# Licentiate Thesis proposal

Context-addressed communication dispatch

Alisa Devlic

Royal Institute of Technology (KTH)

Primary advisor: Prof. Gerald Q. Maguire Jr.
Secondary advisor: Prof. Mark Smith
Committee member: Dr. Fredrik Kilander

15 October 2006

# Table of contents

# 1 Introduction

Communication has always been an essential part of people's lives. They would meet regularly to exchange goods, ideas, and socialize together. 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 techologies, people started to use device-mediated communication and communicate globally. With this increasing globalization, more and more people travel both for business and private purposes, and they want to use various types of mobile devices for communication to be reachable. This phenomenon has brought benefits to society due to technological developments, but people lost some of the emotional and intuitive aspects of their communication. Additionaly, there is a myriad of electronic devices in people's environment for measuring location, temperature, date and time, lighting, noise level, etc. Combining this knowledge of the environment with ambient interaction (interaction between heterogeneous computing devices) we can create new techniques to enhance people's communication experience. One way to achieve this is to make applications and systems *context-aware*, meaning that these applications use context information and react to it, by adapting their behavior according to changes in context.

## *1.1 Problem statement*

The problem to be addressed in the thesis is how to address and communicate with 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. End users have their own preferences of how they would like to receive information from different parties (regarding both communication means and device) and these preferences can change with time and the situation the user finds themself in. The challenge is to overcome these difficulties and to create and build a model that can initiate session to correct users based on these users' current context, establishes and manages context for these sessions, and facilitates exchange of information using the users' preferred communication means in the current context.

## *1.2 Contribution*

The contribution of this thesis work is to find new, innovative uses of context information and to realize a novel approach to *context-addressed communication dispatch*, meaning that information is to be sent to a subset of users whose context matches the specified context. Therefore, destinations will be identified by context instead of by network address (e.g. *all people in the meeting room*). Communication sessions will be triggered based upon context information, where the context information itself will be inferred from the situation, and not explicitly stated by the user. End users will be able to set their preferences regarding how would they prefer to receive information (e.g., as an instant message, using RSS feeds, as a VoIP call, or by file sharing) and on which device they would like to receive it (if they are using multiple devices). Their preferences can be context-based, so that the behavior will change as their current context changes. Entities called *communication dispatchers* will have the responsibility for establishing, maintaining, and terminating context sessions with the end user, as well as delivering content using the user's preferred communication means. An assumption is that a suitable means of content information collection and dissemination exists and that users will be willing to allow their context information to be used by the communication dispatchers.

## 1.3 Background

There have been a number of attempts by researchers to define context, we will examine these in section 2. However, in this paper we have adopted the following definition of context from Dey and Abowd [4]:

> *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.*

The important contribution of this definition is that the context information characterizes the situation from the perspective of an entity with regard to attributes that are application domain-specific. In this thesis we will further focus on only the context information that is relevant concerning initiating communication with an entity.

Context information will be collected from the environment through some automated means (i.e., via sensors). A context model will be used to hide low-level sensing details from applications. This technique facilitates extensibility, since applications don't have to be modified to accomodate different sources of context information and simplifies the reusability of hardware-dependent sensor code due to abstraction and encapsulation, which eases application development. Application developers should be able to simply choose what information from the set of available knowledge is relevant, then use this to determine how to initiate communication to this user.

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.*

A system is context-aware if it can 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 the application's behavior.

- *Passive context awareness*: an application presents the new or updated context to an interested user or makes the context persistent for the user to retrieve later.

With context-aware systems, applications can adapt to the context information and need less explicit input from the user. They need to make decisions based on the information available to them, to make the user perceive a better result and be more productive.

In a global sense, a context-aware system (shown in fig.1) enrolls three types of entities: *context sources* (CS), *context space*, and *context consumers* (CC). *Context sources* provide context information to the context space. *Context consumers* use information that is stored in context space to accomplish different tasks. *Context space* is an abstraction of distributed repository of context information which stores context information produced by context sources and allows it's querying by authorized context consumers.

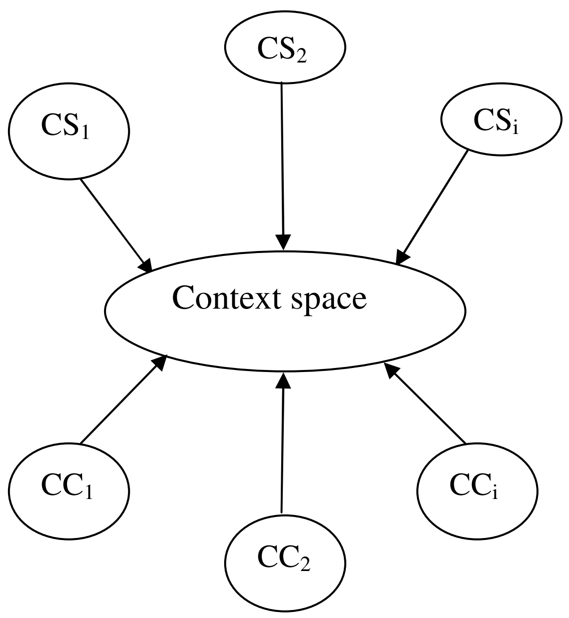
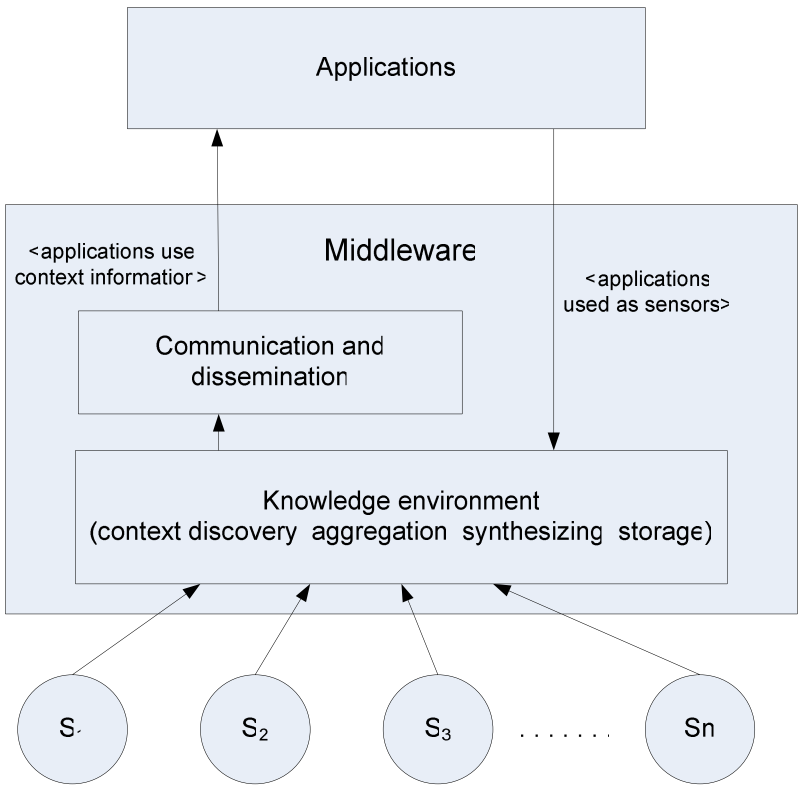


*Figure 1: Context-aware system consisting of the following entities: context sources ($CS_i$), context space, and context consumers($CC_i$)*

A context-aware system consisting of sensors ($S_i$) and middleware supporting (context-aware) applications on top of it, as depicted in fig. 2. Middleware comprises knowledge environment responsible for discovering new sources of context information, aggregating information coming from different sensors, composing existing knowledge into new concepts, and storing context information; and a component for communication and dissemination of context to the applications. Note that applications can contain sensors which provide a context source (i.e., generate context information) while consuming context information from the knowledge environment.


Applications
Middleware
<applications use context information>
<applications used as sensors>
Communication and dissemination
Knowledge environment (context discovery aggregation synthesizing storage)
S1
S2
S3
.......
Sn
*Figure 2: Context-aware system*

### 1.3.1 Sensors

Sensors are hardware or software entities that extract raw data from the device or the environment as input to the context-aware system. Sensors are usually objects that are equipped with some form of computational capacity and have simple sensing and communication facilities. Examples include: location services, a calendar application, microphones, cameras, accelerometers, motion detectors, biosensors, etc.

### 1.3.2 Knowledge environment

The knowledge environment is responsible for collecting and aggregating distributed sensor data, modelling it with higer-level abstractions, synthesizing information to create higher-level context, and storing it for later use (i.e., caching it). It should be capable of discovering and utilizing new sources of context information (sensors).

### 1.3.3 Communication and distribution of context

The component for communication and distribution of context is used to store and disseminate context information from the knowledge environment layer to applications, and vice versa (in the latter case it is assumed that applications serve as new sensors and produce new context information). Such middleware must provide an API to applications that wish to use context information.

This approach enables applications to produce intermediate results which can be used as a new source of context information, by inputting this information as new sensor information to the knowledge environment layer.

### 1.3.4 Context source

Figure 1 showed a context-aware system represented with three types of entities: context sources, context space, and context consumers. This subsection discusses the concept of a context source in the more detail, as illustrated in figure 3. The context source, as mentioned before, generates context information. Generation of context information involves functionalities already specified as responsibilities of the sensors and knowledge environment, such as: obtaining raw data, aggregating this data where necessary, creating context abstractions to be used by applications, composing information into higher-level context, and providing this information through the defined interface as context information. These functionalities are assigned to separate components, called *context providers*, *context aggregators*, and *context synthesizers*.

#### Context providers

*Context providers* utilize data from sensors. They interact with these sensors to obtain raw sensor data and model it to produce context information. They have to have a network address, a notion of time, processing capabilities, and some storage that allows them to perform their tasks. *Context providers* are usually collocated with sensors, but could also be located at a specific node in the network that is responsible for sensor data processing.

*Context providers* create higher-level abstractions of the raw-sensor data which are provided as context information to applications via API. *Context providers* use a common formalism (model) to describe context in a so called context element. They utilize a common interface to place their output (i.e. context information) in the context space.

#### Context aggregators

*Context aggregators* merge context information provided by different context providers and supply it to context synthesizers. *Context aggregators* are usually collocated with context synthesizers.

#### Context synthesizers

As context can be composed of different attributes connected with multiple logical, boolean, or other operators (implemented as functions), a context synthesizer applies these rules (operators) to the supplied data, performs reasoning, and derives new knowledge (context) from it. It should provide new (inferred) context to the system through its context provider.

A context source can be arbitrarily complex, depending on the nature of context information provided (i.e., if it requires aggregation of information from different context providers and synthesizers). Thus, it can range from having only sensors and context providers, to having a complex internal architecture, as shown in fig.3.

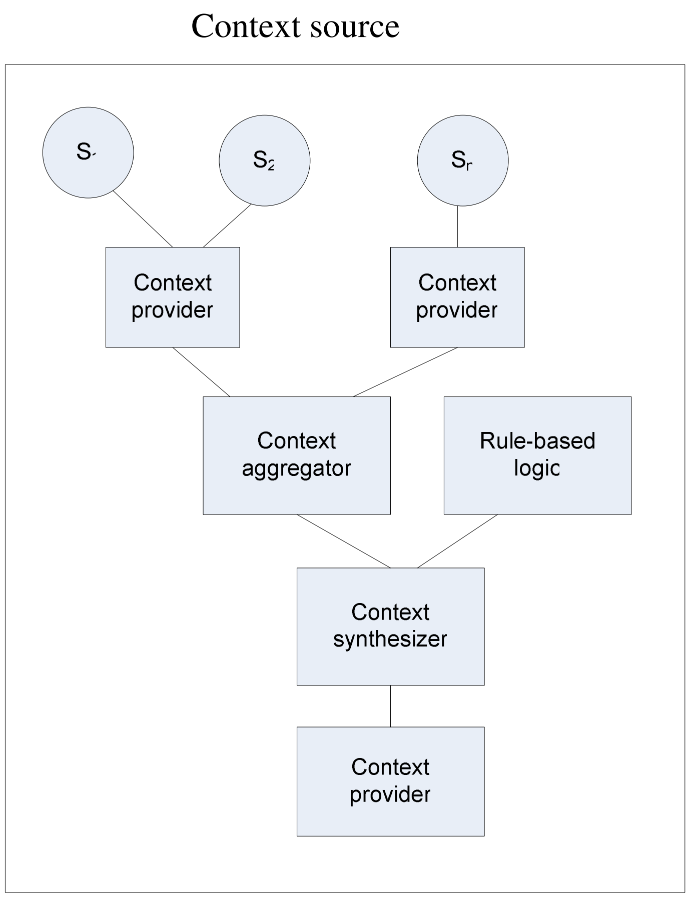


*Figure 3: Context source's architecture*

## 1.4 Use of context information

So far context has been integrated into multiple context-aware systems, creating applications limited only by someone's imagination. By looking at the existing context-aware systems we have identified the following uses of context information:

- *Context-triggered action* – actions are defined as if-then-else rules that specify how context-aware systems should behave in a certain context (e.g. if the user is in a meeting) or how to react to the occurence of some event (e.g., in case of increase of temperature in the server room alert the system manager). An example is the context-aware system in a hospital setting described by Muñoz et al. [32] which sends instant messages written by doctors when a certain set of circumstances is satisfied. The contextual elements this system is aware of include location, time, roles, and device state.
- *Contextual information and commands* – interpretation of user commands depends on the context in which they are issued. An example of contextual commands is the project *lcron* which allows for printing to the nearest printer

[34] and telephone auto-dialing that automatically selects the phone number from different locations having different area codes, handling of extensions, and telephone interfaces [34].

- *Proximate selection* – selecting an object located nearby (e.g. nearest printer) before one further away when the user wants to make use of an object of this type. An example of proximate selection is the Active Badge System [26] which forwards a user's incoming call to the nearest phone to this user.
- *Automatic contextual reconfiguration* – adding and deleting components, or changing connections between components based on current context. PARCTAB [30] is an example of a context-aware system which upon a user's entry to a room adds a connection to an instance of a virtual whiteboard associated with that room. When the user leaves the room, the connection of the user's PARCTAB to the whiteboard is broken.
- *Metadata tagging* – attaching context information to existing data to give more implicit information about objects in the system (e.g. attaching a date and time, identities of people present, and current activity to user's recorded audio clip). An example of such a system is the Context Toolkit [31] that uses context tagging of information to facilitate later retrieval.
- *Terminal adaptivity* – presenting only relevant information to the user on the small screen of a mobile device. A nice example of terminal adaptivity is dynamic adaptation of multimedia services on mobile and fixed terminal devices based on device features and presentation format (e.g., HTML, WML, SMS, E-mail, and speech-driven browsers) [35].
- *Context information filtering* – filtering available information and delivering only relevant information to the user depending on their current context. The Kimura system [29] gathers context information, converts it into the working context, and displays the user's work activities on an electronic whiteboard based on his/her working context.
- *Context sharing* – sharing of context information among communication peers. The Hydrogen project [33] enables the exchange of context information among client devices via WLAN, Bluetooth, etc. when they are in physical proximity.
- *Context-aware service provisioning* – deployment and installation of different services on the user's device based on his/her current context. Service provisioning on different wireless portable devices [36] requires tracking clients, suggests connections to the most suitable resources (either local or remote), maintains session states between client disconnection and successive reconnection (possibly in a different locality or via a different access network), supports changes in the logical set of accessible resources during service provisioning, modifies service management decisions, and enables provisioning of new location-dependent services.

### 1.4.1 Examples of context-aware systems

In this work we make several proposals to improve people's communication based upon utilization of context information. Thus we make several novel uses of context, such as helping to determine:

- *the most appropriate device* the user should use in the particular context (e.g. using a cell phone on the move is appropriate, but while in the theatre or in a meeting it is (very) inappropriate)
- *the most appropriate communication format* the user would like to deliver information in the user's current context (e.g. the user might set preferences regarding receiving e-mail, SMS, facsimile, voicemail, voice calls, instant messaging (IM), etc. depending upon the user's context)
- *change of session parameters* (e.g. if the user is currently engaged in a call, but there is another, urgent-priority, incoming call, the system could utilize information about the user's context and automatically place the original call on hold and switch to the more important call, or if there is now more bandwith available to suggest that the user could add video to their existing audio only session)
- *allocating resources* such as computing resources, bandwidth, memory (e.g. based on nodes' capacity and availability, the system could determine which node should run a specific service, store particular information, etc.)
- *destination address* for context-addressable communication (e.g. to address recipients based on their context rather than their network address or personal/corporate URI)
- *(control) access to information* based on the user's role, where the user's role is a part of his/her context

## 1.5 Research problem revised

The novel approach we would like to realize is *context-addressed communication,* meaning that information is to be sent to a subset of users whose context matches the specified context. This assumes as noted previously that a context collection and distribution system exists and that the user has elected to participate by allowing their context to be utilized. This generalizes the notion of location-based multicast. Therefore, destinations are identified by context instead of by network address, whereby either the address is determined with context (e.g. *all people in the meeting room*), or the content determines the message routing (e.g. *minutes of meeting are sent to participants*). It can be seen as a kind of *context-addressable memory* that knowing a user's context maps a specific context address to the user's SIP URI or network address. This is similar to the use of domain names to provide an abstraction for IP addresses, SIP URIs to provide an abstraction for a domain name, the use of a context address further abstracts the notion of an address.

We would like to create a new mode of communication that could be triggered by context information, where the context information itself is inferred from the situation, and need not explicitly be stated by the user. Therefore, we should create an overlay network capable of creating, modifying, adapting, and maintaining sessions according to contextual parameters. This extends the earlier goal of context-aware delivery (which concerned only the initiation of a session to deliver some content) to *context-aware session control.*

The general idea is presented in the following figure:

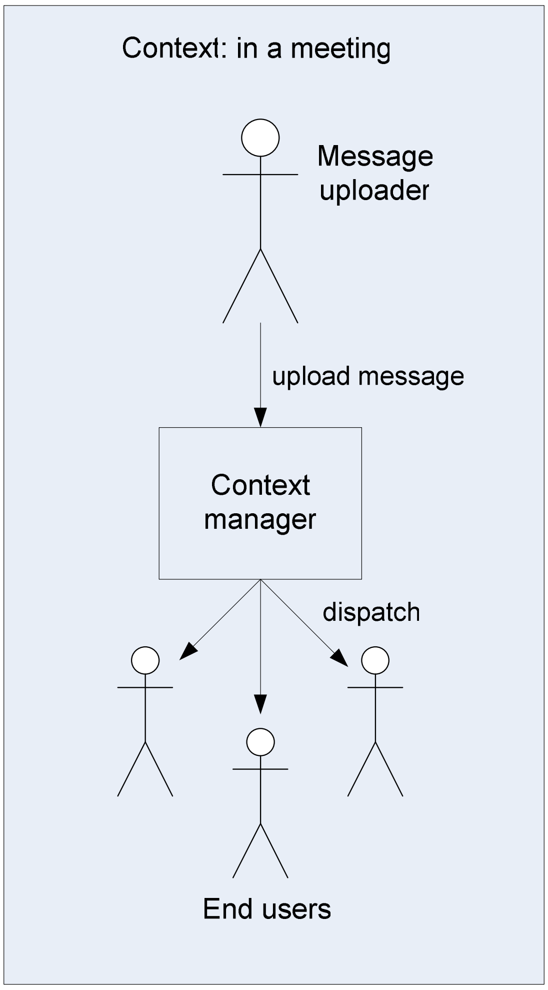


*Figure 4: Context-addressed communication*

One of the users, called message uploader, (e.g. the project manager) wants to send a message to all available users who should attend a meeting (i.e., whose context is not "in a meeting", not "on vacation", and not "with a customer"). Instead of utilizing a list of meeting participants, such a system should be able to automatically compute who should come to the meeting based on users' current context. The message can be any kind of object or media file, e.g. text, image, audio, or video. Users may set their own preferences regarding how would they prefer to receive such a message (e.g. as an instant message, using RSS feeds, as a VoIP call, or by file sharing) and on which device they would like to receive it (if they are using multiple devices). Their preferences can be context-based, so that the behavior changes as their current context changes (i.e., depending upon the time and situation they find themselves in).

Once the context manager receives the message, it must decide to which nodes it should forward this message, in what format, and by which communication means. It should also utilize information about the users' preferences (including when a user is willing to receive such a message, thus it may need to store the message and schedule it for later delivery).

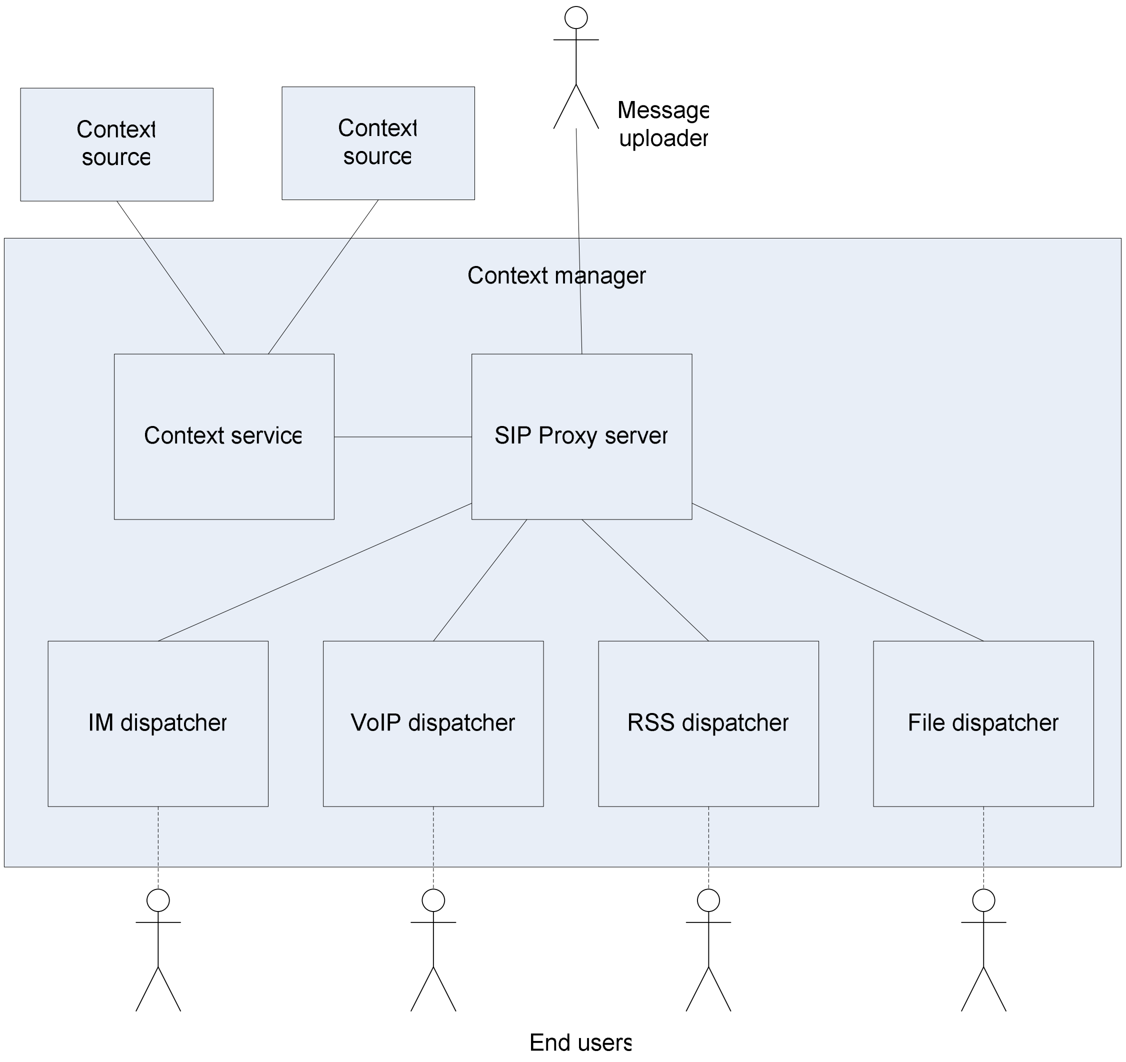


*Figure 5: Context manager components*

A context service is a distributed service that offers functions such as storing, retrieving, or modifying context parameters, and mapping them to particular nodes. For the purposes of this thesis should communicate with context sources and with a SIP proxy server, as depicted in fig.5. Context sources store (publish) their values to the context service. Users' devices can also contain context sources that provide context values to the context service. The context service associates this context with a given SIP proxy server (i.e. its SIP URI) as soon as the updated context is available.

Different types of communication dispatchers take the responsibility for establishing, maintaining, and terminating context-aware sessions with the end user, as well as delivering content using his/her preferred communication means.

Thus the context manager is comprised of components built as mobile middleware, to be able to run it on mobile devices, such as PDAs and smart phones. A secondary goal would be to determine how a future deployment could function in peer-to-peer networks. Such a future vision is shown in fig.6.

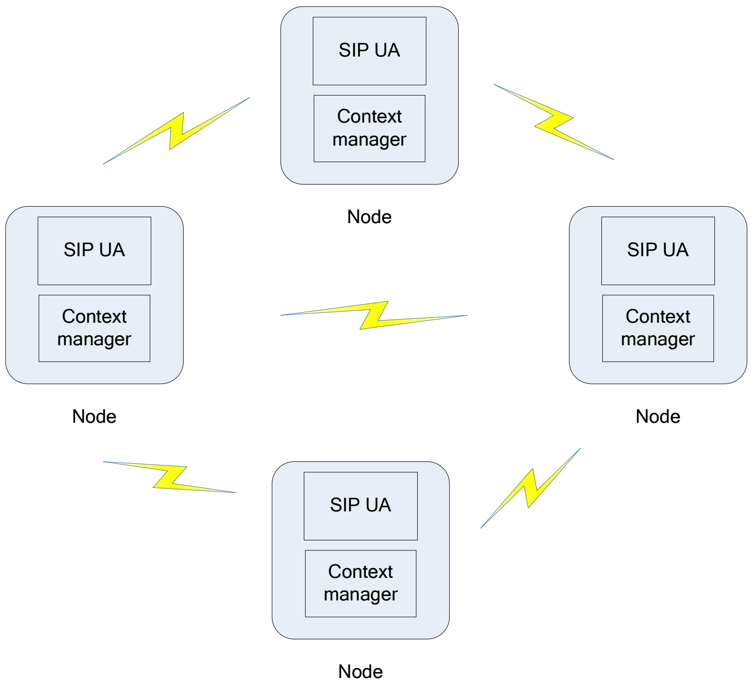


*Figure 6: Peer-to-peer context-addressable communication dispatch*

## 1.6 Reasons for utilizing SIP infrastructure

There are numerous reasons to utilize a SIP [1] infrastructure for this thesis project. First, it is a simple, scalable, text-based protocol, that offers a number of benefits including extensibility and provision for call/session control. SIP has been extended by IETF's SIMPLE (SIP for Instant Messaging and Presence Leveraging Extensions) Working Group to enhance basic protocol with Instant Messaging and Presence (IMP) functionalities. A presence system allows users to subscribe to each other's SIP UAs and be notified of changes in the user's state (note that we see context as an extension of information that is part of the user's state). Instant Messaging (IM) is defined as the exchange of content between a set of participants in near real-time. The SIMPLE extensions to the SIP protocol enable it to exchange messages inside a SIP session and provide an event package mechanism for notification of presence information. Additionaly, SIP provides some mobility support and is suitable for providing IMP services in mobile devices.

## 1.7 Summary of requirements for providing context information

In order to provide suitable context support the system needs to:

- Provide up-to-date, valid context information
- Distribute stored and collected context information
- Gather, model, process, and disseminate context information
- Detect conflicts and resolve context coming from different sources

- Cache context information for reasoning, disconnection, and post-analysis purposes
- Follow policies for authorization in order to control access to context information

## *1.8 For further investigation*

Some potential investigations:

- Determine the cost of utilizing a Domain Name System (DNS) for resolving names (either via a large local database or connectivity to other databases) and mobile IPv6 [24] for network configuration of the nodes. Mobile IP (MIP) [24, 25] will be used for managing mobility in next generation wireless networks. MIP supports mobility of nodes across different network attachment points while maintaining connectivity with other nodes on the Internet (so called corresponding nodes, CNs) by retaining a single mobile IP address. Since IPv6 does not have the limited address space problems which IPv4 has, IP addresses can be assigned for a longer time to mobile devices. Additionaly, IP mobility, in particular MIPv6, can be used to support terminal mobility for multi-access networks, networks that support both cellular access through GPRS/UMTS and for example wireless LAN access.
- Is it feasible to make a decision based on the nodes' resource information (more precisely current battery level, battery power consumption, processes running on the device, processing power, and memory status) as to which node should perform name resolution and run a context manager.
- Can context processing be based on resolving names (similar to DNS mapping of IP addresses to node names).

# 2 Related work

It has been a challenging task for researchers to find a common definition of what context is and what information is included. The first and most used type of context information is location, but over the last few years 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.

Schilit and Theimer, the pioneers of context-aware computing, stated that important aspects of context answer the following questions: where are you, whom are you with, and what resources are nearby [2].

Chen and Kotz [3] have extended Schilit's version of context and classified the context as: *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.

Schilit's classification of context has been further expanded by Dey and Abowd and defined as: *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."* [4]

The above definition includes any information about user's environment that describes the situation of the user. When dealing with context three classes of entities can be identified: *places* (e.g. rooms, buildings), *people* (e.g. individuals, groups), and *things* (e.g. physical objects, computer components). Each of these entities can be described by various context attributes.

A popular way to classify context is to describe it using different context dimensions, such as *virtual* and *physical context* [5]. The goal of context-aware system is to gather both virtual and physical context information and merge this to form a complete view of a user's environment. Virtual context encompasses information about the version of the operating system of the device the user is using, the device's interface capabilities, the available wireless connectivity, email messages sent and received by the user to abstract his/her interaction with others documents edited, ... Physical context may be the presence of another entity (user or device); location and availability of the user's colleagues; the proximity to a particular printer; information indicating user physical activity, such as is the user standing, walking, or sitting; and the current weather conditions. Most existing systems apply physical context information, especially location: the Active Badge System [26], the Cricket Location System [27], the Radar System [28], etc. The Kimura system [29] uses virtual context from the user's desktop actions to help classify, interpret, and visualize other forms of physical and virtual context, such as user's work activities and context information sensed from around the office. The system integrates physical and virtual context information into visualization of the user's disparate activities to help the user interpret and act on this available information. It automatically tracks documents attached to the user's working context, including text files, Web pages, and other application files, as well as other indications of an ongoing activity, such as email messages awaiting replies and outstanding print jobs.

Furthermore, context information can be subdivided into *static* and *dynamic context* information [5]. *Static context* refers to any information related to user's environment that is invariant. This information is mostly obtained from the user. The rest of context is dynamic and may be highly variable: for example a user's location and activity might change from one minute to the next. *Dynamic information* must be accumulated indirectly, by means of sensors and more frequently acted upon. The persistence characteristics should influence the frequency at which context information must be gathered and how long the gathered information is valid. The context information can be retrieved by requests or by activating triggers (i.e., a special form of queries that asynchronously obtain the response when a specified condition is met). Additionaly, past context information may be needed to understand the full state of the environment if the application requires such a context. It should be stored in some kind of repository in a form that can be processed by applications, or the raw data should be stored in the repository and semantic annotations must be added by some other component, i.e. middleware or the application itself. In the former case, middleware can have a dedicated subcomponent to model the information retrieved from the repository and provide it to the application on request. In the latter case, every application has to model the context information it will use. The benefit of this approach is the openess and extensibility, i.e. new context information is easily added to the system. The drawback is the absence of reusability, i.e. the same context information will be modelled by different applications.

There is a concern for the delay between production and consumption of context information, so that users might obtain outdated information. Other sources of concern regard availability (possible broken link between context producer and context consumer(s)) and reliability of context producers to generate context information. Thus, context information has to have a timestamp indicating when it was generated.

Context history includes storing the user and physical contexts over a period of time in order to establish patterns of user mobility and situational behavior in order to predict resource consumption and future context values [5]. The main concerns about maintaining historical context information are memory capacity and privacy, since most context sources are constantly providing context data and the context-aware system must decide what historical information is worth collecting and at what level of resolution. Therefore a large storage capacity is needed and it must offer the possibilities of querying and modifying the stored knowledge at a high abstraction level to both applications and sensors for updating and retrieving context information. The storage can be distributed across multiple devices or it can be implemented in a node attached to a fixed network. In the first case, efficient algorithms (in terms of low resource consumption) have to be implemented to process this information and to extract the meaningful data, whereas in the latter case the lightweight process is not necessary.

I have investigated several research projects that are looking at context-awareness from different angles (network domains and environments); each is decribed in a following section.

## 2.1 Ambient Networks

Ambient networks (AN) project [6] envisages cooperative utilization of heterogeneous networks that belong to different operators or technology domains (ranging from personal area networks (PANs) to satellite networks), on demand, transparently, and without the need for preconfiguration. End users are seen as networks as well, operating their own network of devices in homes or offices, and around their body (such as inter-vehicle networks, body area networks, and sensor networks). All of these networks are integrated into a larger system and comprise a so called *ambient networking*.

Ambient networks evolved from all-IP-based mobile networks. They clearly separate control from transport functions to differentiate between underlying network technology and applications, so that applications have a uniform view of network connectivity. Control functions are isolated and extended to form an *Ambient Control Space (ACS),* which set of functions to guarantee cooperation between networks. Control functions cooperate to implement specific control functionality. A modularization of the control space enables operators to adapt their networks' control functionality to their specific needs, while maintaining global interoperability. A second advantage of modularization of a control space is dynamic, plug-and-play integration of new control space functionality during the lifetime of the network. This improves evolvability and adaptability of deployed infrastructure.

The ambient control space provides three generic interfaces: *Ambient Network Interface (ANI)* (i.e., a network-level interface used for network composition), *Ambient Resource Interface (ARI)* (i.e., a network-level interface to access network resources), and *Ambient Service Interface (ASI)* (i.e., an interface towards the user/application plane).

The Ambient Networks project aims to create a common framework for context-awareness across all functions in the Ambient Control Space in order to adapt service availability and service delivery in heterogeneous networks and dynamic environments [7]. This framework should support collection, processing, management, and dissemination of context information enhanced with context-level agreement negotiations and support for conflict resolution. AN differentiates between user-centric context information targeting end user's applications and network context information used as a basis for control decisions about changes in the network. Network context information can be used to make networks more responsive to user needs and enhance the user's experience by making communication easier and more available [8].

AN has defined an architecture for context management, called *contextware* [37], that coordinates information exchange between different entities in the control space (mobility, media delivery, security, network management, connectivity, multi-access, and sensor network), collects and distributes context information from the environment, as well as provides information at different levels of abstraction. The context information is enhanced via context-level agreement negotiations and support for context resolution. AN believes that the exploatation of network context information will be the basis for control decisions in future networks. Therefore, an open and generic system is designed to provide context information to functions inside and outside the ACS, rather than having all functional entities working on specific methods to derive the required network information.

The overall architecture is designed around two main functional areas: Context Coordination Functional Area (ConCord FA) and Context Management Functional Area (CM FA), one interfacing to other FAs of Ambient Networks and the other for implementing the core internal operations required in the context provisioning system. The ConCord FA is the first point of contact for any context client requesting context information, negotiating the type and quality of context information, and investigating possible context information conflicts with the ConCord FA. The ConCord FA has the following functions, shown in fig.6:

- *Subscription management* – the function that helps to establish associations between context sources and context clients. It is called by both end-user services through the ASI, and by network services (to specify the context information they are interested in).
- *Negotiation management* – the function for negotiation between entities requesting context information and context providers. This negotiation might be requested because of restrictions to access to some information or the requestor might ask for information that is not readily available. To negotiate these cases, this function is called by the Conflict resolution function or by the Context Processing function. It may also cooperate with Negotiation management function in other ANs.
- *Conflict resolution* – the function to identify and manage possible conflicts that arise in the exchange of information between administrative domains. It may call the Negotiation management function if, for example, some negotiation needs to take place to resolve conflicts.

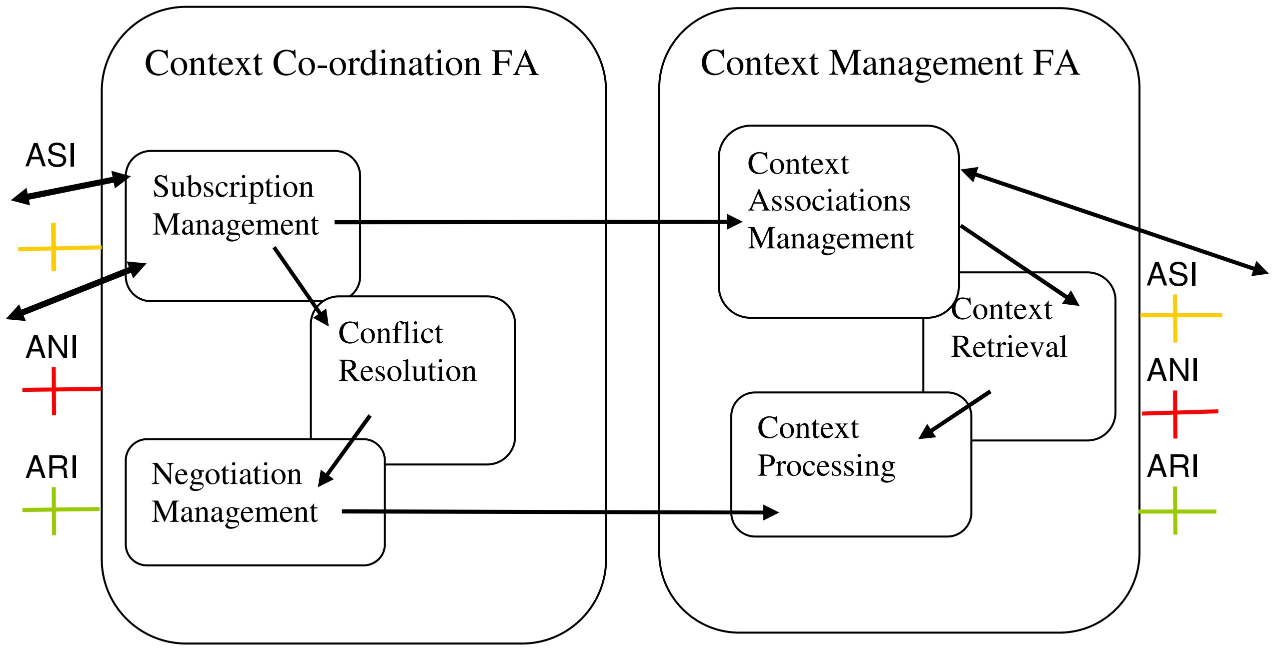


*Figure 6: ConCord FA's and CM FA's functions and relationships*

The CM FA manages context within and across domains. This involves operations, such as collection, modeling, and dissemination of context information to the interested entities, as well as managing the sharing of context information among different domains i.e., cross-domain management. The CM FA is initiated through the

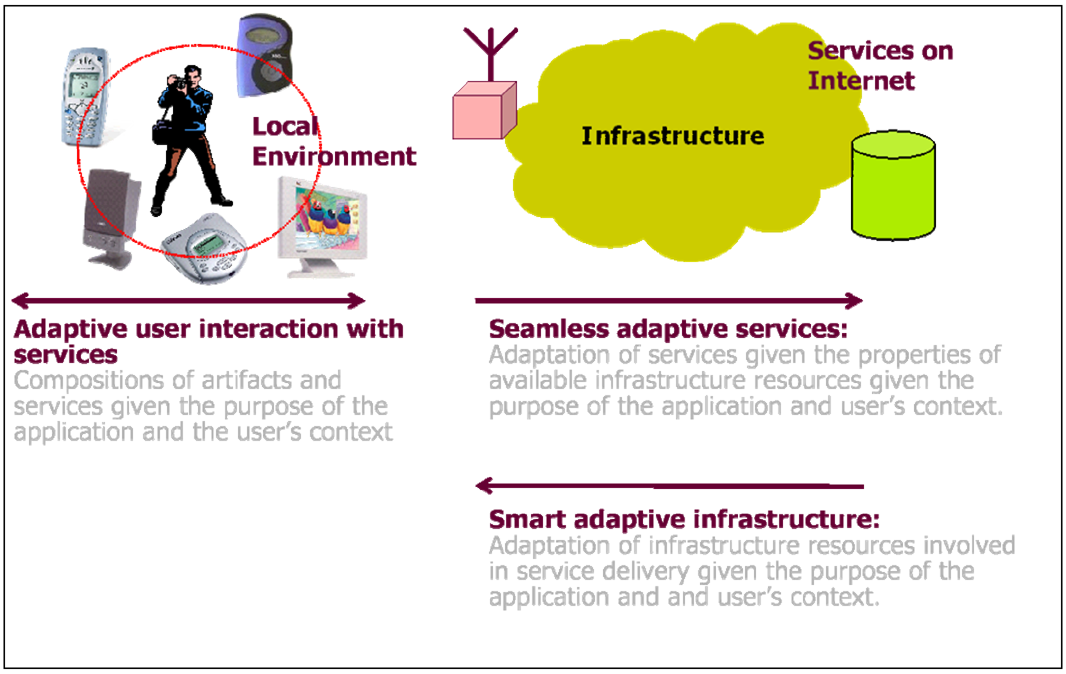
ConCord FA and is also responsible for scheduling interactions between context sources and context clients, monitoring these interactions, re-allocating channels of interaction (in case of context changes), and finally aggregating and composing context according to clients' requirements. The CM FA implements the following functions:

- *Context association management* – the function fulfills the context association requirements in order to manage associations between context clients and different context data, and to notify the subscribing clients of any changes. It calls the Context retrieval and Context processing functions to fulfill the agreed context associations.
- *Context retrieval* – the function is significant as contexts in ANs vary significantly in changing situations. It provides methods to lookup, fetch, search, and index context information in the AN space. It is called by the Context association management and Negotiation management functions. It can retrieve context information from other ANs through the ANI, while the ARI is needed to retrieve information on available network resources. Indexing is important method since it speeds up the retrieval of context information.
- *Context processing* – the function manipulates raw context information based on knowledge from the Context associations management function. It aggregates context information, composes context, and performs filtering and semantic search. It can be called by Context retrieval and Negotiation management function when needed.

The contextware architecture interworks with external systems and functions via ACS interfaces: ASI, ANI, and ARI (these include interaction with outside context sources, context clients, and contextware functions within other Ambient networks). The ASI would be responsible for conveying and receiving information relevant to the application or service enabled across the various networks, which may or may not be Ambient Networks. This information must be mapped or translated into ANI (the interface between two ACS) and ARI (the interface from ACS down to connectivity layer) in order to enable other functions simultaneously such as mobility and QoS.

## 2.2 Adaptive and Context-Aware Services

The Adaptive and Context-Aware Services (ACAS) project [9] aims to create affordable and adaptive support for users to interact with services on the Internet beyond that is possible with existing and planned cellular infrastructures. It has identified three research areas: 1) Adaptive user interaction with services, 2) Seamless adaptive services, and 3) Smart adaptive infrastructure.


Local Environment
Services on Internet
Infrastructure
Adaptive user interaction with services
Compositions of artifacts and services given the purpose of the application and the user's context
Seamless adaptive services:
Adaptation of services given the properties of available infrastructure resources given the purpose of the application and user's context.
Smart adaptive infrastructure:
Adaptation of infrastructure resources involved in service delivery given the purpose of the application and and user's context.
*Figure 6: ACAS concept [9]*

This project seeks to design mobile middleware to enable automatic configuration of services between entities without having to know the services or their properties beforehand. The services envisioned are context-aware and they are built on peer-to-peer protocols.

The project has proposed an infrastructure for acquiring context data from sensors, processing, and distributing context information, called a context information network, that will be used by application and services via a context API. The infrastructure is capable of discovering and managing context information, and utilizes policies to control access to context information. The context information network consists of application, general, and sensor layers, where the sensor layer is closest to physical or virtual sensors, and the application layer provides context information in a format ready for use by applications. The general layer is the core layer which provides context distribution and adds abstractions to the sensor data.

Context distribution is based on the SIMPLE. It allows for subscribing to and publishing presence information to interested entities. The infrastructure consists of a network of Context Management enabled Entities (CMEs) which enable sharing of context information between applications and devices (fig. 7). CMEs usually have a client and server role, because they may need to answer context information requests partly locally and partly by subscribing with other CMEs that have the missing parts of the requested information. In this chain, each CME notifies the subscriber of its local part of the subscription. CMEs also host the context repository for storing context information. The context information is stored in the repository during its validity time, unless stated otherwise.

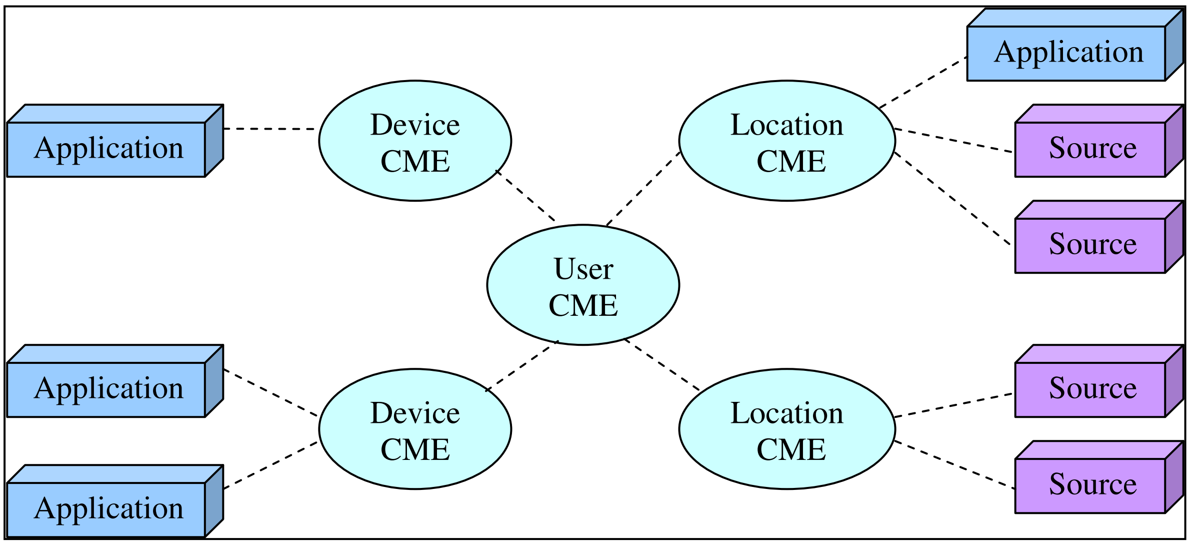


*Figure 7: Context Management Entities*

The another way to share and distribute context is via Context Data eXchange Protocol (CDXP) [10], which is more efficient than the SIMPLE for transport of context information directly between hosts, particularly on wireless links. Another important part of CDXP is that rather than streaming a series of events the requestor can provide a specification of when data is to be reported, e.g., only report a **change** in activity.

The context information itself is described in a context description language based on XML. It is contained in a context element (defined by acas namespace), an object that consists of the following attributes: id, value, datatype, unit, entity-reference, time, source URI, and source content. The XML format allows entities to apply rules for context refinement, translation to other formats, and inference. An example of a context element is:

```
<acas:contextelement id="123c">
        <acas:value datatype="string" unit="location">Home</acas:value>
        <acas:entity-reference rel="acas:dsv.su.se/k2/r7741/l"/>
        <acas:time>Fri Aug 25 00:05:21 CEST 2006</acas:time>
        <acas:source uri="uri:acas:dsv.su.se/k2/csf/apax"/>
</acas:contextelement>
```

The context refinement is possible using Tryton rule language [38], an experimental production language developed by ACAS project. It is used to apply rules to context information and to produce new (refined) context. The rule matches the value of the context information against the value stated in the rule description. If a match is found, it creates the new computed context element.

The project also looks at service discovery mechanisms in local area networks (LANs), metropolitan area networks (MANs), personal area networks (PANs), and proximity-based peer discovery, which will not be covered in this paper. There are proposals for protecting privacy through an anonymizer proxy [11] to hide the actual network address from others – while still allowing them to access some context information.

# 3 Ideas and insights

## *3.1 CPL extensions*

My initial work has lead to some preliminary results in the area of introducing context parameters into a VoIP system [13][41]. The motivation behind this work was to enable the user to control his/her incoming and outgoing calls in a more powerful fashion that currenly available. The use of context information enhanced functionalities of the existing SIP call control services and offered a user the possibility to make decisions of whether to accept an incoming call based on his/her current context. These services are implemented using CPL (Call Processing Language) scripts and their behavior is described using set of rules.

Call Processing Language (CPL) is a language used to describe and control Internet telephony services. It is described in RFC 3880 [14]. It works on top of SIP or H.323. It can be implemented on either network servers or user agents; both can usefully process and direct users' calls. CPL is an XML-based language, simple, extensible, and independent of operating system or signalling protocol. It is not Turing complete, in that it doesn't support recursions, variables, loops, or calls to external programs. These limitations are to prevent users from doing something that could take a substantial amount of time or could lead to problems for the server or other users.

CPL scripts are XML documents. The DTD (Document Type Definition) for CPL is specified in the "cpl.dtd" file available at [15]. A CPL script consists of ancilliary information about the script and call processing actions. Ancilliary information is information which is neccessary 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 when it receives 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, 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. Subactions may not be called recursively.

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.

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

| | | |
|---|---|---|
| Node: | context-switch | |
| Outputs: | context | specific context parameters to match |
| Parameters: | owner | context owner name |
| | | |
| Output: | context | |
| Parameters: | location | location of a context owner |
| | task | task status |
| | activity | activity status |

The definition of these CPL extensions for context is specified in the file "context.dtd". The part of the file that contains CPL extensions is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>

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

 <!-- Context-switch makes choices based on context information. -->
 <!ELEMENT context-switch ( context*, (not-present, context*)?,
                   otherwise? ) >
 <!ATTLIST context-switch
   owner   CDATA   #REQUIRED
 >

 <!ELEMENT context (%Node;) >
  <!ATTLIST context
   location     CDATA   #IMPLIED
   task         CDATA   #IMPLIED
   activity     CDATA   #IMPLIED
  > <!-- at least one and at most three of these attributes must appear -->
```

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 (this assumes that the other user allows this context information to be made access and the actions to be applied). Values of context parameters are specified in the user's ontology document (described in section 3.1.5). The user's context determines which script will be uploaded to the SIP proxy (here implemented using SER, explained further in text; the selection of which script to upload and how this upload occurs is described in section 3.1.2). When the context-switch node is invoked, it will match the context parameters set by the ontology with context values in the CPL script and return a decision of how to process an incoming/outgoing call (accept, reject, redirect, voicemail, etc.).

An example of a CPL script based on this extended CPL is shown below. Here Jim's SIP proxy 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>
```

For our measurements we have utilized a scalable and reliable open source SIP platform, called SIP Express Router (SER) [16] to upload and execute CPL scripts (figure 8). It can act as SIP registrar, proxy, or redirect server; however, here we have used it primarily as a SIP proxy, but we also use it as the registrar to simplify the user's specification of a new set of CPL scripts.

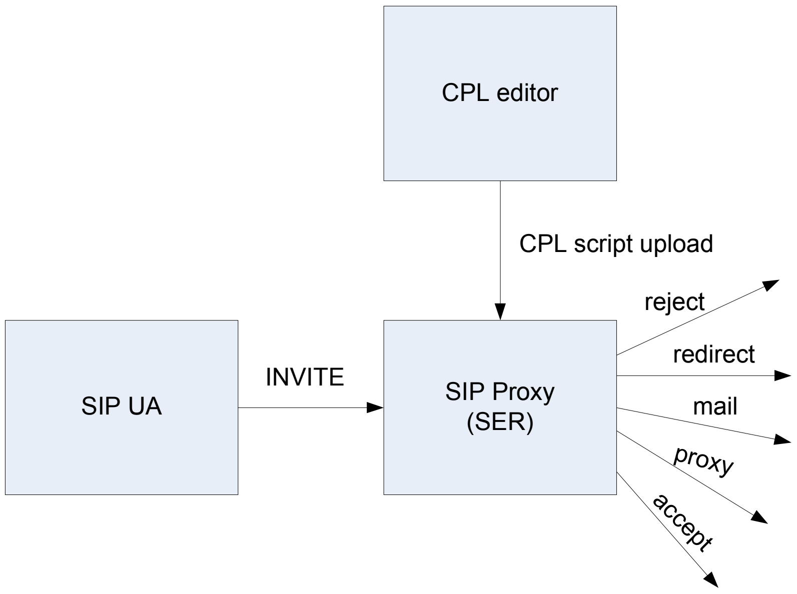


*Figure 8: SIP Express Router*

CPL scripts are uploaded using SIP's REGISTER method or with the aid of graphical programs, such as CPLEd. CPLEd [17] is an open source CPL editor developed in Java. CPLEd is used to download, upload, or remove CPL scripts from servers via HTTP and via SIP (using the REGISTER message). CPLEd can be executed as an applet embedded into a web page or as a standalone Java application. For our testing we have select the script to upload based on an application that offers a user the ability to browse for and to upload an ontology or a CPL script.

The CPL script is parsed after uploading to SER, as depicted in figure 9. It is stored in an external MySQL database [39] and is loaded and executed upon receiving incoming/outgoing call requests delivered by SIP INVITE messages. SER executes the appropriate part of the CPL script that refers to incoming or outgoing call and manages the call routing logic (e.g., accept and route the call to the callee, reject the call, redirect it to some third party, forward it to the user's voicemail, etc.).

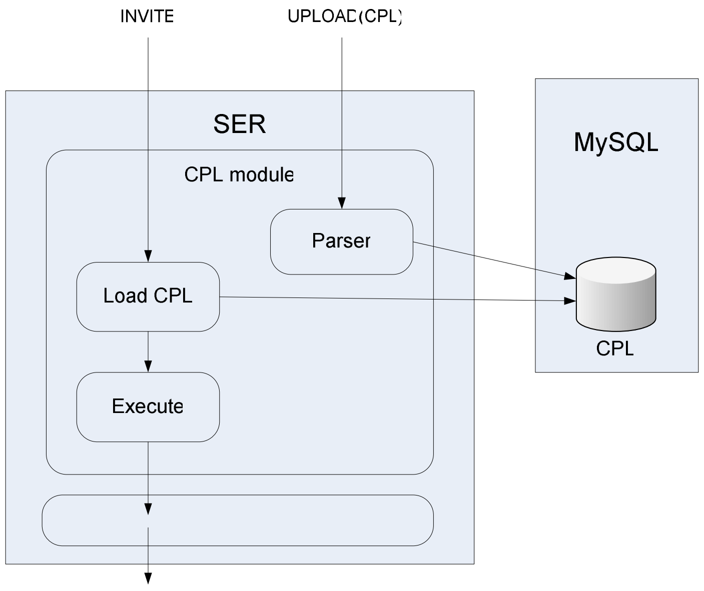


*Figure 9: SIP Express Router*

The idea of this 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. Context parameters are described in a user's ontology document. When the user wants to upload a context-based CPL script (see figure 10), 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 user information 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 user's current context. Before they are uploaded to SER, these CPL scripts are stored in the CPL repository, while ontologies reside in a context repository. Upon receiving a call or INVITE message from a SIP UA, SER loads and executes the user's current CPL script. 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 to retrieve context values set when the ontology was uploaded.

The prototype that we implemented in the lab consists of four components: a client application, a match component, wrapper, and extensions to SER's CPL-C module [40].

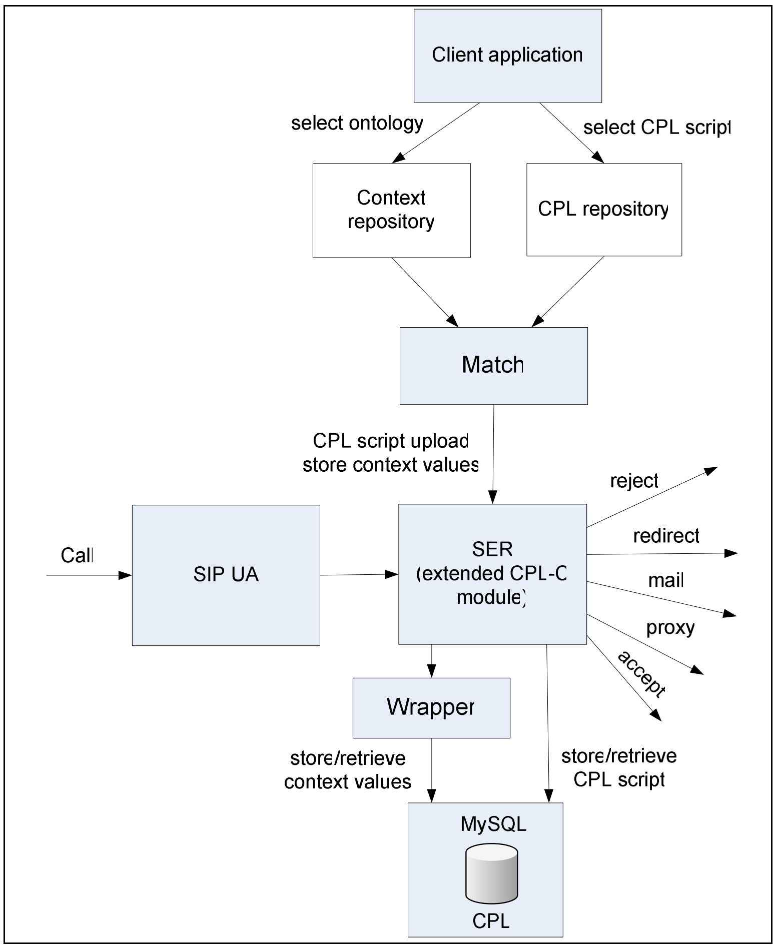


*Figure 10: Context-aware VoIP prototype*

### 3.1.1 Client application

A simple client application is used for uploading ontologies and CPL scripts. It was designed to be used from different machines and different locations, hence the preferred implementation was as an applet (see the following figures).

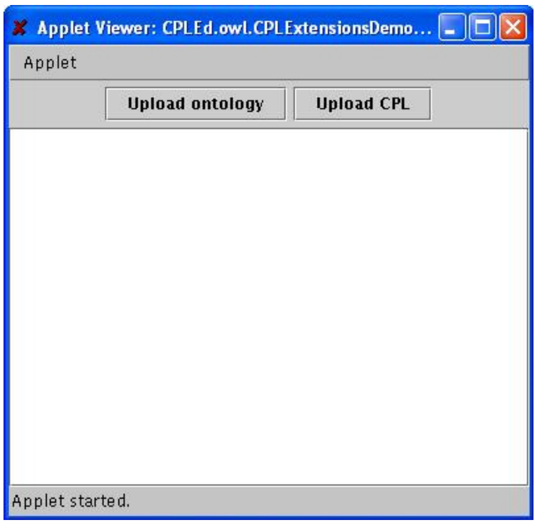
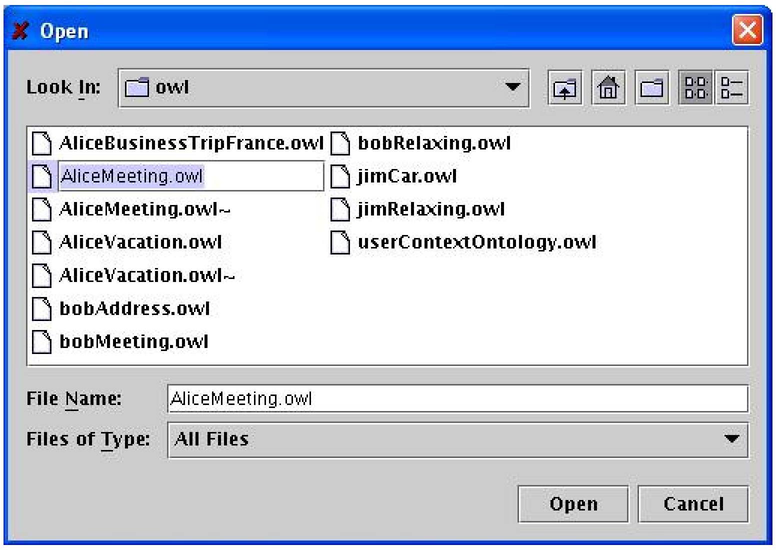


*Figure 11: CPLExtensions applet*

The user can upload an ontology or a CPL script. 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.



AliceBusinessTripFrance.owl
AliceMeeting.owl
AliceMeeting.owl~
AliceVacation.owl
AliceVacation.owl~
bobAddress.owl
bobMeeting.owl
bobRelaxing.owl
jimCar.owl
jimRelaxing.owl
userContextOntology.owl

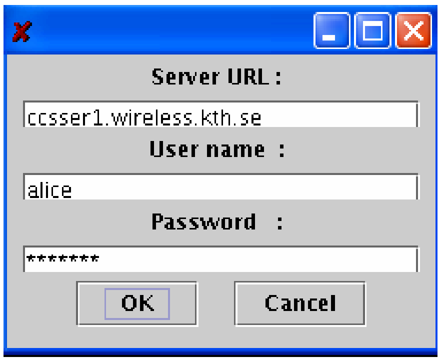
File Name: AliceMeeting.owl
Files of Type: All Files

*Figure 12: File chooser dialog*

Using an applet makes this updating possible from any computer on the Internet. A drawback of this solution is that applet code needs to reside on the same machine as SER is running on, and it can upload only a locally stored CPL script. The applet opens the file chooser dialog (see figure 12) to browse for a file to open (i.e., which ontology to load). Therefore it requires configuration of a suitable java policy on the machine where SER is running and it can run in an applet viewer, but not in a browser. A browser-based version could be implemented that used a service CGI (Common Gateway Interface) script or a PHP generated web page with a radio button to select the ontology.

### 3.1.2 Match component

The match component is responsible for parsing the uploaded 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. If the CPL script was selected by an applet, then the match component uploads it directly to SER. The match component requires a SIP domain name to be exported by SER, the username, and the password of the user to which the CPL script applies (see figure 13). SER will, upon receiving the script, store it in the database under the supplied user's credentials.


Server URL :

ccsser1.wireless.kth.se

User name :

alice

Password :

*******

*Figure 13: Data window*

A free, open source CPLEd project [17] for uploading CPL scripts via SIP and HTTP(S) was extended by adding a CPLEd.owl package to support the functions needed by our match component. This program is written in Java and its source code is provided in [13]. The applet class is CPLExtensionsDemo.java.

### 3.1.3 Wrapper

A wrapper was created to pass context values between java applications (applet and match component) and SER (which was written in C). The context parameters are stored in the database when the ontology is parsed, and retrieved by the wrapper program when the script is executed.

### 3.1.4 CPL-C module extensions

We had to modify the cpl-c module of the SER source code to support our context-switch and context node. This is explained in more detail in [13].

### 3.1.5 Context ontology

A common context ontology is important to enable all systems to cooperate. We chose to describe this ontology in OWL (Web Ontology Language) [42]. OWL provides a means to describe general concepts about basic context and it can be extended by adding domain-specific ontology in a hierarchical manner. One of its strengths is reuse of existing well-defined ontologies, rather then starting from scratch. Based on this context ontology, logic-based reasoning can be performed to check the consistency of context information and to reason using low-level, raw context information to derive high-level, implicit context.

OWL is modeled through an object-oriented approach, and a domain structure is described in terms of classes and properties. There are reasoners available that can perform checking of class consistency and consumption, and other ontological reasoning.

Formalizing all context information would be an enormous and exaustive task. Instead, the more efficient approach is to identify the most frequently used context information or an application-required context information. We have chosen to model this information as in context-dependent CPL scripts, namely: context owner, location, task, and activity. The context ontology model is shown at fig.14.

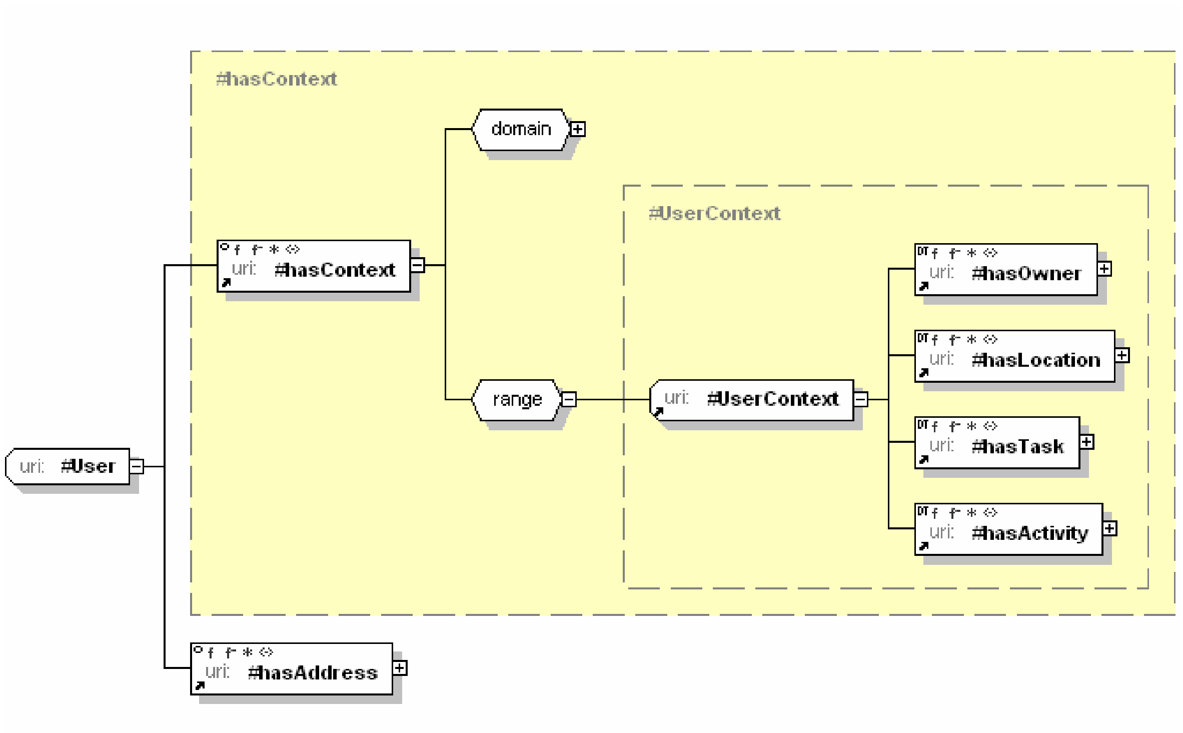


*Figure 14: Context ontology*

The above diagram can be interpreted as follows:

- A user (class User) has context (User Context)
- User's context consists of the following properties: hasOwner, hasLocation, hasTask, and has Activity
- A user has also address, but it is not relevant to this work

### 3.1.6 Measurements

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, 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 the figure 15.

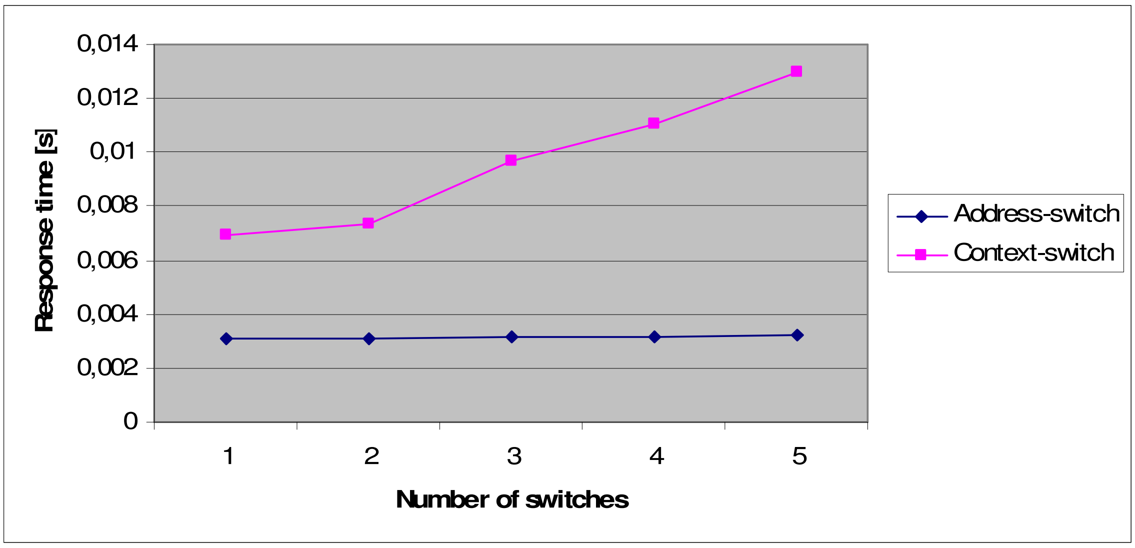


*Figure 15: Comparison of (standard and context-dependent) CPL scripts response times*

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) miliseconds, 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% reponse 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 the fig.15), 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 16 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.

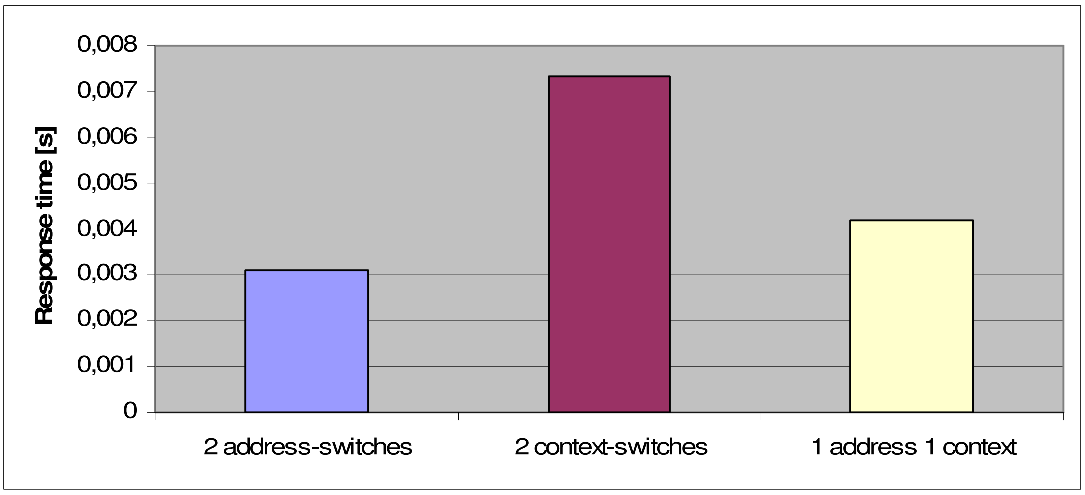


*Figure 16:Comparison of different types of CPL scripts  and thier response times*

## 3.2 Power consumption for different activities: iPAQ measurements

Computing requirements for handheld devices running on batteries are growing faster than the battery capacity [18]. Therefore these devices rely on selecting the right algorithm and behavior to lower their power consumption, along with power management techniques, thus it is very important to determine the impact on the battery lifetime when using different applications, such as using GPS positioning over a Bluetooth interface, WLAN connectivity, or audio and video playing. There are three things to consider: first battery power consumption is not constant, it varies with time as the battery voltage decreases. Second the battery capacity also varies with the load placed on the battery. Third battery life greatly varies with the usage pattern of an individual user and the configuration of the device (in this case an HP iPAQ h5550 [19] running Microsoft's Pocket PC operating system). Usage of some accessories can significantly decrease the battery life. Given an understanding of battery drain of each workload, device designers can design next generation devices with the improved techiques for minimizing power consumption and optimizing the amount of energy provided by the battery and application designers can optimize their applications for the current hardware and software – to live within their power budget.

In our measurements we tried to determine the cost of getting GPS values from a GPS receiver over a Bluetooth interface and to compare it with cost of other services, such as WLAN connectivity or video & audio playing. We measured the battery power consumption for each service and logged the remaining battery capacity every minute (the log file was kept locally on the device). Then we put all these measurements into a single graph to facilitate comparing them. We computed a least squares fit to each of curves, then calculated the slope for each curve.

Measurements have been done with an HP iPAQ Pocket PC h5550. It uses a 1250 mAh Lithium-Ion Polymer removable/rechargeable battery with approximately 10 hours of battery life under optimum conditions and approximately two hours recharge time under optimum conditions. This device also has an internal backup battery that allows the user to swap the removable battery with the system in standby mode. There is also the 2500mAh Lithium-Ion Polymer extended battery that is sold separately, however, this optional battery was not used for any of the experiments.

All measurements were started after the battery was fully recharged, and lasted until the device completely ran out of power (or in the case of the WLAN interface until the operating system shutdown the interface). Logging of battery consumption was performed using a STATUS_CLIENT application [20]. The application was configured to output a measurement every minute with the following information: current time, percentage of the battery power remaining, AC line status, available memory, and storage card free space.

For GPS measurements we used a GlobalSat BT-338 GPS reciever [21] with a Bluetooth interface. The receiver contains a SiRF III low power GPS chipset [22]. The receiver contains a replaceable battery with a large capacity (1700mAh), which enables the BT-338 to run in continuous mode up to 17 hours. When fully charged, it can operate more than 20 hours, in trickle power mode. It has a power saving function, meaning that if the Bluetooth link is not connected to any devices for 10 minutes, the receiver will automatically turn off the power. However, in our experiments it was continuously connected via Bluetooth with the iPAQ.

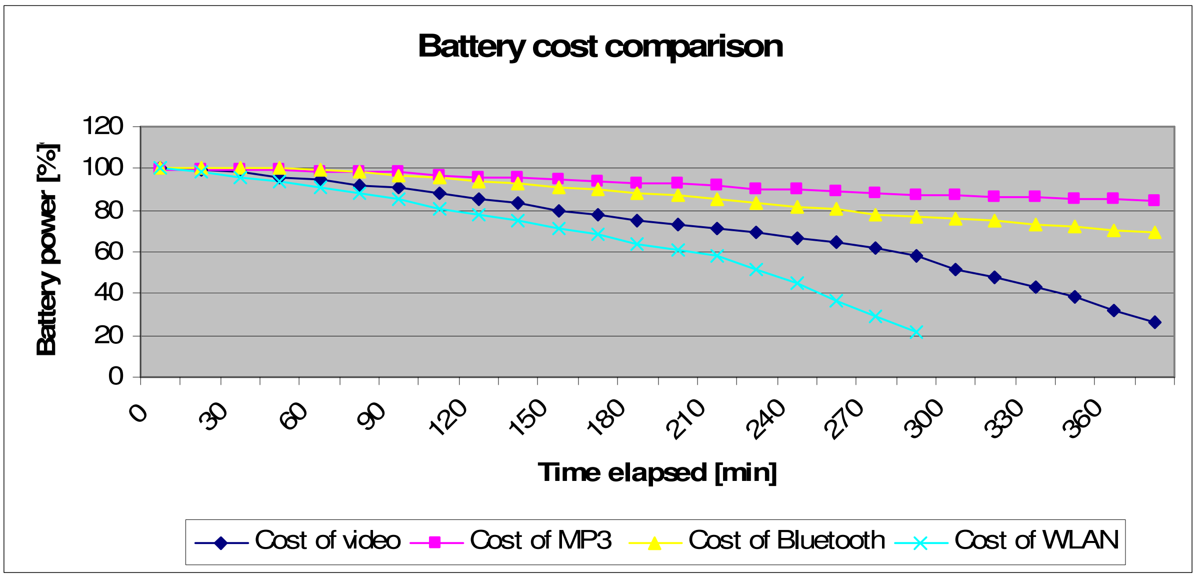


*Figure 17: Battery cost comparison*

The result was that WLAN connectivity was the most expensive operation, followed by video playing, then getting GPS values from the GPS receiver over the Bluetooth connection, and finally audio playing. We saw from the measurements (shown at fig.17) that the cost of getting GPS values from the GPS receiver over Bluetooth connection is actually the cost of Bluetooth connection itself, so when determining the system cost of GPS it is important that the GPS chipset is built into the device itself. However, building the GPS receiver into the device without adding additional battery capacity would mean much greater power consumption within the iPAQ than currently occurs. Thus there is a trade off between communication with another device with its own battery and incorporating the functionality of this other device directly into a single device.

Based on the detailed measurement and the resulting equasions presented in [23], it should be possible to predict battery power consumption for services on an iPAQ. Using this information one could try to optimize the behavior of each of these services.

### 3.2.1 Future work

As a continuation of this work, we will build a *resource allocation model* to describe the effect of running different applications on different types of PDAs. The model will consider the following resources: battery power consumption, available memory, and processing power. The model could be used to predict the actual consumption of resources before initiating a concrete service on a PDA.

We would like to apply this resource consumption model at both the device and system level. Given this model, we will design and create a *resource allocation framework* which can be further used when predicting resource consumption on PDAs by various applications (fig.18). This framework will contain a resource broker component that gathers information about resource state from different PDAs and based on the prediction of resource consumption on each of the devices, it will schedule activities for running different services on different devices, or propose other activities to help the user achieve his task (e.g. "you can share the expense report over Bluetooth, but it will cost you 10% of your remaining battery power, thus you might want to use a flash drive to transfer this file").

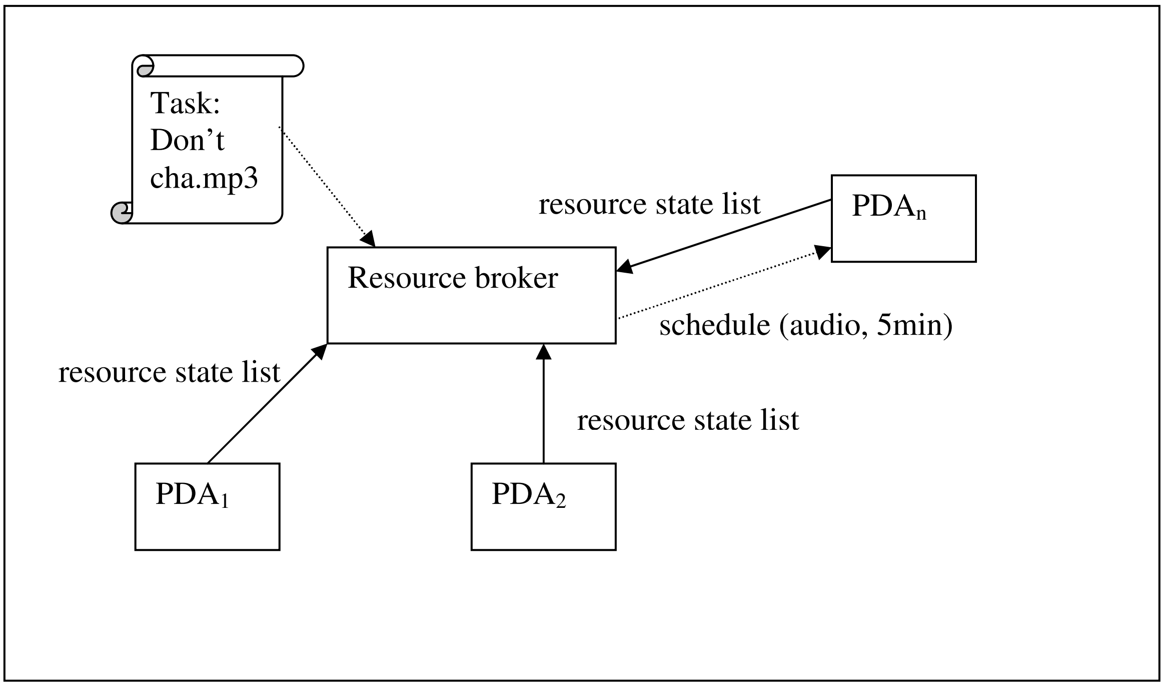
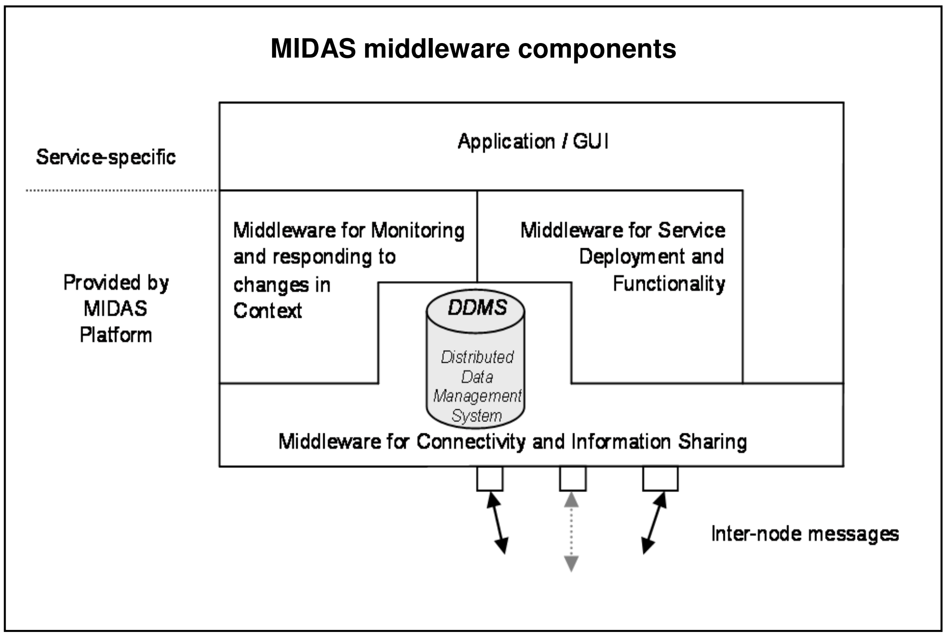


*Figure 18: Resource allocation framework concept*

## 3.3 Allignment with an EU project in which Appear Networks is involved

The basic idea is to realize communication dispatch to end-users using context-based addressing. Context-based addressing enables data packets to be sent to users based on context specified as the destination address of a packet, when those users' current context matches the context specified in the address. Context is collected from sensors in the environment, modelled in high-level abstractions, and mapped to node identifiers (SIP URIs, IP addresses, or node names). Communication dispatchers establish a communication session with end-users using their preferred communication means: instant messaging, VoIP, RSS feeds, or file sharing. We would like to understand this concept in the concept of IMS, thus extending core IMS components (CSCFs – Call Session Control Functions) with context parameters. We believe this would bring an added value to the communication services which could be built on top of it.

The results of this work should be (partly) integrated into the Middleware Platform for Developing and Deploying Advanced Mobile Services (MIDAS) project [12], as there is a requirement for middleware to support context-addressable communication dispatch. Each node should be provided with a single middleware block, that is able to adapt to changes in context and network topology (not only solving problems when it comes to connectivity failures, but also exploiting opportunities, such as higher available bandwidth). This middleware should be able to disseminate messages to different nodes and we propose the concept of communication dispatchers to be used


MIDAS middleware components
Service-specific
Provided by MIDAS Platform
Application / GUI
Middleware for Monitoring and responding to changes in Context
Middleware for Service Deployment and Functionality
DDMS
Distributed Data Management System
Middleware for Connectivity and Information Sharing
Inter-node messages

for this purpose, as depicted in fig.19.
*Figure 19: MIDAS midleware building block components*

The MIDAS project seeks to define and implement a platform to simplify and speed up the task of developing and deploying mobile applications and services. It focuses in particular on making it feasible to provide mobile services where: 1) the number of users is very large, 2) the network may need to be set up at short notice, or for a limited duration, and 3) when infrastructure is limited and some users may have to use ad hoc, peer-to-peer communication links.

All service functions will be realized by entering, retrieving, or responding to changes in data stored in the Distributed Data Management System (DDMS). Nodes update the DDMS by asynchronously exchanging short messages. These are exchanged using *one or more communications mechanisms* – depending on what other nodes are present and what communication links are available (fig. 19).

The middleware will *automatically adapt to changes in network topology*, not only to compensate for problems (e.g. failure of particular links), but also to exploit opportunities offered (e.g. when high-bandwidth connections to central machines are possible).

The middleware will provide structured mechanisms for representing and adapting to changes in user context in a distributed, mobile environment. A service lifecycle model will be defined that enables different infrastructure options to be available during *different phases of providing a service.*

The middleware introduces the concept of context-addressable messaging, meaning distributing information and events to different nodes based on the user's context information. This context-based address will be used to perform routing tasks in mobile ad hoc networks. The MIDAS concept of context-addressable messaging should also be applicable to mobile ad hoc networks (MANETs) for ad-hoc, peer-to-peer communication links, when the ability to infrastructure is limited (or too expensive). It operates at the network layer of the OSI reference model and leads to ontology based routing, i.e., routing which uses context addresses for dynamic addressing, instead of IP-based static addressing. Context addresses will describe a set of nodes matching a certain context. My approach for context-addressed communication dispatch differs from the MIDAS context-addressable messaging approach in that it will use the existing IP layer for routing information. Context-based addressing will be based on mapping a node's name and IP address to the current context assigned to that node. Mapping of nodes addresses to current context parameters is done by context service (as described in section 1.5). The proposed way of communication dispatch will not require changes in the IP layer and will not require reasoning on every node along the route, as it will be the case for MIDAS approach. Thus, it is expected to require less efforts in terms of coding, show better execution on mobile devices, and consuming less energy than the concept of context-based addressing presented in MIDAS.

## *3.4 Future work*

The CPL extensions work was carried out as a part of the ACAS project [9] to investigate the possibilities of intelligent call processing based on context parameters. The plan for the future work is to design and build a SIP proxy server enhanced with context capabilities, that will be able to control the user's incoming/outgoing calls and establish sessions between the users based on their current context and preferences regarding communication means and device used for content delivery. The SIP proxy server will be ported to a mobile device, such as PDA or smartphone.

# 4 Statement of the solution being sought

We would like to realize the concept of context-addressable communication dispatch as a part of mobile middleware. This solution would result in an overall architecture of the system and a prototype on the mobile device as a proof of concept. Performance evaluation and the cost of running the necessary services would be part of thesis work.

# 5 Plan of action

The plan for course work includes the following list totaling 30 points:

- 2G1325 Practical Voice Over IP, 5p (completed spring 2006, Prof. Maguire)
- 2G5664 Wireless and Mobile Network Architectures, 5p (completed spring 2006, Prof. Maguire)
- 2G1704 Internet Security and Privacy, 5p (completed winter 2005, Montelius)
- 2G1722 Developing Mobile Applications, 5p (completed winter 2005, Montelius)
- 6B4015 Developing applications for 3G mobile phones (Applikationsutveckling för 3G telefoner), 5p (completed spring 2006, Karlsson)
- 2E1624 Performance Analysis of Communication Networks, 5p (spring 2007, Fodor)

The plan is to organize this thesis work into several activities, of which each one should result in a publication. Activities correspond to the major research issues of the problem to be solved.

1. Context-aware VoIP system: extensions of CPL with context ontology – this has an initial publication [41]
2. Building context sources
    - identifying sensor information
    - retrieving information from software sensors
    - modelling context, comparison of different approaches for context modelling (ontology, object oriented models, key-value pairs, logic-based models, etc.)
    - synthesizing context
3. Building context service component
4. PDA measurements: predictions and optimizations for resource consumption of running services
5. Context-addressable communication dispatch

# 6 Thesis outline

A rough outline of the licentiate thesis is:
1. Introduction
2. Problem statement
3. State-of-the art of context management
    3.1. Context-aware systems and infrastructures
    3.2. Context representation, acquisition, aggregation, and synthesis
4. Context-aware VoIP system: CPL extensions with context parameters
5. PDA measurements: predictions and optimizations for resource consumption of running services
6. Context-addressable communication dispatch
    6.1. Testbed
    6.2. Measurements: performance evaluation, cost of running services, etc.
7. Conclusion
8. Future work

References

The related publications would be included as appendices.

# 7 References

[1] 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.
[2] B. N. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications", IEEE Workshop on Mobile Computing Systems and Applications, December 1994.
[3] G.Chen and D. Kotz, "A Survey on Context-Aware Mobile Computing Research", Technical report TR2000-381, 2000.
[4] 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, April 2000.
[5] K. Barett and R. Power, "State of the art: Context management", M-Zones deliverable1, http://m-zones.org/deliverables/d1_1/papers/4-01-context.pdf, May 2003, p.69-87.
[6] Ambient Networks, Integrated Project of European IST 6th Framework Programme, http://www.ambient-networks.org/
[7] A. Karmouch, R. Giaffreda, A. Jonsson, A. Galis, M. Smirnov, R. Glitho, and A. Karlsson, "Context Management Architecture for Ambient Networks", Wireless World Research Forum, Toronto, Canada, November 2004.
[8] R. Giaffreda, A. Karmouch, A.Jonsson, A. M. Karlsson, M. I. Smirnov, R. Glitho, A. Galis, "Context-aware Communication in Ambient Networks", Wireless World Research Forum, June 2004.
[9] Adaptive and Context-Aware Services (ACAS) project, http://www.wireless.kth.se/AWSI/ACAS/, Center for Wireless Systems, Royal Institute of Technology (KTH), Stockholm, Sweden, August 2006.
[10] Andreas Wennlund, "Context-aware wearable device for reconfigurable application networks", M.Sc. thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, March 2003.
[11] Wei Li, "Towards a Person-Centric Context Aware System", Licentiate thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, March 2006.
[12] MIDAS (Middleware Platform for Developing and Deploying Advanced Mobile Services) project, Strategic research project (STREP) of European IST 6th Framework Programme, http://www.ist-midas.org
[13] A. Devlic, "CPL extensions", Report for the Practical VoIP course, draft version, http://web.it.kth.se/~devlic/CPL%20extensions.pdf, May 2006.
[14] 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.
[15] CPL XML DTD draft, http://xml.coverpages.org/CPL-DTD-200201.txt, IETF, January 2002.
[16] SIP Express Router, iptel.org, http://www.iptel.org/ser/, April 2006.
[17] CPLEd, a free graphical CPL editor, http://www.iptel.org/products/cpled/, September 2002.
[18] M. A. Viredaz and D. A. Wallach, "Power evaluation of a handheld computer", IEEE MICRO, 2003, pp. 66-74.
[19] HP iPAQ Pocket PC h5500 series product overview, http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00046402/c00046402.pdf
[20] Inmaculada Rangel Vacas, "Context aware Adaptive Mobile Audio", Master thesis, KTH Microelectronics and Information Technology, 2005,

ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/050418-Inmaculada-Rangel-Vacas.pdf
[21] GlobalSat web site, Bluetooth GPS receiver BT-338 description, http://www.globalsat.com.tw/english/products_detail.php?main_id=20&p_id=74
[22] SIRFstar product sheet of SIRFstar III GPS chipset, http://www.sirf.com/Downloads/Collateral/GSC3(f)_6.20.05.pdf
[23] A. Devlic, "iPAQ measurements", Technical report, http://web.it.kth.se/~devlic/iPAQMeasurements.pdf, February 2006.
[24] D. Johnson, C. Perkins, and J. Arkko, "Mobility Support in IPv6", draft-ietfmobileip-ipv6-24.txt, http://users.piuha.net/jarkko/publications/mipv6/drafts/mobilev6.html, IETF Mobile IP WGr, June 2003.
[25] C. Perkins, "IP Mobility Support for IPv4", http://www.ietf.org/rfc/rfc3344.txt, IETF RFC 3344, August 2002.
[26] R. Want, A. Hopper, V. Falco, and J. Gibbons, "The Active Badge Location System", ACM Transactions on Information Systems, January1992.
[27] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The Cricket Location-Support System", Proceedings of the Sixth Annual ACM International Conference on Mobile Computing and Networking (MOBICOM), Boston, US, August 2000.
[28] P.Bahl and V.N.Padmanabhan, "RADAR: An In-Building RF-based User Location and Tracking System", Proceedings of IEEE Infocom 2000, Tel-Aviv, Israel, March 2000.
[29] S. Vovida, E. D. Mynatt, B. MacIntyre, and G. M. Corso, "Integrating Virtual and Physical Context to Support Knowledge Workers", IEEE Pervasive Computing, p.73-79, July 2002.
[30] B. N. Schilit, N. Adams, R. Gold, M. M. Tso, and R. Want, "The PARCTAB mobile computing system", In Workshop on Workstation Operating Systems, p.34-39, October1993.
[31] A. K. Dey and G. D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", available at: http://www-static.cc.gatech.edu/fce/ctk/pubs/HCIJ16.pdf, November 2002.
[32] M. A. Muñoz, M. Rodríguez, J. Favela, A. I. Martinez-Garcia, V. M. González, "Context-Aware Mobile Communication in Hospitals", IEEE Computer, p.38-46, September 2003.
[33] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, and J. Altmann, "Context-Awareness on Mobile Devices – the Hydrogen Approach", In Proceedings of 36th Hawaii International Conference on System Sciences, 2003.
[34] J. Heidemann and D. Shah, "Location-aware scheduling with minimal infrastructure", In Proceedings of 2000 Usenix Annual Technical Conference, San Diego, California, June 2000.
[35] J. Korva, J. Plomp, P. Määttä, and M. Metso, "On-Line Service Adaptation For Mobile and Fixed Terminal Devices", Mobile Data Management: Second International Conference, Lecture Notes in Computer Science, Hong Kong, China, p.252-263, January 2001.
[36] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli, "Context-aware Middleware for Resource Management in the Wireless Internet", IEEE Transactions on Software Engineering, p. 1086-1099, December 2003.
[37] WWI Ambient Networks, IST-2002-2.3.1.4, "Ambient Networks ContextWare", Deliverable 6.1, January 2005.
[38] C.-G. Jansson, M. Jonsson, T. Kanter, F. Kilander, G. Q. Maguire Jr., and Wei L., "Context middleware for adaptive services in heterogeneous wireless networks", In Proceedings of the IEEE 61st Vehicle Technology Conference (VTC 2005-Spring), Stockholm, Sweden, June 2005.
[39] MySQL, http://www.mysql.com, 2006.

[40] CPL-C module of SER, http://www.iptel.org/ser/component/module/cpl_c, iptel.org, 2005.
[41] A. Devlic, "Extensions of CPL with context ontology", In Mobile Human Computer Interaction (MobileHCI 2006) Conference Workshop on Innovative Mobile Applications of Context (IMAC), Espoo/Helsinki, Finland, September 2006.
[42] OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features/, 2004.