NTNU
Norwegian University of
Science and Technology

Conceptualization and Design of a Context-Aware Platform for User-centric Applications

Ana Hristova

Master of Science in Communication Technology

Submission date:	June 2008
Supervisor:	Peter Herrmann, ITEM
Co-supervisor:	José R. Casar, Universidad Politecnica de Madrid
	Johan Montelius, The Royal Institute of Technology

Norwegian University of Science and Technology
Department of Telematics

Problem Description

The goal of this thesis is to conceive a platform prototype to provide context-aware services in a given mobile computing environment, namely advanced e-healthcare. Conclusions should be drawn for architectures, to deal with general mobile, location sensitive and context dependent services.

The project consists of:
o A study of the characteristics and architecture of well-known general purpose platforms for context-aware applications (Context Toolkit, CoBrA, CMF and Gaia).
o An analysis of the application field and identification of the parameters which describe context.
o An analysis of the Context Toolkit and proposals for its future extension.
o Design of an architecture to go from context sensing to context monitoring to final context awareness. Determining the functionalities of the platform's components.
o Implementation of a prototype in Java. The prototype should implement a subset of functionalities and interface with a location-providing platform.
o Experiments to determine the suitability of the architectural issues.
o Making reasonable tuning changes, conclusions and generalization to other environments.

Assignment given: 15. January 2008
Supervisor: Peter Herrmann, ITEM

Abstract

With the appearance and expansion of mobile devices, ubiquitous computing is becoming more popular nowadays and the user and his tasks are becoming the focus of application development. One area of ubiquitous computing is composed by the context-aware systems, systems where applications are designed to react to the constant changes in the environment. The heterogeneity of the different domains where context is a key parameter has generated different approaches for context acquisition and modeling. Thus, a number of platforms have been developed in order to alleviate the process of application development and to set a common practice for building applications and services.

This thesis studies the different platforms, focusing on the Context Toolkit. It examines the design principles, context representation, context acquiring methods and context handling. By developing services for a smart home, it explores the mechanisms applied in the framework and evaluates it. Furthermore, proposals for improving the Context Toolkit's functionality and the application's performance are given, by introducing the concept of quality of context and several enhancements regarding the resource discovery mechanism. Consequently, after gaining hands-on experience with developing a sample context-aware application and analyzing the Context Toolkit, a context-aware platform design is proposed which offers a complete solution defining all the necessary components and their interactions.

Preface

This thesis is done as the final work of the Master's of Science program in Security and Mobile Computing (NordSecMob) attended at the Royal Institute of Technology (KTH) and the Norwegian University of Science and Technology (NTNU). It has been performed remotely at the Technical University of Madrid (UPM) in the spring semester 2008.

The thesis has been supervised by Professor José R. Casar and PhD Ana Bernardos from UPM, while the academic people responsible were Professor Peter Herrmann from NTNU and Professor Johan Montelius from KTH. I would like to thank them for the help and support provided while working on the thesis, especially Ana Bernardos and Professor Casar for offering the valuable topics, suggestions and guidelines for the thesis which lead the work to a successful end.

Most of all I would like to thank my parents for their support throughout the master program, and Aleks and Borche who always encourage me in everything that I do. Without them I wouldn't have completed the thesis project. Furthermore, I also want to thank my friends for all the wonderful moments we've shared during the studying years: Nate, Andrijana, Ivica, Gaby, Daniel, Ines, Sara, Ana, Blagica, Mila, Aleksandar, Eli, Ana and Tomi.

Madrid, June 2008

Ana Hristova

Contents

List of Figures

List of Tables

List of Abbreviations

AHCS	Ambient Home Care System
CC/PP	Composite Capabilities/Preference Profile
CoBrA	Context Broker Architecture
CORBA	Common Object Request Broker Architecture
CSCC	Computer Supported Coordinated Care
CTK	Context Toolkit
ECG	Electrocardiogram
HMM	Hidden Markov Model
HTTP	Hypertext Transfer Protocol
J2EE	Java 2 Platform Enterprise Edition
J2SE	Java 2 Standard Edition
NFC	Near Field Communication
ORM	Object-Role Modeling
OWL	Web Ontology Language
QoC	Quality of Context
PDA	Personal Digital Assistant
RDF	Resource Description Framework
RFID	Radio-frequency Identification
RMI	Remote Method Invocation
RSS	Received Signal Strength
SD	Secure Digital
SMS	Short Message Service
SQL	Structured Query Language
SOAP	Simple Object Access Protocol
TCP/IP	Transmission Control Protocol/ Internet Protocol
UAProf	User Agent Profile
UDDI	Universal Description Discovery and Integration
UML	Unified Modeling Language
WLAN	Wireless Local Area Network
WSDL	Web Services Description Language
XML	Extensible Markup Language

1. Introduction

With the appearance and expansion of mobile devices, ubiquitous computing is becoming more popular nowadays and the user and his tasks are becoming the focus of application development. One area of ubiquitous computing is composed by the context-aware systems, systems where applications are designed to react to constant changes in the environment. However, the importance of context-based systems extends far beyond systems that are designed around information about the location, user identity, device capability, and services related to this information. Moreover these systems can acquire data for the biomedical functions of people or virtual data coming from software applications, which combined in a new way, can be used in a domain that can improve a person's wellbeing, such as healthcare.

Context awareness, considered as a basic property in the future mobile applications, has gained a momentum in the last few years. The heterogeneity of the different domains where context is a key parameter has generated different approaches for context acquisition and modeling. Thus, a number of platforms have been developed in order to alleviate the process of application development and to set a common practice for building applications and services. They address different architectures, design principles, context representation, sensing approaches, and handling context. Study of the state-of-the-art architectures is an inevitable step for getting better understanding of the problems that application developers face. This, together with the definition of the parameters that describe the context, can lead to inferring a better way of managing context and making suggestions for extensions of the Context Toolkit, a context-aware platform chosen for analysis, which would increase the functionalities and aid the developers in modeling and supporting context aware applications.

Thus, by examining the structure of the Context Toolkit, this project presents ideas, methods and issues that will lead to a new design of a conceptual model of a context aware platform that will ease the task of adaptive context-aware applications development and will increase its acceptance.

1.1 Problem definition

There are number of existing context aware frameworks that approach the application design from different perspectives. The goal of this master thesis is to study the existing context-aware platforms, particularly the Context Toolkit, analyze the architecture and context modeling. By developing services in a smart home, this thesis critically analysis the Context Toolkit platform and examines the design practices applied in the framework. Furthermore propose ways that will help building applications, by increasing their quality and reducing the complexity of their creation and aggregation.

In this project we narrow the scope to several issues. We explore the quality of the captured data i.e. quality of information that is used as context information, including: precision, accuracy, freshness, resolution and reliability, and outline several design issues and challenges in the area. Furthermore, we study the resource discovery process imposed from the decoupling of the application from the actual process of acquiring context information. Proposals to improve the current design of the resource discovery are examined together with the option for further decoupling of the acquisition process from the rest of the infrastructure.

By studying these issues the goal is to draw conclusions about best practices in this domain, and design a context aware platform that can establish the basis for its further development and future implementation.

The main problem statements in this project can be outlined as follows:

- Analyze general approaches for building a context aware systems and design principles;
- Define the strengths and weaknesses of the Context Toolkit;
- Infer enhancements and enrichments that could be added to the Context Toolkit for making more easy and functional way of building complex applications;
  - Examine the quality of captured sensor data handling and correspondingly define suitable evaluation parameters;
  - Analyze the resource discovery, a mechanism for automatically subscribing to appropriate widgets, and propose improvements;
- From the performed analysis propose a design of a context aware platform and motivate the decisions taken;
- Define and create a smart home application that will ease the life of elderly or disabled people;
- Experiment in order to determine the suitability of the architectural issues and make appropriate tuning changes and draw conclusions;

1.2 Motivation

There are several reasons for working on this master thesis. On one hand there are a number of platforms that enable developing context aware applications. They defer in architecture, sensing, representation of context data, resource discovery, security and privacy issues, keeping track of historical data, target group they aim towards to etc. In this project we closely examine the Context Toolkit, infer possible add-ons that would increase and improve its functionality and make it suitable for developing more complex and complete context aware applications. Then, we identify the need for a design of a new context-aware platform that will make use of the aspects analyzed previously and will encompass the enrichments outlined. Therefore this report can set the ground for future investigation and can further be used as a cornerstone and give directions for design of better and generally accepted solutions. On the other hand, working on this project gives me the chance to get in-depth knowledge and hands-on experience of a hot topic that will evolve, improve, develop in the years to come and eventually will become inevitable part of normal way of living.

1.3 Report organization

This report analyzes the context aware systems' domain and conceives ideas for a context aware platform that encompasses the modifications proposed. Following is the structure of the report:

Chapter 2
Chapter 2 gives an overview of context aware computing. It defines the concept, introduces the application domains, the components found in a context aware system, challenges in this area and shortly discusses several context aware platforms.

Chapter 3
Chapter 3 evaluates one of the context aware platforms, the Context Toolkit. It describes its architecture and possible ways of building an application, the components involved, the flow of actions and gives critical analysis of its strengths and weaknesses.

Chapter 4
Chapter 4 proposes several enhancements of the Context Toolkit that would enrich the context awareness and enable easy development of complex context aware applications. It speaks about extending the resource discovery mechanism and examines the definition of a possible quality of context evaluation parameters that would describe the widgets and would further influence on improving the performance.

Chapter 5
In this section we draw conclusions on best practices in the domain and we design a context aware platform that can address some of the issues analyzed before and deal with new ones. Chapter 5 presents a proposal for a context aware platform based on the in-depth study conducted on the Context Toolkit.

Chapter 6
Chapter 6 addresses the need for support of the life of elderly and gives a background of the related work in this area. A service is proposed and a prototype for a smart home application is developed. The details about the structure and implementation of the prototype are described in this chapter.

Chapter 7
Finally, chapter 7 offers conclusions from the performed study and provides proposals for future work.

2. Context Aware Computing

2.1 Definition of context

With the appearance of mobile devices, ubiquitous systems have gained popularity and application developers have increasingly focused on making applications that target PDAs, mobile phones, notebooks, smart phones etc and the way to use these devices and new technologies to aid the user in performing its daily tasks. This has resulted in rapid integration of these devices in a person's day-to-day life in a manner that the user and his tasks have been placed in the forepart and are central for service development, suppressing the devices, their connectivity and other technical issues.

One area of pervasive computing is context awareness, a concept first introduced by Schilit and Theimer in 1994 referring to a new class of applications that are aware of the context where they are executed. Focusing on active map service that provides information about the located objects and how they change in time, they use the term context to refer to location and they perceive context-awareness as the ability of mobile applications to discover and react to changes of the environment where they are located. [1]

Schilit later gives a user-centered definition of context stating that context can be characterized by several important aspects: where you are, who are you with, and what resources are nearby. [2] Therefore information about the location is just one segment of the context information, and context is presented as a relation between the user and the environment. Opposite of this Lieberman et al. propose another, application centered definition to context, specifying that context can be considered to be everything that affects the computation except the explicit input and output. [3] With this they define the context awareness as the ability of the system to take action as a response to the context gathered and not just adjust the application interface according to the context and the direct command the user is entering. Consequently many other scientists and researchers have continued working in the domain, expanding and redefining the concept of context, but these views remain to represent two distinct approaches of context awareness in human computer interaction and its further development evolves following these two paths.

When referring to context different context parameters are being given different preference. Context parameters that researches and application developers often list are: location, time, environmental parameters, user activity, device capabilities, identity, network capacity etc. Application developers look into the 'who', 'what', 'where' and 'when' of certain entities and by its analysis they reason about the 'why' of a given occurrence, and program the application logic. However a common way to classify context is by differentiating between physical and logical context.

- Physical context is the one that can be measured by hardware sensors, such as: light, temperature, humidity, sound, movement, location etc.
- Logical context is the one that is inferred by monitoring the user's behavior, his tasks, his physical and emotional state etc.

It very difficult to include each context category in an application and coordinate their interactions, therefore an extraction and identification of the parameters of interest is the basic step when considering the context elements.

2.2 Definition of context-awareness

Schilit et al. [4] have also first discussed about context awareness as the capability of a system to *"adapt according to the location of use, the collection of nearby people, hosts, accessible devices, as well as to changes to such things over time"*. An application with these abilities will be able to sense the environment and correspondingly react to its changes. On the other hand, Pascoe et al. [5] define context-awareness as the ability to detect, gather, interpret and react to context changes in the user's surrounding and changes in its device.

Schilit et al. outline four categories of context aware applications shown in Table 1. The categories are divided according to two properties: whether the system gathers information or executes an action, and whether this is done manually or automatically.

	Manual	**Automatic**
Information	Proximate selection and context information	Automatic contextual reconfiguration
Command	Contextual commands	Context-triggered actions

Table 1: Context-aware application categories depending on Schilit et al. [4]

Proximate selection is when applications manually request information based on is current disposable context, while the automatic contextual reconfiguration category encompasses applications that automatically retrieve information based on the current context. Contextual command applications are the one that execute commands for the user when they are manually instructed to do so and context triggered applications refer to applications that provide automatic execution of commands depending on available context.

Pascoe et al. define context-aware application categories that can match some of the already specified by Schilit, but they emphasize that they allow a context-aware application that will fit in each of this categories and will contain all od its characteristics and will not be limited just to this properties.

Dey and Abowd define a context-aware system as "a system *that uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task*". [6] This definition differentiates from the previous one because it only requires response to context, allowing its gathering and interpretation to be performed by other components, but the definition is given in a broad sense so that it encompasses already developed context-aware applications. These authors outline three categories of context aware applications that deal with presentation of information, automatic execution and tagging of context information, and represent a combination of the categories previously mentioned.

2.3 Application domain of context-aware services

As technology advances and context awareness computing evolves, the number of areas where context awareness is applied increases. Following some of them are listed:

- *Airports:* Emergency response solutions are based on context aware systems which can automatically deliver immediate security alerts to the relevant personnel at the airport, such as: security, maintenance, fire department, police etc. Services directly aimed to passengers, that are closely connected to the location of the passenger are developed as well. They adapt the notifications that are sent to the passengers' mobile devices depending on the users' location and inform them about shopping zones nearby, exits, gates, arrivals and departures.

- *Smart homes:* Context awareness integrated with technology and services through home networking increases the quality of living and can help disabled or elderly people lead safe and independent lives in their homes. It can provide:
    - *Security functions:* Generation of alerts in case of fire, gas leakage, open tap, detection of unknown persons present in the home. These alerts are sent to the user as graphical or audio information, but can be directly sent to the appropriate institution as well e.g. police, fire department etc.
    - *Appliances functions:* Monitoring and control of the appliances in the household such as turning on/off the light, opening/closing the window, turning on/off the air conditioner and heater (tasks that the householders are unable to do), improving energy management etc.
    - *Environmental information:* Overview of the situation in the domicile and the environmental parameters in each room.
    - *Healthcare functions:* Constant monitoring of the person's biomedical functions, sending reminders about daily tasks he needs to perform e.g. medication intake and sending alerts to remote caregivers if an emergency situation is detected.

- *Leisure/Entertainment:* This application area is closely associated with the user's location. Namely, a variety of services have been lately developed that are activated and offered to the user and provide information about nearby restaurants, cinemas, theatres, festivals, concert events, shops, maps and other information related to the area where the user is present.

- *Hospitals/Healthcare:* Context awareness can aid in improving the quality of service in hospitals by implementing interactive hospitals where doctors and nurses are provided information about the diagnosis, the treatment and medial history of patients in their proximity and they are able to carry out video conferences which enable collaborative feedback, discussion and diagnosis with other doctors. These interactive hospitals would also support reminders to patients for medication intake, verification that the nurse is carrying the right medicine, closely monitor the patient's state and raise emergency alerts if necessary, carry out video conferences between a doctor and a patient at home etc.

- *Museums and fairs:* Context awareness is closely related to detecting user's position in a building and guiding them through it. It is commonly used to run applications on portable devices which are location sensitive and provide audio and video information to the user about art pieces in a museum or a gallery, give directions about the way to reach a specific place of interest during a fair, send notifications about stands that might be of the user's interest depending on his already known preferences and enable adaptation of music, sounds and other effects as the user changes the location.

- *Offices:* Usual services that can be found beneficial in office work is locating the position of employees in the company building, additional information about the office where they are located (whether there is a meeting in progress, how many people are present), status of the equipment in the company, environmental parameters measured in the office where a user is present, inferring the activity of the user depending on pending calendar entries etc.

2.4 Design principles of context aware systems

In this section an overview of different architectural approaches when building a context-aware framework is given, together with an emphasize of the units found in most frameworks: sensing infrastructure, inference engine, storage of historical data and several methods for context modeling.

2.4.1 Architectural approaches

There are a number of different architectural designs when building a context-aware system. The selection of the one that an application will adopt should encompass decisions about issues such as scalability, number of users, reusability, location of the sensors, the method of context-data acquisition or the type of devices disposable for usage. Winograd describes three different approaches for managing context and context-aware system components. [7]

- *Widgets*: Widgets are software components that give a uniform way of handling device specific drivers and are coordinated by a widget manager. They hide low level details about the sensing by encapsulating the driver details and physical connection ports, and furthermore provide contextual data directly to the applications. Instead of implementing notifications in distinct way for each device driver incorporated, this approach offers a consistent way of sending messages to widgets, and callbacks - when a certain event is registered by the sensor. A drawback in this approach is that it is susceptible to component failures that might impact the normal functioning of the system.

- *Networked services*: Unlike the previous model, where a centralized component that keeps track of existing widgets is introduced, this model approaches context acquisition sources as independent services. Although locating appropriate processes can be more expensive when it comes to time and communication, with careful selection of specialized protocols this can be surpassed. Applications find services of interest by using discovery techniques (most usually in the same network) and describing the desired process. Although less efficient due to additional networking costs, this model provides robustness. [8]

- *Blackboard model:* Unlike the process-oriented approaches earlier, this approach follows the data-centric view. Instead of sending requests and process callbacks, in this model processes send messages to a common board i.e. a context server, and subscribe to get messages that match certain criteria. All messages pass the centralized server and require two hops in order to reach the application, which decreases the communication efficiency, but on the other hand provides easy configuration and alleviates the addition of new context sensors.

Similarly, Chen introduces three architecture designs for context acquisition: [9]

- *Direct sensor access*: In this approach, there is no division between the sensor data acquisition and its logical processing; everything is tightly coupled in the application. This method was earlier adopted but because of its poor reusability nowadays it is not utilized as much.

- *Middleware architecture*: Good software practices and design impose modularity and separate the business logic from data acquisition. Hence this approach introduces separation of these logics and their implementation in different layers, which furthermore increases the reusability of the sensing infrastructure.

- *Context server*: This design allows access of sensor data concurrently by multiple clients at a time through collecting all sensor data by the server. On the server side, if necessary additional processing is performed, alleviating the mobile device by decreasing the amount of computational power needed for obtaining certain data.

To some extend the categories in these classifications overlap with each other and should not be taken rigorously since they intersect and a combination of two sometimes might be the most suitable design.

2.4.2 Sensing infrastructure

In order to improve reusability of applications and to alleviate the process of building them, a common practice is to separate the sensing logic from the rest of the system. Hence a common basic module for all architectures is the sensing layer.

Context-acquisition begins with sensors. Sensor technology has significantly improved in the past years powered by new solutions that increase quality, reliability, increase the number of parameters that can be measured (temperature, pressure, humidity, acceleration, motion, location, blood pressure etc.) and minimize energy consumption, size and cost. Sensors are sources of contextual data, but the term does not only infer hardware sensors; it encompasses any source of information which provides contextual data and improves the description of a real situation.

Three types of sensors can be identified depending on the way the data is measured and acquired: physical, virtual and logical. [10]

- Physical sensors are hardware sensors which capture physical data. They are the ones that are most common and most widely spread when information about the environment, people, objects, or body functions is needed. They range from temperature sensors, microphones, touch sensors, light sensors, location sensors, motion sensors, biosensors for measuring the human body's functions etc.

- Virtual sensors refer to context data that comes from applications and these sensors are usually software processes. [11] They observe application events, operating system events or network events. For example, a virtual sensor may include schedule's info, calendar's entries and e-mail for determining a user location; then, the level of network congestion and the abilities of the user's device.

- Logical sensors include data inferred from both, physical and virtual, sensors. The information derived with combination of these data sources is usually used to resolve complex tasks e.g. by combining the context data sensed which measure the human body's functions and the person's medical history stored in a database in the hospital, a logical sensor can determine the value of a parameter used as an indicator for the seriousness and urgency of some anomaly detected.

These sensors include explicit information given by the user, implicit information about the abilities of the devices; existing information obtained by other services and captured data by physical sensors. Integrated together, they constitute the sensing module of the architectural model which provides information to the rest of the system.

2.4.3 Context Modeling

After context data acquisition, in order to further efficiently use the obtained data, it needs to be represented and/or stored in appropriate form suitable for further processing. The context model chosen is determined from the general approach of the whole context aware framework and the data processing methods selected. Several most frequently used modeling approaches are:

- *Key-value model*: This is the simplest modeling technique which is widely utilized. It represents contextual information with a key-value pairs which are later used by some matching techniques to perform sensor data discovery.(e.g. TEMPERATURE = 25)

- *Logic based model*: This model is based on facts, expressions and rules. [10] A logic based system manipulates with the elemental items of this model and infers higher level logic by utilizing the already defined rules to deduce new facts.

- *Ontology based model*: Ontology is a description of concepts and their relationships. However, it is not only a classification of concepts; it also includes higher relationships between them and enables interaction between systems that have compatible ontologies. One way of implementing these ontologies is by using the Web Onotlogy Language (OWL) which consists of a set of classes, class hierarchies, set of property assertions, constraints on these elements, and types of permitted relationships between them. [12] While, another alternative is using a knowledge representation language - the Resource Description Framework (RDF). This is a promising model because of the possibility to apply reasoning techniques. [8]

- *Graphical models*: Using the Unified Modeling Language (UML) is another way of representing context, as well as using an extension of the Object-Role Modeling (ORM) with context information. [10]

- *Object*-oriented models: Object-oriented design of context benefits from the common properties object-oriented programming, such as inheritance, encapsulation, reuse, and polymorphism. An architecture exists that uses a class ContextObject, which is inherited by other context–specific classes which implement the common abstract methods, convert data streams to context objects and vice versa, and provide well known interfaces to access the context's logic.

- *Markup languages*: These models have hierarchical structure composed of tags and attributes. User Agent Profile (UAProf) and Composite Capabilities/Preference Profile (CC/PP) are some of the specifications that describe the capabilities of mobile devices and different user agents, enabling the content providers to produce and deliver content suitable for each request.

Research has shown that the most complete modeling technique is the ontology model. It is the most expressive and meets the requirements of most of the systems. [10] However because each architecture tries to meet different goals, this model is not always chosen in a given architecture.

2.4.4 Reasoning methodologies

A challenge in context-aware computing is to manage context appropriately and use it in an intelligent way, by using captured data from different sources in order to deduce new information. Interpreting low-level context data into a higher level ones can be done by using ontologies and logic reasoning, or by using one of the following techniques for inferring context and situations in context aware systems: [13]

- *Artificial Neural Networks*: They perform well regardless the noise level produced when capturing sensor data and they support unsupervised learning of input data. New context data can be easily included, and the algorithm will be able to adapt the context from the new input by recalculating the internal representation of the contexts. [13] Therefore unpredictable context that hasn't been introduced to the system yet can be detected, complex relationships can be modeled and patterns in behavior can be found.

- *Bayesian Networks*: Bayesian network is a probabilistic graphical model that represents a set of variables (known and random context parameters) and their probabilistic relationships. Given all the possible context types and values acquired by the sensors, a Bayesian network would be able to calculate the probability for occurrence of some situation or the probable state of set of variables.

- *Hidden Markov Models*: This is a statistical model where the system that is represented is assumed to be a Markov process with unknown context parameters and the aim is to find out the unknown parameters (context types or an inference of a higher-level situation) from the context data on disposal. HMM requires training phase in order to initially categorize activities and builds statistical memory of sequences of events that are reliable and robust to changes and provide higher-level knowledge deduction. It is mostly used for modeling human behavior, because it is able to recognize sequences of activities. [13]

- *Fuzzy logic*: This is a method for approximate rather than precise reasoning, that ranges the probability of every statement's in the interval between 0 and 1 and is not limited only to two values (true and false).

- *Dempster-Shafer Theory and evidential reasoning*: This is a mathematical theory of evidence where evidence is associated with multiple possible events e.g. sets of events. [14] It is an approach that models judgments with uncertainty.

Although several methodologies have been outlined, none of them can be pointed out as most efficient or beneficial, since they all address different issues. Therefore selection of one is guided by the framework's requirements.

2.4.5 Historical data

Keeping a record of historical data might be useful for the section discussed previously. It can help the system to detect patterns, establish tendencies and predict future behavior of a context parameter. Historical data enable implementation of intelligent learning algorithms which would provide flexible and easy adjustable context-aware services.

Main issues regarding keeping storage with historical data is the memory usage. Since this storage would be memory resource demanding, a storage component needs to be allocated. Context data is usually written in a database which with the use of the Structured Query Language (SQL) enables constructing queries that will enable revealing more complex patterns and detect relationships on a higher level. Context – aware frameworks may also provide an interface for accessing this database and the possibility to store other entities apart from the context data, such as rules, expressions, and higher logic representation.

This storage component is also know as a context knowledge base and can be centralized or distributed. If it is centralized then we have one context server that collects all the sensor info and dispatches it to the appropriate applications, while in the case of a distributed storage components, each application in need for historical data retrieves it from a number of locations without any centralized support.

2.5 Overview of Context Aware Frameworks

In this section different context aware frameworks are shortly introduced, discussed and compared on common criteria. They adopt different architectural styles mainly driven by the context acquisition, different method of context representation, processing logics and reasoning engines, and at times distinct storage approaches and communication patterns.

The following frameworks will be shortly described:

- Context Toolkit
- Context Broker Architecture
- Context Management Framework
- Gaia

2.5.1 Context Toolkit

The Context Toolkit adopts the widget architecture style and its main purpose is to alleviate the development of context-aware applications by providing a common framework that will be taken as a base for further uniform development of services. It consists of widgets which comply with a distributed allocation managed by a central component called a discoverer that keeps record of all running widgets in the network.

A central component in this architecture is the widget, which is a software component that encapsulates the sensor specific communications details and notifies subscribed applications when the context changes. Other components include aggregators, which are extensions of widgets that combine the context info of several widgets, decreasing the processing requirement on the application side; and interpreters which are components that translate certain information from a low-level to higher-level data suitable for processing.

The communication between two components is peer-to-peer, without any direct intermediaries and most of the components inherit the communication capabilities from a superclass called BaseObject, which provides the ability for interconnection between the context aware system's components. The framework adopts the key-value pairs (encoded using XML for transmission) context modeling, has no reasoning engine and has storage support.

This architecture is further analyzed in details in Chapter 3.

2.5.2 Context Broker Architecture – CoBrA

COBRA shown in Figure 1 is an agent based architecture that supports context-aware systems in smart spaces (physical places-meeting rooms, homes, vehicles that are equipped with intelligent systems that enable ubiquitous computing). [15] Its central component is the Context Broker, which maintains and manages a shared contextual model on behalf of the collection of agents and is consisted of four main components: Context Knowledge Base, Context Reasoning Engine, Context Acquisition Module and Privacy Management Module.

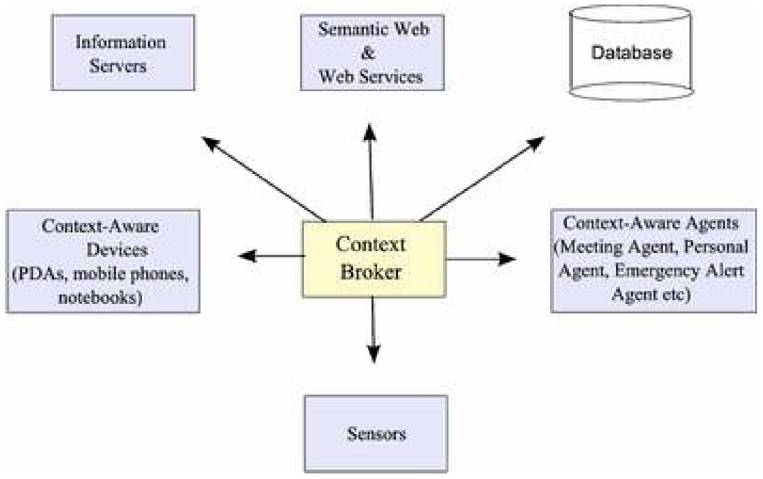

Information Servers
Semantic Web & Web Services
Database
Context-Aware Devices (PDAs, mobile phones, notebooks)
Context Broker
Context-Aware Agents (Meeting Agent, Personal Agent, Emergency Alert Agent etc)
Sensors

Figure 1: CoBrA Architecture

Context is modeled with ontology, which is integrated with a rule based inference engine, and also, the architecture includes its own policy language, Rei, which controls access and enables security and privacy protection. [10]

2.5.3 Context Management Framework

The Context Management Framework is shown in Figure 2 and is an example of a framework that adopts the blackboard architectural design. It is consisted of several entities: context manager, resource servers, context recognition services, change detection server, security component and an application.

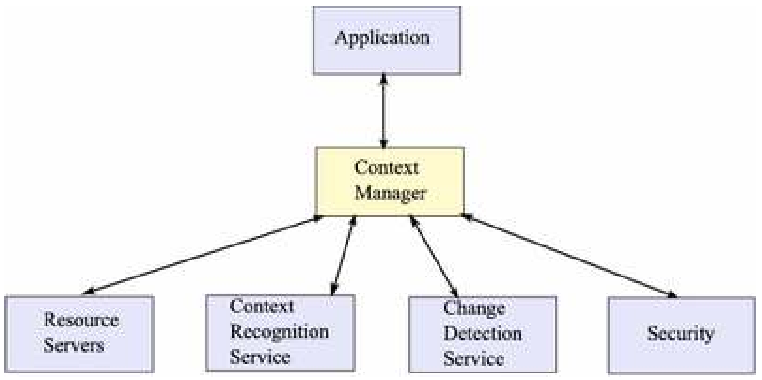


Figure 2: CMF Architecture

The context manager is the central component that manages the blackboard and acts as a central server, it processes the context information acquired from many sources, infers higher level information from them and delivers them to its clients. Data acquisition is performed by the resource servers, while the context recognition services are used on demand by the context manager to deduce complex data out of simple context entities.

What distinguishes this framework from the others, apart from the architectural design, is the advanced way of handling context data represented with ontologies and the usage of fuzzy logic to build higher-level data. However, a drawback of this architecture is that the context manager presents a single point of failure, since application's normal functioning depends directly from it.

2.5.4 Gaia

The Gaia project illustrated in Figure 3 is a distributed middleware infrastructure which extends the typical operating system concept. It is intended to coordinate the development and execution of mobile applications for active spaces, typically a single room and provides the following functionalities: program execution, I/O operations, file-system access, communications, error detection and resource allocation. [16]

It is structured as a traditional file system with a kernel composing the core of the system and applications built on top of it which provide specific services. Here, context is acquired by context providers, classified by the type of information they gather, therefore the architecture distinguishes between three types of context: location, context and events which enables reasoning on a higher level and inferring activities. The framework's processing is implemented into the Context Service Module which performs first order logical operations such as "and" and "or", and applications query this module for obtaining specific context information.

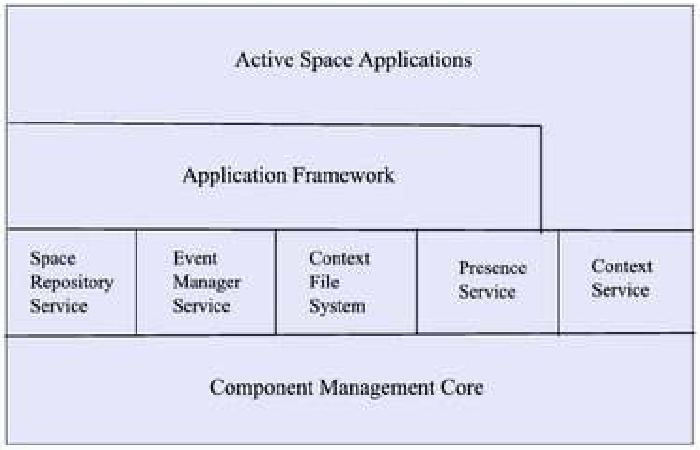


Figure 3: Architecture of Gaia

Context is modeled in a special manner, by using 4-ary predicates consisting of: (<Context Type>, <Subject>, <Relater>, <Object>), a notation that is used for representing context and defining rules. [10] In addition, context is represented as directory, where the path represents the context type and the value (e.g. location = kitchen is represented as: /location:/kitchen/).

2.5.5 Comparison of the context-aware frameworks

On the following table a breakdown of the described context-aware frameworks is given over a set of common criteria, such as: architecture, sensing, contextual model, processing, resource discovery, storage of historical data, and security and privacy.

	Context Toolkit	**CoBra**	**Context Management Framework**	**Gaia**
Architecture	Widget based	Agent based, centralized context broker	Blackboard based	MVC extended
Sensing	Context widgets	Context acquisition module	Resource servers	Context providers
Context model	Attribute-value pairs	Ontologies (OWL)	Ontologies (RDF)	4-ary predicates
Context processing	Context interpretation and aggregation	Reasoning based on OWL schema, rules and inference engine; knowledge base	Context recognition service and fuzzy logic to build higher-level concepts	Context-service module (first-order logic)
Resource discovery	Discoverer component	N/A	Resource servers and subscription mechanism	Discovery service

Historical data	Available	Available	N/A	Available
Security and privacy	Context ownership	Rei policy language	N/A	Supported(e.g secure tracking, location privacy, access control)

Table 2: Comparison of the context-aware frameworks [10]

A common property for all solutions analyzed is the decoupling between the sensing infrastructure and the rest of the system, which increases reusability of context sources within the system. On the other hand each framework has its own format for context representation and uses different communication principles, which in turn makes the interconnection of the frameworks difficult to be accomplished and disables developers to reuse services based on another framework.

Furthermore, almost all systems have well developed resource discovery component and support storage of historical data which later aids the context reasoning. Security and privacy is present in most of the systems, but still in the form of basic security mechanisms which should further be strengthened.

2.6 Challenges in context awareness

Due to the existence of a variety of different context aware frameworks a series of challenges in the domain of context-aware computing arise. They address sensor issues, intelligence and inference, architectural design issues, privacy and ethical issues etc. Following they are analyzed in more details.

- Sensor issues
  - Determining the right number of sensors deployed in the sensor network is another challenge that engineers face. The amount of sensors deployed should be a trade off. On one hand as the number increases, the data they acquire is more accurate and reliable, but then the higher number of sensors deployed consumes computational power in the pre-processing engine and decreases the speed of the learning algorithms. Hence a number of sensors that would provide good performance should be chosen. This can be done through context data selection, which implies that the context data from all the sensors should not be captured. A basic filtering needs to be done and a subset of sensors should initially be chosen for recognizing context and use the redundant ones only in case of uncertainty. In this way, data transmission over the network would decrease together with the processing demand.

  - Automatic restarting of components is another challenge in this area. Often a component fails and it is later manually restarted, which requires close human observation and maintenance of the whole architecture. To overcome this, redundancy in the system can be introduced as well as software components that verify the normal operation of all critical entities in the system.

    - Handling context is sometimes difficult. Developers are forced to include context information which is made available from the context acquisition mechanisms – hardware and software sensors. This makes them adjust the application concept and its future development according to the sensor data obtainable, which in turn limits the scope of the applications that can be developed by guiding its design from this sensor-oriented approach. Therefore, a way of capturing and acquiring information about more parameters, which describe the situation around the user, should be developed.

    - A way to interact with the sensors and relevant actuators is an important point of interest. Feedback the sensors and dynamically managing them during the operational phase can improve the applications performance and the quality of the service offered. Hence, the need for developing a generic way to support this requirement.

- Intelligence and inference
    - Since new contexts are continuously added in the sensor network, continuous adaptive learning is an issue that draws attention. A way of autonomous learning without any user assistance needs to be added to the algorithms in use, so they can better learn the new contexts that are presented to the system, as well as preserve the knowledge for the already known ones.

    - Also, context inference is a complex task that requires a good mechanism for mapping simple captured context data to higher level one. This is not an easy task and requires using ontologies with logic reasoning, or probabilistic reasoning. Research is still ongoing to find the best method that will disregard errors, automatically adapt the system to new types of data, learn autonomously and reason correctly.

- Architectural design issues
    - Making context aware services and components between frameworks reusable, is something that the context-aware systems community should focus more on. At present there are no methods or tools that allow using a component developed with one framework for usage by another. The frameworks should be extended in order to support interoperability between its components and applications on heterogeneous platforms.
    - Service description and discovery is something that each framework handles differently, and it is not specific to context-awareness solely. Services should be described with a language that will alleviate the process of their discovery and the discovery technique should be such that will be able to easily match the service request with its description and would support service and context-acquisition entities relocation. [17]

    - Designing strategies to make the context-aware system work with different levels of user collaboration is needed in the systems. For example, self-adaptation of the system depending on the user preferences should be provided. If user preferences are not provided, the system should adjust and continue working without that info. Also, adjusting the location where some calculations are performed (client or server side) depending on the info needed to be calculated, is another architectural issue of interest.

  - Cooperative and distributed handling of context data should be added in the architectural designs developed. For example, a given device may be able to estimate some parameters that another device is not able to. A way of sharing some of these descriptors with other devices that need them, in order to increase their knowledge base, should be modeled in the architecture.

- Other issues
  - Security and privacy is an issue that hasn't been quite solved yet because its introduction in the system is twofold. As the amount of information which is on the application's disposal increases, the application decisions' quality increases as well. Applications will operate with more data and will be able to make more accurate decisions. On the other hand, the more information is available, the less privacy the user can sustain. Users should be able to control which information they are willing to reveal for application purposes and when they allow their access. This as a consequence decreases user-friendliness of any application, because it becomes cumbersome to constantly reconfigure it, and reduces its usability. Hence, as mentioned, this is an issue that needs good balance in order to preserve both benefits, respecting privacy of users on one side, and confidentiality and easy utilization on the other.

  - Although composition of simple context-aware services appears like an obvious and intuitive task, in reality integration of the components of these systems is not seamless as it seems. [18] It is a complex task whose difficulty lies in solving practical design and implementation problems that have not been predicted. Hence, a better understanding on services composition should be acquired and guidance with best practices should be proposed.

3. Analysis of the Context Toolkit

3.1 Introduction

The Context Toolkit is a platform for distributed context-aware systems. It was developed by three researchers, Daniel Salber, Anind Dey and Gregory Abowd, at the Georgia Institute of Technology in USA during five years and was mainly intended to aid developers when building context-aware applications by providing a suitable framework. [19]

The toolkit represents an implementation of formalism for describing context and its abstractions. Context is represented by data acquired from context widgets which encapsulates the communication with sensors, context combined from several widgets in an aggregator or context translated by context interpreters. The toolkit enables subscription and discovery mechanism so that these contextual data can be easily utilized and appropriate components easily located.

It is a JAVA-based framework deployed in a distributed infrastructure, but components in other programming languages exist as well, such as C++, Frontier, Visual Basic and Python, therefore allowing expansion of the possible application domain. The framework provides several services which will be further analyzed in this chapter: encapsulation of sensors, abstraction and of context data and combining them through interpreters, support of aggregation of context information, independence and persistence of context widgets, storing history of context data, sharing context data in a distributed infrastructure etc [15] Furthermore, the toolkit employs the key-value context model required for defining and storing context data. Attribute-value pairs are used to describe the context information and they are utilized for service discovery by matching the required data according to these pairs.

This section gives an overview of the Context Toolkit. It first describes several modes of application design that the toolkit provides and introduces the architecture, its main components and the way communication is performed between these components. Then, it gives a more detailed description of the sequence of actions that occur when initializing a context-aware application and other components required. Finally, it finishes with the critical analysis of the strengths and weaknesses of this framework.

3.2 Modes of Application Design

Different designs exist for context-aware systems that depend on several parameters like number of users, location of sensors, device capabilities etc. Context Toolkit encapsulates three different design principles of handling context and building an application which directly refers the manner of acquiring contextual information.

The first approach is a simple direct sensor access shown in Figure 4, something that is easily accomplished. This approach does not use any additional layer for acquiring and processing data. The context retrieval is tightly coupled with the application and sensor drivers are hardwired into the application. However several problems exist in this approach. It doesn't support good software design practices and imposes dealing with complex context acquisition of context. This results in poor reusability that leads to rare utilization. On the following picture this design can be examined closer.

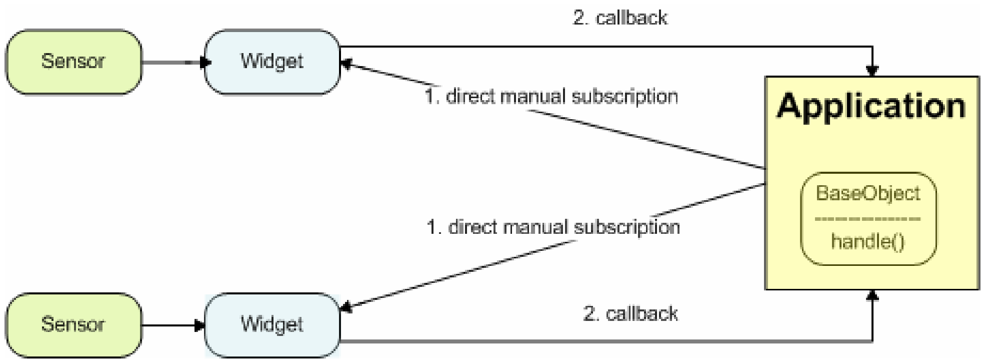


Figure 4: Traditional application design without a discoverer

The second approach shown in Figure 5 utilizes the middleware infrastructure principles. Here the business logic, the user interface and the sensing infrastructure are separated. This layered architecture enables hiding of the low-level sensing details and mostly this separation is accomplished by the discoverer component which acts as a registry where available CTK entities are registered. Applications do not contain details about the widgets they need to communicate with, in fact, they subscribe to the discoverer to dynamically locate the widgets. By this, automatic discovery of widgets

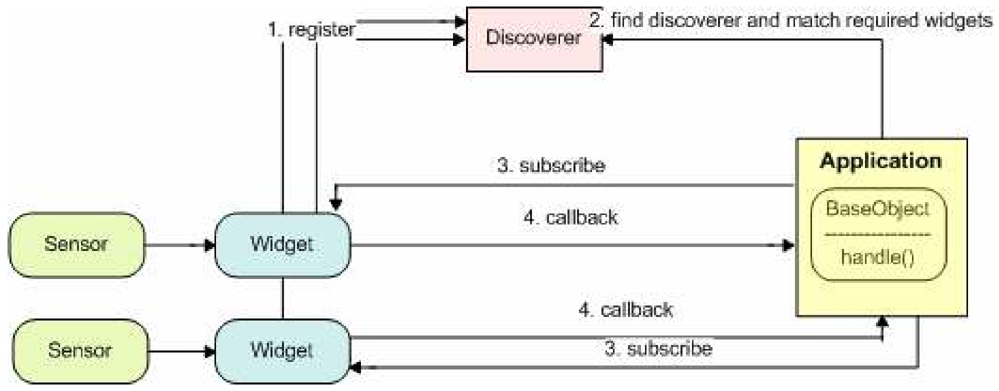


Figure 5: Application design with a discoverer

is provided, the sensing and application layers are divided and access to the remote data sources by multiple clients is enabled.

Further enhancement later added to this design by Newberger and Dey [20] addresses the issue of monitoring and controlling context-aware applications by introducing the concept of Enactor. This enhancement alleviates the designer access to the application state and behavior and allows developers to easily encapsulate application logic in this component. All enactors are managed by Enactor Subscription Manager which manages CTK subscriptions on behalf of the enactors by notifying and subscribing them to widgets that match the enactors' description. On Figure 6 the relation between different components in the third approach that encompasses enactors is presented.

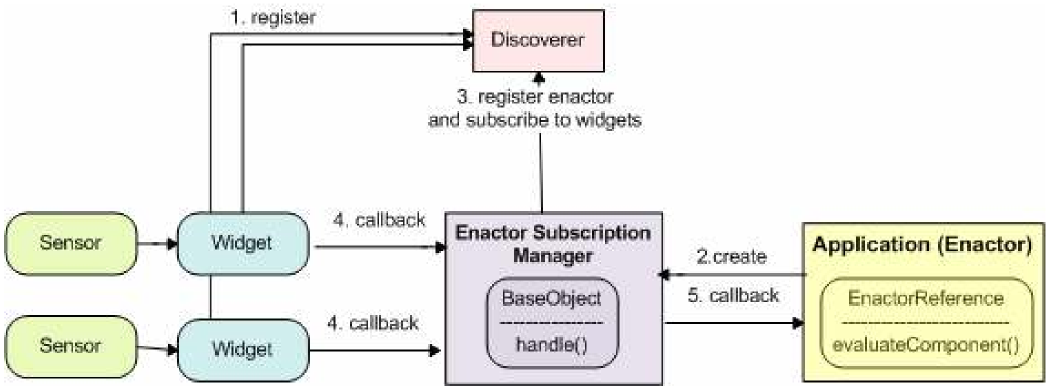


Figure 6: Enactor application design

A context-aware application will usually have several enactors, all encapsulating and processing different kind of context information, such as location enactor, temperature enactor, light enactor etc. But, although the whole application logic can be placed in only one enactor which manipulates with all context data, grouping the functionalities in different enactors enables sustaining modularity and increases its reusability.

3.3 Description of the Architecture

The Context Toolkit's architecture uses the object-oriented approach and consists of several main types of context abstractions further analyzed in this subsection and presented on Figure 7.

The relationship between different entities is presented by a configuration that consists of: two widgets, an aggregator, an interpreter, a service, a discoverer and two applications. All components' existence is independent of any application and its communication with other components is automatic and in line with known network protocols. Every component registers its characteristics to the discoverer, alleviating the discovery process for other components that want to subscribe to it. Therefore, aggregators can easily find appropriate widgets, interpreters can locate widgets, and applications can subscribe to suitable widgets, aggregators and interpreters.

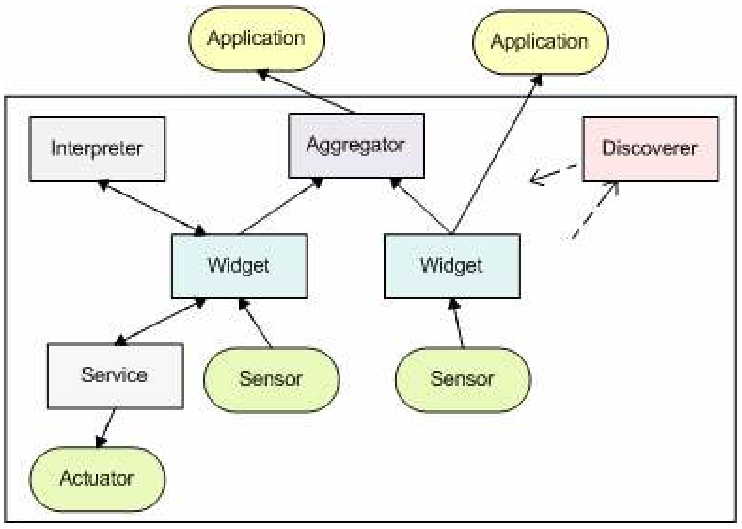


Figure 7: Relationship between Context Toolkit components

As shown on the Figure 7, sensors provide data to widgets. This data can be further translated by an interpreter to comply with a form that is more understandable for an application. The acquired data from several widgets can be aggregated in a single component that extracts and combines the logic for certain entity. Finally, this context is delivered to the application which further manipulates with it and develops the application logic.

Following each of these components is closely examined.

3.3.1 Widget

A context widget is a software component that enables applications to access to context information in the same way as GUI widgets enable the interaction between the user and the application. It encapsulates the context information and provides methods for accessing it.

Some of the benefits that widgets offer are:

- They hide the complexity of the actual sensors used by the application. [21]
- They provide uniform interface to applications;
- They can abstract desired context information that matches the conditions set in the application, e.g. a widget may notify an application when the temperature rises above certain level, but not about variations below that level. With this the widget does not report information that is not of interest for the application.
- They are built as separate building blocks that can be adjusted, combined and reused by multiple applications simultaneously.
- They provide context data on demand or when a change has occurred. More specifically they utilize callbacks that notify the application and the

subscribing components when an event has occurred and the context of interest has changed. Furthermore they can also be queried and pooled by applications.

Each widget class defines a set of constant and non-constant attributes that define the context that the sensors provide, and callbacks which are further delegated to the widget's subscribers. Also it contains the details about how to connect and query data from the sensor.

3.3.2 Aggregator

The aggregator or server, as it is called in the toolkit, combines context information from multiple widgets that are logically connected into a joined repository. It gathers contextual information about a specific entity, like a person or a certain location, and directly provides information related to it or builds higher-level context objects. This alleviates the job of an application developer because it allows the application to subscribe only to a single object that encapsulates the logic of many. Therefore inferring a higher level of context is easier if all pieces of information are available in an aggregator.

The aggregator subscribes to every widget of interest by providing their names and acts as a proxy to the application. [22] This component inherits all the methods and properties that widgets have, and therefore can be subscribed to and be pooled. In addition it automatically inherits all the callbacks and attributes of the widgets it is subscribed to, therefore no special declaration is needed. However, in order to better describe the aggregator's behavior other callbacks and attributes can be added as well.

3.3.3 Interpreter

Another abstraction introduced in the toolkit is the Interpreter. Interpreters transform context information by raising the level of abstraction. They take a piece of information and interpret it into another form or meaning. Usually interpretation has been performed by applications, but by introducing this concept this functionality is separated from the application layer and reusability by multiple applications, widgets, servers and other interpreters is enabled.

When creating an interpreter the developer provides the following information: the incoming and outgoing attributes and an implementation of the InterpretData() method that does that translation.

3.3.4 Services

Apart from aiding the process of acquiring context data, the toolkit defines objects that are able to perform some action in return. Services are components that can execute actions on behalf of the application. This concept has not been introduced from the beginning of the definition of the toolkit; therefore it hasn't been explored as much. The idea is to support output just as collection of input is supported, and by that to influence the environment using an actuator. Similar to widgets, the details about how this is done and the communication with the actuator are hidden from the application and encapsulated in this component. [23] An example would be to change the temperature in a room, after a certain level is reached.

Services can be synchronous and asynchronous. In the case of synchronous services, the service is considered complete when a reply about the status of the successful execution is being sent back to the application. While in the case of asynchronous services, the service notifies the application that the service has been started and feedbacks about results when they will be available.

3.3.5 Discoverer

The discoverer is one of the most important components in the framework because it allows adding a discovery system to the toolkit. It contains a registry of what capabilities the platform provides and registers all the widgets, aggregators, interpreters, and services that are available for use by applications. It contains information about how to contact the widget, the id of the widget, then its' hostname, port number; and also information about what kind of context the component provides, such as: callbacks, constant, nonconstant attributes, input and output attributes. The Discoverer inherits the widget's class attributes and methods and its basic functionality, therefore widgets can subscribe to it, query it and subscribe to notifications.

When a widget is created, it does not know the other components. It can subscribe to another component of interest if it knows its hostname, port number and name. Therefore when created, the widget finds the discoverer, using a multicast communication, and it registers to it by having the discoverer store its description. So, whenever an aggregator or application needs to contact other contextual component they can find the appropriate information by querying the discoverer and matching the component by some or all of its description parameters.

Applications can query the discoverer in two ways: by performing a lookup that resembles the yellow pages lookup and another one that is similar to the white pages lookup. [21] The white pages lookup allows finding a component by stating its name and id, while the yellow pages lookup allows retrieving it by declaring a set of its attributes.

3.3.6 Enactor

Newberger and Dey have identified the need for effective context monitoring and control, hence propose a solution that provides external access to the application logic and enables its manipulation. [20] They have defined a new component, enactor, which enables application developers to define the application logic and expose its design-time and run-time characteristics. The Enactor's structure is shown in Figure 8.

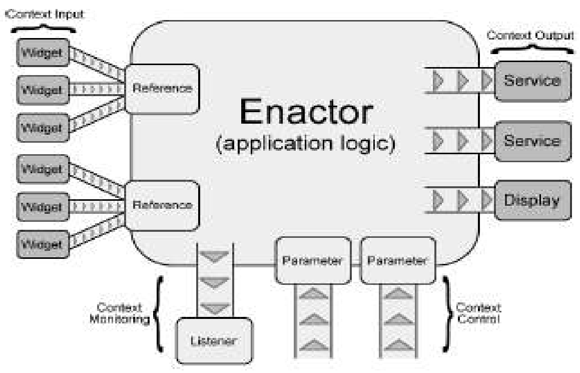


Figure 8: Structure of an Enactor [20]

An enactor is composed of three subcomponents: references, parameters and listeners. References fully describe a widget via a set of attributes and a set of conditions about those attributes. They send queries to discoverers and automatically subscribe to any components that match those queries. [24] The set of parameters constitute the enactor's public view in a distributed environment. Enactors inform the listeners when an action has been invoked and context data received, so that a corresponding action can be taken: a service can be executed, a user can be notified, a message can be written on a display etc. On top of all enactors is the Enactor Subscription Manager which manages subscriptions on behalf of the enactors. It generates discovery queries from the enactor's references and notifies the enactors if any new widget matches those references. It subscribes to that widget and further delegates the callbacks from the widget to the enactor.

Another feature of enactors is that they allow designers to easily enclose application logic in one component giving a compact solution that communicates with all relevant components in an automated way. This can be accomplished by creating an enactor reference for every widget the application needs to subscribe to and listeners that trigger appropriate application behavior. However a recommended approach is to create a separate enactor for different kind of contextual information (e.g. temperature, location) as a way to encourage reusability and preserve modularity. [20]

3.3.7 Communications Infrastructure

One of the requirements set when defining the communications infrastructure of the toolkit has been that the platform supports TCP/IP. This was decided in order to alleviate integration and increase support by many custom built devices, such as: wearable computers, handheld computers, mobile phones and many custom made sensors.

All the previously mentioned objects, i.e. widgets, aggregators and interpreters, extend the capabilities of a single class called BaseObject, shown in Figure 9. The BaseObject encapsulates the communication logic in a distributed environment and all its subclasses inherit its functionality for communication with the rest of the context architecture. This object enables subscribing and unsubscribing to other components, pooling and retrieving historical context and other component's specific data, having a double role and acting as both, a client and a server.

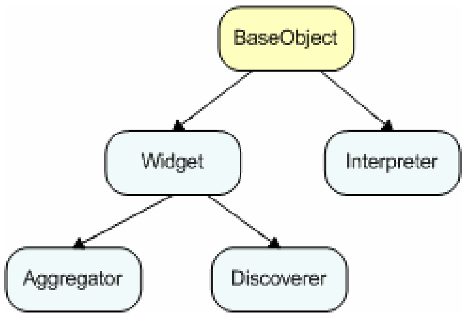


Figure 9: Diagram of object inheritance of the Context Toolkit's abstractions

The BaseObject's default configuration is to support HTTP (Hypertext Transfer Protocol) and XML (Extensible Markup Language). HTTP is used for sending and receiving messages, while XML is used as the language for the data being sent. Its purpose is to facilitate sharing of structured data between different information systems, by defining own elements and to be used on any platform that supports text parsing. [25] XML and HTTP were chosen because they support lightweight integration of distributed components and enable access to heterogeneous platforms with multiple programming languages. The devices that are used in the architecture should only support ASCII parsing and TCP/IP.

The BaseObject can be extended to support other communications protocols and data structures as well. This is accomplished by creating an object that speaks the newly chosen protocol for both incoming and outgoing communication, and utilizing this class when creating the BaseObject. [22] CORBA (Common Object request broker Architecture) and Java RMI (remote Method Invocation) are one alternative to XML and HTTP, but they were not implemented because it was evaluated that they can be deployed on a small number of platforms and would be more demanding on computational resources in comparison with the current selection. [21]

3.4 Flow of actions

There are several flows of actions that need to be closely examined in order to understand the processes that occur within a context-aware application built with the Context Toolkit.

In Figure 10 two sequences of actions are presented. First, the discoverer creation and initialization, and second a widget creation, registration and event handling. Before a context-aware application is started, the Discoverer needs to be run. This object extends the Widget class and inherits all its functionality, therefore inheriting the communication functionality that the BaseObject provides. It creates a CommunicationsObject, which further handles the network communications and exchange of messages, and then the attributes, callbacks, ids and subscribers are initialized. It sets the attributes by specifying the information parameters that the discoverer is storing when registering components, parameters such as: id, hostname, port number, constant and non-constant attributes, classname, input and output attribute elements.

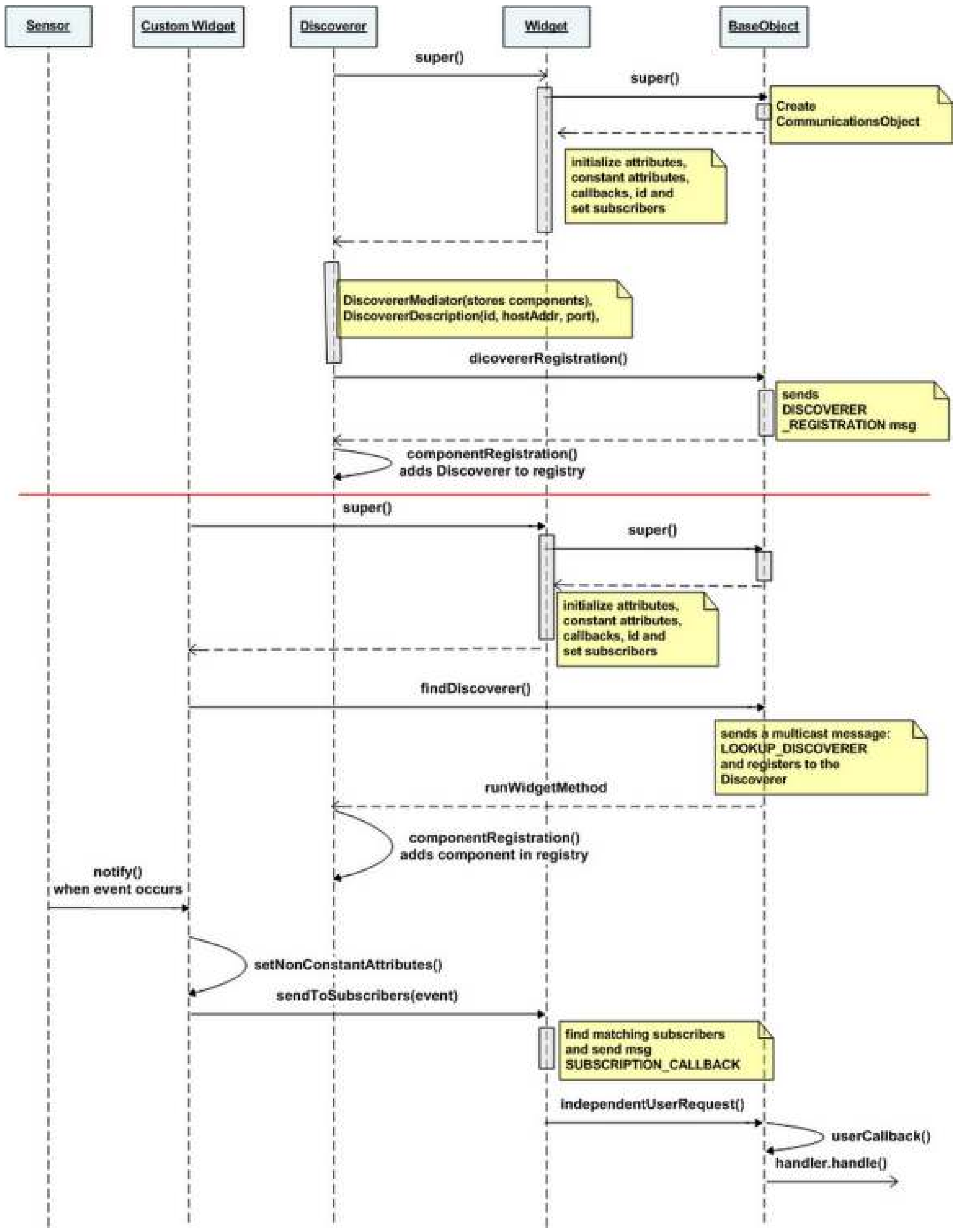

Sensor
Custom Widget
Discoverer
Widget
BaseObject
super()
super()
Create CommunicationsObject
initialize attributes, constant attributes, callbacks, id and set subscribers
DiscovererMediator(stores components), DiscovererDescription(id, hostAddr, port),
dicovererRegistration()
sends DISCOVERER _REGISTRATION msg
componentRegistration() adds Discoverer to registry
super()
super()
initialize attributes, constant attributes, callbacks, id and set subscribers
findDiscoverer()
sends a multicast message: LOOKUP_DISCOVERER and registers to the Discoverer
runWidgetMethod
componentRegistration() adds component in registry
notify() when event occurs
setNonConstantAttributes()
sendToSubscribers(event)
find matching subscribers and send msg SUBSCRIPTION_CALLBACK
independentUserRequest()
userCallback()
handler.handle()

Figure 10: Flow of actions – discovery initialization and widget subscription

Furthermore it creates a DiscovererMediator, an object that creates a log file where it stores all registered components and also handles subscription queries, and then initializes the discoverer's description with its id, hostname and port. The discoverer initialization phase terminates by registering the discoverer's characteristics and description into the discoverer's registry.

Next, a custom widget is run. It first initializes the attributes, callbacks, id and the subscribers (it sets the subscribers by going through the log of its previous subscribers' description and sends them a ping message to check their aliveness before it restarts the subscription). Then, it searches the discoverer by sending a LOOKUP_DISCOVERER message, and after receiving the discoverer's description it registers to it, enabling its later retrieval by other components. The widget now listens for notifications coming from the sensor regarding latest captured events, sets the new attributes' values and sends this event to all of its subscribers who are registered in this widget's subscriptions log file for the specified callback. Subsequently, this event is processed by the handle of the subscribed component.

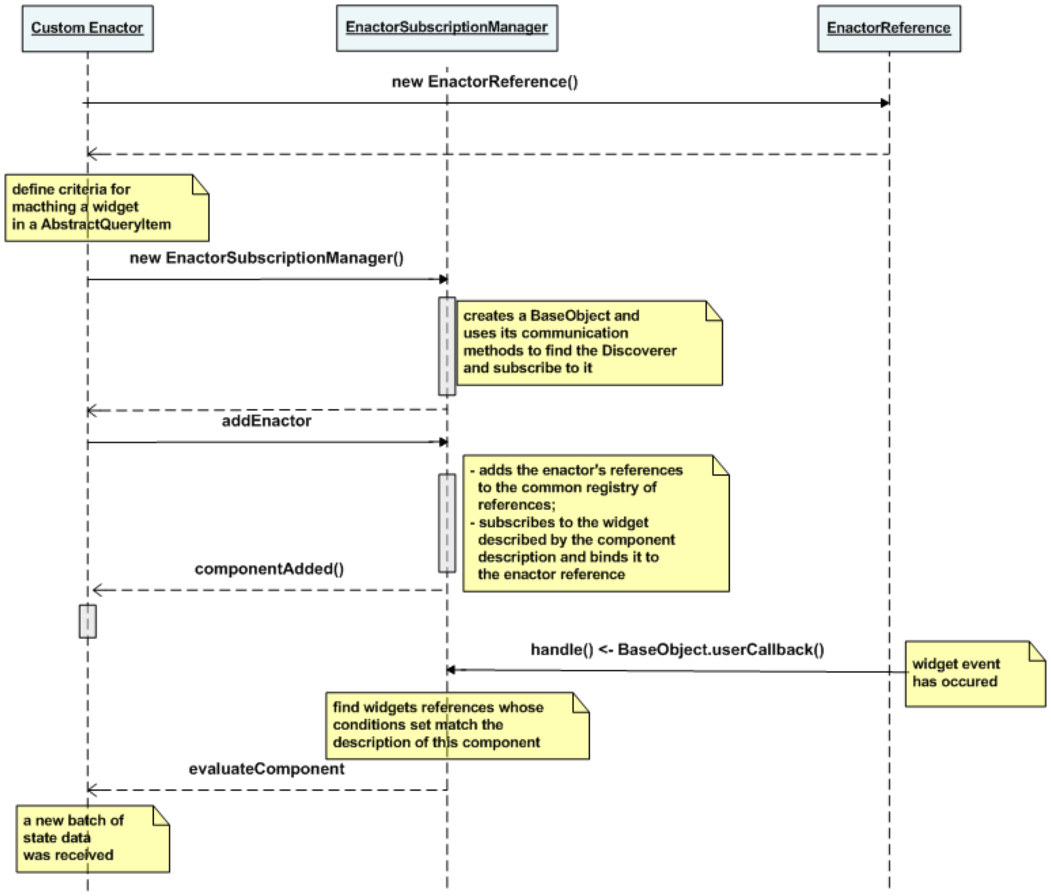

Custom Enactor
EnactorSubscriptionManager
EnactorReference
new EnactorReference()
define criteria for macthing a widget in a AbstractQueryItem
new EnactorSubscriptionManager()
creates a BaseObject and uses its communication methods to find the Discoverer and subscribe to it
addEnactor
- adds the enactor's references to the common registry of references;
- subscribes to the widget described by the component description and binds it to the enactor reference
componentAdded()
handle() <- BaseObject.userCallback()
widget event has occured
find widgets references whose conditions set match the description of this component
evaluateComponent
a new batch of state data was received

Figure 11: Flow of actions – enactor initialization and event handling

In Figure 11, the flow of action regarding the enactor is shown. First an enactor reference is defined, containing the logic for processing a widget's specific information, and a QueryItem object is defined, which contains query criteria used for matching the required widget. EnactorSubscriptionManager object is created, which creates a BaseObject for establishing communication, finding the Discoverer and registering to it. Then, each custom made enactor is added to the already created EnactorSubscriptionManager object which in turn adds the enactor references to the common registry of references and subscribes to the widgets that meet the conditions stated in the query item. If everything is successfully done, an invocation of componentAdded method follows that sets some custom parameters for better manipulation of the widgets matched. When a widget event has occurred, the handle method of the EnactorSubscriptionManager is invoked, which propagates the event handling to the widget reference whose description query matches the widget's characteristics and then delegates the handling of the latest acquired contextual data to the enactor where the component is evaluated.

3.5 Critical analysis

Although still in its early stages, context awareness makes attempts to set the basis for design and development of context-aware applications. The Context Toolkit, as one response to this attempt, addresses problems identified in this domain and suggests a design that deals with some of these issues. Although it successfully implements some approved concepts and newly introduced design principles, nevertheless it contains weaknesses that set the ground for further formulation and development. This section highlights the most distinctive assets and weaknesses of this framework

3.5.1 Strengths

Following some of the strengths and good design practices of this architecture are identified.

- *Reusability*: A main concern for designers of context-aware applications is that there is no common way to acquire and handle context, so developers usually reach to what is the most suitable technique at the moment and implement it without considering its scalability and extensibility. By separating the context acquisition and processing from the application logic, the Context Toolkit enables the widgets, interpreters and aggregators built to be reused. Each application just locates the component of interest, according to the components' description, and subscribes to receive notifications from it. Therefore this software design increases the easiness for the application developer to utilize already handled sensor's information into multiple applications. Applications that are now developed can use contextual information without worrying about the sensor details and how to acquire context from it.

- *Distributed approach*: The designers have chosen a distributed architecture for positioning the components involved, i.e. the widgets and the application no longer need to be running on the same computer. A number of sensors could be spread on different locations, all queried by a single or multiple widgets. The applications are also spread on different computers and communicate with the widgets over a computer network through well established protocols. Hence, choosing distributed computing increases the independence between components and failure of one would not mean malfunctioning of all.

- *Easy extension for support of other protocols*: As far as communication is concerned, another advantage is the communication mechanism. The designers have adopted one set of protocol and encoding language and have based the architecture on it. However, they have set the ground for its extension. Very easily another CommunicationsObject that supports protocols and languages different from the default ones can be created. This object will reference the classes that implement the newly supported communications mechanisms and the communications framework would be easily extended.

- *Resource discovery mechanism*: Earlier, the sensor's location and name should have been known if an application wanted to communicate with the sensor. With the resource discovery mechanism the application describes the component it needs, by using parameters such as constant, non-constant attributes, callbacks, component types (widget, server, interpreter etc) and queries the discoverer. Subsequently the discoverer, which keeps a registry of available components, either returns the desired widget's contact details or if such widget is not available, it subscribes to be notified when such widget is registered by the discoverer.

- *Storage of context data*: Keeping track of captured data and user's behaviour can be beneficial for an application. If a component maintains a history of the context it provides, it could allow applications to later on establish patterns and predict future behavior. The Context Toolkit brings forward former designs and introduces a Storage Interface which, when implemented, keeps account of the last captured context info and flushes it to a database table, if one specified. This way, by saving the attribute values of the widgets, the architecture supports storage and meets any application's need for querying historical context data. Additionally, this lays the foundation for the possibility of implementing learning algorithms that will provide highly adaptable services.

- A*utomatic unsubscribing*. Applications do not register as components into the discoverer's registry, so whenever an application is closed by the user, it is automatically unsubscribed from any widget it was subscribed to. In case it was terminated due to some failure, the widget unsubscribes the application after several unsuccessful attempts for sending data. [21]

- *Push and pool capability*: Applications can process information which were pushed by the widget of interest, i.e. delivered by some kind of a notification, or can either choose to pool for such context information, i.e. to ask for those data explicitly. This allows better flexibility of the application by providing on time and on demand notification about specific context of interest.

- *Context monitoring and control*: An extension was added to the toolkit which enables context monitoring and control by providing external access to the application logic at the toolkit level through standard API. The API is represented through the Enactor, a component that operates independently from any application interface, encapsulates the application logic, gathers context data through references and exposes significant properties through parameters. By this, an external application can monitor a given context data by having an overview of the relevant parameters of an enactor, and also can instruct the enactor to change some conditions that determine its behavior.

- *Easy-updatable architecture*: If some logic has to be changed of a component (i.e. enactor, widget, server, interpreter), that can be done without drastically affecting the reasoning and the logic of another component in the system. If an algorithm for calculating certain output parameters is changed, this modification would not trigger subsequent changes in other interacting components. However, if the change refers the definition of an output parameter (introducing a new one or eliminating an existent one), then this change would demand corresponding changes in the interacting components.

- *Easy-extendable system*: The system based on the context toolkit can be easily extended by adding a new component, whether it is a widget, interpreter or aggregator. Namely, the application does not need to acquire context data from all the described sensor units at once because the architecture allows dynamic modification of the system and consequent subscription which allows easy addition and removal of new components.

3.5.2 Weaknesses

Following some of the weaknesses of this architecture are identified.

- *Discoverer – single point of failure*: The discovery mechanism used is centralized since only one discoverer is utilized. On one hand this adds simplicity but on the other represents a single point of failure by focusing and increasing the load of work for the registration and subscription functionality to only one component. The discoverer when started first checks the aliveness of the components already registered in its database by pinging them several times and noting whether they respond or not. The components that do not respond are removed from the registry and new components that have queried for these kind of components are now notified. If the Discoverer component is in any way compromised, new components won't be able to be registered and applications would not be able to query and locate them since there won't be any central repository that keeps a list of all widgets available. Therefore, the Discoverer, as a mediator for establishing communication implements the centralized approach where the lack of redundancy poses serious threat for further normal functioning of the context-aware applications.

- *Clocks should be precisely synchronized*: Choosing a distributed environment besides some advantages has some downsides as well. All components should keep a clock and should sustain their clocks synchronized. Since in all context-aware application time is a key parameter, a small error and mismatch in its components' clocks might mean applications misbehavior in terms of not capturing the right behavior on time and triggering the appropriate action.

Hence, the dependency on the clocks' tuning increases the frangibility of the architecture.

- *Limited scalability*: Choosing XML and HTTP for distributed communication as well as selecting centralized discoverer architecture, imposes limited scalability when the number of components increases. [21] Scalability in this case measures the system's ability to maintain the usual response time as the user load grows. This can be easily alleviated by implementing another network architecture for distributed systems, e.g. Jini. Jini technology forms a network on the fly, without manual configuration, therefore can be used to create technology systems that are scalable, evolvable, and flexible, as required in dynamic runtime environments, such as our context toolkit and its components.

- *Absence of automatic restarting*: Another restriction is the lack of automatic restarting. It was already mentioned that when a component fails because of some reason, it is automatically unsubscribed from the components it was due to receive callbacks from. But in an environment where constant context notification is needed, a mechanism that supports automatic restarting and its subsequent normal operation would improve the quality and would alleviate the maintenance process.

- *Not interoperable*: A serious drawback of this architecture, as well as for the other, is the interoperability issue. Most of the platforms developed use their own context modeling and handling, and offer this info to applications without having a basic service infrastructure established. Because the sensing mechanism is implemented differently in every framework, the developers of the application user interface have to adapt its communication towards the context toolkit's components that supply the context information, and readapt its interfaces when acquiring contextual data from another context-aware platform.

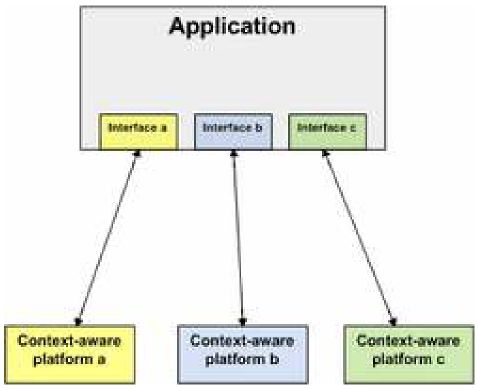


Figure 12: Infrastructure of an application that retrieves information from different context-aware platforms

- *Privacy and security not yet implemented in the official distribution*: Privacy and security are important issues in context-aware systems. Nevertheless, their introduction in the system is twofold. On one hand, the more context information the application has on its disposal, it can carry out better reasoning and operation. On the other hand, by exposing the context data to all application users, the owner of the context data becomes more vulnerable, by allowing access of its personal information to everybody, without any restriction.

Dey, Abowd and Salber, the creators of the toolkit have introduces the concept of Owner. [21] An owner has to be added to each contextual information and has to define a set of rules which describe who has access to the context information and under which conditions. Furthermore the real implementation of this concept involved introducing new objects: MediatedWidget and OwnerPermissions, and modified versions of the BaseObject and Authenticators classes. [10] The Mediated Widget is an extension of the Widget class and contains a widget developer field indicating who owns the context sensed. The OwnerPermissions is an object that receives quires asking for permissions and grants or rejects access. The BaseObject is extended with identification mechanisms, where with the help of the Authenticator one can prove its identity by using public-key infrastructure. [10] However, although introduced, this concept was not implemented it the available official distribution of the Context Toolkit; therefore can be noted that the toolkit at present does not support this feature.

- *No mechanism for resolving conflicts*: The toolkit is lacking an engine for resolving conflicts. Often, there is more than one sensor that gathers the same context information. This information may come with different granularity, distinct level of details specified and those details may be contradicting each other, e.g. different sensors positioned near one of another may measure different temperature values, and only one should be further propagated to the application. Hence a mechanism for resolving these contradicting situations is needed, that would reason the correct data, would give preference on one context data over another or would derive one by processing all of them.

- *Absence of a reasoning engine*: A major drawback for the Context Toolkit is the absence of a reasoning engine and any implementation of intelligence. The context modeling with attribute-value pairs does not have any meaning if not incorporated with additional programming logic. This programming logic decreases the independence of the widgets and tightly couples the model to the rest of the system. A more abstract reasoning engine which will introduce well established mechanisms and rules for higher reasoning needs to be imposed to the system. It will infer over the context data and would make deductions using a form of intelligence.

- *Lack of centralized discoverer of discoverers*: Each set of applications and set of sensor units, geographically very distant, would usually use a separate discoverer, responsible for the given network where these components are run. A centralized discoverer where all discoverers will also be registered is missing in this architecture. Its introduction would alleviate the process of locating required components by the client applications, even though they are not part of the local network.

- *Difficulty of usage the toolkit*: Although the CTK developers aimed at developing a platform that will accelerate context service development and deployment, they partially achieve these objectives. Although the developer has a set of classes that enable uniform interfaces for handling the context data, the developers' effort in working with the toolkit and learning how to use the toolkit in a way beyond the examples provided is not to be neglected. Some of the functionalities offered are not well explained; therefore, documentation is lacking that if provided will alleviate developer's tasks.

4. Enhancements of the Context Toolkit

4.1 Quality of Context

Context-aware applications adapt their behavior depending on the contextual information they receive. The context information has a feature that better explains its quality, called Quality of Context (QoC) and this section defines this concept, analyzes it, proposes an approach that can be adopted by the Context Toolkit and identifies the challenges in this area.

4.1.1 What is it

In 2003, Buchholz, Kupper and Schiffers were the first ones to define the Quality of Context as: [26]

"Quality of context is a*ny information that describes the quality of information that is used as context information. Thus, QoC refers to information and not to process nor the hardware component that possibly provide the information."*

When analyzing the problem area and the characteristics of the sensor data acquired, it can be noted that researchers have distinguished and defined several quality of context parameters that better define and describe the concept of quality of context. Following some of them are outlined:

- *Precision*: It describes how precise are the data that the sensor is producing. Its definition depends on the context that is measured, e.g. for location it can be expressed as number of meters, for temperature as number of significant figures or in general precision can be expressed with percentage.

- *Accuracy or Correctness*: It defines the degree to which the information captured is correct. This parameter can be deduced by using statistical estimation method, determining a value how often the sensor incorrectly captures the data. Although it is difficult to estimate and know whether the value captured is correct, a confidence interval can be defined in order to verify whether the captured value is within this interval or not.

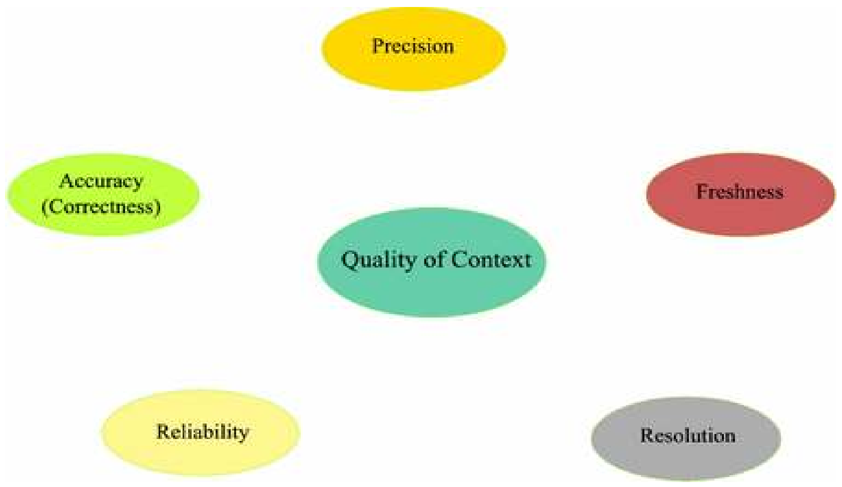


Figure 13: Quality of Context parameters

- *Freshness*: Describes the age of the contextual information, i.e. the difference in time between the latest received contextual information and the currently received one, or the difference between the current time and the time of the last received contextual information. [26] It can usually be calculated by adding a timestamp with every context info delivered and in that manner extracting the time difference. Furthermore, the time when the context data was received is mostly used as a reference in time rather than the exact moment when the data was actually captured, because out of some reason the data might be delayed and the time that is captured as an acquisition time is the delayed one and not the real one. An advantage of this parameter is that it can have a uniform representation no matter of the nature of the contextual information, as it always represents a period of time.

- *Reliability*: Describes the level to which contextual information are available. It gives the completeness of data produced, presented by the ratio between the expected number of context data produced and the total number of output values that should have been produced. This parameter is closely correlated to the sensor's hardware and the error rate it has defined, as well as the reliability of other components that participate in the context acquisition from the widget.

- *Resolution*: Refers to the granularity of the context information. [26] Sometimes the sensors do not provide information that give exact information for a given location, and only measure a parameter in a certain part of that location. However this information is still correlated with the whole location, e.g. the temperature measured can be associated with a room but actually it gives information for a corner in a room, because in the opposite side of the room a heater might be running and the temperature might be higher.

Selecting the quality of context parameters is a trade-off. On one hand the parameters selected should be generic enough so they can be applied to all types of context data, whereas their number should be well chosen so it can be easy implemented as an add-on in as many context-aware frameworks as possible. [27]

4.1.2 Why do we need it

Introduction of a mechanism that handles quality of context issues is needed because the correctness of the information retrieved is of great importance. This is especially critical in applications in the healthcare domain, where on-time, urgent and accurate analysis of the vital signs of the patient's functions as a step closer for preventing attacks, is gaining momentum.

When focusing on quality of context, several aspects should be taken into account, which emphasize why quality of context is necessary:

- Noise in the sensor network may appear from a number of reasons (sensor failure, sensor noise), which consequently introduces errors in the measured data and inaccurate data capture. Consequently this can cause false detection of a critical situation (false positive) or complete overlooking of an existing one (false negative). Therefore, a value that truly reflects the 'predetermined' level of noise should be known, together with appropriate adaptation of the algorithm that deals with the sensed data to detect such noise occurrences and reduce their influence;

- In a context aware system it is common that several sources provide the same type of contextual information and refer to the same entity. Hence, several pieces of context information taken at the same time can relate to the same entity measured, and can deliver distinct data measured, leading to inconsistency. For example, there might be several sources that provide location information (GPS, schedule information, mobile phones, RFID tags and readers etc) all differentiating in its accuracy, having different error rates and providing location with different precision, such as: coordinates, rooms, buildings and areas. This imposes the need of a mechanism that solves these conflicting situations;

- Contextual data is imperfect, which comes as result of: sensor limitations, the situation of the specific measurements (sensors can be affected by heat, cold, humidity), the setting where the sensor is situated, the granularity of the data the sensor produces, the sensor unavailability etc;

Context-aware applications adapt their behavior guided by the context acquired. However, the reasons listed above indicate that applications should adapt their behavior on Quality of Context as well, and this information should be supplied to the application along with the contextual information. Another reason for introducing this dimension to a system would be the connection between QoC and privacy of information. [28] Often a context-owner might like to limit the information that is supplied to the requesters, e.g. a person might not want others to be able to accurately locate him, and would like to supply and reveal less precise location information. Consequently, introducing the concept of QoC imposes itself more as need and requirement rather than as an enhancement in context-aware systems.

4.1.3 Design issues

The previous section emphasizes the necessity of the introduction of quality of context, therefore several design issues that are associated to its implementation will be examined in this section.

a) The value of context data is not the only thing required from the application in order to make an accurate decision. Quality of context parameters should also be propagated to the application along with each context information update. In addition to the parameters, sending a single evaluation parameter that reflects the quality in general and represents a summary parameter can also be sent.

b) Next, a decision should be taken whether the framework should deal with abstract descriptions of the quality parameters or with the exact values that describe them.

   - *Descriptive form*: This approach gives all applications a uniform way of evaluating the quality of context parameters. However, on the downside, sending the parameters in descriptive form limits the application knowledge and imposes usage of the value-description mapping that the widget designer has set before. Moreover, the widget designer and the application developer might have different understanding on the same mapping, e.g. what is high precision for the widget designer, might be medium precision for the application developer.

Context Type	**QoC Parameter**	**QoC Value**
Location	Precision	Low
Temperature	Accuracy	High
Schedule	Reliability	High
Heart Rate	Freshness	High
Location	Resolution	Medium

Table 3: Example of abstract descriptions of the QoC parameters

   - *Exact values*: Sending the parameters in a raw form provides the application with the true and objective parameter values, and enables it to make more accurate decisions on whether they meet the standards for the application or not.

   Both approaches are acceptable and choosing one is a matter of choice that should comply with the architecture's design approach. If we want less processing on the application side and not so exact values, than the abstract descriptions approach should be selected. If we operate with health-critical applications that require precise and full description of all the parameters involved, then the propagation of the exact value of the QoC parameter is recommended.

c) Following, the quality of context mechanism should be integrated with the system's existing mechanism for resolving conflicts. Due to the fact that there can be several sources of the same context information, but from different type of sensors (physical, virtual, logical) and several sensors of each type, it is very likely that they won't always deliver the same data. The concept of QoC can aid in this situation by filtering the data and giving preferences of one sensor data over another by using the quality of context evaluation parameters. A certain threshold for each quality parameter can be set and all the context data that don't meet the requirements will be discarded and will not be further processed. Next, the established conflict resolution mechanism is applied and a certain value for the context type is set.

   As a side note it is advisable that the different sources of the same type of contextual information use the same representation format, so that this info can be later easily processed on a joined scale.

d) Finally, there are several ways how the application will handle the quality of context paradigm:

   - The system can perform conflict resolution and together with the information supplied it can represent how authentic is the given value to the original, actually which is its confidence rate.
   - The user is presented an overview of the different information retrieved by different sensors and is prompted for selection of one. This approach demands more attention from the user, but it gives him power to choose which sensor source it finds more trustworthy.

4.1.4 Proposal for improvements

At present, in the current distribution of the Context Toolkit there is no mechanism that covers quality of context issues. An object that describes the QoC parameters can be added to each widget, sent together with the non-constant attributes with each callback, and passed for further processing to the application side.

The architecture will be extended with several new components: QualityOfContextParameter, QualityOfContextItem, QualityOfContextItems whose class diagram is shown in Figure 14, and a modified Widget and ComponentDescription class:

- The QualityOfContextParameter class contains (name, value, type) tuple that describes a parameter;
- The QualityOfContextItem class contains a list of all the quality of context parameters for a certain contextual type and a description for the context information;
- The QualityOfContextItems class contains all the quality of context items related to a single widget, e.g. one widget may provide QoC information about temperature, light and humidity;

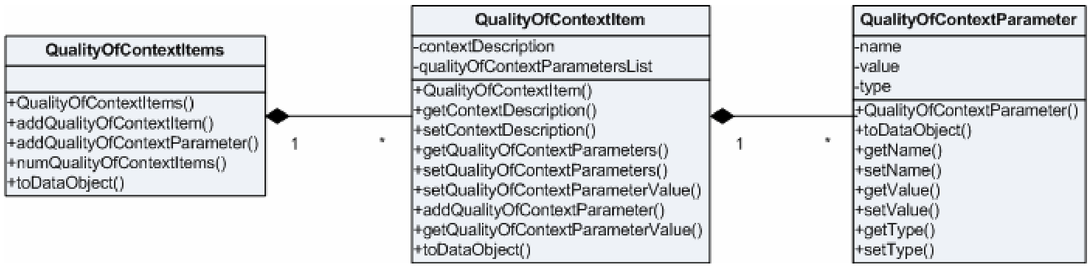

QualityOfContextItems
+QualityOfContextItems()
+addQualityOfContextItem()
+addQualityOfContextParameter()
+numQualityOfContextItems()
+toDataObject()
1
*
QualityOfContextItem
-contextDescription
-qualityOfContextParametersList
+QualityOfContextItem()
+getContextDescription()
+setContextDescription()
+getQualityOfContextParameters()
+setQualityOfContextParameters()
+setQualityOfContextParameterValue()
+addQualityOfContextParameter()
+getQualityOfContextParameterValue()
+toDataObject()
1
*
QualityOfContextParameter
-name
-value
-type
+QualityOfContextParameter()
+toDataObject()
+getName()
+setName()
+getValue()
+setValue()
+getType()
+setType()

Figure 14: Class diagram of the new components introduced related to the QoC

- The QualityOfContextConstants class contains the definitions of the QoC parameters;
- The ComponentDescriptor class now contains QualityOfContextItems which are passed to the widget's subscriber together with the non-constant attributes when every callback processed;
- The Widget class has been extended with several methods for initialization of the QualityOfContextItems and the way the callback's ComponentDescription is built.

With the adoption of the widget model, the Context Toolkit has separated the sensing infrastructure from the application logic and therefore tends to use the widgets as context acquisition components without implementing any logic in specific. Consequently, more suitable approach for the Context Toolkit would be to propagate the exact value of the quality of context parameters and let the application make its own estimations, an approach shown in Figure 15.

When the EnactorReference is defined and coupled with a certain widget for new data notifications, a requirement for the QoC parameters can be also declared. If the data obtained do not meet the demands and do not conform to the threshold values set, the system either disregards them or re-requests them. Furthermore, the Context Toolkit does not offer any mechanism for conflict resolution, so a simple implementation can involve introducing a summary value that unites all QoC parameters by giving different weights to the parameters. So, if a conflict occurs, it is resolved by referring to the summary QoC value in a manner that the context data with a higher summary QoC value is taken into account over the others.

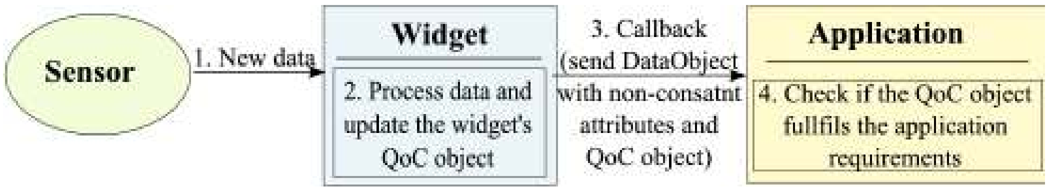

Sensor
1. New data
Widget
2. Process data and update the widget's QoC object
3. Callback (send DataObject with non-consatnt attributes and QoC object)
Application
4. Check if the QoC object fullfils the application requirements

Figure 15: Sequence of actions when handling QoC

4.1.5 Challenges in the QoC

Although there are many analysis in this area, there are still some challenges that remain to be solved:

- *Representation format*: Sensors from the same type often provide the contextual data in a different format. Standardization needs to be done so that the chosen format could express the sensors measurement as authentic as possible and in the same way, which will further alleviate its processing.

- *Rules for calculation of the parameters*: Establishing a bundle of rules to make the acquisition services calculate its quality of context parameters in a standardized way, so that we can be sure that they are all using the same criteria.

- *Integrating QoC with resource discovery*: Frameworks should be extended to support context description with the QoC parameters, which in turn will allow location of context sources and sending notifications depending on the QoC definitions and values. [29]

- *Trustworthiness*: Keeping track of the performance of a context source, how many contradictions it has encountered and whether it complies with other contextual data from the same type, should aid in building a reputation for a certain source that can be extracted as another QoC parameter. Different applications should be able to share the trustworthiness knowledge on the context sensors and infer better decisions.

- *Recalibrating the parameters*: Recalibrating the QoC descriptors during the operation time depending on the learned behaviour and history tracked.

4.2 Resource Discovery

4.2.1 Introduction

The designers of the Context Toolkit have selected the widget architectural approach in order to separate the application logic from the sensing infrastructure. In this manner they manage to hide the details about the sensors connectivity and driver's configuration in a widget component and ease the application development. This in turn increases: the reusability of the context acquiring process by using the same retrieved context data for multiple applications, its maintainability in terms of easy substitution of a single component without influencing the normal operation of others, and the extensibility of the system by seamlessly adding new components.

In order to allow the decoupling, the system must be able to provide a mechanism for locating the widgets, to get information about the host where they are running and the port on which they can be accessed. Therefore the decoupling imposes the support of resource discovery.

Presently the resource discovery process of the Context Toolkit is illustrated in Figure 16 and occurs in the following order:

- A widget is started and it is registered in the Discoverer's database. The following data are stored: name, hostname, host address, port, version, type, constant and non-constant attributes, callbacks and services. The Discoverer "pings" all components in its list to ensure that they are functioning normally. If the component does not answer, then it is removed from the list.

- An application is run which sends a multicast message containing its id, port and hostname, in order to locate the Discoverer;

- The application describes the widget of its interest by constructing a query object which contains some or all of the parameters that describe a widget (usually type, constant and non-constant attributes);

- The application queries the Discoverer for such components and once it receives the widgets' description, it directly communicates with the widgets and subscribes to them;

- Once the application is shut down, it is automatically unsubscribed from the widget's subscribers list. If the application terminates due to some failure then it is unsubscribed after several consecutive unsuccessful notifications.

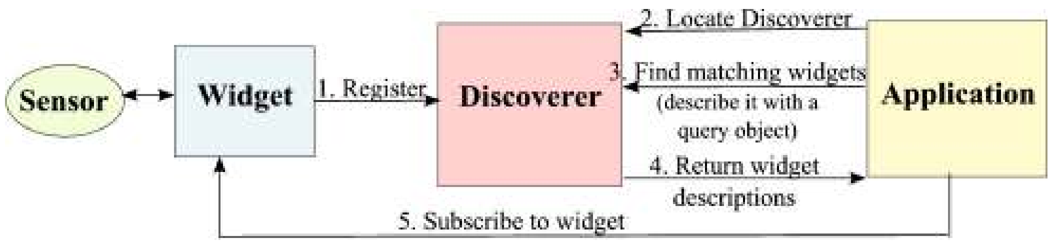

Sensor
Widget
1. Register
Discoverer
2. Locate Discoverer
3. Find matching widgets (describe it with a query object)
4. Return widget descriptions
Application
5. Subscribe to widget

Figure 16: Sequence of actions when locating a resource

4.2.2 Proposal for improvements

The current resource discovery design is quite functional as it is, still improvements can be introduced in the system to advance its performance. This section examines several issues and motivates the given proposals.

- The current implementation of the Context Toolkit offers the following parameters that describe a component: name, hostname, host address, port, version, type, constant and non-constant attributes, callbacks and services. This description can be further extended by adding the QoC parameters (precision, accuracy, reliability, freshness and resolution) as criteria for matching widgets. Other criteria for further widget selection can be the context information type, e.g. location, temperature, light, weather etc. This spans the description capability and makes the user filter the right widgets with higher precision.

- As mentioned in section 3, the discovery mechanism in the Context Toolkit is based on a centralized approach. This imposes the risk that this component is a single point of failure in the system. If it stops functioning out of some reason, new components can not be registered and consequently will not be able to be located by the applications.

  One option is to impose redundancy in the system by replicating the discoverer, which would no longer make the system be physically centralized and would enable rapid recovery in the event of failure. But as solving one, this opens another problem of keeping the consistency between all instances of the Discoverer. Consistency is hard to implement, keeping copies may be expensive and depends how often updates are performed. Therefore, this is a trade-off between reliability and performance.

- Presently, only widgets that are run on near by devices are located. An extension would be to focus on resource discovery on distant hosts, not just close by devices. One approach is to introduce a centralized discoverer-of-discoverers, where all discoverers register. The discoverers are all independent and still reachable between each other. When sending the request for matching a widget, a flag can be introduced which can indicate whether the search should be performed only in the local Discoverer or in all discoverers by querying the central master Discoverer. Searching through other discoverers will decrease the response time but will give more extensive results.

- Since interoperability is an issue that all frameworks should aim for, a discovery mechanism that will be universally accepted is another aspect of this issue. Service Architectures such as Web Services are lately targeting service integration on the Internet and in enterprise architectures, and use different discovery mechanisms. [30] They try to provide advanced search mechanisms with very small error retrieval rate, by extending service description languages and enabling more accurate matching of desired services. Therefore, using the Web Services model of Service Provider, Service Broker and Service Requester is already widely spread and might provide just the approach needed for the context-aware frameworks.

- The resource discovery implemented at the moment requires that the application finds out the hostname, address and port number of all the widgets it needs to subscribe to. If we want to increase the decoupling and make the retrieval of this communication information unnecessary, the widget model can be combined with the blackboard model.

  A mediator (alike a common board) component can be introduced and will be in-between the widgets and the applications. The application will send the widget descriptions it is interested in, and it will subsequently directly receive notifications from the mediator component. On the other side, widgets will send notifications when a new event occurs only to the mediator and then the mediator will redirect the notifications to the appropriate applications. The mediator would process widget registrations, widget notifications and would handle the application's subscription to the widgets. This approach increases the decoupling but also increases the time spent in communication.

In absence of perfect approach, the selection of one applied for the Context Toolkit will come with some advantages and disadvantages at the same time. Choosing one is a trade off between parameters such as: reliability, performance, scalability, extensibility, interoperability etc. and its selection should generally be guided by the demands that the framework would like to meet.

5. Design of a Context Aware Platform

By analyzing the context-aware application domain and design principles, focusing on the Context Toolkit, its architecture, information flow, advantages and disadvantages in its design and possible add-ons, in this section we draw conclusions on best practices in the domain and we design a context aware platform that can address some of the issues analyzed before and deal with new ones.

5.1 Design principles

In order to ease and generalize the development of context-aware applications, an abstract framework will be conceived that should offer generic base for further development and implementation. Following several principles that the framework will be build on are outlined.

5.1.1 User-centered design

The architecture offered should comply with the user centered design, which should optimize the user experience, in a manner that the user does not adapt to the system, but forces the system to adapt to the user's behavior. Also, it does not only follow the user's actions and make adjustments according to that, but also performs usability analysis and testing where it examines the correctness of the assumptions made and improves its future reasoning. It considers the user's interface preferences and model the system actuation taking into account the user's privacy constraints. Therefore, it positions the user in the center of its concerns.

5.1.2 Layered approach

As mentioned in the previous chapters, an infrastructure that decouples the sensing infrastructure from the rest of the system is what all frameworks should support. Although there is no best solution, the one that is a combination and integration of several architectural approaches is often the most suitable one. The architecture proposed possesses characteristics that describe three different architectural approaches: widget, blackboard and networked services approach. As shown on Figure 17 the framework is roughly divided into three layers: application layer, context manager layer and context provider layer. The context provider follows the widget model and standardizes the context data acquisition and its decoupling from the sensors infrastructure by applying software components called widgets. All the retrieved data from the context provider are sent to the context manager for further processing following the blackboard model, and then made available for the applications as services complying with the network services approach.

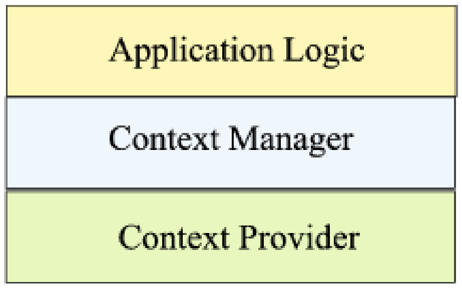


Figure 17: Layered approach

Context provider layer: Consists of sensors, classified as physical, virtual and logical, and presents the data acquisition part of the system. It acquires the sensor data by coupling each sensor with a standardized component that further delegates the contextual information. This layer provides the upper layer with sensor data from multiple sources in a reliable way, enables reuse of the contextual data from multiple clients and makes the framework easy extendible by allowing adding of new context providers. Apart from context data acquisition, it offers a mode to feedback the sensors and thus achieving interaction of the application with the sensors. This adds value to the services offered and improves performance of the system as a whole.

The context provider layer contains very little or no logic at all, and is meant to serve only as an interaction layer with the sensors.

Context manager layer: Consists of several modules and implements the logic of the framework. It processes the contextual data obtained and is responsible for storing this data. The data acquired is usually raw information that needs to be interpreted and higher level information need to be reasoned. The problem of detecting and solving conflicts is also addressed at this level, together with data manipulation depending on the quality of the data sensed. Moreover, as privacy and security of the sensitive information becomes key issue, the framework should handle this aspect as well. Finally, this layer offers data to the application layer through a public interface that is easily understandable and accessible from as multiple applications at a time.

Application logic layer: This is the implementation of the client, realized in the upper, third layer. It implements the logic of a custom created application, the events that should be triggered when certain conditions are met and the adjustments that need to be done in the interface depending on the context information and the user's situation.

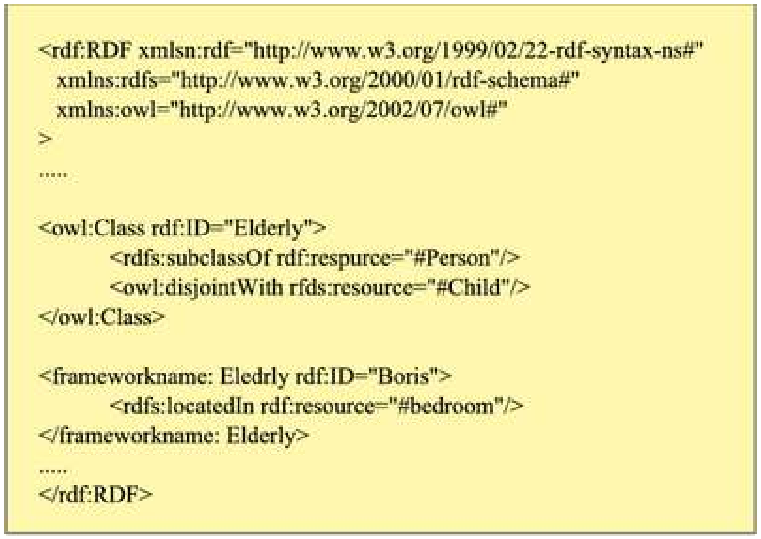
5.1.3 Ontology-based context model

Determining the context model of the representation of the context data is an important step that influences the whole system design and performance. Since the key-value pair model already reviewed requires additional programming effort, the ontology based model was chosen for this architecture.

The ontology provides a vocabulary for representing knowledge about a domain and for representing specific situations in a domain. [31] This model describes context in an expressive way independent from the programming language and aids the system by applying intelligent reasoning techniques by using first-order logic. An ontology language that can be adopted is the Web Ontology Language, OWL, an XML language designed to be interpreted by computers, chosen because of its higher expressiveness in comparison to other similar languages.

Following is an example of basic representation of contextual data expressed with this model using predicates and later on Figure 18 using OWL:

- Location (Aleksandra, Kitchen)
- Light (Living room, On)
- HeartRate (Borche, Normal)
- Temperature (Bedroom, 25)



```
<rdf:RDF xmlsn:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
>
.....

<owl:Class rdf:ID="Elderly">
        <rdfs:subclassOf rdf:respurce="#Person"/>
        <owl:disjointWith rfds:resource="#Child"/>
</owl:Class>

<frameworkname: Eledrly rdf:ID="Boris">
        <rdfs:locatedIn rdf:resource="#bedroom"/>
</frameworkname: Elderly>
.....
</rdf:RDF>
```

Figure 18: Context ontology written in OWL

These statements can be combined with other more complex predicates and can infer higher-level information about people, objects, situations and other entities.

An open source popular tool based on Java that can be used to model ontologies is Protégé, which is an ontology editor and knowledge-base framework. Furthermore, Jena Semantic Web Framework can be used as a tool to reason upon the ontologies defined. This is a Java framework that includes a rule-based inference engine and provided programmatic environment for RDF, OWL etc.

5.1.4 Context retrieval

The contextual data should be delivered to the Context Manager layer for further processing and from there sent to the application layer. In order to be more flexible, the architecture should support the two basic modes of context retrieval and delivery:

- Push: Enables automatic context delivery to all subscribed components.
- Pull: Enables on demand context data retrieval, done by addressing a special query to the context manager layer.

When initiating these types of retrieval methods, the client has to state the context information needed, the accuracy of the data requested expressed through the quality of context parameter values and possibly other requirements.

5.1.5 Service description and discovery

The discovery mechanism in a context-aware system in a pervasive setting is an important issue. The sources used are not stable and always available, so up-to-date listing of the currently available resources and its discovery should be always obtainable. Hence, the context manager layer exports available services that can query the system or can subscribe to receive automatic notifications when a new event is detected for particular context data or a higher-level contextual information.

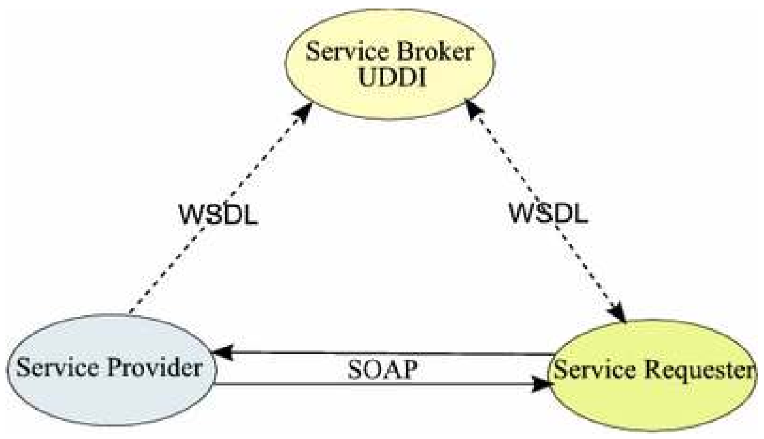


Figure 19: Web Services architecture

Currently the industry is imposing convergence to a single service-oriented architecture. Therefore a way of modeling service descriptions with the Web Services model presented on Figure 19 is proposed. Web Services are APIs that can be accessed over the network and be executed on a remote host. They encompass communication using XML messages that comply to the Simple Object Access Protocol, SOAP, where clients request for services which are earlier described using the Web Service Description Language, WSDL. [32] This enables the architecture to use well established modeling languages and protocols for communication which increases its support for interoperability, since many of the other frameworks could be augmented to support this already well understood and spread design.

Moreover, the service provider can explicitly register its services into a Web Services Registry, such as the Universal Description Discovery and Integration, UDDI, and make them available for usage. UDDI is a XML registry for publishing services, discovering each other and defining how the services interact over the Internet. [33] Furthermore it enables service discovery by performing yellow and white pages lookup.

The link between service description and discovery is obvious and the web services paradigm offers a way of handling both. The context manager can publish the contextual data, together with required input parameters (e.g. quality of context parameters), it provides them in the form of services using WSDL, and the application can search into a service registry for description of appropriate services.

5.2 Components

The platform proposed is given on Figure 20 and provides the fundamental abstraction elements necessary for developing a context – aware application. This section looks into the components that each of the layers consists of.

- *Widgets, aggregators, interpreters*: Main building elements in the first layer are the widgets, aggregators and interpreters, a design inherited from the Context Toolkit earlier discussed. All sensors in the layer are closely coupled with widgets, software components that encapsulate the communication and driver details, which further delegate the data. Data from several widgets can be integrated in an aggregator, offering a higher-level abstraction of context data, or the context data can be translated and mapped from one set to another set of values with the help of interpreters.

  Contextual data offered from a set widgets, aggregators and interpreters constitute a context provider. And context providers are formed based on reference to the same logical group of widgets (e.g. widgets that capture context info in the same building).

- *Sensing Infrastructure Manager:* A sensing infrastructure management block is a component that manages physical resources, and keeps track of the state of all the widgets in the lowest layer, as well as their description, location and capabilities.

- *Context Provider Registry*: The context manager needs to keep a record of all the context providers subscribed to it. Therefore, different context providers are stored and listed in a registry component that contains an overview of all context information sources available.

- *Storage*: Context storage is a component responsible for storing the historical information, keeping record of the chronological sequence and changes of data retrieved for each sensor, enabling future retrieval. This data further enables performing analysis, to help in detecting a pattern and predict future behavior.

The stored context data should be organized in tables and a database implementation can be selected for its deployment, which provides the architecture with a high-resource storage element. This alleviates the process of data analysis by allowing the construction of complex queries with the use of the Structured Query Language, SQL.

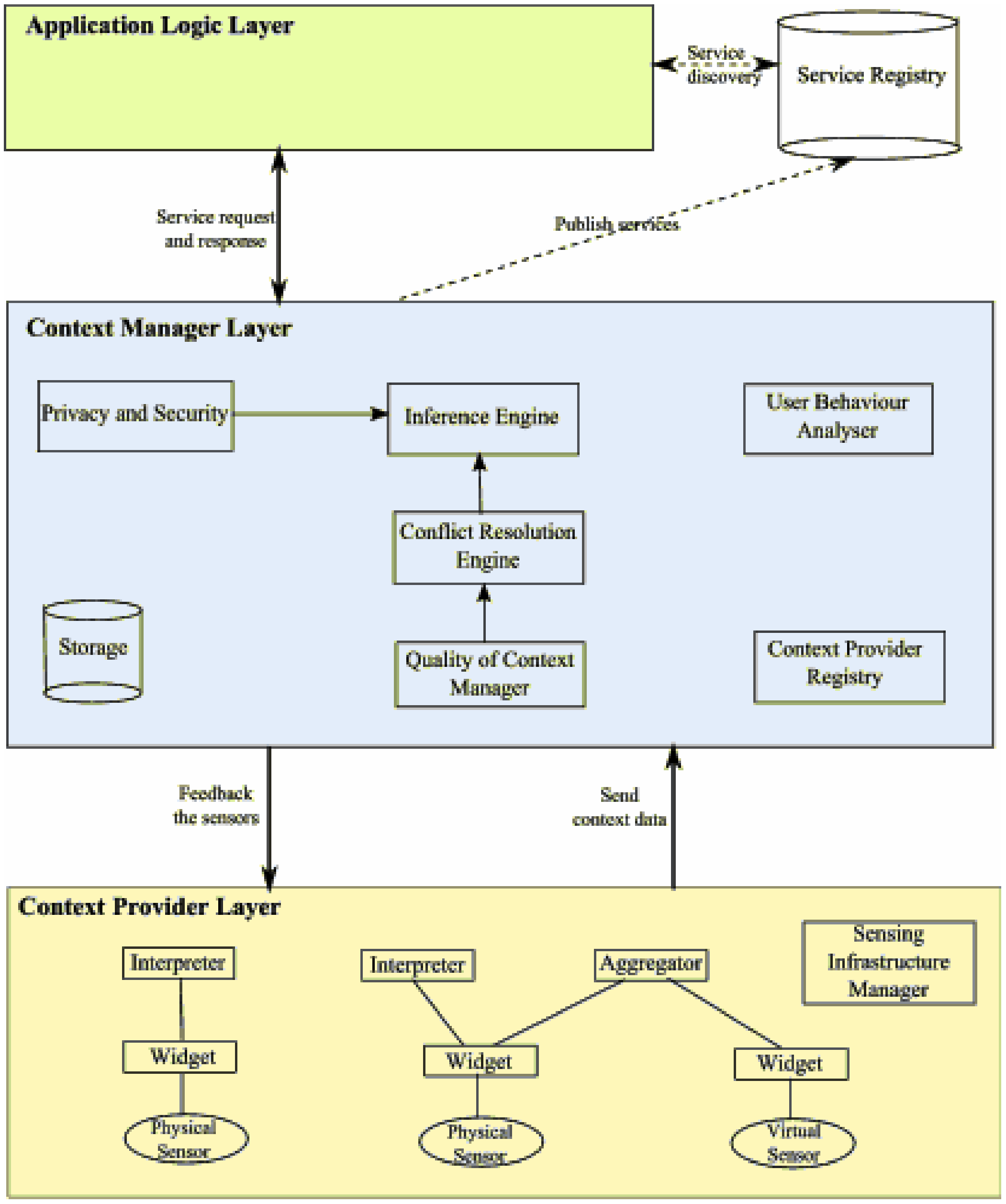


Figure 20: Proposed context-aware platform architecture

- *Quality of Context Manager*: A manager that evaluates the captured context data and assesses it on several parameters is an important module of the context manager layer. As mentioned earlier, precision, accuracy, freshness, reliability and resolution, were chosen as descriptors of the concept of quality of context and provide a mean of measuring how good data is. In order to achieve better expressiveness, precision and unambiguity the parameters' values should be represented with exact values rather than using descriptive vocabulary.

This component takes the applications service request, analyses the demands referring to quality of context defined in it and further processes information that meet these requirements. It may just filter data that meet the demands or filter them and return exact information about the quality of context overall characteristics in a summary value which the application may want to display to the user.

- *Conflict Resolution Engine*: A component that the Quality of Context Manger needs to be integrated with is the Conflict Resolution Engine. After the requested context data have been filtered depending on whether they satisfy the quality of context requirements, contradictory values can be detected, captured by different sensors that measure the same parameter. This component offers a way of dealing with the ambiguity of the context and makes a preference of one sensor data over another. The preference can be based on the:

  - User decision: the user makes an explicit selection of the sensor data it prefers;
  - Quality of context: the sensor data with a higher summary value for the quality of the context data is selected;
  - Analysis of historical data: An analysis is performed based on the previous two aspects (user decisions, quality of context historical data) and context data is selected based on prediction for higher reliability.

- *Inference Engine*: In order to recognize new and deduce more complex contextual information, an inference engine is a necessity in the architecture. By using the knowledge obtained and the historical data already stored, this component represents the intelligent part and introduces the 'awareness' in the context-aware system.

  The ontology based contextual model earlier chosen, declares classes, rules and relationships between them. It already supports some form of intelligence that can be augmented by adding the ability to make further inference by using intelligent reasoning methods, like logic or probabilistic reasoning, fuzzy logic, Bayesian networks etc.

- *User Behavior Analyzer*: The user-centric design is a requirement that the system should fulfill, and it makes an attempt to do so by adding the user behavior analyzer component into the system. By using reinforcement learning which is a sub-area of machine learning it analysis the user behavior and its interaction with the application.

  Reinforcement learning utilizes the user's feedback, which is automatically tracked in order to modify the system's behavior. [34] The method is actually learning what action to take in order to maximize a reward signal. Two features make the technique distinguishable: trial-and-error search and delayed reward. [35] If the user selects one feature, but then switches to another from the same category and uses that one for a longer period of time, the later might be considered as a preferred one and switching to the second one can be interpreted as a negative feedback for the earlier feature. The analyzer registers the user's behavior and its interaction with the system and affects on improving the application performance and increasing user satisfaction.

- *Privacy and Security Module*: This module introduces the concepts of security in the context-aware platform. However, its consolidation in the system is a sword with double edges. On one hand it enables the user maintain its privacy and control which information he makes available to the rest of the users. And on the other hand, it increases the need of user's interaction with the system, by frequently reconfiguring the security preferences for every piece of contextual information it provides.

  A basic functionality this module provides is authentication. It is a necessity because often in healthcare applications the data provided is personal and access should be allowed only to verified users. Proving somebody's identity should be fast, easy and over a secure communication link.

  Another functionality is to guarantee the privacy of the context information it offers by providing access control mechanism. One approach is to assign an owner to each context information provided. Therefore, full and partial access can be granted to all or specific users or access can be completely denied, resulting in absence of context delivery to the application. Another approach is to use policies to express the rights allowed, and the security rules the system and the data it provides, should comply to.

  Finally, keeping the context data protected is another capability of this module, which enables even stronger protection. Depending on the application domain that the context manager supports, encryption of context data can be deployed or can be omitted.

5.3 Flow of actions

Following, the general sequence of actions in the proposed context-aware framework are described.

A set of sensors, widgets, aggregators and interpreters comprise a context provider which locates and registers to a context manager component. A context manager can operate with the data of multiple context providers and keep a record of the subscribed providers by registering them in the context provider registry. The widgets acquire sensor data and delegate these data to the upper layer, which stores them for later inference and pattern detection.

When an application needs a certain contextual information, it searches for it in the service registry (e.g. UDDI) by performing a white pages lookup (by specifying the exact name or id of the service), or a yellow pages lookup (by specifying attributes that define and describe the service). Once a service is matched, the appropriate context manager is communicated and a service request is sent.

The context manager contains a security module that authenticates the user of the application. If the authentication is not successfully completed the service request is denied, otherwise a normal flow of actions follows. Moreover, whether the response delivered to the client will contain the data requested, depends on the rights the authenticated user has been granted and the policies established. If the user requesting the data is not allowed to access certain information then the service response will not deliver them.

The context manager retrieves the latest data stored for the context info requested, or in case of a higher level data, it retrieves several entities from the database. Furthermore, the manager processes this data through the quality of context manager, to eliminate all the data that do not fulfill the QoC requirements specified in the service request. Then, the subset of data is run through the conflict resolution engine, which solves conflicts and deduces the most probable value for certain context when several distinct values are detected. If a higher-level data has been requested, the inference engine draws conclusions depending on the last obtained data and the historical information, otherwise the data processed is prepared for the service response.

Moreover, the applications operation and users' satisfaction is improved by analyzing the user's behavior. Namely, features that the user most frequently utilizes can be automatically reconfigured. They are made more accessible by dynamically reorganizing the interface and adapted depending on user's interaction.

Last, the architecture proposed provides all the essential building blocks needed for a robust platform that can match the wide range of demands and should be able to alleviate the work of the application developers for developing complete, stable and reliable context-aware applications.

6. Context awareness in healthcare

With the reduction of size and price of computing devices, ubiquitous computing has become part of our reality, being present in almost every part of our surroundings and starting to cover more segments of modern way of living. This was made possible by combining a number of technologies available, such as: telecommunications, electronic engineering and computer science. [36] In particular, the development of wireless networks, mobile communications, intelligent sensors and their integration has brought intelligence in a wide range of settings, making context – aware applications possible more than ever. These services have been targeting home environments, hospitals, conferences, workplaces, elderly nursing homes but many more are yet to come.

Smart homes apply gathering contextual information and reasoning in order to provide a series of functions into one's household. They encompass information about the need for food supply, devices' status in the domicile and their control, energy consumption, alarm system, medication consumption, access monitoring.

This section focuses on smart homes and their design to alleviate everyday life of elderly people who have decided to live at home. First, the needs of elderly or dependable people are identified, and then a background of services already provided in this domain is given, followed by an analysis of a prototype of the application implemented.

6.1 Need for support of the life of elderly

A deeper analysis of the challenges that elderly and dependable people face is needed to properly identify the focus of the applications already developed.

A study performed in Finland by Sauli Tiitta [37] has pointed out two factors that affect the quality of life of elderly people: immobility and solitude. The study was conducted by choosing four voluntary elderly persons documenting their activities and commenting which tasks they find troublesome performing. The study came to the conclusion that elderly people lack the support of social interactions and e.g. would need assistance in building new friendships especially with people that live close by. It has also identified security as a feature that they are very concerned about.

Furthermore as the world population is aging rapidly, the number of people aged over 65 is increasing for 800 000 per month. [38] A big percentage of this people are starting to face the inability to live independently and they require the assistance of another individual in conducting the day-to-day activities. Therefore, elderly are confronted with the need to be moved to a nursing home. Unfortunately, studies show that once elderly are moved to a nursing facility they tend to get depressed and increase the number of doctor visits.

Loosing the independence has a negative impact on the elderly and dependable people and finding a way to make use of the technological achievements would impact positively on prolonging the transfer to an institutional care. In addition, development of services that would aid in daily activities would also offer a cost effective way in preserving the independence. Therefore gaining information about the state of the environment and automating processes in the household would significantly alleviate people's life.

Another point for preserving the independence is to increase the monitoring of the elderly person's vital functions i.e. health monitoring. This would exclude the need for a person to be closely supervised in an institution, by providing the means of performing the monitoring in a home environment, by creating devices that measure the critical parameters and rise notifications and alarms if abnormalities occur. This out-of-hospital monitoring has already been in the focus of researchers and offers benefits to the medical staff as well by providing insights about the effectiveness of the medical treatments at home. [39]

Applying context awareness in a home environment can be found beneficial for elderly and dependable people from several perspectives: it can improve social interactions, support independent living, increase the support for performing daily activities and carry out health monitoring. By this, the need for nursing care could be avoided or postponed, elderly people's independence will be sustained, and the quality of life will be improved.

6.2 Background

This section reviews the research literature, and intends to provide an objective view of the progress that has been done in the area of context awareness in smart home with emphasis on applications useful for elderly or dependable people. It captures the dynamism of the advancement and gives overview of the applications proposed by different research groups.

Choi, Shin and Shin [40] propose a context-aware middleware that provides automatic control of devices in a smart home environment depending on user's preference. Their prototype measures six parameters: pulse, body temperature, facial expression, room temperature, location and time; and based on the measured values it learns and predicts the user's preference. The user behavior and choices that he makes are captured and stored by a user's profile manager so that the appliances′ (TV, air conditioner, projector, light) setting can be predicted based on the learned behavior.

Another research group consisting of Baek, Lee, Lim and Huh [41] have designed an intelligent home care system based on context information. This system minimizes the users′ intervention and increases its autonomy by automatically triggering services based on the context reasoning. In addition, it encompasses a way of resolving context conflicts and service interactions, analysing scenarios where two different users give opposite commands related to services such as: turning on/off the light, alarm clock setting etc and dealing with them by proposing priorities. Baek, Choi and Huh [42] subsequently extend this work by deploying a location tracking system and a sensor platform that acquires data on heat and illumination. The location tracking service presents user's position on a 3D home map and the home appliance control system manages optimal automatic control of devices at home such as air conditioner, heater, light etc.

Lee, Kim and Huh [43] describe context-aware based home services that make people's life more comfortable, easy and efficient, by providing automatic intelligent services in relation with the location, time and situation. The authors suggest several services, ranging from answering doorbell (greeting the visitor appropriately depending if there is somebody at home, or choosing who should answer the door if there is more than one person at home based on location and preference), seamless transfer of the transmission from one display device to another as the user moves around the house and watches TV, reminders to turn off some devices while cooking as well as recipes outline on a display near by, and reminders for better management of the home (such as receiving notification to turn off the light, gas stove and some electronic devices).

Another research group has been examining ubiquitous healthcare services and has proposed a wearable sensor system which measures the biological functions of a person. [44] The infrastructure allows two categories of services: remote health monitoring and self health check. In the case of remote health monitoring, the sensors measure information about the heart rate, blood pressure, body temperature and respiration. The user subscribes to a health monitoring service through Internet, sends this information and thus allows doctors to monitor his state. In the case of self health check service, a service is downloaded to the wearable computer in advance. The service determines the health status based on the mentioned context information and offers an overview to the user of his health condition through a comprehensive GUI on a portable computer. This group has later extended their research by adding another healthcare medical device which can measure blood pressure, blood sugar level and body mass index. [45]

Korel and Kao [46] have also addressed the issue of context awareness in body sensor networks. They are implying the importance of monitoring the vital signs of elderly or patients with chronic cardiac anomaly through measuring the physical and bio signals derived from body sensors (ECG, heart rate, blood pressure, oxygen saturation and sweat volume/rate). Information on other context elements such as environment temperature, person's condition and others are significant for determining unnatural behaviour of some body functions and by that preventing mortality on time.

Another approach is the design of a social alarm system based on wearable sensors and monitoring of the condition that will fit the needs of elderly and dependable people. [47] Social alarm systems usually consist of activating an alert button that is usually worn on the wrist or the neck. However, due to the fact that a person not always has this device with himself or is not able to press the button, Korhonen, Pavilainen and Särelä have suggested an intelligent alarm system that provides remote monitoring of a user's state. The implemented system consists of three units: a wrist unit, a base station and software for receiving and routing alerts. The wrist unit has the already mention button but in addition also has incorporated movement sensors, sensitive to muscle movements on the wrist as well as the whole hand and body movements. Help is automatically requested if no movement is observed for longer period. The system also triggers notifications delivered to the emergency person if it has detected that the user does not wear the wrist unit and therefore a warning is sent to the user.

Helal, Giraldo, Kaddoura, Lee, El Zabadani and Mann [48] have developed a mobile patient care giver assistant which is deployed on a smart phone and its main objective is to catch the attention of people with Alzheimer's Disease and notify them about an action they have to do, by playing a message on the phone, by playing a sample of the task on the nearest displays in the house which the user is facing, or playing an action notification and instructions recorded by relatives and friends. Then, a general reminder system is included, which reminds the user about medicine consumption or a scheduled appointment. When taking certain medicine, the medicine box is scanned for verification that the right medicine will be taken and also a control of the medicine left in the box is performed so that refill can be done on time.

Similar system for monitoring and assisting medication consumption at home has been developed by Agarawala, Greenberg and Ho [49] and by Fishkin and Wang. [50] The later have developed a prototype of the system consisting of a portable pad that combines Radio Frequency Identification (RFID) tags and a sensitive scale to detect when a bottle has been picked up or placed down and measures the changes in the weight. This data is processed and is supplied to an application that reminds a user how many pills need to be taken and when.

Bardram has introduced his visions and on-going research in the domain of healthcare in a hospital environment, and some of the services he suggests can be found appropriate for usage in a context aware home as well. [51, 52] He proposes an interactive bed with a touch sensitive display. The bed has sensors that detect the position of the patient, the identity of the person standing besides the bed and the medicines that are carried. In this way the system controls if the right medication is supplied to the right person and verifies the dosage on the screen attached to the bed. In addition a number of body functions can be measured and be displayed on the screen, as well as displaying the patient's record.

The review of the applications in the domain of context aware home and healthcare of elderly and dependable people indicates that this application area is in an early stage of evolution, but it is gaining momentum because if ubiquitously deployed it will succeed in improving the quality of people’s life. As noticed, some of the work of the research groups has been overlapping i.e. the functionalities that the applications are offering are close to one another, but they differ in their implementation and technologies used. As there is a wide range of opportunities in this field of research, new applications that require competence and know-how of several disciplines, should be designed and be developed to better shape the focus of this area and in the same time alleviate people’s life.

6.3 Prototype of a smarthome application

This chapter explains and documents the application that has been developed in order to better understand the structure and mechanisms of the Context Toolkit, its advantages and drawbacks, the aspects of the architecture where improvements are needed and finally to infer techniques and approaches for designing a new context-aware platform.

Taking into account the previous discussion about the necessity to support the life of the elderly in a home environment and the background literature study of the applications developed in this area we choose to implement an application that is targeting both, elderly people living at home and caregivers which remotely monitor patients and react accordingly to the situation. To the elderly the application should offer reminder functionality in conducting daily activities and to the caregiver it offers a monitoring tool about the person’s behavior within his home, his condition and the state of the environment where the user is located. The system is a type of an Ambient Home Care System (AHCS) that can be extended to establish a Computer Supported Coordinated Care (CSCC) network for an elderly residence or a hospital.

The application chosen to be implemented serves as a proof of concept. It is an on-going work that will continue in order to fulfil the description given in the next subsections.

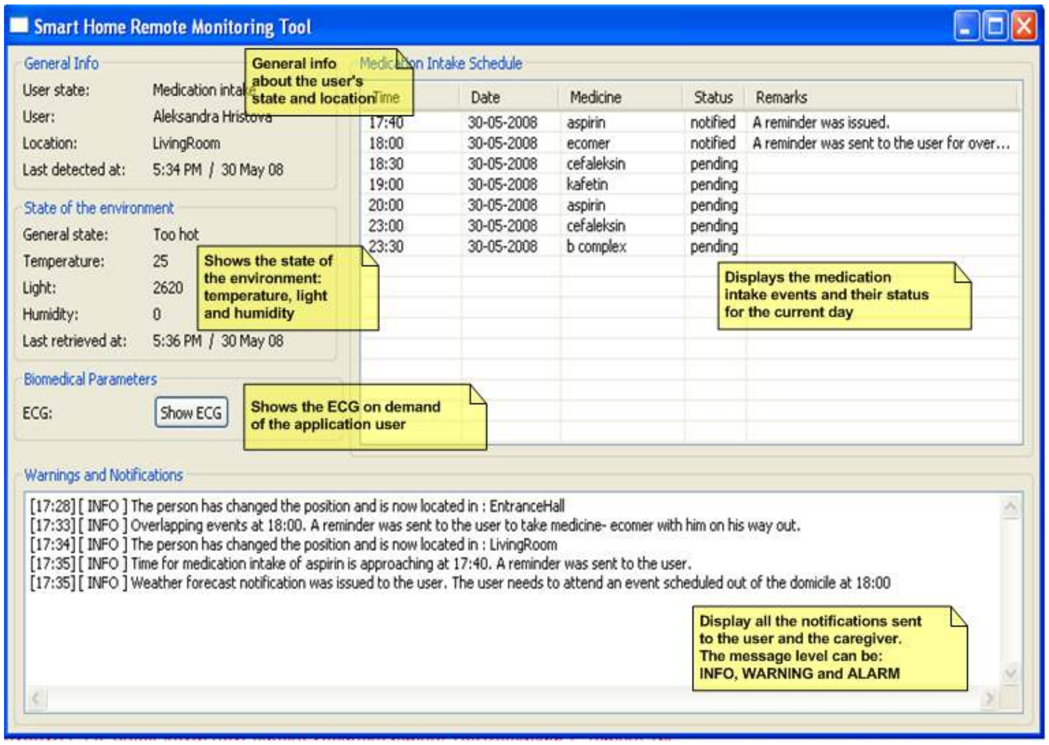
6.3.1 General description

An application’s screen shot is shown on Figure 21. The application is thought as a remote monitoring tool of an elderly person. In order to show its capabilities a selection of example scenarios is further listed:

- Borche is at home and receives a notification that it is time to take a medicine. After taking the medicine, if the medication bottle is almost empty a notification is sent to him that he needs to refill the supplies for that medicine.

- Aleksandra is about to leave the house for a meeting with her friend scheduled outside of the domicile. While the appointment is scheduled, she has to take a medicine. Therefore, she receives a reminder to take the medicine before leaving the house and a notification about the weather forecast for the rest of the day.

- Light on is detected in a room for a longer time where Ivica has not been present for a while. Send a notification to Ivica that he should turn off the light. (This can be further extended into automatic regulation of the light)


Smart Home Remote Monitoring Tool

General Info

User state:	Medication intake
User:	Aleksandra Hristova
Location:	LivingRoom
Last detected at:	5:34 PM / 30 May 08

State of the environment

General state:	Too hot
Temperature:	25
Light:	2620
Humidity:	0
Last retrieved at:	5:36 PM / 30 May 08

Biomedical Parameters

ECG: Show ECG

Medication Intake Schedule

Time	Date	Medicine	Status	Remarks
17:40	30-05-2008	aspirin	notified	A reminder was issued.
18:00	30-05-2008	ecomer	notified	A reminder was sent to the user for over...
18:30	30-05-2008	cefaleksin	pending	
19:00	30-05-2008	kafetin	pending	
20:00	30-05-2008	aspirin	pending	
23:00	30-05-2008	cefaleksin	pending	
23:30	30-05-2008	b complex	pending	

Warnings and Notifications

```
[17:28] [ INFO ] The person has changed the position and is now located in : EntranceHall
[17:33] [ INFO ] Overlapping events at 18:00. A reminder was sent to the user to take medicine- ecomer with him on his way out.
[17:34] [ INFO ] The person has changed the position and is now located in : LivingRoom
[17:35] [ INFO ] Time for medication intake of aspirin is approaching at 17:40. A reminder was sent to the user.
[17:35] [ INFO ] Weather forecast notification was issued to the user. The user needs to attend an event scheduled out of the domicile at 18:00
```

General info about the user's state and location

Shows the state of the environment: temperature, light and humidity

Displays the medication intake events and their status for the current day

Shows the ECG on demand of the application user

Display all the notifications sent to the user and the caregiver. The message level can be: INFO, WARNING and ALARM

Figure 21: Screenshot of the monitoring tool

- The caregiver, Natasha, instead of working in the patient's home, can work from distance and monitor several patients at a time. She receives information about the environmental conditions of the patient's home (light, temperature and humidity), the location of the patient within his home, the electrical activity of his heart, and information whether the patient has taken the medications on time.

- Daniel's ECG shows abnormal rhythm of the heart. The caregiver can immediately verify the other parameters in the environment and the state of the person and take urgent actions if necessary.

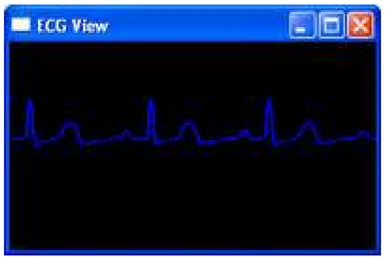


Figure 22: Screenshot of the ECG view

6.3.2 Functions

Following are the functionalities of the application and the information it covers:

Functionality	Description
Automatic positioning	The application can automatically locate and identify a patient within his home.
Schedule overview	Caregivers can have insight of the scheduled medication intake events of the patients.
Environmental parameters	The caregiver's monitoring tool features information about the state of the user's home environment: light, temperature and humidity.
Weather forecast information	Informs the patient about the weather before leaving the house.
Medication intake reminder	Patients at home should be able to get notifications for each medication intake scheduled in the calendar. The reminder issuing depends on the patient's state and location. Patients also receive a reminder for medication intake when overlapping events scheduled on different locations are found.
Biomedical functions	The caregiver's application monitors the ECG of the user.

Table 4: Functionalities of the application

At present the notifications sent to the person monitored are only showed in the notifications panel of the application. Further development of the application can encompass sending notifications through SMS.

6.3.3 Reasoning engine

The application logic is implemented using a rule-based reasoning engine that is based on "if-then-else" rule statements. The actions triggered usually depend on several conditions depending from the information retrieved from various widgets, closely observed on a joined timescale.

- General
    - Display a notification in the monitoring tool when the user changes the location;

- Status
    - User/elder/person's status:
        - Medication pending: This status will be set 20 minutes before the scheduled medication intake;
        - Scheduled appointment: if an appointment is scheduled in the next 60 minutes.

    - Status of the environment:
        - Normal;
        - Too hot: temperature 10C above the normal temperature;

        - Too cold: temperature 10C below the normal temperature.

- State of the environment
    - If the temperature exceeds the limits for normal condition for +/-5 C, send a notification to the user that he needs to turn on the air conditioner/heater;

    - If the temperature surpasses the upper limit for 15C, send an alarm to both the caregiver and the elderly resident, to check the given room where a possible fire might have started;

    - When the user is in the same location for more than 15 minutes, check the light in the other rooms of the user's home and if the light is on send a notification to the user to switch off the light in those rooms;

    - If the user changes the location, display the environmental parameters for the room where the user is currently present;

- General schedule
    - If an event is scheduled outside of the person's home, send a reminder to the user about the event and a notification about the current weather conditions and the forecast for the rest of the day. The notification should be sent 30 minutes before the event;

    - If the user is located in the entrance hall, an event is scheduled outside of the home and a reminder has already been issued, then the user is about to leave for the scheduled event;
        - If the weather forecast says it might be raining, issue a notification to the user to take an umbrella;
        - If the weather forecast says that the current temperature is below 0C, issue a notification to the user to take a hat and gloves.

- Medication intake schedule
    - Display all the medications that the user needs to take till the end of the day;

    - If a medication intake overlaps with an event scheduled outside of the person's place of residence, 30 minutes before the event is scheduled send a reminder to the user that he should take the medication with him;

    - Update the status of each medication scheduled event in the remote monitoring tool:
        - Pending - Status set till 20minutes before the medication intake event is scheduled;
        - Notified - Status set when a notification for the medicine intake has been sent to the user. 20 minutes before this event locate the user:
            - If user is in the kitchen, send a reminder;
            - If user is asleep, wait for issuing the reminder till the exact time for medication intake comes;
            - If the user is not in the kitchen, wait till 5 minutes before the medication intake time and then issue the reminder;

        - Forgotten - If the medication was not taken after 20 minutes of the scheduled time, this status will be set. A warning will be resent to the user and a notification to the caregiver;

    - If the number of pills left in the medication bottle is 5, issue a warning to the user that the certain medication needs to be refilled and also a notification to the caregiver about this situation.

- Health
    - Monitor the ECG of the person on demand.

6.3.4 Widgets architecture

This subsection discusses the widget architecture that the application benefits from. In Figure 23 the interconnection between the Context Toolkit components is presented. The application makes use of a number of context information. Each

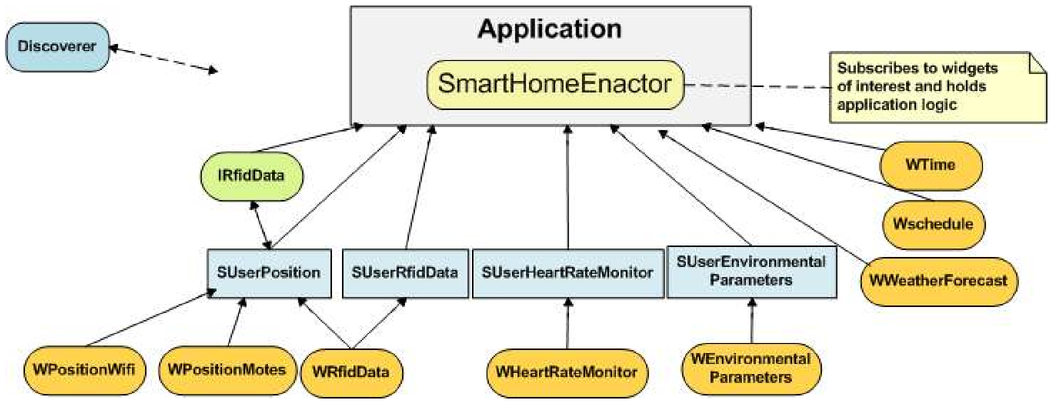


Figure 23: Widgets' architecture

context info is retrieved from a widget or aggregator component which correspond to a given sensor: physical, virtual or logical. Examples of physical sensors are the motes deployed in different positions and measure the temperature, humidity, light etc. Examples of virtual sensors are the public interfaces Google Calendar and Google Weather utilized in the presented infrastructure, and an illustration of a logical sensor is the UserPosition aggregator, which combines information from several physical and logical sensors (Wi-Fi localization, motes localization and RFID localization). Moreover, Table 5 gives an overview about the types and functionalities of the components implemented.

All the widgets, except of the aggregators, are run in advance - before running the application. During application initialization, the following aggregators are initialized: UserPosition, UserRfidData, UserHeartRateMonitor and UserEnvironmentalParameters. The UserPosition aggregator subscribes to the position widgets (Wi-Fi, motes and RFID) and is responsible for aggregating the position information referring to the application's user. The same applies to the rest of the aggregators; they subscribe to the corresponding widget and extract only the context data referring to the application user.

Component Type	Sensor Type	Component	Functionality
Widget	Physical	PositionWifi	Positions users with a Wi-Fi positioning system
Widget	Physical	PositionMotes	Positions users depending on the RSS signal received from static motes
Widget	Physical	RfidData	Gathers data about read RFID tags
Aggregator	Logical	UserPosition	Gathers information about person's position
Aggregator	Logical	UserRfidData	Gathers information about read RFID tags from a specified user's device
Widget	Physical	HeartRateMonitor	Acquires ECG data
Aggregator	Logical	UserHeartRate Monitor	Gathers ECG data for specified user
Widget	Physical	Environmental Parameters	Acquires environmental parameters (temperature, light, humidity)
Aggregator	Logical	UserEnvironmental Parameters	Gathers environmental parameters for locations referring to a specific user
Widget	Virtual	WeatherForecast	Acquires data about the weather forecast
Widget	Virtual	Schedule	Acquires data about a person's scheduled events
Widget	Virtual	Time	Generates callbacks with certain frequency
Interpreter	/	RfidData	Interprets the read RFID tag (it can be a location indication or a medication identification)

Table 5: Architecture components and their functionalities

6.3.5 Technology background

This subsection gives an overview of the technology used to make the already described services possible.

- *Software components*

  - Java was chosen as the programming language for the development of the application since it is built on top of the Context Toolkit which is developed in Java. In particular Java 2 Standard Edition, J2SE, was used for the application development and Java 2 Platform Enterprise Edition, J2EE, was used for developing Java Servlets that render an xml file as a response to a service request.

    Java Servlets were chosen for extracting some of the logic (e.g. motes based positioning) in order to make the functionality reusable by several different applications.

o The information about the user's data, medication, location and connectivity details is stored in the open source database – MySQL version 12.

o The common functionalities that other developed applications also need to access, are placed in the Apache Tomcat web server version 5.5.

o Google Calendar API, which enables client applications to view and update calendar events that match certain criteria.

o Google Weather API, which provides current weather conditions and weather forecast for the rest of the day as well as three more days in advance.

- *Hardware components*

o Alive Wireless Heart Monitor presented on Figure 24 is a wireless health monitoring system that conducts real time transmission of measured parameters via Bluetooth to PDA, smartphone or PC and performs further analysis. It measures ECG, acceleration and heart rate. [53]

Figure 24: Alive Wireless Heart Monitor

For the purposes of the application, the heart rate monitor sends the ECG measurements to the PDA and the PDA forwards it to the Tomcat web server from where they are processed and delivered in XML format to the appropriate widget on its request.

o The application benefits from data gathered from a deployed wireless sensor network which includes the following components:

- MoteWorks 2.0 Software Platform, which enables: MoteView client application, a server gateway (XServe) and sensor devices (XMesh networking protocol);

- Crossbow's MICAz which is 2.4 GHz mote module presented on Figure 25 that measures: temperature, light, humidity, acceleration etc. [54]

Figure 25: MICAz mote module

- Crossbow’s MDA100, MDA300, MTS400 and MTS 420 data acquisition boards. [55]

o SDiD 1010 shown on Figure 26 is a Near Field Communication/Radio Frequency Identification Secure Digital Card, designed to plug into a PDA, smartphone or other hand-held device with an SD slot. The card offers NFC two-way communications and RFID read/write capabilities.[56]

For the application’s needs, this card has been inserted into a PDA that the user is always carrying with him. It is used for reading RFID tags, for the purpose of identifying a medicine taken or as an additional source for acquiring location information (RFID cards with written location are utilized as a simulation for door sensors).

Figure 26: SDiD 1010 NFC/RFID SD Card

o WLAN user positioning: Most newer mobile phones and PDAs come with the possibility to connect to Wi-Fi networks. In addition, the access points become increasingly present in the surrounding area, being capable of offering functionality other than communication, such as localization of mobile devices. Hence, the application takes advantage of the already deployed WiFi localization system and Figure 27 displays the elements and connections of the system for acquiring location data. In particular, the exact location of the access points is known and used by the ‘getLocation’ servlet implemented in the Tomcat web server to retrieve the user’s position.

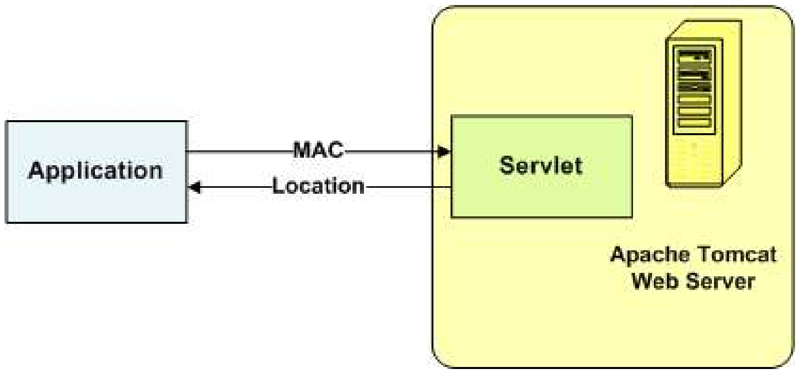


Figure 27: Elements and connections for acquiring location data

6.3.6 System overview

A high level system overview of the prototype solution for a ambient home care application is shown on Figure 28. There are several elements that compose the framework: a remote monitoring tool, a set of reusable widgets running independently

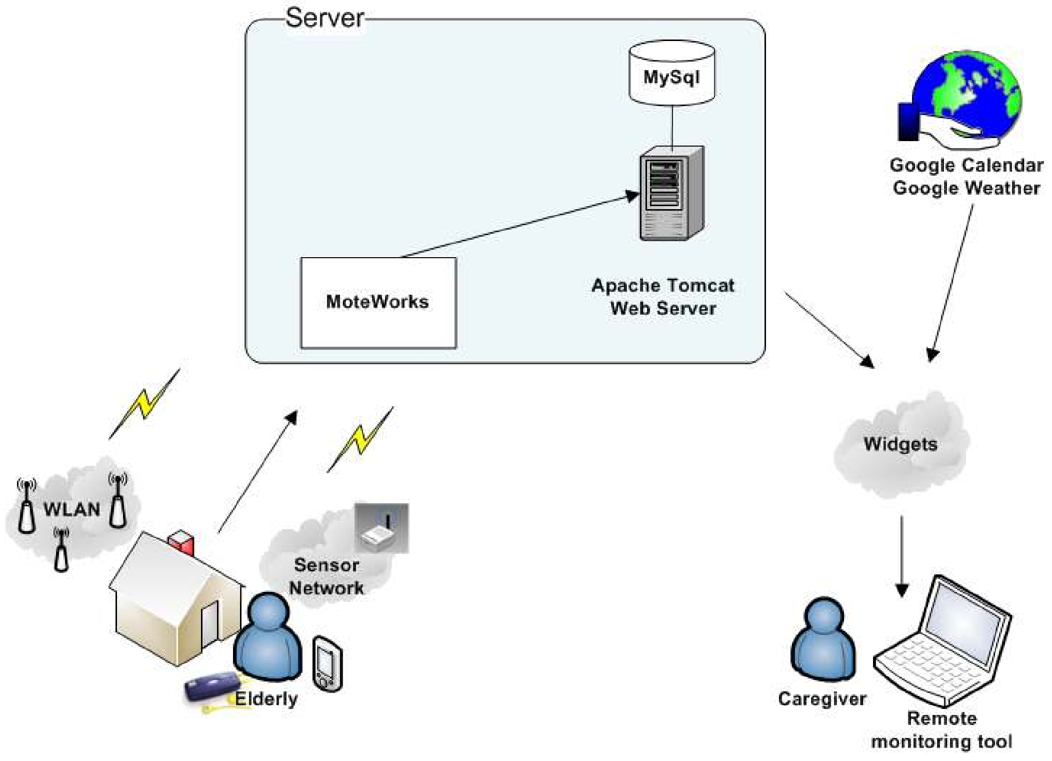


Figure 28: System overview of the prototype solution

from the application, a collection of sensors distributed in the person's home, a set of devices that the person always carries with him (PDA/mobile phone with RFID reader, heart monitor), access to some public web services (Google Calendar and Google Weather), and access to Apache Tomcat web server that encapsulates the processing of some of the data that the widgets deliver to the application.

The widgets are run beforehand and independently from the applications. When the remote monitoring tool is started it initializes several aggregators that acquire user specific information and subscribes to the widgets of interest by matching them according to certain criteria (constant and non-constant attributes defined in the widget), which in turn supply the application with notifications about new sensed context data.

The distribution of the access points enable localization of the user within his home. The WifiPositionWidget queries the web server which successively calculates the user's position and sends back a structured data with this info.

The data that the sensors acquire is sent over-the-air and gathered by the MoteWorks server. This info is requested from a servlet deployed on the web server, and a structured data is delivered to the EnvironmentalParametersWidget on its demand.

The same applies to the read RFID tags. The read data is being sent to the web server, processed, structured depending if it is a medication data or a location data and further delegated to the RfidDataWidget.

Instructed from the application, the ECG view can be activated. The HeartMonitorWidget then starts querying the web server for ECG data retrieved for the specific user that wears the heart monitor. The retrieval is terminated when the view is close.

All these data is merged and its interaction is aggregated into a joined application logic that displays state data and generates notifications in an application which is run on location remote from the position of the elderly person.

7. Conclusion and future work

The main goal of this thesis is to conceive and design a platform for context-aware services in a mobile-computing environment. In this chapter a conclusion is given based on the analysis performed followed by additional guidelines for future work.

7.1 Conclusion

This thesis is focusing on exploring the Context Toolkit, as a sample of a context-aware framework, detecting its strengths and weaknesses, identifying the possible enhancements and successively proposing a design of a new context aware framework.

From the analysis performed on the Context Toolkit, it can be concluded that it possesses a number of good design principles such as: reusability, distributed approach, resource discovery mechanism, storage of context data, automatic unsubscribing, context monitoring and feedback etc. However, weaknesses are present as well: the discoverer is a single point of failure, there is a need for clocks synchronization, there is no automatic restarting, there is lack of interoperability support, privacy and security are not yet implemented, there is no reasoning engine, and there is an absence of a conflict solving mechanism.

In this thesis, some of these issues were closely looked into and examined. For example, a quality of context mechanism is proposed for improving the quality of the application by making it more reliable and offering service adequate to the application developer's or user's demands. Also several proposals were suggested as a possible way of improving the resource discovery mechanism. Moreover, a matter that would need further focus is the interoperability, in particular making the context data and services provided from one platform available for discovery and usage by others as well. A mechanism has to be developed that will interface with different platforms and will reuse the context sources implemented with different frameworks by providing a common interface that everybody can interact with.

A prototype application has also been developed for the purpose of evaluating the Context Toolkit and obtaining hands-on experience of building a context-aware application. Each widget was easily developed, but integration of all the services has not been a seamless task, since the way they interact between each other has to be carefully established and implemented. Another difficulty was the testing; setting up all the involved context parameters on a joined timescale that matches the present, at times was found somewhat troublesome. Also, dealing with different technologies required studying all the sensors' specifications and interfaces just to be able to extract the context info needed for the application.

Consequently, after gaining practical experience with developing a sample application and analyzing the Context Toolkit, a conceptual model of a context-aware platform was proposed. It defines the components necessary for building context-aware applications, and the interactions between them. The suggested design is general and should fulfill the demands of a number of application areas of the context-aware application development. It is a combination of several studied architectural designs. It comprises the widget paradigm, the blackboard model and complies with the networked services approach.

Some of the platforms reviewed totally lack important modules for building a complete context-aware application. Therefore, when designing the platform several aspects were taken into consideration such as: the necessity of quality of context handling, conflict resolution, reasoning engine and a privacy and security module. Approaches such as user-centered design, prediction of user's preferences and testing these assumptions are important for the design of the platform therefore an analyzer of the user behavior was also a component added in the proposed architecture. Furthermore, a general flow of actions has been defined that presents the main idea of the system's functioning.

Finally, the designed platform does not drastically defer from the other analyzed. Most of them follow the layered approach and have similar abilities because they are based on similar methodologies. What defers in the proposed model is the completeness. It covers the central points of concern that improve the performance and the overall quality of the developed applications.

7.2 Future work

Further work in this area encompasses several aspects, outlined as following:

The prototype was developed in order to get familiar with the Context Toolkit, its drawbacks, missing functionalities, sample design principles, software practices, technologies and examine the need for its future expansion. It has not been built in order to satisfy scalability issues and it was developed without the intention to go in real production. It serves only as a proof of concept. However, there are issues in its current implementation that can be further improved. The application could be expanded to support remote monitoring of several persons at a time, sending SMS messages to the elderly and his family members in case of an alerting situation and enriching the application with more context information in order to get more exact information about the current situation. New services can also be offered: fall detection, abnormal heart rate state alarm, nutritional recommendations, fridge monitoring system (by tracking the items left inside the refrigerator and constructing the shopping list) etc.

Concerning the Context Toolkit, its future development can cover: complete implementation of the quality of context mechanism including a conflict resolution engine which will be standardized and will adapt on the information provided, extending the widget description with other parameters that describe it in greater details, introducing redundancy in the system and replicating the Discoverer component for achieving better reliability, and adding a module that will aid interoperability by making the context info more available to other frameworks. Moreover, proposing solutions for the other weaknesses detected and their implementation can also be a topic for further work.

Last, the proposed context-aware framework in this thesis is general. It defines the architectural design principles and identifies the necessary components. Further work would include more detailed specification of the building elements and selection of the algorithms that some need to define. This should be followed with an implementation of the proposed solutions and sample application development which will serve for evaluation purposes.

8. References

[1] Bill N. Schilit, Marvin M. Theimer, “Disseminating Active Map Information to Mobile Hosts”, IEEE Network, Volume: 8, Issue: 5, September/October 1994, pp. 22-32, available at: http://impact.asu.edu/~cse591uc/papers/00313011.pdf, last visited: June 2008

[2] B. Schilit., M. Adams, R. Want., “Context Aware Computing Applications”, Workshop on Mobile Computing Systems and Applications, December 1994, pp. 85-90, available at: http://www.ubiq.com/want/papers/parctab-wmc-dec94.pdf, last visited: June 2008

[3] Jesper J. Bisgaard, Morten Heise, Carsten Steffensen, “How is Context and Context-awareness defined and Applied? A survey of Context-awareness”, available at: http://www.csconsult.dk/rap/inf7_con.pdf, last visited: June 2008

[4] Bill Schilit, Norman Adams, Roy Want, ”Context-Aware Computing Applications”, Workshop onMobile Computing Systems and Applications, 1994. Proceedings., Santa Cruz, CA, USA, available at: http://sandbox.xerox.com/want/papers/parctab-wmc-dec94.pdf, last visited: June 2008

[5] J. Pascoe, N. Ryan, D. Morse, “Using While Moving: HCI issues in fieldwork”, ACM Transactions on Computer-Human Interaction (TOCHI), Volume 7, Issue 3, September 2000, pp. 417 - 437, available at: http://delivery.acm.org/10.1145/360000/355329/p417-pascoe.pdf?key1=355329&key2=9722834121&coll=GUIDE&dl=GUIDE&CFID=34047412&CFTOKEN=41193238, last visited: June 2008

[6] Anind K. Dey, Gregory D.Abowd, “Towards a Better Understanding of Context and Context-Awareness”, Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, Vol. 1707, 1999, pp. 304-307, available at: ftp://ftp.gvu.gatech.edu/pub/gvu/tr/1999/99-22.pdf, last visited: June 2008

[7] Terry Winograd, “Architectures for Context”, Human Computer Interaction, Volume 15, no. 4, 2000, pp. 263-322, available at: http://hci.stanford.edu/~winograd/papers/context/context.pdf , last visited: June 2008

[8] Oriana Riva, “A Conceptual Model for Structuring Context-Aware Applications”, Fourth Berkeley - Helsinki Ph.D. Student, Workshop on Telecommunication Software Architectures, June 2004, available at: http://citeseerx.ist.psu.edu/viewdoc/summary;jsessionid=8199F1E754DF6887BED7B3223EC8F345?doi=10.1.1.97.3855, last visited: May 2008

[9] H. Chen, T. Finin, A. Joshi, “An ontology for context-aware pervasive computing environments”, Special issue on Ontologies for Distributed Systems, Knowledge Engineering Review, Vol.18, No.3, 2004, pp.197-207, available at: http://www.cs.umbc.edu/~finin/papers/ijcai03OntologiesCAPCE.pdf, last visited: June 2008

[10] Matthias Baldauf, “A survey on context-aware systems”, Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, 2007, pp. 263 - 277, available at: https://berlin.vitalab.tuwien.ac.at/~florian/papers/ijahuc2007.pdf, last visited: June 2008

[11] Jadwiga Indulska, Peter Sutton, “Location Management in Pervasive Systems”, Conferences in Research and Practice in Information Technology Series; Vol. 34, Pages: 143 - 151 , Adelaide, Australia, 2003, available at: http://crpit.com/confpapers/CRPITV21WIndulska.pdf, last visited: June 2008

[12] Wikipedia – Web Ontology Language (OWL), available at: http://en.wikipedia.org/wiki/Web_Ontology_Language, last visited: June 2008

[13] Barbara T. Korel, Simon G. M. Koo, “Addressing Context Awareness Techniques in Body Sensor Networks”, 21st International Conference on Advanced Information Networking and Applications Workshops, Volume 2, Niagara Falls, ON, Canada, May 2007, pp. 798 - 803, available at: http://ieeexplore.ieee.org/iel5/4221005/4224052/04224203.pdf?arnumber=4224203, last visited: June 2008

[14] Kari Sentz, “Combination of Evidence in Dempster-Shafer Theory”, Sandia Technical Report, SAND 2002-0835, Sandia National Laboratories, Albuquerque, NM, April 2002, available at: http://www.sandia.gov/epistemic/Reports/SAND2002-0835.pdf, last visited: June 2008

[15] Abhishek Singh, Michael Conway, “Survey of Context aware Frameworks - Analysis and Criticism”, UNC-Chapel Hill ITS, The University of North Carolina, 2006, available at: http://its.unc.edu/teap/tap/core/caf_review.pdf, last visited: May 2008

[16] Kristian Ellebæk Kjær , "A Survey of Context-Aware Middleware", Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering, Innsbruck, Austria, 2007, pp. 148-155, available at: http://delivery.acm.org/10.1145/1340000/1332069/p148-kjar.pdf?key1=1332069&key2=3382834121&coll=&dl=&CFID=34049128&CFTOKEN=45233244, last visited: June 2008

[17] Julien Pauty, Davy Preuveneers, Peter Rigole, Yolande Berbers, "Research Challenges in Mobile and Context-Aware Service Deevelopment", Future Research Challenges for Software and Services Conference, pp.141-148, Vienna, Austria, 2006 available at: http://www.loms-itea.org/publications/frcss06.pdf, last visited: June 2008

[18] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges", Personal Communications, IEEE Wireless Communications, Volume: 8, Issue: 4, Aug 2001, pp. 10-17, available at: http://www.cs.cmu.edu/~aura/docdir/pcs01.pdf, last visited: June 2008

[19] Christoph Endres, Andreas Butz, Asa MacWilliams, "A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing", Mobile Information Systems Journal, 1(1), Jan-March 2005, pp. 41-80, available at: http://www.medien.ifi.lmu.de/pubdb/publications/pub/butz2005ubicompsurvey/butz2005ubicompsurvey.pdf, last visited: May 2008

[20] Alan Newberger, Anind Dey, "Designer Support for Context Monitoring and Control", Technical Report IRB-TR-03-017, Intel Research, Berkeley, June 2003, available at: http://www.intel-research.net/Publications/Berkeley/070920031122_141.pdf, last visited: May 2008

[21] Anind K. Dey, Gregory D. Abowd, Daniel Salber, "A Conceptual Framework and a Toolkit for Supporting the rapid Prototyping of Context-Aware Applications", Human-Computer Interaction Journal, Vol. 16 (2-4), 2001, pp. 97-166, available at: http://citeseer.ist.psu.edu/cache/papers/cs/32861/http:zSzzSzwww.cc.gatech.eduzSzfcezSzctkzSzpubszSzHCIJ16.pdf/dey01conceptual.pdf , last visited: May 2008

[22] Anind K. Dey, Daniel Salber, Gregory D. Abowd, Masayasu Futakawa, "An Architecture to Support Context-Aware Applications", GVU Technical Report, GIT-GVU-99-23, Georgia Institute of Technology, June 1999, available at: ftp://ftp.gvu.gatech.edu/pub/gvu/tr/1999/99-23.pdf , last visited: May 2008

[23] Context Toolkit - Tutorial, available at: http://contexttoolkit.sourceforge.net/documentation/tutorial/, last visited: May 2008

[24] Adrian K. Clear, Stephen Knox, Juan Ye, Lorcan Coyle, Simon Dobson, Paddy Nixon, "Integrating Multiple Contexts and Ontologies in a Pervasive Computing Framework", in Contexts and Ontologies: Theory, Practice and Applications, Riva Del Garda, Italy, August 2006, pp. 20–25, available at: http://www.cs.ucd.ie/UserFiles/publications/1148314115874.pdf, last visited: May 2008

[25] Wikipedia - XML, available at: http://en.wikipedia.org/wiki/XML, last visited: May 2008

[26] Thomas Buchholz, Axel Kupper, Michael Schiffers, "Quality of Context : What It Is And Why We Need It", In Proceedings of the Workshop of the HP Open View University Association 2003 (HPOVUA 2003), Geneva, July 2003, available at : http://media.cs.tsinghua.edu.cn/~qinwj/readings/paper/buchholz-hpovua03.pdf, last visited : May 2008

[27] Tobias Zimmer, "QoC : Quality of Context - Improving the Performance of Context-Aware Applications", Adjunced Proceedings of Pervasive 2006, September 2006, available at : http://www.pervasive2006.org/ap/pervasive2006_adjunct_4E.pdf, last visited : May 2008

[28] Kamran Sheikh, Maarten Wegdam, Marten van Sinderen, "Middleware Support for quality of Context in Pervasive Context-Aware Systems", Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops, 2007, pp 461-466 , available at: http://portal.acm.org/citation.cfm?id=1263543.1263826&coll=&dl=ACM, last visited: June 2008

[29] Michael Krause, Iris Hochstatter, "Challenges in Modelling and Using Quality of Context (QoC)", Mobility Aware Technologies and Applications, 2005, pp. 324-333, available at: http://www.springerlink.com/content/f72510g13705x3t1/fulltext.pdf, last visited: June 2008

[30] Giovanni Cortese, Massimiliano Lunghi, Fabrizio Davide, "Context-Awareness for Physical Service Environment", Ambient Intelligence: The Evolution of Technology, Communication and Cognition Towards the Future of Human-Computer Interaction , 2005, pp. 71-97, available at : http://books.google.com/books?id=sACug4nZDmUC&printsec=frontcover&dq=Giovanni+Cortese,+Massimiliano+Lunghi,+Fabrizio+Davide,+%C2%A8Context-Awareness+for+Physical+Service+Environment%C2%A8,+&lr=&source=gbs_summary_r&cad=0, last visited : May 2008

[31] Tao Gu, Hung Keng Pung, Da Qing Zhang, "A Service-Oriented Middleware for Building Context-Aware Services", Journal of Network and Computer Applications, Volume 28, Issue 1, January 2005, pp.1 – 18, available at: http://www1.i2r.a-star.edu.sg/~tgu/gutao/paper/SOCAM_gutao.pdf, last visited: June 2008

[32] Wikipedia – Web service, available at: http://en.wikipedia.org/wiki/Web_services, last visited: June 2008

[33] Wikipedia-UDDI, available at: http://en.wikipedia.org/wiki/UDDI, last visited: June 2008

[34] Petteri Nurmi, Patrik Floréen, "Reasoning in Context-Aware Systems", 2004, available at : http://www.cs.helsinki.fi/u/ptnurmi/papers/positionpaper.pdf, last visited : June 2008

[35] Reinforcement Learning e-Book, available at: http://www.cs.ualberta.ca/%7Esutton/book/ebook/node7.html, last visited: June 2008

[36] Giuseppe Riva, "Ambient Intelligence in Health Care", Cyberpsychology & Behaviour, Volume 6, Number 3, 2003, pp. 295-300, available at: http://labstudenti.unicatt.it/doo/autori/Username%20n.%2007/p295_s.pdf, last visited: June 2008

[37] Sauli Tiitta, "Identifying elderly people´s needs for communication and mobility", Proc. Include, Helen Hamlyn Research Centre. London, England, March 2003, pp. 266-271, available at: http://www.hiit.fi/~tiitta/articles/Identifying_elderly_peoples_needs_for_communication_and_mobility.pdf, last visited: May 2008

[38] Oxford University: "Injuries of Aging Person's Report", 2008, available at: http://www.jr2.ox.ac.uk/bandolier/band25/b52-2.html, last visited: May 2008

[39] I. Korhonen, J. Parkka, M. Van Gils, "Health Monitoring in the Home of the Future. Infrastructure and Usage Models for Wearable Sensors"; Engineering in Medicine and Biology Magazine, IEEE Volume 22, Issue 3, May-June 2003, pp. 66 - 73, available at: http://ieeexplore.ieee.org/iel5/51/27287/01213628.pdf, last visited: June 2008

[40] Jonghwa Choi, Dongkyoo Shinn, Dongil Shin, "Research and Implementation of the Context-Aware Middleware for Controlling Home Appliances", IEEE Transactions on Consumer Electronics, Volume 51, Issue 1, Feb 2005, pp. 301 - 306, available at: http://ieeexplore.ieee.org/iel5/30/30482/01405736.pdf?arnumber=1405736, last visited: June 2008

[41] Seungho Baek, Hyunjeong Lee, Shinyoung Lim, Laedoo Huh, "Managing Mechanism for Service Compatibility and Interaction in Context-aware Ubiquitous Home", IEEE Transactions on Consumer Electronics, Volume 51, Issue 2, May 2005, pp. 524 - 528, available at: http://ieeexplore.ieee.org/iel5/30/31480/01467996.pdf?arnumber=1467996, last visited: June 2008

[42] Seung-Ho Baek, Eun-Chang Choi, Jae-Doo Huh, ¨Design of Information Management Model for Sensor Based Context-Aware Service in Ubiquitous Home¨, International Conference on Convergence Information Technology 2007, Gyeongju, Republic of Korea, Nov. 2007, pp. 1040-1047, available at: http://www.ieeexplore.ieee.org/iel5/4420216/4420217/04420396.pdf?tp=&isnumber=4420217&arnumber=4420396, last visited: June 2008

[43] Hyunjeong Lee, Jongwon Kim, Jaedoo Huh, "Context-Aware Based Mobile Service for Ubiquitous Home", The 8th International Conference Advanced Communication Technology 2006, Volume 3, Korea, Feb. 2006, available at : http://ieeexplore.ieee.org/iel5/10826/34122/01625957.pdf?isnumber=34122&prod=C

NF&arnumber=1625957&arSt=+4+pp.&ared=&arAuthor=Hyunjeong+Lee%3B+Jongwon+Kim%3B+Jaedoo+Huh, last visited: June 2008

[44] Dong-Oh Kang, Hyung-Jik Lee, Eun-Jung Ko, Kyuchang Kang, Jeunwoo Lee, "A Wearable Context Aware System for Ubiquitous Healthcare", Proceedings of the 28th IEEE, EMBS Annual International Conference, New York, USA, Aug-Sep 2008, pp.5192-5195, available at: http://ieeexplore.ieee.org/iel5/4028925/4030573/04398624.pdf, last visited: June 2008

[45] Dong-oh Kang, Kyuchang Kang, Hyung-jik Lee, Eun-jung Ko, Jeunwoo Lee, "A Systematic Design Tool of Context Aware System for Ubiquitous Healthcare Service in a Smart Home", Future generation communication and networking (fgcn 2007), Volume: 2, Dec. 2007, pp. 49-54, available at: http://ieeexplore.ieee.org/iel5/4426076/4426188/04426202.pdf?isnumber=4426188&prod=CNF&arnumber=4426202&arSt=49&ared=54&arAuthor=Kang%2C+Dong-oh%3B+Kang%2C+Kyuchang%3B+Lee%2C+Hyung-jik%3B+Ko%2C+Eun-jung%3B+Lee%2C+Jeunwoo, last visited: June 2008

[46] Barbara T. Korel, Simon G.M. Koo, "Addressing Context Awareness Techniques in Body Sensor Networks", 21st International Conference on Advanced Information Networking and Applications Workshops 2007, Volume 2, Niagara Falls, Canada, May 2007, pp.798 - 803, available at: http://ieeexplore.ieee.org/iel5/4221005/4224052/04224203.pdf?arnumber=4224203, last visited: June 2008

[47] Ilkka Korhonen, Paula Paavilainen, Antti Särelä, "Application of Ubiquitous Computing Technologies for Support of Independent Living of the Elderly in Real Life Settings", Proc. UbiHealth 2003: 2nd Int'l Workshop Ubiquitous Computing for Pervasive Healthcare Applications, Seattle, Washington, October 2003, available at: http://www.healthcare.pervasive.dk/ubicomp2003/papers/Final_Papers/2.pdf, last visited: June 2008

[48] Sumi Helal Carlos Giraldo,Youssef Kaddoura,Choonhwa Lee, Hicham El Zabadani, William Mann, "Smart Phone Based Cognitive Assistant", Proc. UbiHealth 2003: 2nd Int'l Workshop Ubiquitous Computing for Pervasive Healthcare Applications, Seattle, Washington, October 2003, available at: http://www.healthcare.pervasive.dk/ubicomp2003/papers/Final_Papers/14.pdf, last visited: June 2008

[49] Anand Agarawala, Saul Greenberg, Geoffrey Ho, "The Context-Aware Pill Bottle and Medication Monitor", In Video Proceedings and Proceedings Supplement of the UBICOMP 2004 Conference, Nottingham, England, September 2004, 4 minute video and 2-page summary, available at: http://grouplab.cpsc.ucalgary.ca/grouplab/uploads/Publications/Publications/2004-Pillbottle.UBICOMP.pdf, last visited: June 2008

[50] Ken Fishkin, Min Wang, "A Flexible, Low-Overhead Ubiquitous System for Medication Monitoring", Intel Research Seattle Technical Memo IRS-TR-03-011, October 2003, available at:

http://informationmediary.com/media/pdf/IntelSystemForMedicationMonitoring.pdf, last visited: June 2008

[51] Nathalie Bricon-Souf, Conrad R. Newman, "Context Awareness in Health Care: A Review", International Journal of medical informatics 76, 2007, pp. 2-12, available at : http://www2.chi.unsw.edu.au/pubs/souf_newman_07.pdf, last visited : June 2008

[52] Jakob E. Bardram, "Hospitals of the Future- Ubiquitous Computing support for Medical Work in Hospitals", In Proceedings of UbiHealth, Workshop on Ubiquitous Computing for Pervasive Healthcare Applications, Seattle, Washington, October 2003, available at :
http://www.healthcare.pervasive.dk/ubicomp2003/papers/Final_Papers/13.pdf, last visited : June 2008

[53] Alive Technologies – Alive Heart Monitor Specification, available at: http://www.alivetec.com/products.htm, last visited: June 2008

[54] Crossbow Home Page - MICAz 2.4GHz specification, available at: http://www.xbow.com/Products/productdetails.aspx?sid=164, last visited: June 2008

[55] Crossbow Home Page - MDA Data Acquisition Boards, available at: http://www.xbow.com/Products/productdetails.aspx?sid=178, last visited: June 2008

[56] Wireless Dynamics Inc – SdiD 1010 Specification, available at: http://www.wdi.ca/docs/SW06-0007-DS%20-%20SDiD%201010.pdf, last visited: June 2008