Context-aware Wearable Device for Reconfigurable Application Networks

Andreas Wennlund
Royal Institute of Technology (KTH)
Stockholm, Sweden

Masters of Science thesis performed at
Wireless Center, KTH
Stockholm, Sweden

Industrial advisor: Dr. Theo Kanter
Examiner: Prof. Gerald Q. Maguire

This version was last updated April 29th, 2003

Department of Microelectronics and Information Technology (IMIT),
Royal Institute of Technology (KTH), Stockholm, Sweden.

Abstract

Context information available in wearable devices is believed to be useful in many ways. It allows for hiding much of the complexity from the user, thus enabling simpler user interfaces and less user interaction when carrying out tasks on behalf of a user, as well as enabling network operators to provide a better interface to third-party service providers who will provide and deliver wireless services. Using the available context information from the wearable device, optimization of service delivery in wireless networks, such as setting up optimal delivery paths between two wearable devices, may be possible without using a third party to do negotiations.

In order to fully enable context-awareness, a clear model for how to sense, manage, derive, store, and exchange context information must be defined. This will then provide the platform needed to enable development of context-aware applications that can exploit the possibilities of context-aware computing. The model must take into consideration parameters such as memory usage and power and bandwidth consumption, in order to be efficient on all types of platforms and in all types of networks. It must also be modular enough to survive replacing and upgrading of internal parts.

Today little research is available about sensing context information, sensor management, APIs towards other applications, and how and how often to present context information to applications. Since context aware computing relies heavily on the ability to obtain and represent context information, sensing strategies greatly affect efficiency and performance. It is therefore of great interest to develop and evaluate models for carrying out these tasks in order to exploit the results of context awareness research. This thesis will identify and design several components of such a model, as well as test and evaluate the design, in order to be able to make conclusions to whether is lives up to the expectations stated.

In order to make the proper design decisions, a full understanding of the context-awareness research area and the goals and purposes of context-aware computing are required. To understand the entire picture is crucial to find a suitable solution. Therefore, determining an efficient sensor input and management strategy, along with a powerful and flexible API for applications, which are the goals of this thesis, fully qualifies as a significant master thesis assignment.

Sammanfattning

Information om bärbara enheters omgivning som kan göras tillgänglig i enheten, tros kunna vara användbart på många sätt. Det kan möjliggöra gömmande av komplexitet från användaren, vilket ger enklare användargränssnitt och mindre användarinteraktivitet, när utförandet av uppdrag från användaren sker, eller underlätta för en nätverksoperatör som tillhandahåller ett bättre gränssnitt gentemot en tredje part, som tillhandahåller och levererar trådlösa tjänster. Genom att utnyttja tillgänglig information om omgivningen från en bärbar enhet, kan man optimera leverans av tjänster i trådlösa nätverk, så som att hitta en optimal kommunikationsväg mellan två bärbara enheter, utan att använda sig av förhandlingar med en tredje part.

För att till fullo möjliggöra ett sådant omgivningsmedvetande, krävs en tydlig modell för att uppfatta, förfina, lagra och utbyta det data som beskriver omgivningen. Denna modell kan då utgöra en plattform som möjliggör utveckling av omgivningsmedvetande applikationer, som kan utnyttja och reagera på de data som beskriver omgivningen. Modellen måste ta hänsyn till parametrar så som minneskonsumtion och batteri- och bandbreddsförbrukning, för att vara effektiv på alla typer av plattformar och i alla typer av nätverk. Den måste också bestå av tillräckligt väl separerade moduler för att klara av byten och uppgraderingar av dess beståndsdelar.

Idag finns endast lite tillgänglig forskning om insamlandet av omgivningsdata, hanteringen av sensorer, gränssnitt gentemot mot applikationer och hur och hur ofta omgivningsdata skall presenteras för applikationer. Eftersom omgivningsmedvetenhet beror av möjligheten att införskaffa och representera omgivningsdata, påverkar strategier för att uppfatta omgivningen både effektivitet och prestanda. Det finns därför ett stort intresse i att utveckla och utvärdera modeller för utförandet av dessa uppdrag och för att utforska forskningsresultat om omgivningsmedvetande. Denna rapport identifierar och konstruerar flera komponenter till en sådan modell, samt testar och utvärderar denna för att kunna dra slutsatser om huruvida den lever upp till de förväntningar som finns.

För att kunna göra en fullgod konstruktion, krävs en ingående förståelse i forskningsområdet omgivningsmedvetande och syften och mål med densamma. Att förstå den övergripande bilden är nyckeln till en passande lösning. Konstruktion av effektiva strategier för att uppfatta omgivningen, tillsammans med ett kraftfullt och flexibelt API gentemot applikationer, vilket är målen med denna rapport, kvalificerar sig därför som ett examensarbete.

Acknowledgements

The author wishes to thank: Prof. Gerald Q. Maguire - for his positive attitude and never ending stream of feedback, Dr. Theo Kanter - for great support and engagement. Asim Jarrar and Roberto Casella - for ideas, feedback, and corporation. Gloria Dixon-Svärd- for her patience and help with administrative matters, and finally all employees at DSV sitting close to my room, for letting me in and out of my room, with the help of their keys.

Foreword

This Master of Science thesis discusses work done partly in conjunction with work done by Asim Jarrar and Roberto Casella. Thus a complete solution to addressed problems will be presented in this thesis **and** two others: “GGSN Support for Opportunistic and Adaptive Mobile Communication” by Asim Jarrar and “Reconfigurable Application Networks through Peer-2-Peer Discovery and Handovers” by Roberto Casella. Solutions presented in this thesis, are intended to be compatible with solutions presented in Jarrar’s and Casella’s work.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

1.1 Overview of the Problem Area

The increasing use of wireless infrastructure and portable technologies, such as laptop, cellular phones, and Personal Digital Assistants (PDAs), puts new demands on network infrastructures and network accessibility. Ad-hoc networks have changed the role of the user in the network. The demand for constant connectivity along with device and user mobility puts high demands on the networks, as interruption of on-going communications or transactions should be avoided.

Mobile users need to automatically discover services relevant to the their current location, without needing to have complete prior knowledge of the communication environment. In such a network, the infrastructure should be self-configuring, and should take in account the context of both users and their communication.

This rapid growth of wireless-enabled environments has increased the necessity for service discovery protocols in order to dynamically provide users with the services they wish and require. The user must be able to move among different wireless networks keeping the same user-identity while communicating via a multi-access mobile device or multiple devices. Thus, where the wearable devices are equipped with multiple communication interfaces (GPRS, WLAN, Ethernet, etc.), roaming and handovers must be possible between any combinations of types of networks. Accessing each type of network requires different routines and so making the connectivity transparent to the user becomes an attractive feature.

The user must be able to ask for services, provided by various devices (e.g. printers, cameras, public screens, etc.), or simply utilize software services, such as file, audio, or video access, which are device independent. In such a situation, a robust and self-configuring wireless network is essential to enable computational devices or users to communicate with each other both autonomously and in response to a user's request. Ad-hoc networks are characterized by dynamic links between nearby nodes, and thus the network structure may change based on the node's mobility and/or node failure. Since wearable devices should adapt dynamically to these changes, they should be smart enough that the user only notices changes in communication speed, for example even this may often be hidden by buffering data before handing over to a slower network, or routing packets via another network.

Furthermore, a user must be able to discover a service that depends on the user’s location. This enables new location based applications, such as service discovery and local peer-to-peer applications, to be performed more reliably and allows context-sensitive computing to create and exploit an user profile or environment profile by extracting an user's context and actions. The user only has to ask for the service he needs and need not know how it will be delivered. Since the user might not have complete knowledge of the network topology or the location of the desired resources, discovery should be performed automatically (i.e., with little or no interaction with the user).

Making wearable devices and making the nodes in networks context aware can be a good solution to the above problems. Gathering and propagating context information in networks can be used to make decisions enabling networks and hosts to dynamically adapt to the changing network environment and user demands.

Since the late 1980s research has studied context-awareness and explored its possibilities in different areas [35]. One area of great interest (for the future) is a network to which wearable devices are wirelessly connected, where context-awareness is believed to be able to play an important role. Context information available from the wearable devices, with the help of various sensors, and other wearable devices, along with sensor management software and analysis, are believed to be useful when making decisions about user interaction as well as enabling network operators to provide a better interface to third-party service providers who seek to provide and deliver new wireless services. Using the context information present in the wearable device, optimization of user interfaces and user interaction are possible as well as optimization of service delivery in wireless networks, for example, by setting up optimal delivery paths between two wearable devices, without using a third party to do negotiations (as in server based systems).

In order to enable this context information sharing, sensing, storing, exchanging, and analyzing context information must be efficient. Much context information can be made available in a wearable device, but probably little of it will be of interest at any given point in time. Different context will be needed in different situations. Selection of what context information to share with whom and when, becomes an important part of the solution.

Information must be exchanged between hosts, and propagated through the network (and perhaps between networks) in order for hosts to act and react to this information. At the same time, exchanging this information must leave room for communication, and must not consume too much bandwidth. Context information must also, when exchanged between terminals, be represented in such ways that all parties can understand it. A common agreement on how to identify and represent context information is needed.

Once a way of exchanging context information has been established, applications that act upon this "knowledge" can be developed, thus meeting the new demands on networks. This information will be used by devices to make intelligent decisions regarding routing of user traffic, service delivery, and application software.

Even though context awareness promises less user interaction and great possibilities for applications, there still are some difficulties that remain. Lack of full understanding of the nature of context and changes in context, as well as the lack of common conceptual models and tools to support service providers to develop context-aware applications, dampens the anticipated potential of context-aware computing. Existence of such models and tools are believed to be essential for experiencing context-awareness [22][27].

1.2 Problem Specification

In a network environment such the one described in the previous section, there are some (new) requirements on the devices, protocols, and applications involved. The aim is to make the network *user centric* [16][17], to make the details of accessing the network and roaming hidden from the user at all times. Switching between different network interfaces on a device, handovers between networks, and authentication are never exposed to the user. This section will identify and describe some "key requirements" of the above environment. Suggested solutions to some of them will be presented later in this thesis and in conjunction with the work done by Casellla [3] and Jarrar [15].

1.2.1 Keeping Connectivity and Identity Despite Mobility

For a user to be connected to a network at all times, a device equipped with at least one network interface is needed. To be able to access different types of networks, such as GPRS, WLAN, and LAN, different types of network interfaces that can hand over communication between (vertical handover) them are required [8][20][21][34]. While constantly moving around in and between networks, a user must maintain a constant identity in order to be **reachable** by any other user at all times. This identity must be independent of the user's current device, IP address and MAC addresses, since these will change as the user switches between networks interfaces, and devices. The identity should be only user dependent, thus enabling a high degree of mobility.

1.2.2 Sensing Context Information

For a device to be context aware it needs to collect information about its context - sensing. Context information available on a device (e.g. through integrated sensors) must be sensed in an effective fashion. Considerations must be taken to what context information is needed, when it is needed, and how to sense the information from the environment. What information is valuable at a given time will probably vary, and so sensing context information in an effective way, becomes a very important issue. Context information that changes quickly must be sensed more frequently than more stable information to guard against stale information. Accurate prediction of changes in context will most likely be one of the keys to efficiently allow context awareness. Since many of these sensing processes will most likely run on a wearable device powered by batteries, considerations must also be taken to power consumptions.

1.2.3 Storing Context Information

Once context information is sensed it need to be stored in order to be available to context-aware based applications. Effective storage of this context information will enable smart application to reach a high degree of context awareness. Several layers of abstraction may be needed and rules to derive more knowledge used. Good replacement policies will ensure minimal memory consumption (an issue in most wearable devices today), while also avoiding stale data. Traces of context data history may be needed as a basis for making predictions of future changes, and to recognize patterns in user behavior, etc [5].

1.2.4 Modeling Context Information, Services and Entities

In order for context information to be exchanged, there must exist a way of identifying and representing every possible type of information. Searching for a service or an entity faces the same problem. In order for two peers to understand each other, they have to have a consensual knowledge of how to identify and describe context information, services, and entities. This model must be known and accepted by all involved parts, and perhaps also exchangeable between them. The model must provide means for naming, resolving, and binding of context information, services, and entities [13]. The matter of identifying services and discovery of services are further discussed in [3].

1.2.5 Accessing Context Information

Once context information is stored, it needs to be available to any application to use as necessary. This can include carrying out a specific task triggered by a change in a sensor's value or analyzing the data to derive more knowledge. In any circumstance a standardized interface to access stored context information, will enable applications to run anywhere in the network. Context-aware applications can be downloaded to a device to carry out a task, and then be thrown away. But the same application could carry out the same task, by accessing data over the network. In order for such applications to be more general, a common interface to the context information is desirable. Otherwise applications must be developed separately, for each type of context information storage.

1.2.6 Exchanging Context Information

Not all context information needed by a context-aware application may be available on the same device, and so means for devices to exchange context information is desirable. A protocol, that takes into consideration both bandwidth and power consumption is required, since exchange of context information is most definitely not the user's principle reason for communication, and on a wearable device this is unlikely to be the most important local processing. With such a protocol knowledge can propagate through the network (and between networks), thus devices can "learn" from each other.

1.2.7 Keeping Context Information Up to Date

As wearable devices will enter and leave a network, knowledge obtained by them is not ensured to be persistent in that network. However, devices new to a network would most likely benefit from knowledge generated by other devices in the same situation earlier (consider passing knowledge on to the next generation). Information would be lost, unless some device capable of storing large amounts of context information was constantly present. This matter is discussed further in [15].

1.2.8 Service Discovery

For wearable devices to be able to access services as they enter new networks, means for service discovery are necessary. Several strategies for how to discover a service may have to be supported, since different devices may communicate through different network interfaces, or access different types of networks. Efficient ways for describing a service, locating a service, and accessing a service based on context-awareness, must also be provided. This matter is discussed further in [3].

1.2.9 Problem Statement

In order to fully enable context-awareness, a clear and efficient model for how to sense, manage, derive, store, and exchange context information must be designed. The model must allow for replacements and upgrading of its internal parts as they improve, for it to survive. It must offer a clear interface to context-aware applications capable of providing and delivering the available context information, as they might need it. This will then constitute the platform needed to enable development of context-aware applications that can exploit the possibilities of context-aware computing. The model must take into consideration parameters such as memory usage and power and bandwidth consumption, in order to be efficient on all types of platforms and in all types of networks. Wearable devices are likely to experience low bandwidth and low power at times, and must still be able to benefit from the model. It is the goal of this thesis to propose, design, and evaluate a solution to such a model that takes into account the requirements identified above in this section together with Roberto Casella [3] and Asim Jarrar [15].

2 PREVIOUS AND RELATED WORK

2.1 Background

As wearable devices are being developed at an increasing rate, becoming smaller in size, consuming less power, showing a steady increase in performance and using wireless communications, investigating *context awareness* and *ubiquitous computing* is becoming more and more interesting. The main idea of utilizing these two concepts is to produce a higher degree of intelligent behavior in wearable devices and networks, simplifying or even eliminating some interactions with the user, resulting in simpler user interfaces and better services. The applications and computing are hidden as much as possible from the user and even the devices themselves. Devices can be placed all around us providing services and resources to users that move around in the networks (and even between networks). The goal is to enable the devices and applications to anticipate the needs and actions of a user, and act upon them automatically and prepare to serve the users in advance [37]. Mark Weiser, who introduced the concept ubiquitous computing in 1991 [38], also describes *calm technology*, as the resulting effect when most of the computation and action are hidden from the user resulting in a calmer environment (from the user's perspective) [36]. Tuulari [35] accounts for Negroponte's analogies to a butler: a butler knows about all the people in a house and their whereabouts and knows beforehand what services will be expected from them. The butler is fully aware of his surroundings in the house (context awareness). By allowing a wearable device to be more aware of its surroundings, and using context -aware applications, one step is taken towards the goals of context awareness and ubiquitous computing. An important part of context awareness is obtaining context information. This thesis will focus mainly on this aspect, which involves *sensing data* and *exchanging data with other devices*.

2.1.1 Wearable Devices

This thesis uses the term wearable device to define a device that can be carried and operated with one hand (at the same time). This device may have wireless communication capabilities, but does not depend on connectivity for it to operate. A PDA is a good example of a wearable device that is fully operational without connectivity. Mobile phones, on the other hand, have functionality (e.g. calendar, phonebook, clock, games) without being connected to the network, but to fully function it depends on connectivity. In this thesis, it is assumed that a wearable device has wireless connectivity at all times. For simplicity, it is also assumed that the details of the connectivity (access points, network providers, etc.) are hidden from the user. Under these assumptions, a mobile phone would be fully operational at all times.

2.1.2 Context-Awareness in Wearable Devices

Using context awareness in wearable devices can be very advantageous if designed properly. Some discussion and definition of the term "context awareness" will provide a basis for our reasoning and enable us to draw some conclusions about the *possibilities* and *limitations* in context-aware computing.

2.1.2.1 Introducing Context-Awareness

There are many views and descriptions of the terms "context" and "context awareness" presented by many authors.

Schmidt, et al. [29] describe context as "knowledge about the user's and IT device's state, including surroundings, situation, and to a less extent, location". Here the definition of *context* requires understanding the exact meaning of the words *state* and *surroundings*, but it is a first step towards an understanding.

Dey and Abowd [6] uses a slightly different formulation: context is described as "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". Here the word *situation* must be clarified in order to reach an understanding of what context means. Dey and Abowd [6] also assumes that user interaction must be present, something that others do not.

Tuulari [35] refers to Schilit and Schmidt when describing context awareness as: "Knowledge of the environment, location, situation, user, time and current task. Context awareness can be exploited in selecting application or information, adjusting communication and adapting user interface according to current context".

2.1.2.2 Categorising Context Awareness

Tuulari [35] divides context awareness into two parts: *Self-contained context awareness* and *Infrastructure-based context awareness*, where the former implies context awareness achieved without any outside support and the latter is context awareness achieved by outside support (such as larger systems and infrastructure). Using these definitions one can say that a wearable device without any connectivity can only achieve self-contained context awareness, whereas a wearable device with connectivity can also achieve infrastructure-based context. The latter can only be achieved using wireless communication; the former can be achieved independently of connectivity.

Chen and Kotz [4] use Schilit's division of context into three categories and then add one more, to achieve a better understanding of the concept:

- *Computing context* includes network connectivity and bandwidth, communication costs and nearby resources such as printers, displays and workstations;
- *User context* includes user profiles, user location and nearby users and people; and
- *Physical context* includes lighting, temperature and humidity.

And the added category:

- *Time context*, that includes time of day, week, year, and also the season of the year.

Chen and Kotz [4] then use the following definition of context: "Context is the set of environmental states and settings that either determines an application's behaviour or in which an application event occurs and is interesting to the user".

This is followed by definition of context awareness as:

> "Active context awareness: an application automatically adapts to discovered context, by changing the application's behaviour. 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".

2.1.2.3 Summary of Context Awareness in Wearable devices

This thesis will utilize as context any information and data derivable directly or indirectly by a wearable device about its state, environment, and surroundings. This data and information can be either self-contained or infrastructure-based and can be categorized as one sees fit to serve a certain purpose. Any two wearable devices can exchange their contexts in order to update or collect context information. A wearable device can also derive more context information by looking at available information and then drawing conclusions. This context information can then be used by any application running on the wearable device (or on a server in the network that has received this information) in order to achieve a higher degree of autonomy and intelligence and to provide a simpler interface to the user or to provide better services. Note that any applications running (with or without interaction with the user) are also part of the context information.

Not all context information is relevant to all applications or to the user, at any or all times, and so one must carefully consider what context information to deliver to the applications at a specific time and therefore also what context information to discard or ignore. Consideration must be taken to how to represent and store context information as well as in what format and how it should be passed on to applications that need the information.

Context awareness is when applications (running on the wearable device or remotely) use the (stored) context information by making decisions of how to act based on that information. Context awareness can be either passive or active as discussed above.

2.1.3 Obtaining Context Information - Sensing

Context information is obtained by collecting data from sensors or by exchanging data with other wearable devices or from devices attached to the network. A sensor placed on a wearable device can provide self-contained context. Additionally sensors can be placed anywhere and their data can be transmitted to any wearable device over the network. Sensors are typically physical sensors (hardware) that measure physical conditions (temperature, acceleration, etc.), but a sensor may also be of the logical type (software) that captures more abstract information like user activity, status of running applications, and patterns of user behaviour. Sensing of the network (from the network's point of view) and network nodes is addressed in [15].

2.1.3.1 Examples of Context Information to Sense

Any information available at any time can be seen as context information. For example:

Identity	User name and user ID
Spatial information	Location, orientation, speed, and acceleration
Temporal information	Time and date
Environmental information	Temperature, humidity, air quality, light level, and noise level
Physiological measurements	Blood pressure, heart rate, respiration rate, muscle activity
Availability of resources	Battery level, display, network connectivity, and available bandwidth
Social situation	Whom the user is with, and who are other nearby users
Nearby resources	Accessible devices and hosts
Computational information	Software operated by the user, level of current user interaction, user settings and preferences, and known patterns of the user's behaviour

2.1.3.2 Examples of Sensors

There exists several techniques to obtain context information and several types of sensors are available today. For example:

- Voice recognition, finger print scanners, iris and retinal scanners, smart card readers, and other sensors for recognition and authentication of users
- GPS, IR, and RF based positioning systems (indoors), compasses (e.g. the Cricket Compass, [24]), accelerometers, and other sensors for location and orientation
- Watches, clocks, and other time sources
- Thermometers, humidity sensors, and light sensors, and other environmental sensors
- Pulse rate sensors, skin resistance, and other physiological sensors
- Voltmeters, ohm meters, current meters, and other sensors for physical measurements
- Software that monitors user interaction, network capacities and quality, localises nearby users and resources, and other sensors monitoring non physical context

By analysing the sensor data, the wearable device may conclude what the user is doing and where the user is. Then applications can use this information to reason, make decisions, and act.

Some devices have only one sensor attached to it and only one application that constantly reads the sensor and passes information on to the user (e.g. a digital thermometer). In this case a simple model for how to store and deliver sensed data to the application is sufficient. However, devices that have many types of sensors attached and that can run a variety of applications are being developed (e.g. SmartBadge [2][30][31]). This requires more complex and well-designed models for how to store and represent sensed data in order for applications to use it properly. Carefully designed protocols and APIs will enable adding and removing sensors or applications without having to redesign the entire model.

2.1.4 Sensor Management Systems and Data Fusion

A sensor management system manages, co-ordinates and integrates sensors to enable a higher level of context-awareness. The ultimate goal of the sensor management systems is to optimise the sensing of context information from a set of available sensors. The basic objective of a sensor management system is to sample the “right” sensor at the “right” time. Sensor management exists on several layers of abstraction. The lowest layer is concerned with sensor hardware to be able to sample a sensor. Higher level layers deals with scheduling strategies, decision-making and data fusion. Sensor management system designs and techniques are further discussed in [9] and [39].

Data fusion can be used to reach a higher level of context-awareness. New context information can be determined by conclusions base on already available information from several sources. Also, data from one source can be verified (or discarded) by fusion data from multiple other sources, which can be used to enhance reliability of uncertain data [27].

The presence of sensor management and data fusion is of great importance to context-aware applications, since they hide the details of accumulating context information. The applications can instead focus on acting upon the information obtained. This way, sensor management and data fusion acts on behalf of one or more context-aware applications. This thesis will not fully explore the area of sensor management and data fusion, but merely suggest that any context-aware systems solution may truly benefit from it.

2.1.5 Reconfigurable Application Network

Kanter [16][17] introduces the term *user-centric computing and communication* to describe a network where the focus is as near to the user as possible. The network provides transparent connectivity between devices in it, hiding details such as current network-operator and means of network-access. From the user’s point of view the network is invisible since the user need not know any connectivity details, but can instead focus on using the applications and services available through the network. In such a network, a service is not coupled to a network operator (as many services are today) and so the network focuses entirely on the user and how to provide the user with resources and services independent of the type of connectivity. The focus is on service delivery and on the application layer. This view of the network creates an easier to use network for both users and service and application providers since it separates the communication profiles of the wearable devices from the access networks and transport networks.

In a network as described above, nodes can cooperate in order to deliver services optimally through the network and through peer-to-peer discovery they can create ad hoc networks to route traffic to and from services. The topology of the network (either ad hoc or infrastructure based) changes as nodes are entering, leaving, and moving around, thus the network reconfigures itself as it changes in order to optimise service delivery. Context-aware wearable devices fit perfectly into this type of network, since they are able to make decisions, based on their context, about how to participate in service discovery and delivery.

2.1.6 Context-Aware Devices in the Network

In the network several context aware devices will exist and act. Some of them (carried by users) will be moving around most of the time, and some of them (such as printers, workstations, displays, and some sensors) will not be moving at all. All of them could still be considered as sources of context (sensor) data, if they make available the same interface to other devices and the network. This way, one could look at every device as a sensor platform device or a context-aware device. Even sensors attached to a moving device could be viewed separately as a sensor platform device (both the host device and other devices can use this viewpoint as well as the sensor can view its host as yet another sensor platform or context-aware device). Users will operate some devices and some will have a very high degree of autonomy. Service discovery in the network will enable devices to locate other devices and exchange context information or services with each other. Application providers and service providers will also be able to deliver better services to devices in a network by using context information, using both information received from the device as well as the discovery service. Information such as location and current connectivity can be used in order to deliver appropriate services.

2.1.7 Session Initiation Protocol (SIP)

The session initiation protocol (SIP) is a protocol developed to assist in providing telephony and conferencing services across the Internet. SIP is being defined via the Internet Engineering Task Force (IETF) standards process and is described in RFC2026 [12]. It is used to establish, change, and tear down “calls” between one or more users in an IP-based network. It is best described as a control protocol for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet multimedia conferences, voice over IP calls, and multimedia distribution. SIP supports session descriptions that allow participants to agree on a set of compatible media types. It also supports user mobility by proxying and redirecting requests to the user's current location using SIP-servers. SIP uses its Universal Resource Identifier (URI) to address other SIP users. The SIP URI is bound to a contact address by registering with location services.

A more thorough discussion of the protocol with respect to the scenario described in section 1.1, can be found in [3].

2.1.8 Mobile IP

Mobile IP [20] is a protocol that enables mobile hosts, to maintain connectivity when moving between different IP sub-networks. The protocol identifies a mobile host with a globally unique IP address, called a home address, regardless of what subnet on the Internet the host is currently connected to. A home agent (HA) that resides in the mobile host’s home network will know whether the mobile host is connected to its home network or elsewhere. When a mobile host wants to connect to a network other than its home network, i.e. a foreign network, it will need an IP address from that (sub-) network, frequently provided by a foreign agent (FA). Every sub-network has it’s own foreign agent. The mobile host receives a temporary care-of address from the foreign agent and sends a UDP packet to its home agent containing its temporary care-of-address. The home agent will then forward all incoming data to the care-of-address in the foreign network. The home agent and the foreign agent will create an IP tunnel between them for this forwarding for as long as the mobile host remains in the foreign network.

Mobile IP efficiently hides the constant changes of care-of addresses of a mobile host at the cost of longer routes for IP packets. However, higher-level applications do not need to be concerned with details of how to communicate with moving hosts. A downside to mobile IP today is that most hosts do not have a globally unique IP address, but instead a private IP address from a private IP network [33].

2.1.9 Context Widgets

D. Salber [27] states that in order to fully exploit context awareness, conceptual models and tools must be provided, to support the development of context-aware applications. A suggested solution to this uses *context widgets*, software components that provide context information to applications. Widgets can be of different types and have different attributes. Applications can register with a widget that will trigger callbacks when changes occur. Widgets have several components with different responsibilities. The lowest level interfaces to a physical sensor, and hides the lower level details of how to control the specific sensor. The middle layer is concerned with abstracting data and combining data from the lower level. The highest level coordinates the underlying components and provides the callback interface to applications.

One advantage of this design is that it clearly separates low-level details of sensor hardware and sensor values, from higher-level context information interesting to applications. Parts of a widget can be upgraded or replaced without affecting the applications [7][27]. One disadvantage of the design is that there may exist several widgets on a device, each one able to provide one type of context information to an application, and so, the application must know which widget to register with.

2.1.10 Modeling of Context Information

In order for context information to be exchanged between any two peers, a consensual model [13] to identify all types of information to be exchanged is needed. Each peer must know how to bind the identity of any information to internal representations of the corresponding information, and also to resolve internal representations of information to determine their consensual identity. Just like in natural language, when words are used to identify concepts and context is used to avoid ambiguous meaning, a well-formed structure will allow applications to do the same.

Held et al. [13] states that a model for naming and structuring of context information should be present throughout the entire process of sensing, storing, managing, and exchanging context information. Several requirements for a model that can support this, is then identified:

- The model must me **structured**. This will allow faster searching for context information since filtering is possible. Also, structured information will help avoid unambiguous interpretations of information.
- The model must be **interchangeable**. This allows for peers to exchange information about the model between themselves.
- The model must be **uniform**. Uniform representation of all types of context information will ease interpretation of information.
- The model must be **extensible**. In order to support future type of context information, the model must allow extensions.
- The model must be **standardized**. Since context information will be exchanged between different devices and systems, a standardized model will enable system independency and interoperation.

A structure for context information with relationships between different types of information, that will help distinguishing different types of information, can meet these requirements.

2.2 Related Work

Research on context awareness as well as on wearable devices began in the late 1980s [35] and has been rapidly evolving since. Today there are many research projects that develop and test context awareness, mostly by designing small systems. This section presents research in the area of context-awareness in wearable devices, conclusions drawn, and solutions suggested.

2.2.1 Active Badge

The Active Badge was developed between 1989 and 1992 at AT&T. Wearers of an Active Badge can be located in a building where the system is installed. The badge repeatedly transmits an infrared signal identifying itself. In the building, several networked sensors, each with a unique network address, are placed to receive the signals from the badges. Each sensor provides the system with information about the location of the badges whose signals it receives [1].

In terms of simple services, the Active Badge can be used to forward a phone call to the phone nearest to the wearer [4][18].

2.2.2 Xerox PARCTAB

The development of the PARCTAB started in early 1992 at Xerox PARC. The PARCTAB system uses palm-sized portable devices wirelessly connected to a network through infrared transceivers. The device is intended to be carried at all times and the location of each device is always known by the system. The device uses the resources in larger computers by interaction with applications running on these remote servers rather than executing services on the device itself [4][18][23].

2.2.3 SmartBadge

As a prototype to future smart card systems, the SmartBadge has been developed at Hewlett-Packard Laboratories. The SmartBadge measures 64x110 mm and is equipped with several sensors such as a 3-axis accelerometer, temperature sensors, humidity sensors, and light level sensors [31]. An infrared, PCMCIA, USB, and compact flash port, provides many ways for communication. An Intel Strong ARM processor enables application to run on the badge. The SmartBadge can be powered by batteries and is carried around clipped to the user's belt or chest pocket, like any badge. The SmartBadge have been used in an engineering course on mobile communication at the Royal Institute of Technology, in order to investigate the role of smart badges in a location and environment aware mobile infrastructure [2].

2.2.4 MyCampus

An agent-based environment for context aware mobile services, developed at Carnegie Mellon University was evaluated on their campus. It is implemented using smart agents running on a PDA that uses a context server to obtain context. The agents can for example suggest places to dine on campus based on the user's schedule (where their classes are), their position, and expected weather. Restaurants and weather information are obtained from the server. The user's schedule can be either on their PDA or on a server along with other relevant user data. Users can download task specific agents from the system to their PDAs at any time in order to access services [26].

2.2.5 Guide Systems

Context awareness is very useful in this type of applications and services. The user carries a wearable device aware of its position and based on it, applications shows relevant information on a screen. That information can reside either on the device itself or at a central server [4][18].

2.2.6 Prediction of Sensor Readings

Goel and Imielinski [11] suggest how to decrease power consumption by predicting sensor readings. The system controlling a set of sensor makes predictions of the sensor readings and sends them to the sensors. The sensors will then only send values if the prediction was incorrect. Correct predictions, will then decrease the amount of data sent from the sensors. Since sending data consumes more power than receiving data, this approach will be interesting when controlling sensors powered by batteries, since if prediction is successful, it will prolong the life times of (the battery for) the sensors. This approach assumes that the controlling/prediction mechanism resides on a host that does not need to be concerned with power consumption, which will not always be the case in the situation described in the introduction to this report.

2.2.7 Anne Ren's Licentiate Thesis

Ren [25] introduces a model for storing context information called a Context-aware Knowledge Base (CKB). The Context-aware Knowledge Base consists of two parts: the Context Information Base (CIB) and the Knowledge Base (KB). The Context Information Base is a standard database that stores low level knowledge such as sensor data. The Knowledge Base is a higher-level store that holds knowledge that can be derived from the Context Information Base. Context-aware applications can search the Context Information Base and Knowledge Base in order to derive more knowledge, and store that in the Knowledge Base. To access the Context-aware Knowledge Base, the widely used Lightweight Directory Access Protocol (LDAP, [41]) is used.

2.2.8 Theo Kanter's Dissertation

Kanter [16] suggests solutions for both representing and storing context information as well as what protocols to use when exchanging context information between wearable devices. His solution consists of layered components in the application layer and above, specifically an Extension Layer and a Mobile Knowledge Layer.

Context information obtained within the device as well as information exchanged with other peers is stored in an Active Context Memory (ACM). This ACM not only stores information, but can also create new information on its own by reasoning. In this way, an ACM builds up a model of the surrounding environment. This resulting model can be exchanged and thus merged with models from other peers. The ACM belongs to the Mobile Knowledge Layer.

The extensible Service Protocol (XSP) is an event based general-purpose protocol for exchanging context information between ACMs. Using XSP a wearable device can discover other peers using XSP and register with them. Whenever an event occurs (new information arrives) that information is passed on to the registered subscribers (known by the ACM). This way information is propagated to peers in a network, so that eventually all peers in a network have the same knowledge and thus know about the network and other resources in a given context. XSP is based on the Extensible Markup Language (XML, [40]) and belongs to the Extension Layer. Kanter suggests using the Session Initiation Protocol to maintain constant identity for users, and to carry XSP packets.

A great strength of Kanter's solution is that it is very flexible and allows both the ACM and the type of services searched for and subscribed to, to change without having to change the model and the protocols. Since its lowest layer protocol, SIP, can use any transport protocol, this solution does not depend on lower-layer functionality.

2.3 Prerequisites

In order to fully benefit from this thesis the reader needs to be familiar with and understand the basic concepts and fundamentals of data and computer communications, including wireless communications such as Global System for Mobile Communication (GSM), General Packet Radio Switching (GPRS), third generation cellular system (3G), and Wireless Local Area Networks (WLAN) as well as basic computer communication knowledge. Understanding the principles and functions of communication protocols, specifically the TCP/IP stack, is also assumed.

This thesis will **not** focus on the intelligent software that is part for using context information, but a good understanding of artificial intelligence (AI) and its current capabilities, will help the user understand the potential of context-aware implementations and solutions [19].

3 DESIGN

This section will present a suggested design that takes into account the key issues described in the introduction (section 1.2). It builds upon the earlier work described in section 2.

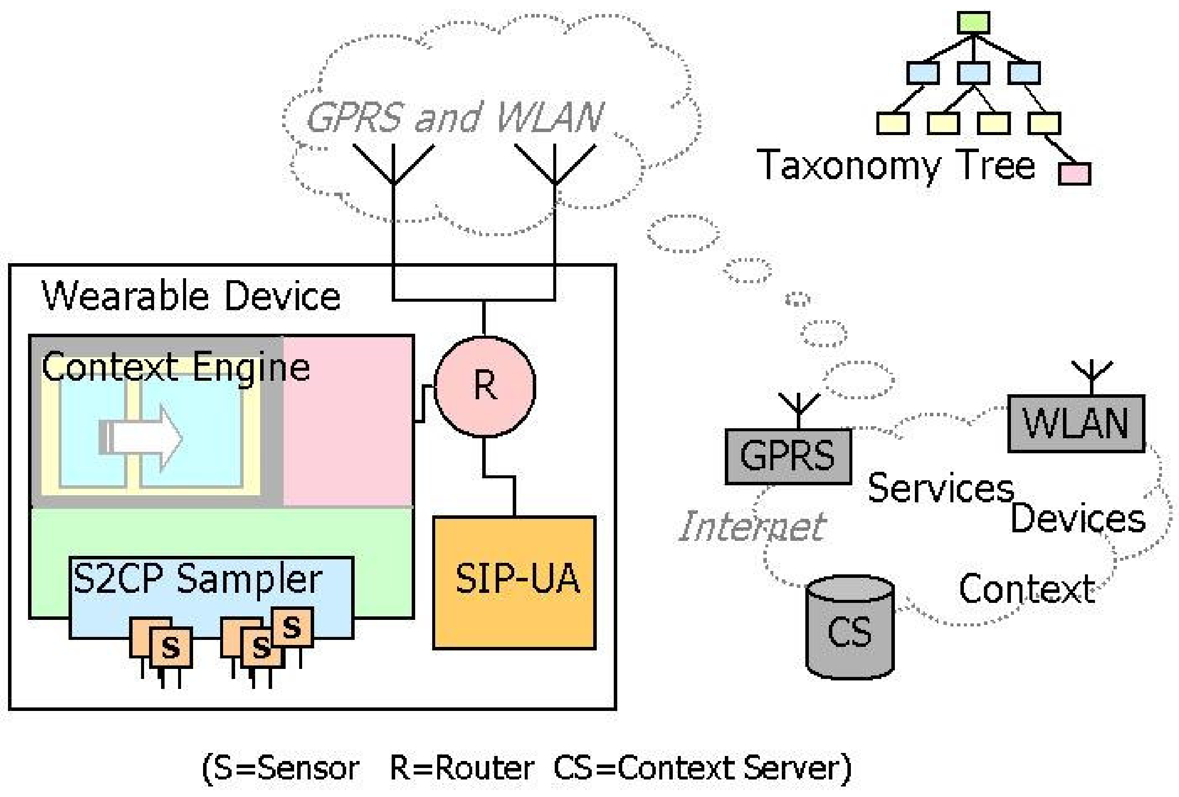

Fig. 1 Design Overview

3.1 Overview

The solution aims to be flexible and modular, in order to enable changes in existing infrastructures and upgrades and improvements of the solution itself. Considerations of power consumption (as the design is intended to be implemented on wearable devices) and low bandwidth consumption (to leave room for the core communication) were also important guiding principles.

3.1.1 Vertical Handover and SIP

Maintaining constant connectivity to networks, using multiple communication interfaces can be made possible using vertical handover. Vertical handover is beyond the scope of this thesis and is further discussed [21][34]. However, this thesis suggests using techniques of vertical handover to meet the users' demands for constant connectivity.

To provide each user with a unique id, independent of location and physical device, SIP and a SIP-UA [12] were used. SIP itself is independent of the underlying transport protocols and provides support for mobility. A SIP-session can be used to control other communication sessions between two peers and hence, adaptation to changing contexts of the peers is possible.

3.1.2 Sensor Sampling Control Protocol

To obtain sensor data, a new protocol, the Sensor Sampling Protocol (S2CP, details in section 3.2), is proposed and designed. It separates low-level details (such as reading physical sensors) from the more abstract level, which makes decisions of when to sample each particular sensor. This protocol allows easily adding and removing sensors to a device, without affecting other components in the solution, as well as allowing for replacing the abstract part of context information gathering with alternative algorithms. The two endpoint-processes of a full S2CP implementation consists of the S2CP Sampler and the S2CP Controller.

3.1.2.1 S2CP Sampler

S2CP Sampler handles the low level, device dependent part of gathering context information. The sampler will handle sensor sampling for a specific set of sensors on a specific device and deliver data to the S2CP Controller using the Sensor Sampling Control Protocol. Several S2CP Samplers may run on the same device.

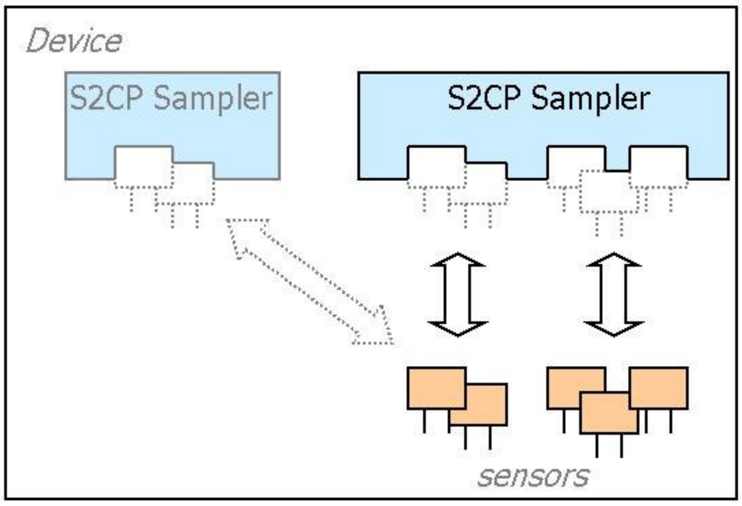

Fig. 2 S2CP Sampler

3.1.2.2 S2CP Controller

The S2CP Controller handles the abstract device independent part of gathering context information. The controller will control S2CP Samplers that have announced their presence (most likely samplers on or close to the same device). The S2CP Controller decides when to issue sampling requests and what sensors (or type of sensor) to sample.

The Sensor Sampling Protocol provides a clear separation of low level and high-level computation. It allows for changing of S2CP Controllers as they become smarter or smaller, and similarly S2CP Samplers may be replaced. Attaching new sensors to a device only requires starting up a new S2CP Sampler capable of sampling the new set of sensors and will require no modification of the S2CP Controller.

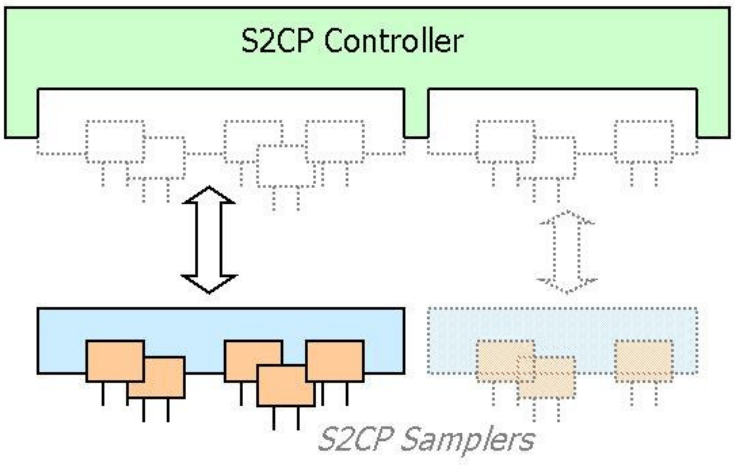

Fig. 3 S2CP Controller

3.1.3 Context Storage and Context Storage Access

Gathered Context information will be stored in a Context-Aware Knowledge Base (CKB) as described by Ren [25], since it provides more levels of abstraction of knowledge. In order for different application to be able to access the Context-aware Knowledge Base (sometimes over the network), a lightweight and simple interface is provided using the Lightweight Directory Access Protocol (LDAP, [41]). Standardized protocols are preferable due to their widespread availability. By specifying an access interface to the CKB, context-aware applications can be made more device-independent. They can be downloaded to a device to do a specific service and then be thrown away.

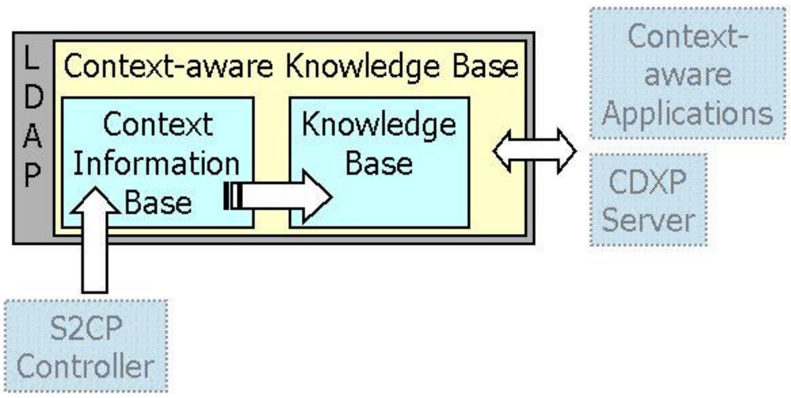

Fig. 4 Context Storage and Context Storage Access

3.1.4 Modeling of Context Information

To enable unambiguous naming and identification of context information, services, and entities, the use of a taxonomy tree is proposed. Relations between context information are represented by how nodes in the tree are connected. Each level down the tree moves towards more and more specialization. Context information, a service, or an entity, can be identified by referencing the correspondent not in the tree (e.g. by specifying the path from the root node of the tree). By using XML to describe the structure, nodes can be added to the tree and then distributed to other peers, thus allowing extensibility. The details of describing and exchanging such a tree is out side the scope of this thesis. However, its functionality and presence is assumed in the overall solution, and it meets the requirements for a sufficient model of context information as identified by Held et al. [13] and discussed in section 2.1.10. Fig. 5 shows an example of a taxonomy tree for context data. Both CDXP and S2CP use this model to identify context information. This way, no internal mapping is needed, as context information moves between applications.

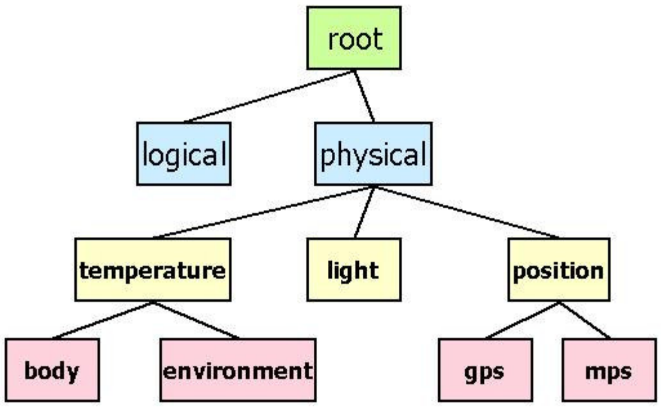

Fig. 5 Taxonomy Tree

3.1.5 Context Data eXchange Protocol

Even though LDAP allows for searching for context information, it is request-response based and this would require the requester of information to be fairly active and to constantly (or at some interval) look for updated information. This may result in unnecessary traffic if the requests are made over the network. However, the new and designed Context Data eXchange Protocol (CDXP), solves this. It is subscription-based, thus the responder will send notifications to the requester when context information is updated. Since the responder is more likely to know or to be able to predict the behavior of a specific source of context information (maybe by analyzing context information other than what the requester requests), this will simplify the interface to a requester and reduce use of network resources.

Another solution is to deliver the notifications using the functionality of SIP, as data changes, and use this to trigger an LDAP request to retrieve the new data. However, this goes against the aim for modularity, since the solution will then rely on or depend on integration of SIP, LDAP, and an application that handles the triggering of LDAP requests on a device. Using CDXP instead of a combination of SIP and LDAP, enables a more modular solution. Also, a full CDXP implementation is much smaller than a SIP-UA implementation.i

3.1.6 Context Engine

Sensor management will be handled by integrating the S2CP-Controller with a CDXP-Server and a simple Context Information Base (from the CKB), forming a Context Engine. The sensor manager can be implemented to do data fusing, prediction, and scheduling in an efficient manner since it will be aware of both what sensors are available and what context information is needed by subscribing CDXP-clients. This enables taking advantage of multiple sources of data to do a better job in deciding what data to send and when to send it. For example, it may use acceleration data to determine when (and how often) to sample other sensors for new data. It may even use the acceleration data to decide how often to sample acceleration data.

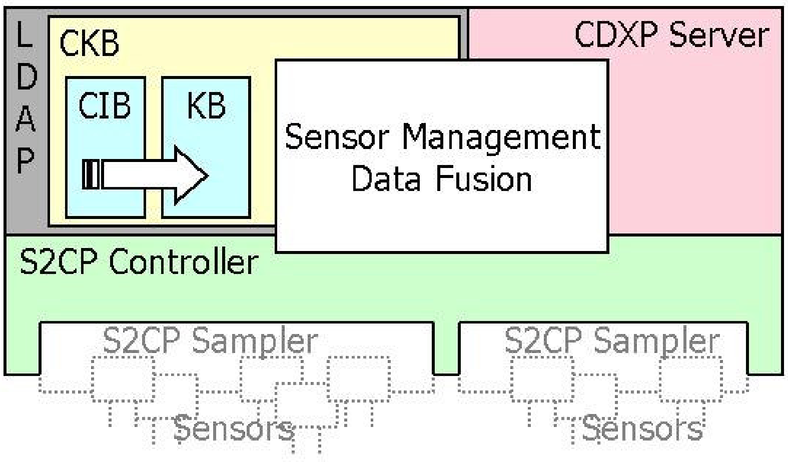

Fig. 6 Context Engine

3.1.7 Context Server

In order to keep context information up to date in a network, by accumulating context information generated by devices in a network, and to enable that context information to be passed on later to devices in the same network, the use of a Context Server, as described in [15], is proposed. The purpose of the Context Server is to distribute knowledge, using the Context Data eXchange Protocol, in order to enable hosts in an access network to make intelligent decisions regarding routing, service delivery, and context knowledge management. The hosts can be either mobile or fixed, service providers, or even another context server.

There should be at least one context server in each access network. This server will have several links and interfaces to different network nodes, and to other servers that are connected to various databases that already exists. The main purpose of a context server is to retrieve data from other access networks and to use this data to support mobile users. The context server will start exchanging context data with other hosts as it becomes aware of their presence (i.e. when they enter the network or area where the context server exists), in order to accumulate and distribute knowledge generated at the hosts.

The context server also tries to maintain knowledge about the network that the hosts cannot provide, by communicating with different access points in the network. These access points can in turn subscribe to information from the context server in order to dynamically reconfigure the network and adapt to changes in the network. Using this design, the context servers can be used to get information about neighboring networks, assist during handovers, locate other hosts, finding suitable access points, learning about network quality of service, and much more.

3.1.8 Service Discovery Using SIP

For wearable devices to be able to access services as they enter new networks, service discovery using SIP is proposed. The purpose of the discovery procedure is to allow users to discover services that are offered by other hosts in the network. Service discovery messages will be carried in SIP packets and SIP will be used to contact other peers during the discovery procedure. Hosts can register their available services with the context server. Upon discovering a service, the context server may be queried in order to find hosts offering this service. If the location of the service is already known, a unicast discovery is sent. If the discovery is in a small area network, multicast discovery can be used. Based on the situation, the discovery application performs the appropriate action. Once a service is located, the service's host is contacted directly. Services and entities are represented using a taxonomy tree such as described in section 3.1.4.

This type of service discovery will require a SIP-UA with extended service discovery functionality on each host. It relies on SIP functioning in all types of networks, including GPRS, and that SIP can be used despite the presence of private network addresses and network address translation (NAT). Each host must be able to communicate using SIP on all its different network interfaces.

Details of the service discovery using SIP and how to enable SIP communication using private network addresses and GPRS are further discussed in [3].

3.2 Sensor Sampling Control Protocol

In order to control the sampling of data from sensors on a device, a new protocol, the Sensor Sampling Control Protocol (S2CP) is proposed and designed. Its purpose is to separate reasoning about context data at a more abstract layer from details about how to sample the data from specific hardware (sensors). The protocol uses its own timeouts and acknowledgements to be able to run over both UDP and TCP. This allows devices to use UDP, which will result in fewer packets being sent, thus decreasing power consumption.

S2CP allows a clear and simple separation of hardware dependent code and more general hardware independent code. Attaching a new (set of) sensor(s) to a device only requires adding a small application for reading values from this new (set of) sensor(s).

For the S2CP to be fully modular, a means to identify types of sensors as well as to represent sensor values is required. Since a sensor value is context information, S2CP will use the same taxonomies as CDXP. This will also simplify the design of the Context Engine, since no internal translation from CDXP to S2CP is needed.

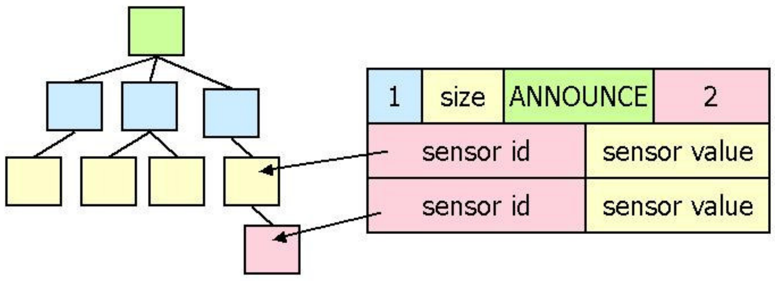

Fig. 7 S2CP ANNOUNCE message

Note that some details are left unspecified by the protocol. This is to enable implementations to take into account parameters such as power consumption and expected available bandwidth on the intended platform. Also a well-known port to be used by the S2CP controller has yet to be requested or assigned. Such an assignment would be necessary if the protocol were to be widespread [14].

3.2.1 Role of S2CP Sampler

The S2CP Sampler is hardware dependent and is implemented for a known set of sensors on known hardware. The sampler need not be smart at all, it must only be able to read and deliver data upon request. Upon start-up, the sampler will send an announce message to a known S2CP Controller port on the (same) host. This message will also contain the set of sensors that the sampler can sample from. The sampler then waits for sampling orders from the controller, and samples its sensors as orders come in. Adding more sensors to a device will require replacement of the S2CP Controller or adding a new one responsible for sampling the new set of sensors.

Upon start-up, the S2CP Sampler sends an ANNOUNCE message to the S2CP Controller (see Fig. 7). The controller then adds the sampler to its list of samplers and replies with an ACKNOWLEDGEment.

Upon receiving a Sample order message from the S2CP Controller, the S2CP Sampler will sample the sensors ordered and reply with a SAMPLE RESULT message (see Fig. 8).

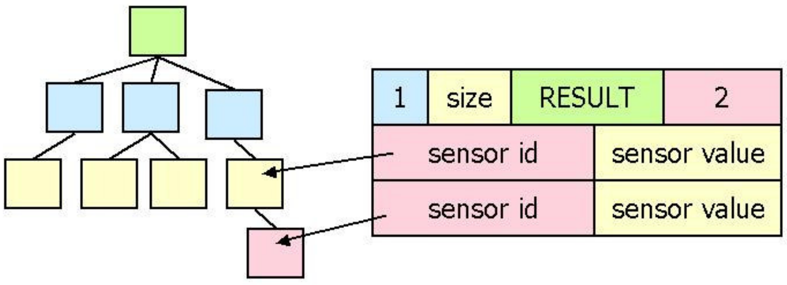

Fig. 8 S2CP SAMPLE RESULT message

3.2.2 Role of S2CP Controller

The S2CP Controller is hardware independent and is implemented for a more general set of sensor types. It can implement high levels of sensor management, do reasoning, predictions, etc. The S2CP Controller listens for sampler announcements on a known port, and maintains a list of known samplers and their sensor types. By accessing data in the Context-aware Knowledge Base, it issues ordering for new sensor data from the S2CP Samplers. As S2CP Controller implementations get smaller, faster, smarter, and can handle additional types of sensors, each controller can be replaced by newer versions, without having to replace the S2CP Samplers.

Whenever the S2CP Controller wishes to sample a certain sensor, it sends a SAMPLE ORDER message (see Fig. 9) to the S2CP Controller responsible for that particular sensor making a request for the requested type of value:

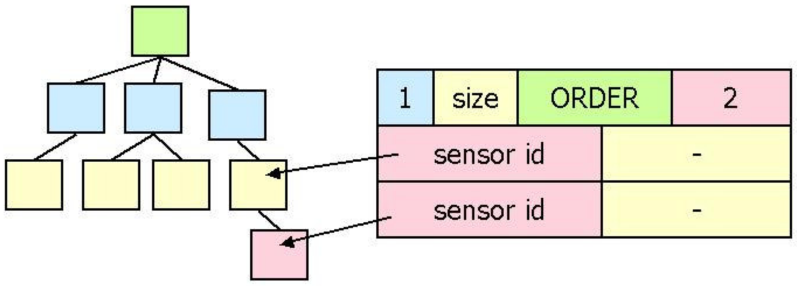

Fig. 9 S2CP SAMPLE ORDER message

3.2.3 S2CP Session

The S2CP Session starts when an ANNOUNCE message sent by a S2CP Sampler has been ACKNOWLEDGEd by the corresponding S2CP Controller. When a timeout occurs on either the sampler or the controller, a PULSE message is sent that must be ACKNOWLEDGEd. It is up to each S2CP implementation to decide how many unacknowledged PULSE messages will be sent before a session is assumed to be terminated. When a S2CP sampler or controller shuts down it is encouraged to send a SHUT DOWN message to corresponding samplers/controller to avoid termination of the session by repeatedly sent PULSEs. The S2CP state diagram is shown in Fig. 12.

3.2.4 S2CP Sequence Number and Acknowledgement

Since S2CP is designed to be able to use both UDP and TCP, it must be able to handle retransmission of lost (or altered) packets. Each packet sent (with the exception of the ACKNOWLEDGE message) has a sequence number that is incremented for each packet sent during the S2CP session. Since S2CP sessions may be alive for very long periods of time resulting in large amount of packets sent, the sequence number eventually wraps around [32]. Thus duplicate packets with the same sequence number may appear, however this is a most unlikely situation. Also wrap-around will only be confusing if more than 2^N packets are generated during a PULSE interval. S2CP packets are acknowledged one by one (and as a direct response), at all times keeping the send queue fairly short and with very recent packets, and so the likelihood of sending a packet with the same sequence number as another packet already in the send queue is very small. Wrap-around problems may be avoided by not allowing unacknowledged packets to be kept in the send queue for more than a certain time. If the session is still alive (packets with larger sequence number or a pulse message has been received) packets in the send queue may safely be discarded, since they contain old data and is most likely not interesting to the recipient. It is also possible to solve the wrap-around problem using timestamps as suggested in [32]. The S2CP Sampler sets the first sequence number when sending the ANNOUNCE message to the S2CP Controller. For each packet sent in the session, the sequence number must be incremented (with the exception of the ACKNOWLEDGE message). When acknowledging a packet, the sequence number in the packet to be acknowledged is used. Packets sent, that need to be acknowledged, are put in a send queue and removed when acknowledged. It is up to each implementation to decide upon the size of the send queue and the time interval at which packets in the queue are being resent.

The S2CP ACKNOWLEDGE message will consist only of the sequence number of the message to be acknowledged and the method is set to ACKNOWLEDGE (see Fig. 10). Upon receiving the message, the recipient will remove the corresponding message from its send queue.

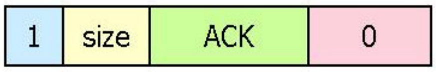

Fig. 10 S2CP ACKNOWLEDGE message

3.2.5 S2CP Dynamic Timeouts

In order to minimize the need for sending PULSE messages, S2CP may use dynamic timeouts; the S2CP Controller, that has an estimate of the time when the next SAMPLE ORDER packet will be sent, can include that time in the SAMPLE ORDER packet. The S2CP Sampler will then know when to expect the next packet and can adjust its timeout timer accordingly. It is recommended to adjust the timeout timer to at least twice as long as the controller indicates. This will allow for one packet to get lost, or the controller to be busy, without causing PULSE messages. The S2CP Controller should not have to adjust its timeout timer the same way the sampler does, since the sample result packet is expected as a direct response to the sample order message. Thus, the S2CP controller uses static time outs. This way both parties will adjust their timeouts to the expected behaviour of the corresponding peer, minimizing the PULSE messages.

Whenever a timeout occurs a S2CP PULSE message will be sent. The message will have the method set to PULSE (see Fig. 11). Upon receiving a PULSE message, the recipient must send an acknowledgement.

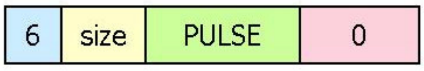

Fig. 11 S2CP PULSE message

3.2.6 S2CP Sensor Management Issues

Sensor management is the responsibility of the S2CP Controller. The goal of sensor management is to avoid sending sensor sample orders that will result in a sensor sampler result with an unchanged sensor value. Results with only little change of a sensor value must also be avoided. At the same time, great changes in sensor values must also be avoided since that means that data has been inaccurate for an unknown period of time. Thus, the sensor manager must strive to read a sensor at the exact time when it has changed enough to be considered as a change. This way, valid data will be provided with minimal traffic between controller and sampler, and minimal work for the sampler. The sensor manager must be able to predict the time when a sensor is expected to report a changed value, and also able to dynamically adapt and modify its predictions when they prove to be wrong.

3.3 Context Data eXchange Protocol

The purpose of the Context Data eXchange Protocol (CDXP) is to provide a simple way of exchanging context data between peers in a network. CDXP can use both UDP and TCP as transport protocol. Acknowledgements and timeouts are used to ensure correct communication.

CDXP is a subscription-based protocol, and uses notifications whenever data subscribed for has changed. An extra feature, prediction, may be used in order to prolong the lifetime of the devices' battery. CDXP offers functionality needed not provided by LDAP.

The current version of the protocol is based on XML [40] to make it easy to design, use, and develop. Adding new messages to the design still allows for older implementations to work since they will simply ignore what they do not recognize.

Note that some implementation details are left unspecified by the protocol design. This is to enable implementations to take into account parameters as power consumption and expected available bandwidth on the intended platform. Also a well-known port to be used by the CDXP server has yet to be requested or assigned. Such an assignment would be necessary if the protocol were to be widespread [14].

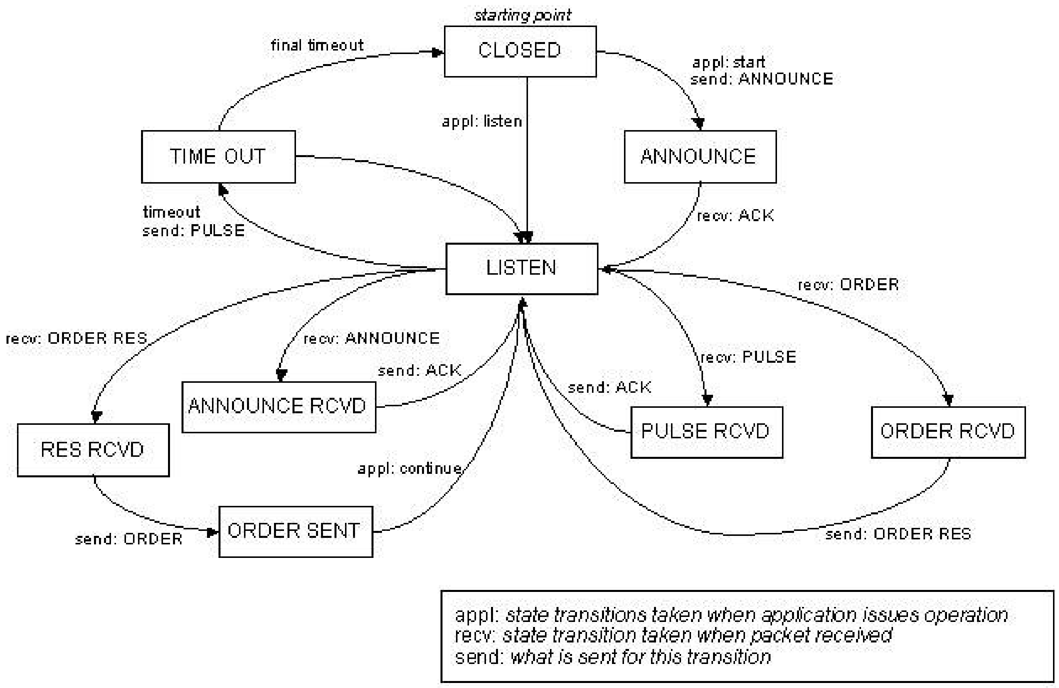

Fig. 12 The S2CP State Diagram

3.3.1 CDXP Client

The CDXP Client acts on behalf of an application that needs context information. It is responsible for retrieving the desired context data either locally or by contacting a CDXP Server on another host. A CDXP Client can use the local CDXP Server in order to retrieve local context. This way, a context-aware application does not have to implement both an LDAP client and a CDXP client. When the client wishes to initiate a subscription it contacts the CDXP Server on the server host (possibly the same host as the client) on a known CDXP Server port. It then receives notifications from the server host, and passes them on to the application the client is acting on behalf of.

3.3.2 CDXP Server

A CDXP Server is responsible for handling subscription queries from CDXP Clients. The CDXP server binds to a known port. When a request for a subscription is received, the server decides if it is willing to deliver the desired subscription to the client (based on client identity, data availability, workload, etc.). If the server decides to serve the client, it adds the client to its list of subscribers. Whenever context data changes on the server host (a CDXP Server can access and control the local CKB) the server checks to see if any client in its list of subscribers is subscribing to that data and if so, it sends a notification with the new data to the client. If the data is not updated for a period of time the server must send a keep-alive message to let the client know that the session is still alive.

3.3.3 CDXP Session

The CDXP session starts when a SUBSCRIBE message sent by a CDXP Client has been ACKNOWLEDGEd by the corresponding CDXP Server. When a timeout occurs on either the client or the server, a PULSE message is sent that must be ACKNOWLEDGEd. It is up to each CDXP implementation to decide how many unacknowledged PULSE messages that will be sent before a session is assumed terminated. The recommended way to terminate a session is to ACKNOWLEDGE an UNSUBSCRIBE message from the client (assuming this message results in there being no more subscriptions in the session). The CDXP state diagram is shown in Fig. 13.

There are several messages that can be sent during a CDXP session. Each message is described in section 3.3.5.

3.3.4 CDXP Packet

The CDXP header contains the following fields:

FROM ADDRESS	source address
FROM ADDRESS TYPE	type of from address (SIP URI, IPv4, IPv6)
FROM ENTITY ID	source entity type identified using a taxonomy tree
TO ADDRESS	destination address
TO ADDRESS TYPE	type of to address (SIP URI, IPv4, IPv6)
TO ENTITY ID	source entity type identified using a taxonomy tree
MESSAGE ID	unique sequence number
METHOD	CDXP method
EXPIRES	an expire-time for the SUBSCRIPTION

The CDXP body contains a list of data:

DATA ID	context data type identified using a taxonomy tree
VALUE	context data value
EXPIRE TIME	time when the data is no longer valid
SOURCE ADDRESS	address of the data source (e.g. when data is forwarded, this may be different than from address)
SOURCE ADDRESS TYPE	type of source address (SIP URI, IPv4, IPv6)
SOURCE ENTITY ID	entity type of the data source

A CDXP packet may look like the following:

```
<Cdxp>
          <version>1 Beta</version>
          <messageID>1</messageID>
```

```
            <from>
                        <entityID>wearable device</entityID>
                        <address type="IPv4">192.168.0.1</address>
            </from>
            <to>
                        <entityID>context server</entityID>
                        <address type="IPv4">192.168.0.2</address>
            </to>
            <method>NOTIFY</method>
            <date>2003-02-10 12:10:23</date>
            <expires>2003-02-10 12:14:46</expires>
            <contextData>
                        <dataID>Humidity</dataID>
                        <value>26</value>
                        <expire>2003-02-10 12:14:46</expire>
                        <entityID>wearable device</entityID>
                        <address type="IPv4">192.168.0.1</address>
            </contextData>
</Cdxp>
```

3.3.5 CDXP Messages

The client-to-server messages are: SUBSCRIBE, UNSUBSCRIBE, PREDICT, PULSE, and REFRESH. The server-to-client messages are NOTIFY and PULSE. The ACKNOWLEDGE and the NOT ACKNOWLEDGE message can be sent and received by both client and server. Any message not understood by either server or client will be ignored.

3.3.5.1 SUBSCRIBE

The SUBSCRIBE message is sent by the client to initiate a session. Data in a subscribe message is a list of context data that the client wishes to subscribe to. The SUBSCRIBE message must be ACKNOWLEDGEd or NOT ACKNOWLEDGEd or the client will retransmit the SUBSCRIBE request until either is received. This is to ensure proper handling when using UDP as a transport protocol. It is up to the client implementation to decide for how to wait between resending the SUBSCRIBE message, and how many to send before giving up. It is also up to the server to decide if and when to answer a SUBSCRIBE request.

3.3.5.2 UNSUBSCRIBE

Whenever a client wishes to stop receiving context data is must send an UNSUBSCRIBE message to the server telling what data it wishes to stop subscribing to. If the client has subscribed to several data, only the data in the UNSUBSCRIBE message will stop being delivered. An UNSUBSCRIBE message must be ACKNOWLEDGEd by the server or the client will retransmit at defined intervals until ACKNOWLEDGEment is received. This is to ensure proper handling when using UDP as a transport protocol.

3.3.5.3 PREDICT

If the CDXP server is running on a wearable device, a client subscribing to data from the server may use the PREDICT message. The purpose of the PREDICT message is to minimize the number of messages sent from the server. A PREDICT message contains predicted data, i.e. the data that the client expect to receive in the next NOTIFY message from the server. The server then only needs to send a NOTIFY message if the prediction is wrong. A PREDICT message does not have to be acknowledged.

3.3.5.4 REFRESH

If the CDXP client wishes to receive NOTIFY messages more frequently, it may encourage the server to do so using a REFRESH message. The REFRESH message contains a request for the data that the client wishes to have sent. A REFRESH message does not have to be acknowledged, nor does it have to be replied to by a NOTIFY message from the server, since it may be busy or want to save power. This message an expressed wish to receive a NOTIFY message. A PREDICT message with an intentionally wrong prediction will have the same meaning.

3.3.5.5 NOTIFY

Whenever context data subscribed to by a client changes, the server sends the client new data in a NOTIFY message. The NOTIFY message must be acknowledged by the client to ensure proper communication.

3.3.5.6 PULSE

In order to ensure the persistence of a session, PULSE messages can be used. When data subscribed to by the client does not update frequently enough to send a new NOTIFY within the time out interval, an PULSE message must be sent to keep the session alive. The PULSE message must be ACKNOWLEDGEd by the client to ensure proper communication.

3.3.6 CDXP Sequence Number and Acknowledgement

Since CDXP is designed to be able to use both UDP and TCP, it must be able to handle retransmission of lost (or altered) packets. Each packet sent (with the exception of the (NOT) ACKNOWLEDGE message) has a sequence number that is incremented for each packet sent during the CDXP session. Since CDXP sessions may be alive for very long periods of time resulting in large amount of packets sent, the sequence number eventually wraps around [32]. Thus duplicate packets with the same sequence number may appear, however this is a most unlikely situation. Also wrap-around will only be confusing if more than 2^N packets are generated during a PULSE interval. CDXP packets are acknowledged one by one (and as a direct response), at all times keeping the send queue fairly short and with very recent packets, and so the likelihood of sending a packet with the same sequence number as another packet already in the send queue is very small. Wrap-around problems may be avoided by not allowing unacknowledged packets to be kept in the send queue for more than a certain time. If the session is still alive (packets with larger sequence number or a pulse message has been received) packets in the send queue may safely be discarded, since they contain old data and is most likely not interesting to the recipient. It is also possible to solve the wrap-around problem using timestamps as suggested in [32]. The CDXP Client sets the first sequence number when sending the first SUBSCRIBE message to the CDXP Server. For each packet sent in the session, the sequence number must be incremented (with the exception of the (NOT) ACKNOWLEDGE message). When acknowledging a packet, the sequence number in the packet to acknowledge is used. Packets sent, that need to be acknowledged, are put in a send queue and removed when acknowledged. It is up to each implementation to decide the size of the send queue and the time interval at which packets in the queue are being resent.

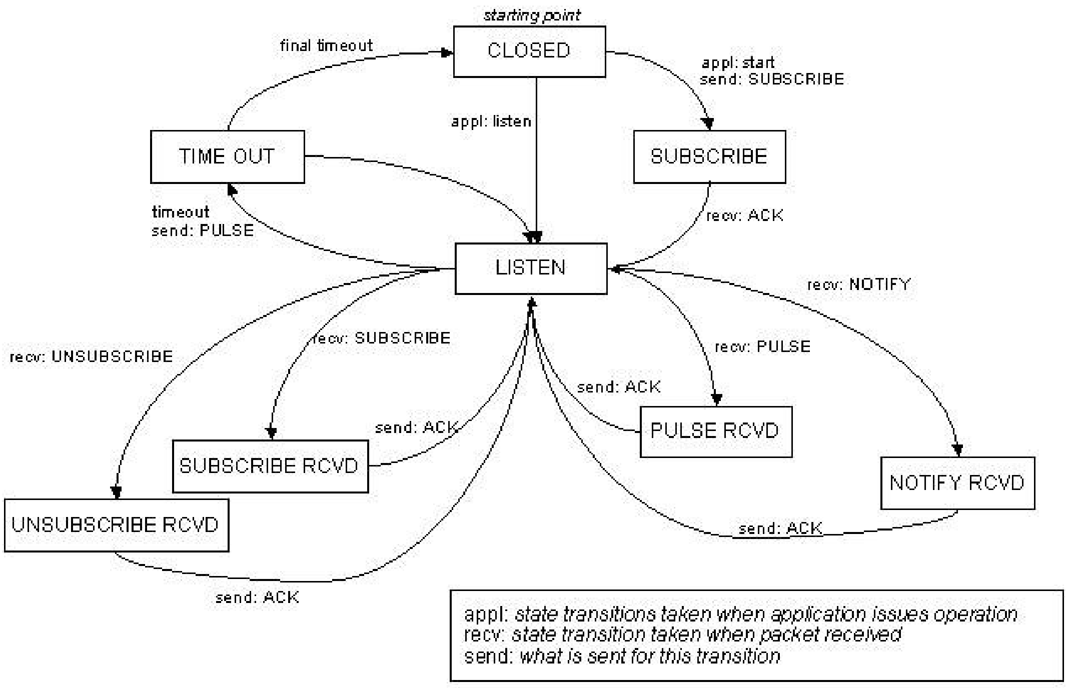

Fig. 13 The CDXP State Diagram

3.3.7 CDXP Dynamic Timeouts

In order to minimize the need for sending PULSE messages, CDXP may use dynamic timeouts; the CDXP Server sets the expire time on each NOTIFY message telling for how long the data is expected to be valid. The CDXP Client will then know when to expect the next NOTIFY packet and can adjust its timeout timer accordingly. It is recommended to adjust the timeout timer to at least twice as long as the controller indicates. This will allow for one packet to get lost, or the server to be busy, without causing PULSE messages. The CDXP Server should not have to adjust its timeout timer the same way the client does, since an acknowledgement is expected as a direct response to the notify message. Thus, the CDXP Server uses static time outs. This way both parties will adjust their timeouts to the expected behaviour of the corresponding peer, minimizing the PULSE messages.

3.3.8 CDXP Server Issues

A CDXP Server may take into consideration many things like how many clients to serve at the same time, whom to serve, etc. The protocol itself does not specify this, since some servers may or may not run on wearable devices. Also, the user might want to be able to change server settings, or settings may change as context changes (e.g batteries are low).

3.4 Context-Aware Knowledge Base

The use of a Context-Aware Knowledge Base (CKB) as described by Ren [25] is proposed. The Context Engine will write its gathered sensor data to the Context Information Base. Context-aware applications may operate on the data and knowledge contained in the CKB. The CDXP Server will accept subscription requests for context data that can be found in the CKB and send notifications as the data changes. Each instance of data in the CKB will have an expire-time associated with it. It is predicted and set by the Context Engine and context-aware applications as they update the data, since they are the more likely do be able to do qualified predictions. Thus, the CDXP Server does not need to do any computation when setting the expire-time for the data in a CDXP notify packet. It simply uses the value from the CKB.

3.5 Context Engine

The context engine integrates a S2CP Controller, a CDXP Server, and a CKB, in order to enable maximum possibilities for sensor management and data fusion. The S2CP Controller part, will give an overview of, and complete control over the available sensor types. The CDXP Server part will provide knowledge of what context information is needed by context-aware applications. Thus the context engine can use this knowledge to enhance sensor management, for example, by stop ordering sampling from sensors, who's data is not needed by any application at a give time. For determining of what context information is available for applications to subscribe to, both the CKB and the S2CP Controller can provide information. Since the context engine integrates the S2CP Controller, the CDXP Server, and the CKB, it is provided with as much information as possible for making the best possible sensor management, as well as accurate prediction of expire time for context data.

3.5.1 Predicting, and Adapting to, Changes of Sensor Values

Predicting changes of context information is the main responsibility of the context engine. Exact predictions will minimize the S2CP sample order messages, and CDXP notify messages.

The first step towards accurate prediction is deciding how much a certain context data (sensor value) must actually change before it should be considered changed (i.e. the CKB will be updated and a CDXP notify message will be triggered). The aim of the context engine is then to order a sample of a sensor exactly after the sensor value changes beyond this threshold. If the sampling occurs earlier than that, it will only result in the S2CP Sampler application doing unnecessary work wasting power. If the sampling occurs long after the change, it will increase the risk of reading a great change, meaning that data in the CKB has been inaccurate for some period of time. Accurate decisions on when to order sampling do not only affect the sampling, but also the CDXP communications. Since CDXP packets uses an expire tag on each context data, the CDXP client will discard the data when it expire and expect a notify message from the server around that time. This means that the CDXP Server will have to send a notification to the client with the same data value and a new expire time, or the client might assume packet loss and start sending pulse messages to determined whether the server is still available or not. Perfect prediction will result it minimal effort on the server side in terms of sampling sensors and S2SP communication, and it will minimize the number of CDXP packets sent to clients.

At the same time, the context engine must be sure that in between samples, the value has not changed beyond the threshold and then changed back. Shannon's sampling theorem (also known as the Nyquist criterion) states that the sampling frequency must be at least twice the maximum frequency to be measured. Whenever Shannon's sampling theorem is not fulfilled, aliasing occurs. For example, to sample the rotational frequencies of the three hands of a watch, a sample with a frequency of at least twice per minute is needed. This means that the context engine must also take into account the frequency of change of each controlled sensor, as well as changes in frequency, and be sure to request samples at least twice the current frequency.

Thus, the context engine needs to know (or learn) the frequency of change for each sensor it controls. It must also be able discover and adapt to changes of the frequency, always requesting sampling of a sensor with at least twice the frequency of the expected (and hopefully correct) frequency of the sensor. The sensor engine must be able to do this effectively for a wide range of sensors, and for a large number of sensors. Despite the fact that this will put high demands on the context engine, there is an advantage of moving the responsibility of predicting the behavior of each sensor up from the S2CP Sampler layer, since the context engine is able to take into consideration, how changes of several sensors are related. For example, if a light sensor reports increasing light levels, it is likely that a nearby thermometer will report a slightly increasing temperature. If motion sensors report that a device is changing orientation, it is likely to experience changes in light and potentially changes in temperature.

3.5.2 A Simple Design for Adapting to Changes of Sensor Values

Implementing a context engine that will be able to meet all demands stated in the previous section will most likely involve AI. This way, the context engine can learn about the sensors behavior and the relations of their behavior as it runs. It can even learn how to adapt to changes of sensor behavior. However, AI is outside the scope of this thesis. A simpler design for adapting to changes of sensors values will be presented.

The design uses two threshold values, T_{low} and T_{high}, to determine whether to change the sampling frequency or not with T_{high} being the same as the threshold value for triggering a CDXP notify message. Further it uses two sampling frequency limits, F_{low} and F_{high}, which sets the bounds for valid sampling frequencies. Finally, a step value, Δ, is used as the value with which to change the current sampling frequency, F_{cur}. Three sensor values are used, S_{ref} , S_{cur}, and S_{not}, in order to compute the change of a sensor since the last reading and since the last reading that triggered an update in the CKB.

The Context Engine sends the first sample request and sets both S_{ref} and S_{cur} equal to the resulting sensor value. After $1/F_{cur}$ seconds it requests the next sample.

- If the resulting sensor value, diff(S_{cur}, S_{ref}), is between T_{low} and T_{high}, F_{cur} is left unmodified.
- If diff(S_{cur}, S_{ref}) < T_{low}, the context engine assumes that the sampling order was made too soon, and so F_{cur} is set to $F_{cur} + \Delta$. S_{ref} is the set to S_{cur}
- If instead diff(S_{cur}, S_{ref}) > T_{high}, the context engine assumes that the sampling was made too late and so F_{cur} is set to $F_{cur} - \Delta$. S_{ref} is then set to S_{cur}
- Whenever diff(S_{cur}, S_{not}) > Thigh, the CKB is updated and S_{not} is set to S_{cur}.

F_{cur} will never be allowed to exceed F_{low} and F_{high}. Also, Δ may decrease as F_{cur} approaches F_{low} and F_{high}.

3.6 Summary

The proposed solution takes into account the key issues identified in the introduction (section 1.2). It suggests the use of SIP [12] to enable constant identity, vertical handovers to enable constant connectivity [21], and it proposes the Sensor Sampling Control Protocol (S2CP) to gather context information. It further proposes the use of a Context-Aware Knowledge Base (CKB, [25]) to store context information and access context information (using LDAP [41]), and taxonomy trees to identify and represent context information. It proposes the use of Context Data eXchange Protocol (CDXP) to exchange context information and the use of a Context Server [15] to keep context information alive. Finally it proposes service discovery using SIP [3]. The proposed solution is modular, separated via protocols with clear interfaces.

4 DESIGN EVALUATION

4.1 Testing and Validating the Modularity of the Design

The aim of the suggested design was to provide a modular platform for sensing, managing, storing, and exchanging context data. Thus it must allow for replacing S2CP samplers and context engines. In order to test and validate theses aspects of the design, different types of samplers and engines (with different sensor management) were implemented. By running different combinations of these components, verification of the design modularity was possible. The following hardware and software was used during testing of the modularity:

4.1.1 SmartBadge4

As a sensor platform, the SmartBadge [2][30][31] was used. The SmartBadge was equipped with one 3-axis accelerometer, two temperature sensors (front and back), one light dependant resistor, and battery meter. The sensors were accessed by reading a byte of data from a file mapped to the specific sensor. Conversion tables were then used to translate this byte value into relevant context data values (temperature, light level, etc.). The SmartBadge was powered by two sets of 1.2-volt batteries in series and carried clipped on to a user's belt.

4.1.2 S2CP Sampler

Three types of S2CP sampler were implemented and tested against the other components of the design, in order to confirm that replacement was possible. All the S2CP samplers were implemented in C (on Linux) and compiled for both x86 and ARM architectures, and run both on an Intel processor based laptop and the SmartBadge (with the exception of the S2CP SmartBadge Sampler).

4.1.2.1 S2CP SmartBadge Sampler

The first type of S2CP sampler, called the S2CP SmartBadge Sampler, was implemented in C and compiled for the ARM architecture. It is capable of reading all sensors mounted on the SmartBadge (see sections 2.2.3 and 4.1.1). Access to the sensors are provided using files, and so reading a byte from a file will deliver the analog to digital converted sensor value of the sensor corresponding to the file. This will of course only run on the SmartBadge, thus this type of sampler is only available for the SmartBadge. Once a sensor value is read, it must generally be converted from the one-byte value (between 0 and 255) to a more useful value. For this, conversion tables are used, indexed by an internal sensor identity and the byte value. The table lookup returns a value that can be used in S2CP and CDXP (such as temperature in Kelvin, light level in Lux, etc.). The conversion tables were provided by the results of sensor calibration made by Prof. Maguire. The S2CP SmartBadge Sampler is around 24K bytes in size (ARM compiled version), including the S2CP communication implementation, but without a complete set of conversion tables. A simple conversion table that converts byte values to integer values is 1K bytes in size. An index table, used to determine what conversion table to use, is around 40 bytes in size (for 10 sensors).

Note! When reading the sensors on the SmartBadge, variations will appear due to analog to digital conversion, and so a sample of a sensor really consists of a sample burst of several samples. The average value is then reported back as the sample result. A sample burst of 100 samples takes about 6 ms to carry out.

4.1.2.2 S2CP Simulated Sampler

The second type of S2CP sampler, called the S2CP Simulated Sampler, was implemented in C and compiled for both the ARM and the x86 architecture. Instead of reading real sensors, this sampler generates sensor values at random. The same conversion table lookup procedure as for the S2CP SmartBadge Sampler is used. This sampler was used mostly for developing and debugging of the other components of the solution. The x86 version of this sampler is around 24K bytes in size, including the S2CP communication implementation.

4.1.2.3 S2CP Recording Sampler

The last type of S2CP sampler is called the S2CP Recording Sampler. Instead of reading real time sensor values, it reads the values from a file with prerecorded sensor values (see section 4.3.2). When the sampler is started, it notes the time. Upon any S2CP sample request, the offset time is calculated. Offset time, the recording frequency, and the number of sensors recorded, is then used to compute the offset to the correct byte in the file, which is then read. The same conversion table lookup procedure as for the S2CP SmartBadge Sampler is used. The S2CP Recording Sampler was implemented to enable repeated testing with some sets of real sensor data without having to move around with the SmartBadge. The x86 version of the S2CP Recording Sampler is around 30K bytes in size, including the S2CP communication implementation.

4.1.3 Context Engine

Three types of context engines with different sensor management strategies were implemented in C, and compiled for both the ARM and the x86 architecture. All three context engines implements a simple database, an S2CP controller, a CDXP server, and a sensor manager. The S2CP controller, the CDXP server, and the sensor manager run as separate threads within the same process. This is to minimize the number of processes on the SmartBadge. Only the sensor managers differ in the implementations. The x86 versions of the context engines are around 44K bytes in size, including S2CP and CDXP communication implementation and CDXP parsing. The following types of sensor managers were implemented in the context engines: Polling Sensor Manager, Constant Sensor Manager, and Dynamic Sensor Manager. Each is described below.

4.1.3.1 Polling Sensor Manager

The first type of sensor management sends S2CP sampler requests at the highest possible frequency. This ensures minimal delay between changes in sensor values and triggering of CDXP notifications. This sensor manager was implemented to emulate optimal sensor management from the CDXP point of view, and was used as a comparison against the other types of sensor managers during testing. From the S2CP point of view, this strategy consumes the most bandwidth and generates lots of work for the S2CP sampler and this sampling rate is likely to be unnecessary to provide accurate sensor data.

4.1.3.2 Constant Sensor Manager

The second type of sensor manager schedules S2CP sample requests at constant time intervals. The intervals can be optimized for known frequencies of changes for certain sensors; following testing which will increase performance (by reducing unnecessary sampling). The purpose of this extremely simple sensor management strategy is for comparison with the performance of the Dynamic Sensor Manager. This strategy enables good control over bandwidth and power consumption, but will most likely perform poorly for sensors with changing frequencies.

4.1.3.3 Dynamic Sensor Manager

The last type of sensor manager implements the algorithm presented above (see section 3.5.2). The purpose of this type of manager is to show that dynamic adapting to changes in sensor values are needed in order to achieve good performance of the design rather than to suggest precisely how adaptation should behave. Since a sensor manager will most likely have to manage sensors with both constant and varying frequency behaviors, it must be able to adjust to all kinds of sensor behaviors. Investigating this suggested use of a dynamic manager, measurements give some indications of how to improve upon this dynamic sensor management.

4.1.4 Simple CDXP Client

In order to trigger the context engine to start controlling the S2CP sampler, updating the data base, and sending CDXP notifications, a simple CDXP client, capable of initiating a subscription and reporting notifications to standard output was implemented and tested. It is around 30K bytes in size including CDXP communication and parsing.

4.1.5 Context Server

Testing against the context server as implemented by Asim Jarrar shows that the implementations in C and Java were compatible with the XML-formatted Context Data eXchange Protocol. The context server waits for hosts to enter the network, and upon discovery of their presence (e.g. by receiving a CDXP subscription message), it starts subscriptions on all context data the new host is willing to share. It maintains the subscription for as long as the host allows. It also grants subscriptions to other CDXP clients, provided that the context data subscribed to can be made available, directly or indirectly, on the context server. When testing the design, a context engine was started on a host, and from the same host a simple CDXP client sent a subscription to the context server to trigger the server. The context server then began subscribing to context data available on the host.

4.1.6 Summary

Different combinations of S2CP samplers and context engines were both run on and off the SmartBadge in order to confirm the modularity of the design. No timing measurements were made during this part of the testing. Measuring speed and size of the implementations are quite irrelevant, since they are all prototypes. It is most likely that implementation sizes and complexity will change with smarter (more dynamic) sensor management. The purpose of the testing was to verify the protocols and modular design.

4.2 Comparing Context Engines (Sensor Managers)

Since the amount of work done by the S2CP samplers and the number of packets send using the S2CP and CDXP, must be kept at a minimum in order to minimize bandwidth and power consumption, it is of great interest to try to isolate and investigate the source of performance limitations; specifically the context engine. Since both the S2CP and CDXP depend on the context engine’s implementation.

A context engine that can predict the time of change of a sensor can send fewer S2CP sample requests, resulting in both fewer packets sent and less work for the S2CP sampler. Accurate prediction will also affect the CDXP side, since unnecessary notifications are not sent (the CDXP server must only send notifications whenever data expires).

When making a detailed investigation of sensor management performance using the components described in section 4.1, changes of sensor values, process and thread switches, semaphores, network communication, will cause each test run to be different from the previous ones. No two test runs will be exactly the same, and the exact behavior of the separate modules cannot be known. Therefore prerecorded sensor data from files were used as input when comparing sensor managers to assure the same sensor behavior. To ensure the exact same input over a large set of test runs, a special test tool was used (see section 4.2.2). Several types of test data were used during testing and evaluation (see section 4.2.1).

4.2.1 Test Data

In order to enable comparisons of performance between the different types of sensor managers, prerecorded data was used. This ensures minimal differences in sampling results during different test runs (variations are due to system calls, thread swapping, and process switching). Another advantage of using recorded data is that the time parameter can be skewed in order to obtain slower or faster changes of the sensors, and thus produce new sensor behaviors. This also avoids the need to move the SmartBadge to change sensor values.

The sampled sensor values were written to a file. At the beginning of the file, information about the number of sensors, sensor identities, sampling frequency, and the number of samplings are stored. The sampling results were written (one byte per sensor) consecutively. Finally an end tag is written to enable checking for complete files. To reach a specific value in the file, an offset depending on time passed since the start, the number of sensors, and the frequency at which the sensors were sampled, and the sensor id must be calculated:

Sample round = time passed / time between samples
(cycles are obtained by sample round = sample round % total number of samples)

Round offset = sample round * the number of sensors

Sensor offset = sensor values are stored in the same order as their ids are written in the file header

Total offset = round offset + sensor offset + header size

4.2.1.1 Recordings of real data

To be able to test and evaluate the performance of the sensor managers, some realistic input data is needed. Using the SmartBadge, data was recorded and written to a sensor-recording file. Using a simple sampler recorder (8K bytes in size), run on the SmartBadge, sampling was done a 1000 samples/second of all 8 sensors, which is close to the maximum sampling frequency possible when reading all the sensors in each sample round (it takes approximately 600 microseconds to read all the sensors on the SmartBadge). The recording implementation samples all available sensors and then waits in a loop until it is time for the next round of samples to be read.

Several recordings were done with the SmartBadge clipped to the chest pocket or the belt. Recordings of working in front of the computer, walking around in corridors, stairs and elevators, and some outdoor walking were done. Each recording was around 10 minutes long. A 10 minutes sample file with recordings of 8 sensors with a sample rate of 1kHz (a sample every ms) is around 5M bytes in size.

4.2.1.2 Generation of synthetic data

In order not to be limited by the behaviors of the available sensors, synthetic sensor data was generated. This is to simulate other types of sensors as well as possible incomplete activities during real recording. In order to test the sensor managers with sensor behaviors not available from the sensors on the SmartBadge, generated sensor data was used. A recording file with three different types of sensors was used; one with constant rate of change (see Fig. 14), one that alternates its rate of change between a slow and a fast frequency (see Fig. 15), and one sensor changing at random (see Fig. 16). All three synthetic sensors alternate between three different levels. This is to enable testing of the sensor managers with different thresholds for triggering of CDXP notifications.

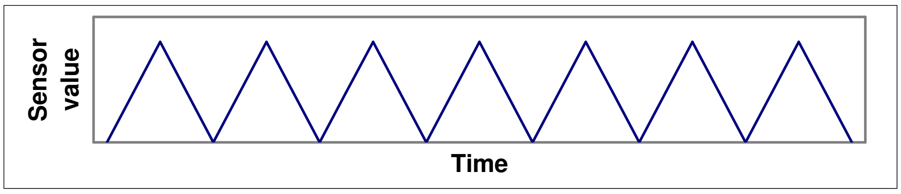

Fig. 14 Periodic Sensor

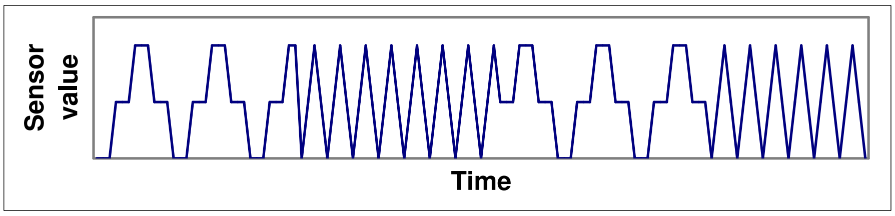

Fig. 15 Alternating Sensor

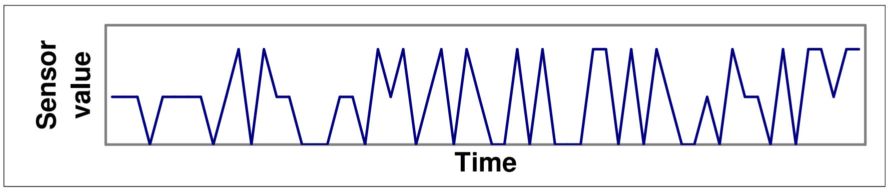

Fig. 16 Random Sensor

4.2.2 Test Tool

Extracting the sensor manager implementations from the context engines, modifying them, and inserting probes for measurement created a test tool that was used for evaluation of the different sensor managers described in section 4.1.3. This ensures the exact same sensor input in every test run and enables running tests with different threshold values and parameters and avoids noise due to other processing.

The test tool uses sensor recordings as input and evaluates all three sensor-management strategies in parallel. It uses an internal time variable that serves as “real” time, and three strategies each use their own variables to indicate when their next sampling request will occur. In between these times they “sleep”. Counters keep track of how many S2CP and CDXP packets are sent as well as how many S2CP packets were not sent, that should have been. As soon as a strategy’s sleep time has passed, it is “woken” up, meaning that it will send a S2CP sample request to get new data, and a CDXP notify with that new data (because the previous data’s validity has expired). It then computes the time of the next wakeup. The optimal strategy wakes up with the same frequency as the sample frequency of the recording file, but updates its counters only when data has changed enough to trigger a CDXP notification. This emulates perfect prediction of data expiration. Also every time a strategy wakes up it checks to see if it has missed sending a S2CP sampler request, by checking against the number of S2CP sampler requests sent by the optimal strategy. If the difference has increased since the last wake up, the same amount of packets as the increase of the difference have been missed. Data about sleep times that are set by the different strategies can be written to file to be used later for graphical display. This shows how the input data changes (through the optimal strategy output) and how the strategies adapt (or don't adapt) to these changes.

4.2.3 Measurements

During test runs with recorded data fed to the test tool, measurements of how many S2CP sample requests (which is equal to the number of CDXP notifications sent) and how many times no CDXP notification were sent for a sensor change were recorded. This will show, how much bandwidth is consumed by S2CP and CDXP; as well as how much work has to be done by the S2CP sampler.

4.3 Test Results

4.3.1 Modularity

By running all three types of S2CP Samplers against all three types of context engines, the modularity of the Sensor Sampler Control Protocol was verified. No modifications were needed. This worked well given that the context engine knows how to handle the sensors values reported by the S2CP samplers. By running both the Simpler CDXP client and the Context Server against the context engine, the modularity of CDXP was verified.

4.3.2 Recordings of Real Sensor Data

Real sensor data was recorded when sitting in front of a computer, walking around in corridors, and walking outdoors. The sensors were sampled at a rate of 1000 samples/second and the values shown are raw sampled values (i.e. no averaging computing was done), which explain the noise in the diagrams. Fig. 17 and Fig. 18 shows the temperature when walking around in a building and then going outside (the move from indoor to outdoor occurring after around 330 seconds). Fig. 19 shows a histogram over temperatures in Fig. 17 (with the exception of the time when the indoor to outdoor transition was made). This shows that even though there is a lot of noise, the temperature range is very narrow.

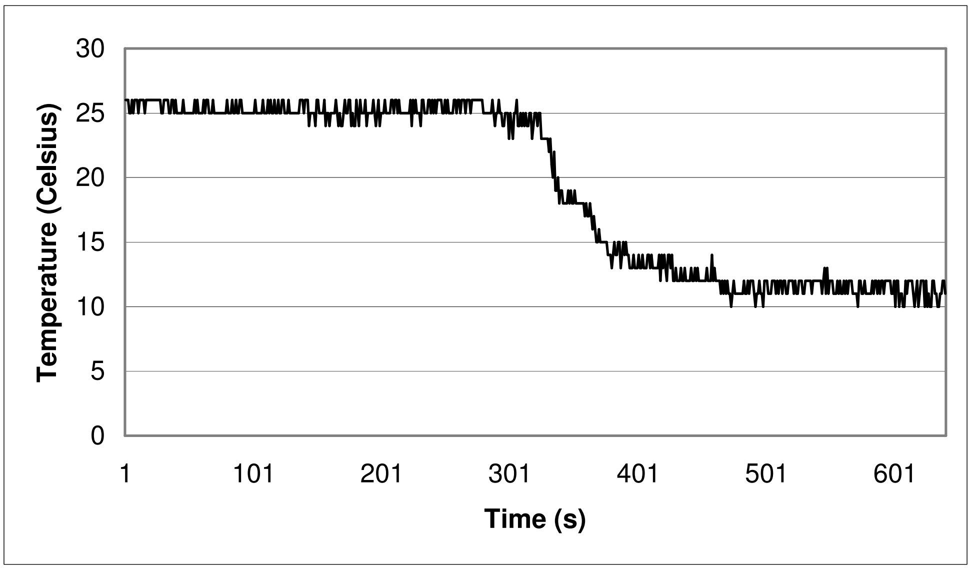

Fig. 17 Recording of SmartBadge Front Side Temperature

4.3.3 Sensor Managers

The dynamic sensor manager was tested using different recording files as input. This section presents and discusses the results of these tests. Test results are shown in Table 1, Table 2, Table 3, and Table 4. The percentage difference from ideal is not always a good measurement on performance since, in theory a sensor manager can send the same number of packets as the optimal manager and still miss a huge number of packets. Hence, this value only indicates performance if the number of packets missed is low.

4.3.3.1 Periodical Sensor

Table 1 shows the test results when using the periodic sensor (see Fig. 14) as input to the dynamic sensor manager. The three levels from the sensor are mapped to the values 0, 2, or 4 using the same type of table lookup procedure as with real sensors.

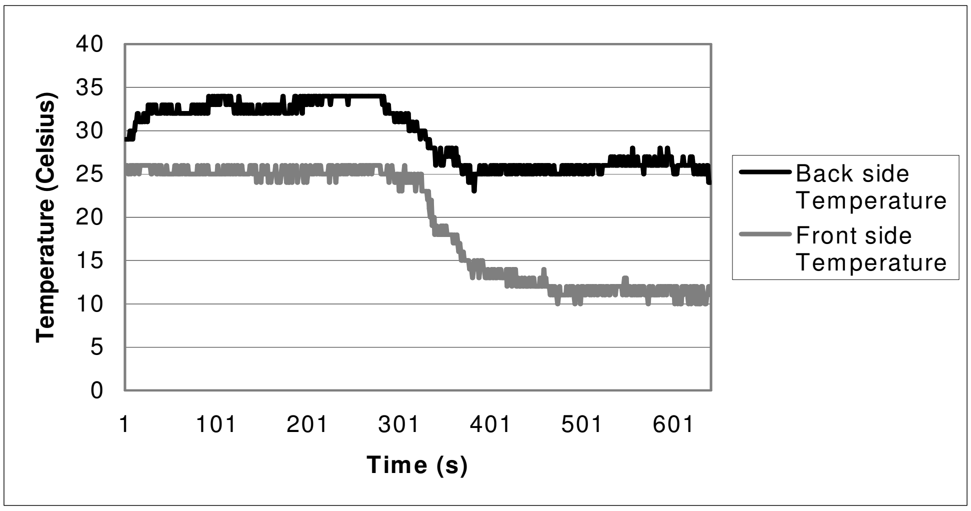

Fig. 18 Recording of SmartBadge Front and Back Side Temperature

Row 1-4 in the table shows the results for constant reading intervals with different frequencies. It is clear that the case when samplings occur at twice the frequency of change or more, no packets are missed and an optimal number of packets are sent. Thus, it is clear that sampling at twice the frequency of change has the optimal result. For a periodic sampling of sensors with constant frequency, sensor managers with constant sample request frequency will have no packets missed as long as their frequency is at least twice the frequency of the sensor. However, they will send unnecessary packets for frequencies higher than twice the sensor's frequency.

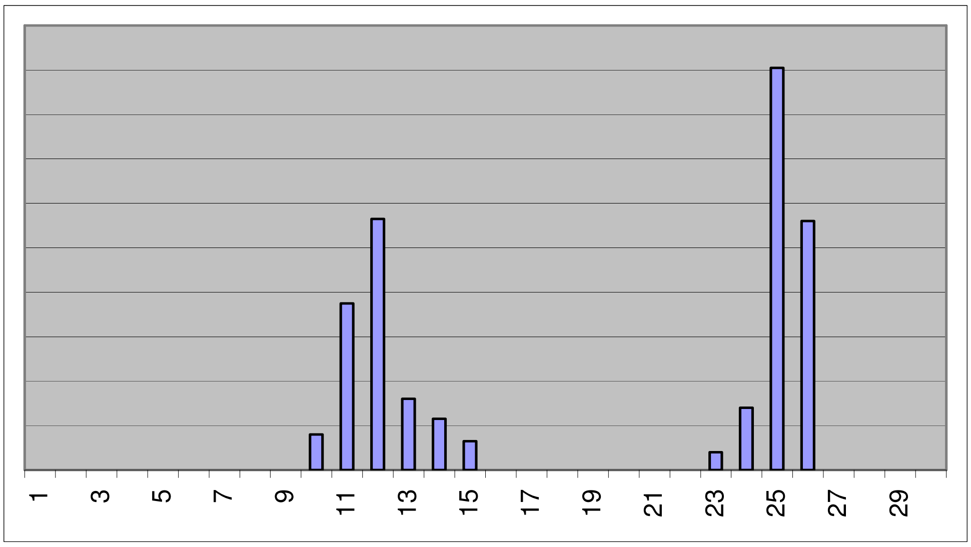

Fig. 19 SmartBadge Front Side Temperature Range

Table 1. Test Results for Periodic Sensor

	Sensor	Frequency (Hz)	No. rounds	T low	T high	F high (Hz)	F low (Hz)	Delta (ms)	Packets sent	Optimal	Percentage difference from ideal	Packets missed
1	periodic	0.25	81914	-	3.0	1	1	-	81914	40957	2,00	0
2	periodic	0.25	81914	-	3.0	0.5	0.5	-	40957	40957	1,00	0
3	periodic	0.25	81914	-	3.0	0.25	0.25	-	20479	40957	0,50	20478
4	periodic	0.25	81914	-	3.0	0.2	0.2	-	16383	40957	0,40	24574
5	periodic	0.25	81914	1.8	3.0	0.2	0.1	2000	16383	40957	0,40	24574
6	periodic	0.25	81914	1.8	3.0	0.25	0.1	2000	16383	40957	0,40	24574
7	periodic	0.25	81914	1.8	3.0	0.5	0.1	2000	40957	40957	1,00	0
8	periodic	0.25	81914	1.8	3.0	0.75	0.1	2000	8200	40957	0,20	32759
9	periodic	0.25	81914	1.8	3.0	1	0.1	2000	81914	40957	2,00	0
10	periodic	0.25	81914	2.2	3.0	0.2	0.1	2000	8193	40957	0,20	32764
11	periodic	0.25	81914	2.2	3.0	0.25	0.1	2000	16383	40957	0,40	24574
12	periodic	0.25	81914	2.2	3.0	0.5	0.1	2000	40957	40957	1,00	0
13	periodic	0.25	81914	2.2	3.0	0.75	0.1	2000	9104	40957	0,22	31854
14	periodic	0.25	81914	2.2	3.0	1	0.1	2000	8194	40957	0,20	32764

The second group of results (row 5-9) all have T_{low} set to 1.8. When sampling at lower frequencies than twice the rate of change packets are missed. In the first two cases (row 5 and 6) F_{high} limits the sampling frequency to lower than twice the rate of change, resulting in missed packets and poor performance. Fig. 20 shows the behaviour of the dynamic sensor manager for the settings shown in the 6th row in Table 1.

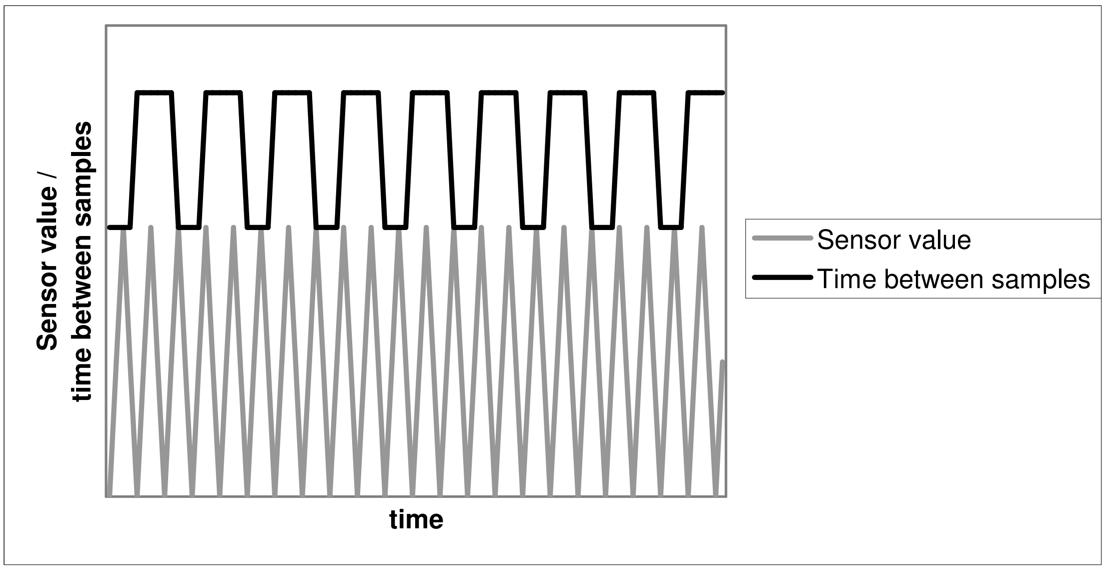

Fig. 20 Behavior of Dynamic Sensor Manager for Periodic sensor (settings as in row 6 in Table 1)

The third case (row 7) reaches optimal result, as the sampling rate is twice the frequency of change. Starting at higher sampling frequencies shows different behavior. The fourth result (row 8) shows poor performance due to unwanted timing in samplings that discovers no change in value, hence the sampling frequency moves towards F_{low}. Fig. 21 shows the behavior if the dynamic sensor manager for settings as in the 8th row in Table 1.

The third group of test results (row 10-14) all has T_{low} set to 2.2. This makes the dynamic sensor manager shows a greater tendency to decrease sampling frequency, since many changes will be below 2.2. Thus, they show behaviors like the one shown in Fig. 21.

4.3.3.2 Alternating Sensor

Table 2 shows the test results when using the alternating sensor (see Fig. 15) as input to the dynamic sensor manager. Again, the three levels of the sensor are mapped to the values 0, 2, or 4.

Also in this table, the first four rows in the table show the results for constant reading intervals with different frequencies.

The second group of results (row 5-8) has few packets missed, since the sampling frequency will always be at least twice the fastest rate of change. The unwanted side effect of this is that a large number of unnecessary packets are sent.

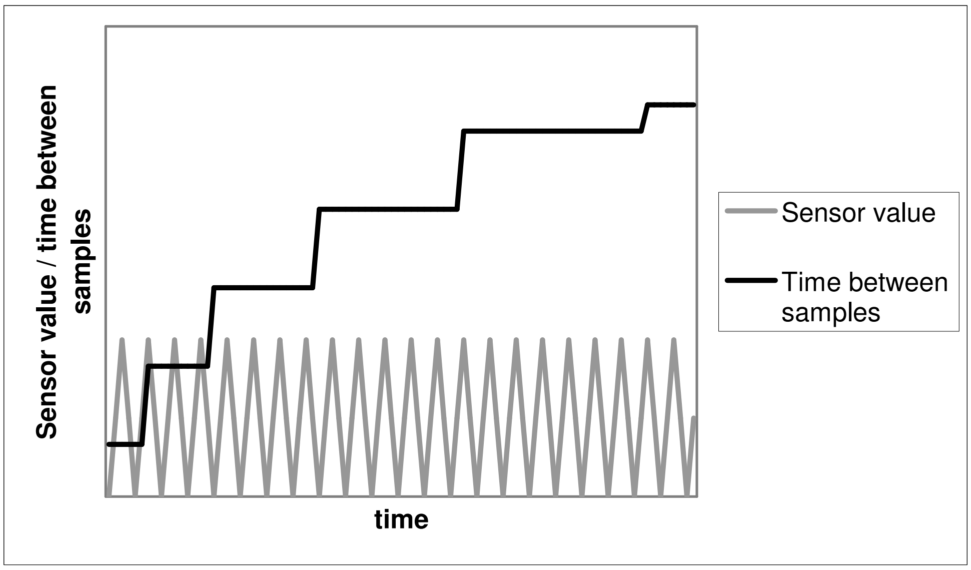

Fig. 21 Behavior of Dynamic Sensor Manager for Periodic sensor (T_{low} set to 2.2)

For this type of sensor, the dynamic sensor manager seem to perform best when sampling frequencies are between 0.5 and 0.083 Hz which is twice the two rates of change of the sensor.

Table 2. Test Results for Alternating Sensor

Sensor	Frequency (Hz)	No. rounds	T low	T high	F high (Hz)	F low (Hz)	Delta (ms)	Packets sent	Optimal	Percentage difference from ideal	Packets missed
1 alternating	0.250/0.083	81914	-	3.0	1	1	-	81914	26451	3,10	0
2 alternating	0.250/0.083	81914	-	3.0	0.5	0.5	-	40957	26451	1,55	0
3 alternating	0.250/0.083	81914	-	3.0	0.25	0.25	-	20479	26451	0,77	9812
4 alternating	0.250/0.083	81914	-	3.0	0.2	0.2	-	16383	26451	0,62	11775
5 alternating	0.250/0.083	81914	1.8	3	1	0.5	250	40967	26451	1,55	1
6 alternating	0.250/0.083	81914	2.2	3	1	0.5	250	40958	26451	1,55	1
7 alternating	0.250/0.083	81914	1.8	3	1	0.5	500	40973	26451	1,55	1
8 alternating	0.250/0.083	81914	2.2	3	1	0.5	500	40963	26451	1,55	1
9 alternating	0.250/0.083	81914	1.8	3	0.5	0.166	4000	30717	26451	1,16	3414
10 alternating	0.250/0.083	81914	2.2	3	0.5	0.166	4000	24743	26451	0,94	4268
11 alternating	0.250/0.083	81914	1.8	3	0.5	0.166	2000	34131	26451	1,29	0
12 alternating	0.250/0.083	81914	2.2	3	0.5	0.166	2000	33277	26451	1,26	0

Fig. 22 shows an example of dynamic sensor behavior for the alternating sensor.

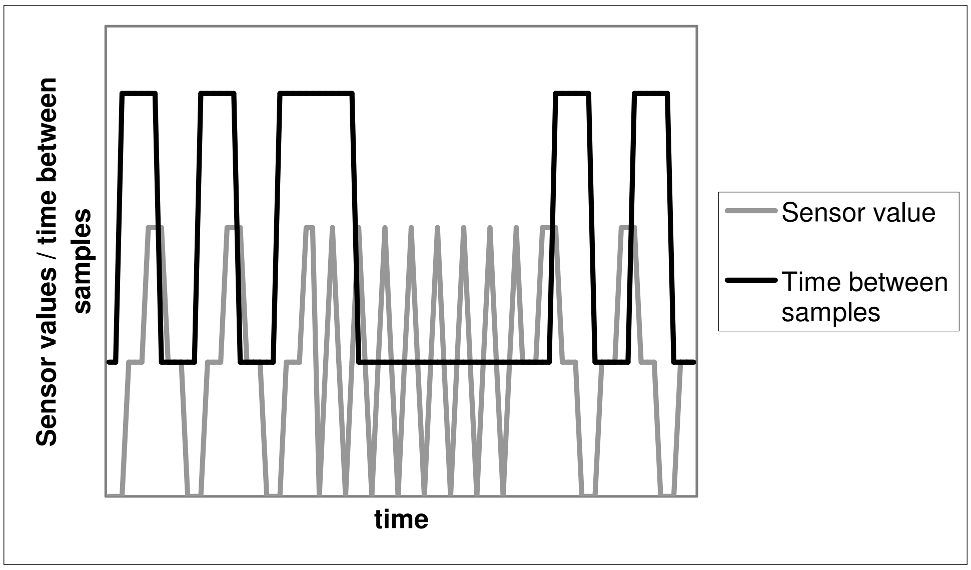

Fig. 22 Behavior of Dynamic Sensor Manager for Alternating Sensor

4.3.3.3 Random Sensor

Table 3 shows the test results when using the random sensor (see Fig. 16) as input. Again, the three levels of the sensors are mapped to the values 0, 2, or 4.

Table 3. Test Results for Random Sensor

	Sensor	Frequency (Hz)	No. rounds	T low	T high	F high (Hz)	F low (Hz)	Delta (ms)	Packets sent	Optimal	Percentage difference from ideal	Packets missed
1	random	< 0.5	81914	-	3.0	1	1	-	81914	27376	2,99	0
2	random	< 0.5	81914	-	3.0	0.5	0.5	-	40957	27376	1,50	4551
3	random	< 0.5	81914	-	3.0	0.25	0.25	-	20479	27376	0,75	10852
4	random	< 0.5	81914	-	3.0	0.2	0.2	-	16383	27376	0,60	13111
5	random	< 0.5	81914	1.8	3.0	1	0.1	2000	11043	27376	0,40	17465
6	random	< 0.5	81914	1.8	3.0	0.5	0.1	2000	10999	27376	0,40	17380
7	random	< 0.5	81914	1.8	3.0	0.25	0.1	2000	10223	27376	0,37	17759
8	random	< 0.5	81914	1.8	3.0	0.2	0.1	2000	9829	27376	0,36	18002
9	random											
10	random	< 0.5	81914	2.2	3.0	1	0.1	2000	9083	27376	0,33	18648
11	random	< 0.5	81914	2.2	3.0	0.5	0.1	2000	9071	27376	0,33	18640
12	random	< 0.5	81914	2.2	3.0	0.25	0.1	2000	8995	27376	0,33	18671
13	random	< 0.5	81914	2.2	3.0	0.2	0.1	2000	8948	27376	0,33	18712

Explaining the behavior of the dynamic sensor manager in this case, requires thorough investigations and analysis of each of the test runs. For this type of sensor, the best way to ensure no packets are missed is to keep the sampler order request frequency at twice the maximum frequency of change for the sensor. This causes a great amount of packets to be sent. Looking at the behavior of the manager shows that it does in fact adapt to the changes of the sensor (see Fig. 23).

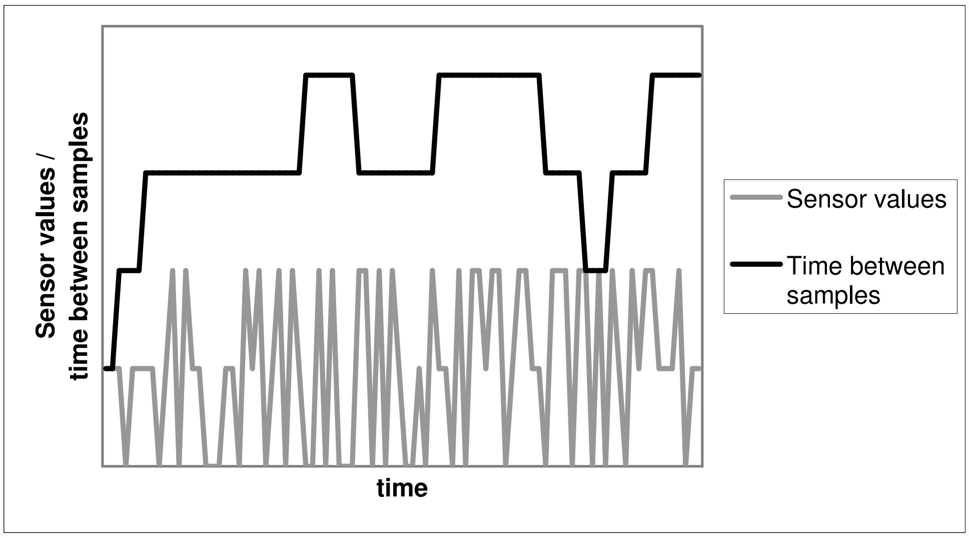

Fig. 23 Behavior of Dynamic Sensor Management for Random Sensor

4.3.3.4 SmartBadge Front Side Temperature Sensor

Table 4 shows the test results when using the SmartBadge front side temperature sensor as input (see Fig. 17). Sample bursts are not used, however, after converting the sensor value to Celsius using the table lookup procedure, the sensor shows fairly stable results. At around 5.5 minutes (around 330 seconds), the sensor was exposed to outdoor temperatures.

Table 4. Test Results for SmartBadge Front Side Temperature Sensor

Sensor	Frequency (Hz)	No. rounds	Tlow	Thigh	Fhigh (Hz)	Flow (Hz)	Delta (ms)	Packets sent	Optimal	Percentage difference from ideal	Packets missed
1 temp.	<500	639994	-	2	1000	1000	-	639994	2209	289,72	1
2 temp.	<500	639994	-	2	100	100	-	64000	2209	28,97	507
3 temp.	<500	639994	-	2	10	10	-	6400	2209	2,90	1632
4 temp.	<500	639994	-	2	1	1	-	640	2209	0,29	2134
5 temp.	<500	639994	1.1	2	1000	100	2	6246	2209	2,83	1636
6 temp.	<500	639994	1.1	2	1000	1	100	646	2209	0,29	2133
7 temp.	<500	639994	1.1	2	1000	0.1	1000	70	2209	0,03	2194
8 temp.	<500	639994	1.1	3	1000	0.1	1000	70	8	8,75	2

For all settings of the dynamic sensor manager, the sampler request frequency moves towards F_{low} and the manager shows behaviors like that shown in Fig. 24. It seems like the dynamic sensor manager have problems adapting to the great change in temperature at around 330 seconds. Lowering F_{low} does not seem to improve performance as the number of missed packets increase. Keeping F_{low} relatively high, results in large number of packets being sent, and even so changes in temperature packets are missed (i.e. the values in packets missing from the samples received). Fig. 24 shows the behavior of the dynamic sensor manager with the settings from the last row in Table 4.

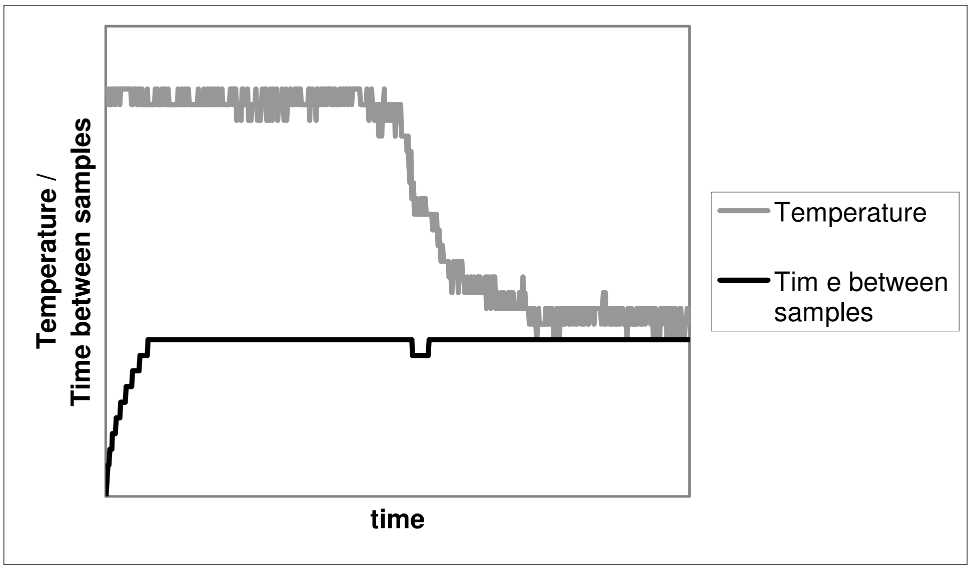

Fig. 24 Behavior of Dynamic Sensor Manager for SmartBadge Front Side Temperature Sensor

5 CONCLUSIONS

The design presented in this thesis provides means to sense, manage, store, and exchange context information using the S2CP sampler and controller, the context engine, and the CDXP client and server. Along with the context data modeling solution (the taxonomy tree) it provides a platform to be used by context-aware applications.

The design allows for replacement of, or changes within, the components of the design without affecting the rest of the design. The two protocols, S2CP and CDXP, interface between the modules and enable the desired modularity. Depending on the platform, combinations of S2CP samplers and context engines can be altered to fit the requirements for that specific platform, such as memory, power, and communication capacity. As components improve (get smaller, faster, smarter, etc.) they can be replaced.

Using S2CP enables separation from hardware (sensor) dependent application and abstract sensor management applications. Thus, adding and removing sensors on a platform does not affect the S2CP controller (provided that it can handle the new types of sensors added).

Using CDXP enables exchanging and propagating context data, in a convenient fashion. The recipient need not know anything about the source of the data including its expected behavior. It is the responsibility of the sender of context data to consider the behavior of the data source.

The aim of the design is to minimize both bandwidth and power consumption. The fewer S2CP packets sent, the less work has to be done by the S2CP samplers, decreasing both bandwidth and power consumption. The fewer CDXP packets sent the less bandwidth is consumed. From this point of view, fewer packets sent means less bandwidth **and** lower power consumption. The two protocols S2CP and CDXP do not require much communication to work properly, but instead bandwidth consumption (and power consumption for the S2CP sampler) seems to rely on the performance of sensor management.

The behavior of the sensor manager (the context engine), strongly affects the number of packets sent or packets missed, both of which should be kept at a minimum. It is feasible to keep the sampling rate at twice the frequency of change at all times in order to obtain optimal behavior, something that sensor management used in the tests did not do. Thus it affects the overall performance of the design. The suggested dynamic sensor manager does **not** perform well enough for the different types of sensors it was tested on. It does not adapt to changes of signal frequencies well enough and it does not stabilize on a good frequency when sensors signals show periodic rates of change. More advanced sensor management must be integrated into the solution for it to show good performance. Sensor management probably must do data fusion as well as incorporate AI, in order to further improve performance. At the same time, the size of the implemented context engine (with the sensor manager) must be small in order to run on a wearable device. Several sensor managing strategies may be implemented in the same context engine, and based on the context be used interchangeably in order to further optimize context engine behavior.

Threshold values used to trigger CDXP notifications heavily affect the number of packets sent (see Table 4) and must be taken into careful consideration. Different subscribers for context data may have different preferences on threshold values for different situations, and so the context engines may have to allow for (and be able to handle) adaptation to different demands from the subscribers. Also, the value delivered in a CDXP notify packet, may have to include more than just a simple value, for example additionally including variance or a set of values.

Finally, the solution allows for improvements, without giving up its fundamental design principles. The fundamentals of the solution are separation of low-level knowledge of how to sense context data, from the more abstract management of context data. It includes a consensual agreement on how to model and describe context data and finally a model to exchange context data between hosts without the need for knowledge about the behavior of the data source at the recipient side. Minor changes in S2CP and the CDXP as well as in the modeling solution of context data, may improve the solution, but does not require reconsiderations of its fundamentals.

6 OPEN ISSUES AND FUTURE WORK

The solution presented is intended to provide a platform for future work in context-awareness. It suggests models to gather, store, evolve, and exchange context information. It leaves for future work to explore what to do with the context information, once available. It also leaves for future work looking into a variety of security issues related to the model.

6.1 Security

This thesis has **not** focused on any of the aspects of security that may be relevant. It leaves for future work to evaluate how to best extend the model to handle these aspects.

6.1.1 Device Integrity

An investigation of policies of device integrity is perhaps of great interest, since not all users are willing to share their context with everyone. Policies dealing with questions of what to share, with whom to share, and why to share, context information may be tested and evaluated. This concerns not only the technical aspects of the matter, but also psychological and user behavior related matters [38].

6.1.2 Communication Privacy

As context-awareness is sometimes dependent on exchanging context information between devices, issues concerning communication privacy may be relevant. Devices may want to hide their information from others and need authentication of users and to encrypt their communication. A thorough investigation of authentication and encryption models (e.g. IPSec, PKI, and AAA) suitable for the models presented in this thesis may be of great interest.

6.2 Artificial Intelligence

6.2.1 Sensor Management and Data Fusion

There are several approaches, algorithms, and theories in the area of sensor management and data fusion [9][10]. This thesis merely introduces the subject and emphasizes the importance of its role. Sensor management and data fusion are probably one of the key issues for increasing performance in context-aware wearable devices.

6.2.2 Context-awareness

The quality of context-aware applications, that responds to changes of context and derives more context information, will probably be a delimiter of the possibilities in the area of ubiquitous computing and context-awareness. Further investigations and development of context-aware applications is of great interest.

6.3 CDXP Improvements

The presented CDXP design leaves some interesting issues to explore. The prediction functionality is added to the protocol with the aim to enable less communication and work for wearable devices. However, this feature has not been explored in this thesis, and so it is left as future work to further investigate this functionality [11].

Also an investigation of the details of what should trigger a CDXP notify packet, would be of great interest. Most likely CDXP notifications must hold more than just a simple sensor value, and threshold values will most likely have to be of a more advanced kind. Should the subscriber state what is a significant change for its subscription or is it up to the CDXP server to make this decision. What type of value should be sent with the CDXP notifications (value, variance, etc.).

7 REFERENCES

[1] ActiveBadge web page, http://www.uk.research.att.com/ab.html. Accessed October 2002.

[2] H.W.P. Beadle, G.Q. Maguire Jr., and M.T. Smith, *"Using Location and Environment Awareness in Mobile Communications"*, Proc. EEE/IEEE ICICS'97, Singapore, September 1997.

[3] R. Casella, *"Reconfigurable Application Networks through Peer-2-Peer Discovery and Handovers"*, Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH), Master of Science Thesis performed at KTH Wireless, Stockholm, Sweden. To appear.

[4] G. Chen and D. Kotz, *"A Survey of Context-Aware Mobile Computing Research"*, Department of Computer Science, Dartmouth College, 2000

[5] B. P. Clarkson, *"Life Patterns: structure from wearable sensors"*, Massachusetts Institute of Technology, September 2002.

[6] A. K Dey and G. D. Abowd, *"Towards a Better Understanding of Context and Context-Awareness"*, Graphics, Visualization and Usability Center and College of Computing. Georgia Institute of Technology, Atlanta, GA, USA, September 1999.

[7] A. K. Dey, The Context Toolkit (web page), http://www.cs.berkeley.edu/~dey/context.html.

[8] J. Ervenius and F. Tysk, *"Dual-mode Capability in a WLAN-equipped PC for Roaming and Mobility between WLANs and GPRS Networks"*, School of Electric Engineering and Information Technology, Royal Institute of Technology (KTH), Master of Science Thesis at Wireless LAN Systems, Ericsson Enterprise AB, Stockholm, Sweden, 2001.

[9] G.W. and K.H., *"Sensor management –what, why and how"*, DSO National Laboratories, Informatics Laboratory, Singapore. Published by Elsevier, Information Fusion 1, pp. 67-75, April 2000.

[10] H. W. Gellersen, et al., *"Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts"*, Department of Computing, Lancaster University, Lancaster, UK, October 2002.

[11] S. Goel and T. Imielinski, *"Prediction-based Monitoring in Sensor Networks: Taking Lessons from MPEG"*, Computer Communication Review, vol. 35, no. 5, pp. 82-98, October 2001.

[12] M. Handley, et al., *"SIP: Session Initiation Protocol"*, RFC 2543, 1999

[13] A. Held, S. Buchholz and A. Schill, *"Modeling of Context Information for Pervasive Computing Applications"*, Proc. of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI2002), Orlando, FL, USA, July 14-18, 2002

[14] Internet Assigned Numbers Authority, http://www.iana.org, accessed March 2003.

[15] A. Jarrar, *"Context Server Support to Mobile Adaptive Communication"*, Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH), Master of Science Thesis at the Royal Institute of Technology (KTH), Stockholm, Sweden. To appear.

[16] T. Kanter, *"Adaptive Personal Mobile Communication, Service Architecture and Protocols"*, Doctoral Dissertation, Department of Microelectronics and Information Technology, Royal Institute of Technology (KTH), June 2000.

[17] T. Kanter, *"Going Wireless, Enabling an Adaptive and Extensible Environment"*, Ericsson Radio Systems AB, Kista, Sweden, [year]

[18] M. Korkea-aho, *"Context-Aware Applications Survey"*, Department of Computer Science, Helsinki University of Technology, April 2000.

[19] J. McCarthy, *"What is Artificial Intelligence"*, Computer Science Department, Stanford University, Stanford, CA, Jul 2002. http://www-formal.stanford.edu/jmc/.

[20] MobileIP (to be edited) http://www.ietf.org/html.charters/mobileip-charter.html.

[21] G. Mola, personal communications regarding his upcoming thesis, January 2003, Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH), Master of Science Thesis. To appear.

[22] H. I. Myrhaug, *“Towards Life-long and Personal Context Spaces”*, SINTEF Telecom and Intormatics, Trondheim, Norway, March 2001.

[23] PARCTAB web page, http://www.ubiq.com/parctab/parctab.html. Accessed October 2002.

[24] N. B. Priyantha, et al., *”The Cricket Compass for Context-Aware Mobile Applications”*. MIT Laboratory of Computer Science, [year]

[25] A. H. Ren, *“A Smart Network and Terminal Framework for Supporting Context-aware Mobile Internet and Personal Communications”*, Department of Microelectronics and Information Technology, Licentiate thesis at the Royal Institute of Technology (KTH), Stockholm, Sweden, October 2002.

[26] N. M. Sadeh, et al., *“MyCampus: An Agent-Based Environment for Context-Aware Mobile Services”*, Carnegie Mellon University, Pittsburgh, PA, USA, 2002.

[27] D. Salber, *”Context-Awareness and Multimodality”*, IBM, T. J. Watson Research Center, Hawthorne, NY, USA, May 2000.

[28] B. Schilit., et al., *“Context-Aware Computing Applications”*, Dissertation, Computer Science Department, Columbia University, USA, 1994.

[29] A. Schmidt, et al., *“Advanced Interaction in Context”*, TecO, University of Karlsruhe, Germany, 1999.

[30] Mark T. Smith and Gerald Q. Maguire Jr., SmartBadge/BadgePad version 4, HP Labs and Royal Institute of Technology (KTH), http://www.it.kth.se/~maguire/badge4.html, 2002-08-12.

[31] Mark T. Smith, *“Smart Cards: Integrating for Portable Complexity”*, IEEE Computer, August 1998, pp. 110-112, 115.

[32] W. Richard Stevens, *“TCP/IP Illustrated, Volume I: The Protocols”*, Addison-Wesley, 1994, ISBN 0-201-63346-9.

[33] G. Söderström, *“Virtual networks in the cellular domain”*, Department of Microelectronics and Information Technology (IMIT), Royal Institute of Technology (KTH), Master of Science Thesis, Telia Research AB, Sweden, February 2003.

[34] J. Tourrhilles and C. Carter, *“P-Handoff: A protocol for fine grained peer-to-peer vertical handoff”*, Proceedings of PIMRC, 2002.

[35] E. Tuulari, *“Context aware hand-held devices”*, VTT Technical Research Centre of Finland, ESPOO, 2000.

[36] M. Weiser and J. S. Brown, *“Designing Calm Technology”*, Xerox PARC, http://www.ubiq.com/weiser/calmtech/calmtech.htm, December 1995.

[37] M. Weiser, *“Ubiquitous Computing”*, http://www.ubiq.com/hypertext/weiser/UbiHome.html, 1996.

[38] M. Weiser, *"The Computer for the Twenty-First Century"*, Scientific American, pp. 94-10, September 1991

[39] N. Xiong and P. Svensson, *“Multi-sensor management for information fusion: issues and approaches”*, Department of Data and Information Fusion, FOI, Stockholm. Published by Elsevier, Information Fusion 3, pp. 163-186, October 2001.

[40] XML, http://www.xml.com. Accessed November 2002.

[41] W. Yeong, et al., *“Lightweight Directory Access Protocol”*, RFC 1777, March 1995.

8 ABBREVATIONS

AAA	Authentication, Authorization, Accounting
ACM	Active Context Memory
AI	Artificial Intelligence
API	Application Programming Interface
CDXP	Context Data eXchange Protocol
CIB	Context Information Base
CKB	Context-aware Knowledge Base
FA	Foreign Agent
GPRS	General purpose Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
HA	Home Agent
ID	Identity
IETF	Internet Engineering Task Force
IP	Internet Protocol
IR	Infra Red
KB	Knowledge Base
LDAP	Lightweight Directory Access Protocol
MAC	Medium Access Control
PDA	Personal Digital Assistant
PKI	Public Key Infrastructure
RF	Radio Frequency
RFC	Request For Comment
S2CP	Sensor Sampling Control Protocol
SIP	Session Initiation Protocol
SIP-UA	SIP User Agent
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network
XML	eXtensible Markup Languare
XSP	eXtensible Service Protocol