions with different communities during a normal day typically exhibit a pattern. These communities can be broadly categorized as family, friends, and colleagues. We have assumed that all individuals in a given class will be treated in the same manner (with regard to context access). Context information is often considered sensitive information whose distribution should be controlled and limited. Therefore, during the rock festival, Paul might share his location only with his friends & family. While to his colleagues he might simply indicate that he is "Out of the Office". In contrast, during working hours, a person's family & friends do not need to know that he is currently in a meeting; they might only be able to learn that Paul is "At work".

This example clearly shows the need for setting and following privacy constraints for both content sharing and context distribution. It also demonstrates the benefits of using the user's social relationships to reduce the burden upon the user – while preserving the desired user control. Moreover, it indicates the need for context dependent privacy restrictions, because user preferences may differ for different situations. In this paper, we focus on how to allow users to control the distribution of their context information based on their current context and their social relationships to the requesting user. These social relations are inferred based on the user's daily communication patterns.

[1] Note that this is the user's own media content and not that of a performance taking place at the festival.

III. ARCHITECTURAL VIEW

The MUSIC context middleware is responsible for collecting, organizing, managing, and sharing context information as acquired by sensors with the proper context clients. Once collected, context information is cached in context storage for the period determined by the nature of information and application needs. In MUSIC, reasoners are specialized context sensors which process existing context data in order to compute higher-level context data. Sensors have plug-in components with a mechanism to activate or deactivate the sensing mechanism (previously described in [3]).

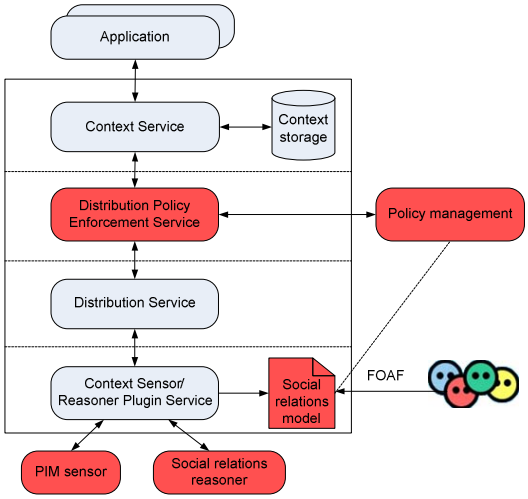


Figure 2. MUSIC Context middleware architecture

To implement our approach, we extend the existing MUSIC context management system (depicted in Figure 2) with a *Personal Information Management (PIM) sensor* which captures features from the user's daily communication patterns; a *Social relations reasoner* (applies a probabilistic rule-based reasoning to the user communication context (provided by the PIM sensor) in order to infer the user's social relationships with other users he/she was in contact with) which are stored in the user's *Social relations model*; the *Distribution Policy Enforcement Service* creates distribution policy requests and sends them to the *Policy management* module for evaluation. This *Policy management* module scopes the dissemination of context information based on the user's social relationship with the query initiator and the user's current situation. The *Policy management* module accesses the user's *Social relations model*. This model contains the user's contacts divided in social groups in the FOAF format. The different parts of this model can be distributed over multiple proxy servers to facilitate processing requests for context information. After obtaining a decision from the *Policy Management* module, the *Distribution Policy Enforcement Service* enforces this decision by forwarding the context information in the allowed scope to the *Distribution Service* or canceling the context distribution and notifying the *Context Service* about the rejection of the request. The *Policy management* module is handled by distributed proxy servers as it is computationally demanding. Sensors and reasoners do not need to run locally on the device where the application is launched, but can be accessed remotely via the *Distribution Service* (which is not described in this paper).

IV. INFERENCE MECHANISM OF USER'S SOCIAL RELATIONS

In order to infer the users' social relationships from logged communication activities via their mobile phone and email we follow an approach similar to that proposed in [4]. However, we modified this approach, as several prerequisites differ:

- Unlike [4] and [5], only a very limited amount of log-data is utilized (with regards to both the number of involved users and the recording time). However, we consider this to be more realistic for our application domain and as the data is collected by each participant in the communication – there is less concern about privacy. This leads to better scaling with the number of users, since each user builds their own model.
- Only log-data about communication activities via mobile phone and email are available, no information about proximity was incorporated.
- Our goal is to utilize only simple inference rules. These rules are intended to be easily understandable by human developers & users and to be updated by a learning mechanism incorporating user feedback.

Of these differences, the limited amount of log-data raises some new challenges, as special care has to be taken to avoid over constraining of the inference rules due to the limited training data. Therefore, a primary goal was to develop a robust approach that is applicable for a limited amount of log-data and does not rely on extensive (historical) logging of communication activities. The rules, once derived by a developer are further adjusted or modified by incorporating appropriate user feedback.

A. Collection of log data

The PIM sensor which acquires data about the users' communication activities (i.e., incoming/outgoing phone calls and SMS/MMS messages) runs as a background C# application on Windows Mobile phones. Each time the phone is synchronized using ActiveSync, incoming e-mails are also recorded. For each communication activity, a log entry with a timestamp, contact information, and duration is made. For email, the sender & receiver email addresses, and the subject are logged. If the user has specified a category for a contact in his address book, then the category of any contact involved in the conversation is also recorded in the log. Logs are stored in the phone's memory and later uploaded to a server for post processing and relationship inference.

B. Inference approach

The proposed inference approach for deriving user social relations from mobile phone and email log-data consists of five steps (see Figure 3):

1. The data collected by the PIM sensor are parsed and converted into standard comma separated value (CSV) format, in order to make it available to other tools.
2. A set of features is derived that might be used to distinguish between the different classes or categories of the corresponding communication partners. Examples of potentially useful features are 'contact data available' and 'number of activities' (this might lead to inference rule such as 'if there is no contact data available and there is only limited communication, then the corresponding user is regarded as a stranger).
3. The usefulness of the selected features is evaluated on the log-data and the features that best contribute to discrimination into the categories of communication partners are retained.
4. These features are utilized for rule-based data-mining approach, e.g. PART [6], in order to derive inference rules based on the log-data. The derived rules are manually inspected by the developer. Rules that hint of overfitting of data are discarded or modified.
5. These rules are used to create a classifier and a simple Bayesian network in order to estimate the confidence of the classifier in its decision. If the classifier is not able to satisfactorily classify the log-data, then go back to step 2.

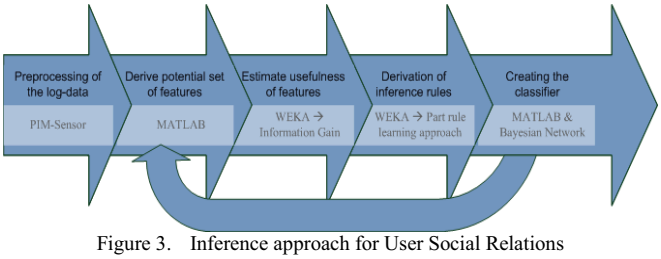


Figure 3. Inference approach for User Social Relations

In the following subsections each of the steps following the preprocessing step are described in more detail.

1) Derive potential set of features (Step 2)

The general objective in our proof-of-concept application was to distinguish between four different categories of communication partners: (1) stranger, (2) colleague, (3) friend, and (4) family member. For this purpose, we have derived 16 candidate features, shown in first column of Table I (each with a value stratified as being low/medium/high, except for Contact data available which has only a yes/no value).

TABLE I. DERIVED AND SELECTED USEFUL SET OF FEATURES

Derived potential set of features	Selected useful features	Abbreviations for selected features
(1) Contact data available	Yes	[ContactData]
(2) Number of communication activities	Yes	[nInteraction]
(3) Number of phone calls	Yes	[nPhoneCalls]
(4) Number of phone calls at work time	No	
(5) Number of phone calls in free time	Yes	[nPhoneCallsFreetime]
(6) Number of phone calls at week end	Yes	[nPhoneCallsWeekend]
(7) Average duration of phone calls	No	
(8) Number of SMS/MMS	Yes	[nSMS]
(9) Number of SMS/MMS at work time	No	
(10) Number of SMS/MMS in free time	No	
(11) Number of SMS/MMS at week end	No	
(12) Number of Emails	No	
(13) Number of Emails at work time	No	
(14) Number of Emails in free time	No	
(15) Number of Emails at week end	No	
(16) Number of Business Emails	Yes	[nBusinessMails]
(subject containing e.g. 'meeting', 'deadline', 'review')	No	
Additional features		
(17) Number of SMS/MMS on Friday or Saturday night	Yes	[nSMSFrSaNight]
(18) Number of phone calls on Friday or Saturday night	Yes	[nPhoneCallsFrSaNight]

For extracting the features described above, we imported the CSV-files into MATLAB [7] and applied appropriate scripts. The statistical features are normalized with regard to the average value of all interactions of a user and discretized to the values 'low' (< 0.75*average), 'medium' (>= 0.75*average, <= 1.5*average), and 'high' (> 1.5*average).

As this set of features proved to be insufficient to discriminate between the four categories of communication partners (in particular between 'family member' and 'friend') it was enhanced by including two additional features:

(17) Number of SMS/MMS on Friday or Saturday night
(18) Number of phone calls on Friday or Saturday night

Without these two rules the classifier is not able to perform significantly better in distinguishing between 'family member' and 'friend' than a random classifier.

2) Estimate usefulness of features (Step 3)

In order to estimate the ability of the features to contribute to the discrimination of the four different categories of communication partners, we exported the log-data along with the two added features as a standard CSV file. This file was imported into WEKA (Waikato Environment for Knowledge Analysis) [8]. WEKA is a data-mining tool that allows applying a number of different feature selection algorithms on the log-data enriched with the new features. We have decided to calculate the Information Gain [9] achieved by a feature in order to estimate its usefulness. Information Gain is a well known measure to select appropriate features and is commonly used in Decision Tree Learning [9], which is quite similar to our rule-based approach. As a result, the set of features shown in the second column of Table I was selected. According to the Information Gain measure, all of the other features did not really contribute to the discrimination of the four categories of communication partners and therefore were discarded.

3) Derivation of inference rules (Step 4)

While the previous steps are quite similar to existing approaches, such as presented in [4], in the next step we go in a different direction than others, in order to cope with the additional challenges and objectives as mentioned in Section IV. To derive the inference rules we again utilized the WEKA tool and its facilities to create classifiers from training data. As our objective is to provide easy to understand inference rules, we applied the PART rule learning approach [6] to create a rule based classifier. However, the derived rules are **not** directly used. First they are manually inspected by the developer, to see if they hint at overfitting or if they correspond to the general understanding that a developer has about the characteristics of communication with a partner of a certain category.

For example, rules which incorporate a large number of different features and allows only a single value for most of the features, hints at overfitting. The same applies for rules that do not really reflect the common understanding of communication characteristics of a category. In this respect, we have to answer the question why we have chosen this approach, instead of utilizing other classification approaches and automatically tuning parameters or rules to avoid overfitting. Usually, in order to detect overfitting the training data are split up in a training set and a test set. The training set is used to create the classifier, whereas the test set is used to check if the classifier adequately generalizes the remaining data. This approach can be used to automatically tune parameters and rules. We have done the same, experimenting with different approaches. However, our experiences have shown that it is quite difficult to find an appropriate setup with a very limited amount of log-data and that the automatically tuned classifiers had a poor performance. We have also split the training data into one training set and two test sets, where one test set was used to detect overfitting and the other to check if the tuned classifier adequately generalizes the second test set as well. However, this was not always the case. Deriving simple and easily understandable rules and asking the user who is able to incorporate general knowledge about communication characteristics has revealed to be more reasonable.

For our proof-of-concept application, we derived the following rules:

(1) if (nInteraction == low and ContactData == no) or (ContactData == no and nPhoneCalls == 0 and nSMS == 0) then category = 'stranger'
(2) if not('stranger') and (nBusinessMails > low or nPhoneCallsFreetime <= medium and nPhoneCallsWeekend <= medium) then category = 'colleague'
(3) if not('stranger') and not('colleague') and nPhoneCallsFrSaNight >= medium and nSMSFrSaNight >= medium then category = 'friend'
(4) if not('stranger') and not('colleague') and not('friend') then category = 'family'

Note that the rule (4) is a result of the PART rule learning approach and the order of filters: the first filter identifies strangers and then out of these colleagues can be extracted. Assuming that only friends and family members are left, the most useful rule filters friends out of the remaining group. Finally, the group that remains are family members.

4) Creating the classifier (Step 5)

The corresponding classifier is directly created from the inference rules shown above. However, as these rules may not directly reflect the communication characteristics in every case, it is crucial to be able to estimate the confidence of the classifier in its classification result. For this purpose, we derive a simple Bayesian network from the set of rules. The conditional probability tables (CPTs) of the nodes are estimated by simply counting the occurrences in the training data that matched the user's self reported social relations whose context data was logged. From this simple Bayesian network the probabilities for a certain category can be calculated. The entropy derived from these probabilities gives a hint on the confidence the classifier has in its classification result.

C. Incorporating user feedback

Incorporating user feedback is still work in progress and also a key part of future work. Therefore, in this subsection only general ideas are presented.

If the rules reflect the training data, then usually the classification result corresponds to the category with the

highest probability. However, if the classification is characterized as high entropy, then the classification can be regarded as unsure and the user has to be asked for feedback. This user feedback can be used to create new instances of labelled training data and to adjust the Bayesian Network. For this purpose, the user chooses a weight[2] to specify the influence of the new training data on the current CPTs. Obviously the updated CPTs affect the entropy values of the classifications. If the update of the CPTs results in greater uncertainty in the classification results, then the classification rules have to be adjusted. To determine the updated rules, PART rule-learning is applied again. If the resulting rules are similar to those already used in the classifier, but with changed comparison values, then the rules can be automatically adjusted. However, if the CPTs are adjusted to a large extent, then the rules may become obsolete and the structure of the rules (leaving out features, incorporate new features) will change. In this case, the rule is presented to the user in an easily understandable expression, e.g. in English language, and he/she is asked if this rule is characteristic for their communication activities with the corresponding category of communication partner.

We are aware that estimating the usefulness of rules is not always a trivial task for the user and that asking the user for such feedback is not desirable. However, learning the rules completely autonomously we regard as not feasible at the present, because of the very limited amount of log-data to be expected. Therefore, currently we consider it unavoidable to ask the user for feedback as mentioned above. However, this has to be investigated more thoroughly in the future.

D. *Evaluation results*

For our proof-of-concept application we ran the PIM sensor for three different users with a recording time of one week. This resulted in 331 communication activities with 41 different communication partners in total: 122 interactions with 15 partners for $user_1$, 121 interactions with 13 partners for $user_2$, and 88 interactions with 13 partners for $user_3$. These numbers highlight again the very limited amount of available log-data and therefore the difficulty of the problem.

Despite this limited data, using the simple inference rules presented above, 33 out of the 41 different communication partners were classified correctly with regard to the users' self reported social relations: 13 out of 15 for $user_1$, 11 out of 13 for $user_2$, and 9 out of 13 for $user_3$. This corresponds to classification successes of about 87%, 85%, and 69%, respectively. Of these, 4 out of the 8 incorrectly classified communication partners are friends that are classified as colleagues. The remaining 4 incorrectly classified partners were more or less spread equally among colleagues, friends, and family members. Unfortunately, a classification success of about 69% for $user_3$ is not very satisfactory. The reason for these incorrect classifications is that the $user_3$ had the smallest number of total interactions. Because of the lack of data about these contacts, there is little evidence to base a decision on. Hence it would be better to add a fifth category – not yet classified. However, more important than the pure classification success is the fact that all the incorrect classifications were judged to be unsure, therefore, the user would have been asked for feedback.

In conclusion, 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. 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.

We are aware that our assumption of mutual exclusive categories of social relationships is not valid in general (e.g., a colleague can also be a friend). In fact, such relationships are seen in our classification results. We plan to extend our approach to detect characteristics of the different categories without having a discriminating classification problem in mind. However, first it has to be investigated if such a limited amount of log-data is sufficient to derive such characteristics.

V. USER SOCIAL RELATIONS MODEL

The user's social relations model stores relations between users inferred by the context inference mechanism. Several formats are emerging to store relations in social networks [24][25]. We chose to store inferred relations using the FOAF format, an RDF-based language already used by several social web services, but extended to categorize the *type of relationship*. A previous proposal to extend FOAF with relationship types is described in [26], where the property foaf:knows, the only option offered by the FOAF ontology, is extended by sub-properties such as: colleagueOf, friendOf, etc. that can be used in our case to distinguish which type of relationship is inferred between two users.

We have added relationship tags to the FOAF Person element, as showed in Figure 4, where the Person "cristina" is defined to be a colleague of Person "massimo", while Person "francesco" is defined to be a friend.

```
<foaf:Person rdf:nodeID="massimo">
    <foaf:name>Massimo Valla</foaf:name>
    <foaf:firstName>Massimo</foaf:firstName>
    <foaf:surname>Valla</foaf:surname>

    <rel:colleagueOf>
      <foaf:Person rdf:nodeID="cristina">
        <foaf:name>Cristina Fra</foaf:name>
      </foaf:Person>
    </rel:colleagueOf >

    <rel:friendOf rdf:nodeID="francesco"/>
</foaf:Person>
```

Figure 4. User social relations model using FOAF extended to support relationship types

The use of FOAF format to represent inferred relationships is also useful if external applications or web services need to access this information in a standard way. Several proposals are emerging to offer social network and relationship information to external services, the most promising one being OpenSocial

[2] The value is in the interval [0.0, 1.0]. A value of 1.0 indicates that the communication partner with unsure classification is very representative of the communication behaviour of the user. A value of 0.1 indicates that the communication partner is representative only to a limited extend, and 0.0 means not representative at all.

API [27]. Our *Social relations reasoner* implements the OpenSocial API in order to be able to export this information externally. Finally, the inferred relations could be integrated with information obtained from external social networks, for example by periodically merging FOAF data exported by such networks with the data inferred by our inference mechanisms.

VI. DISTRIBUTED CONTEXT POLICY MANAGEMENT

Based on the social relations inferred in Section IV, a user can define policies controlling the sharing of his sensitive context information with other users, based on their relations. The goal of these context policies is to grant different degrees of access (i.e., a particular context scope) for the user's different social relation groups. This context scope can be conditioned upon the user's context (i.e. time, place, or activity). The social relations are identified by the FOAF ontology mentioned in Section V. The membership in these groups is automatically inferred, as explained in Section IV.

Each context distribution policy contains a target and one or more policy rules (such as shown in Figure 5). The policy rule can be seen as a set of quadruplets "*relationship, resource, action, context condition*" in which *relationship* is the requesting entity's attribute corresponding to his/her social relationship with the user to which the policy refers to, *resource* corresponds to the context scope associated with this relationship, *action* is to grant or deny read or write access to the actor for the corresponding resource, and *context condition* specifies the user's context in which this action is taken.

Target:
Resource: context information=location
Policy rule:
Relationship: FOAF relation = friendOf
Resource: context scope = street address
Action: read allowed
Context condition: activity = clubbing

Figure 5. Context policy example

Figure 5 depicts a policy for accessing location information of a given user in a certain context. Using this policy, a user indicates that his friends may access his street address when querying about his location if his/her activity is set to clubbing.

A. Design of context policies based on user's social relations

Figure 6 illustrates the same policy presented in Figure 5, but expressed using the XACML Policy Language. In this standard language, policies are defined for a given target, which can be a resource, an actor (called a "subject"), an action, or an environment involving these elements, and they are composed by a set of rules. For each rule, the policy writer defines an effect if the rule is evaluated as "true" (permit or deny), an optional target, and an optional condition. When evaluating a policy, each rule is evaluated using a rule-combining algorithm indicated in the policy definition.

In our example, the policy target is the resource (this policy limits access to location information); however this is not shown in Figure 6 due to limited space. This policy contains only one rule, whose action is to read a particular resource value and which defines, as a condition that the subject has to be in FOAF relation "friendOf" with the context owner.

```
<Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:SimpleRule1"
Effect="Permit">
  <Target>
    …
      <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <ResourceAttributeDesignator
AttributeId="urn:ist-music:names:context:model:concept:value"
DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">urn:ist-
music:names:context:model:concept:value:address</AttributeValue>
      </ResourceMatch>
    …
 </Target>
 <Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <SubjectAttributeDesignator AttributeId="relation"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
       <AttributeValue
DataType=http://www.w3.org/2001/XMLSchema#string>friendOf</Attribute
Value>
     </Apply>
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <ResourceAttributeDesignator AttributeId="activity"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
       <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">clubbing</Attribu
teValue>
     </Apply>
   </Apply>
 </Condition>
```

Figure 6. Policy rule in XACML

It is worth noting in Figure 6 that the granted resource value when requesting a resource is described through a string ("*urn:ist-music:names:context:model:concept:value:address*"), indicating a street address. This string refers to a particular context value represented in the MUSIC Context Model [11]. By referring to concepts represented in the MUSIC ontology, we intend to allow the definition of context distribution policies over all context concepts recognized by a MUSIC application, such as the Instant Social sharing platform [2].

B. Context-aware policy management architecture

The context-aware policy management architecture is shown in Figure 7. It was inspired by [10], but extended to support a distributed and decentralized policy management that controls access to context information based on the social group the context requestor belongs to and the current context of this requested entity. The distributed approach currently utilizes the user's personal proxies.

The architecture involves three main entities: *context client*, *context source* (which is the requested user's context provider), and a *proxy server*. The *context client* sends a query for context information to the *context source*. We will assume for this discussion that the query initiator is the user's friend. The context source's Policy Enforcement Point (PEP) retrieves from the Social relations model the query initiator's relationship with the user, creates a context scope request in XACML format, and sends it to the remote Policy Decision Point (PDP) at the

Home proxy server for evaluation. The PDP sends a request to the Policy Information Point (PIP) for policies related to this decision request, which retrieves these policies from the Policy Information Base (PIB). In order to evaluate which policies are applicable for the received request, the PDP queries the Context Information Base (CIB) via the Attribute Information Point (AIP) for the missing (context) attributes. After evaluating the Target element of the retrieved policies and identifying an applicable policy, the PDP evaluates the (context) condition of this policy. Note that the CIB subscribes at the *context source* node to receive updates on context values which are needed for this purpose. When a match is found, the PDP makes a decision and sends it back in the response to the PEP on the *context source* node. After obtaining the authorization decision from the *proxy*, the PEP enforces this decision. If the decision permits the access to the user's location information, the *context source* retrieves its value from its own CIB, formats it in the granted scope, and sends this scoped value in the response to the *context client*.

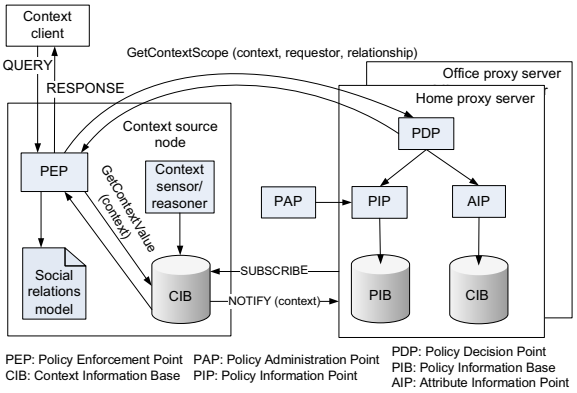


Figure 7. Context-aware policy management architecture

The Policy Administration Point (PAP) is the entity that creates policies and sends them to the PIP at the *proxy server*. After it has received a policy, the proxy's PIP should eliminate the rules which conditions refer to the groups for which this proxy does not manage policies, and store such a modified policy into the PIB. Therefore, in our example, a home proxy would process only rules related to family and friends.

Currently we accommodate context policies that follow the XACML standard. However, since XACML is designed for static attributes and the context information we want to introduce in the rule's condition is volatile, we plan to modify the design of these policies to allow rules to be context dependent - by creating a context-switch and inserting the rules into context conditions. This idea would follow a similar approach to [12], where we extended CPL (Call Processing Language) scripts with context parameters; however, instead of call logic actions there would be policy rules (see Figure 8).

Such a context policy design would enable the proxy to evaluate only the set of rules that are relevant to the user's current situation. This along with dividing the policy management load on several proxy servers responsible for a particular group (or a set of groups) should potentially improve the time to respond to a context query.

```
<context-switch owner="alisa">
  <context activity="clubbing">
    <!-- rule1: allow friends to see your current address -->
    <!--rule2: allow family to see your current city scope -- >
  </context>
   <context day="workday" and timeOfDay="[9a.m.,5p.m.]">
     <!--rule3: allow colleagues to see your current address -->
   </context>
  <otherwise>
    <!--reject-->
  </otherwise>
</context-switch>
```

Figure 8. An example of a CPL policy with the context-switch element

VII. RELATED WORK

Context reasoning derives higher level context information from context sensors. Probabilistic reasoning approaches like Bayesian Networks or hidden Markov models seem to be well suited for this purpose [13]. User social relationships can be inferred from context and used for sharing context information based on policies that utilize social relations. To the best of our knowledge, existing work on inference of user social relations has not yet been part of designing context management and sharing systems. However, inference of user social relations has been a research topic for a long time in the area of social network analysis [14]. Although there are many approaches for mining social relationships, such as [15], [16], and [17], these approaches differ in the number of users, the number of involved social groups, and the nature of the utilized data which they handle. However, only a few researchers have addressed the challenge of analyzing user social relations from interaction data, i.e. based on email and phone logs.

Eagle *et al.* [4] and [5] focused on predicting user social relations. They used interaction data including logs of phone calls and text messages, along with location and proximity information gathered from the cell information of the mobile phone and its Bluetooth interface. In both studies extensive log data were collected for the activities of around 100 users. For the inference task, they applied well known multivariate analysis methods, learned Gaussian mixture models, and well established methods of feature selection. However, we faced additional challenges because we did not have location or proximity data and we tried to infer user social relations from a very limited amount of log data. It is well known in data mining that it is very difficult to infer information from a quite limited amount of data [18]. While properties or behavioral characteristics may be stable over a large group of individuals, the characteristics of individuals may differ quite a lot.

Policies are used for granting (or denying) users access to given resources such as security or network management. Kamienski *et al.* [21] apply policies to the management of ambient networks. Policies are expressed using an extended version of the XACML [20]. Such policies are used to manage network composition in a general architecture for PBMAN (Policy-Based Management of Ambient Network), which is an extension of the IETF PBM framework [22]. Nupur [10] used policies based on the XACML for access control of context information within the ambient network using a centralized Policy Management System (PMS) architecture, which may

lead to scalability problems. Although XACML is presented as a standard for access control policies, Toninelli *et al.* [19] and Verlaenen *et al.* [23] propose other languages for policy description. They each argue that policy definition and conflict resolution in pervasive or service-oriented environments demand reasoning capabilities that can be obtained by combining OWL and rule-based languages.

VIII. CONCLUSION

We have presented a rule-based inference approach for deriving user social relations with his/her communication partners based on the log-data collected by their mobile phone. This approach uses a PART rule-based data mining to derive relevant inference rules, which in turn are used to create a classifier and a simple Bayesian network to provide confidence values. User feedback is incorporated to adjust the Conditional Probability Tables of the Bayesian Network and tune the inference rules in order to obtain better classification results. The proposed approach was evaluated on data obtained from three users monitored for a week, resulting in classification success rates of 87% for $user_1$, 85% for $user_2$, and 69% for $user_3$, despite simple rules and very limited log-data.

The derived social relations with the contacts are stored in the FOAF ontology extended with social relation terms. We proposed these relations to be used as attributes when creating context-aware policies to check if the requesting entity belongs to a particular social group. Thus, there is no need for explicitly stating in the policy all the actors that the policy refers to. Context policies also specify the user's context in which a policy action is executed.

In future, we plan to evaluate our social inference approach with at least 12 users to prove its success - and apply it to publicly available data for mining in addition to our data. We will extend this approach to support classification of the user's communication partners to more than one social relationship category. Finally, we will extend these policies with a "context-switch" to enable rules to be context-dependent and perform a performance analysis of this context policy management.

ACKNOWLEDGMENT

The authors of this paper would like to Prof. Gerald Q. Maguire Jr. for fruitful comments to this research work.

REFERENCES

[1] EU IST MUSIC project, Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environments, http://www.ist-music.eu, 2008.

[2] L. Fraga, S. Hallsteinsen, and U. Scholz, "Instant Social – Implementing a Distributed Mobile Multi-user Application with Adaptation Middleware", In Proceedings of the First International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS), Oslo, Norvay, June 2008.

[3] N. Paspallis et al., "A Pluggable and Reconfigurable Architecture for a Context-aware Enabling Middleware System", In Proc. 10th International Symposium on Distributed Objects, Middleware, and Applications (DOA'08), Monterrey, Mexico, Springer-Verlag, November 2008.

[4] N. Eagle, S. A. Pentland, and D. Lazer. "Mobile Phone Data for Inferring Social Network Structure". In Social Computing, Behavioral Modeling, and Prediction, pp. 79-88, January 2008.

[5] N. Eagle, A. Pentland. 2006. "Reality Mining: Sensing Complex Social Systems", Personal and Ubiquitous Computing, vol 10(4), pp. 255-268.

[6] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization", In Proc. 15th International Conf. on Machine Learning, pp. 144–151, Madison, Wisconsin, USA, July 1998.

[7] MATLAB, The language of technical computing, http://www.mathworks.com (last visited on January 2009.)

[8] University of Waikato, WEKA - Waikato Environment for Knowledge Analysis, Data Mining Software in Java, http://www.cs.waikato.ac.nz/ml/weka/ (last visited on January 2009.)

[9] S. Russell and P. Norvig, "Artificial Intelligence: A modern approach (second edition)". Prentice Hall International, January 2003.

[10] Nupur Bahtja, "Policy Management in Context-Aware Networks", Master of Science Thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, April 2007.

[11] R. Reichle et al., "A Comprehensive Context Modeling Framework for Pervasive Computing Systems", In Proceedings of the 8th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS), Oslo, Norway, Springer Verlag, June 2008.

[12] A. Devlic, "Extending CPL with context ontology", In Mobile Human Computer Interaction (Mobile HCI 2006) Conference Workshop on Innovative Mobile Applications of Context (IMAC), Espoo/Helsinki, Finland, September 2006.

[13] W. Dargie, "The Role of Probabilistic Schemes in Multisensor Context-Awareness", In Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07), pp.27-32, March 2007.

[14] S. Wasserman and K. Faust, "Social Network Analysis, Methods and Applications". Cambridge, UK: Cambridge University Press. 1994.

[15] G. Kossinets and D. J. Watts. "Empirical Analysis of an Evolving Social Network", Science 311, pp. 88-90. 2006.

[16] H. Ebel, L-I. Mielsch, and S. Bornholdt. "Scale-free topology of e-mail networks", Phys Rev E 66: 35103. 2002.

[17] W. Aiello, F. Chung, and L. Lu, "A random graph model for massive graphs", Annual ACM Symposium on Theory of Computing, Proceedings of the thirty-second annual ACM symposium on Theory of computing, pp. 171–180, 2000.

[18] J. M. Kleinberg, "Challenges in mining social network data: processes, privacy, and paradoxes". In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '07), pp. 4-5. 2007.

[19] A. Toninelli, R. Montanari, L. Kagal, and O. Lassila. Proteus, "A Semantic Context-Aware Adaptive Policy Model", Eighth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '07). IEEE Computer Society, pp. 129-140, 2007.

[20] T. Moses (ed.), "OASIS eXtensible Access Control Markup Language (XACML) Version 2.0". OASIS Standard, 1 Feb 2005.

[21] C. Kamienski, J. Fidalgo, R. Dantas, D. Sadok, and B. Ohlman, "XACML-Based Composition Policies for Ambient Networks", Eighth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '07). IEEE Computer Society, pp. 77-86, 2007.

[22] R. Yavatkar, D. Pendarakis, and R. Guerin, "A Framework for Policy-based Admission Control", IETF RFC 2753, January 2000.

[23] K. Verlaenen, B. De Win, and W. Joosen, "Policy Analysis Using a Hybrid Semantic Reasoning Engine", Eighth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '07). IEEE Computer Society, pp. 193-200, 2007.

[24] The Friend Of a Friend (FOAF) project, http://www.foaf-project.org/, last visited on January 2009.

[25] XHTML Friends Network (XFN), http://gmpg.org/xfn/, last visited on January 2009.

[26] I. Davis and E. Vitiello Jr, "RELATIONSHIP: A vocabulary for describing relationships between people", http://vocab.org/relationship/, last visited on January 2009.

[27] Google OpenSocial API, http://code.google.com/apis/opensocial/, last visited on January 2009.

ISSN 1653-6347
TRITA-ICT-COS-0902
ISRN KTH/COS/R--09/02--SE

www.kth.se