GECAF: A GENERIC AND EXTENSIBLE FRAMEWORK FOR DEVELOPING CONTEXT-AWARE SMART ENVIRONMENTS

Angham SABAGH

School of Computing, Science and Engineering
University of Salford, Salford, UK

Submitted in Partial Fulfilment of the Requirements of the
Degree of Doctor of Philosophy, September 2011

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my supervisor Dr. Adil Al-Yasiri for his contributions of time, ideas, experience and patience in making this PhD thesis possible. His enthusiasm and motivation encouraged me to achieve all the requirements sought for.

I would also like to thank my family, parents, sisters, and brother for their continuous patience and support; especially to my husband for his encouragement and assistance throughout the period of study.

Lastly, I offer my regards and gratitude to all those who offered their support throughout this work.

ABBREVIATIONS

4W1H	Who, Where, When, What, and How
5W1H	Who, Where, When, What, Why, and How
A-CoBrA	Activity-Context Broker Architecture
ADC	Analogue to Digital Converter
API	Application Programming Interface
CADBA	Context-aware Architecture Based on Context Database
CADL	Context-aware Architecture Deployment Language
CALA	Context-Aware Learning Architecture
CALA-ONT	Context-Aware Learning Architecture – Ontology
CAMUS	Context-Aware Middleware for URC System
CASS	Context-Awareness Sub-Structure
CCA	Context Collecting Agent
CDL	Context Description Language
CID	Context Identification Code
CLIPS	C Language Integrated Production System
CMF	Context Management Framework
CoBrA	Context Broker Architecture
COBRA-ONT	Set of ontologies provided by CoBrA
CONON	CONtext ONtology
CORTEX	CO-operating Real-time senTient objects: architecture and EXperimental evaluation
CPU	Central Processing Unit
DB	Database
DOM	Document Object Model
EID	Event identification

FIFO	First In First Out
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile communications
GUI	Graphical User Interface
HTML	HyperText Markup Language
ID	IDentification
IIS	Internet Information Services
IR	Infrared
ISIS	Internal Situation Store
KS	Knowledge Source
LDAP	Light weight Directory Access Protocol
MVC	Model View Controller
MySQL	My Structure Query Language
ORM	Object Role Modelling
OS	Operating System
OWL	Web Ontology Language
OWL-DL	Web Ontology Language Description Logic
PC	Personal Computer
PDA	Personal Digital Assistant
PERKAM	PERsonalised Knowledge Awareness Map
PHP	PHP: Hypertext Preprocessor
QoS	Quality of Service
RAM	Random Access Memory
RDF	Resource Description Framework
RF	Radio Frequency
RFID	Radio Frequency Identification

RID	Rule Identification Code
ROM	Read Only Memory
RPC	Remote Procedure Call
RPN	Reverse Polish Notation
RWM	Real World Model
SICL	Smart-Its Context Language
SOAP	Simple Object Access Protocol
SOCAM	Service-Oriented Context-Aware Middleware
SQL	Structured Query Language
ubi-UCAM	Unified Context-Aware Application Model
UDDI	Universal Description, Discovery and Integration
UML	Unified Modelling Language
USB	Universal Serial Bus
WCAM	Watcher, Controller, Action, and Model
WiFi	Wireless Fidelity
WSDL	Web Services Description Language
XML	eXtensible Mark-up Language

ABSTRACT

The new pervasive and context-aware computing models have resulted in the development of modern environments which are responsive to the changing needs of the people who live, work or socialise in them. These are called smart envirnments and they employ high degree of intelligence to consume and process information in order to provide services to users in accordance with their current needs. To achieve this level of intelligence, such environments collect, store, represent and interpret a vast amount of information which describes the current context of their users. Since context-aware systems differ in the way they interact with users and interpret the context of their entities and the actions they need to take, each individual system is developed in its own way with no common architecture. This fact makes the development of every context aware system a challenge. To address this issue, a new and generic framework has been developed which is based on the Pipe-and-Filter software architectural style, and can be applied to many systems. This framework uses a number of independent components that represent the usual functions of any context-aware system. These components can be configured in different arrangements to suit the various systems' requirements. The framework and architecture use a model to represent raw context information as a function of context primitives, referred to as Who, When, Where, What and How (4W1H). Historical context information is also defined and added to the model to predict some actions in the system. The framework uses XML code to represent the model and describes the sequence in which context information is being processed by the architecture's components (or filters). Moreover, a mechanism for describing interpretation rules for the purpose of context reasoning is proposed and implemented. A set of guidelines is provided for both the deployment and rule languages to help application developers in constructing and customising their own systems using various components of the new framework. To test and demonstrate the functionality of the generic architecture, a smart classroom environment has been adopted as a case study. An evaluation of the new framework has also been conducted using two methods: quantitative and case study driven evaluation. The quantitative method used information obtained from reviewing the literature which is then analysed and compared with the new framework in order to verify the completeness of the framework's components for different

situations. On the other hand, in the case study method the new framework has been applied in the implementation of different scenarios of well known systems. This method is used for verifying the applicability and generic nature of the framework. As an outcome, the framework is proven to be extensible with high degree of reusability and adaptability, and can be used to develop various context-aware systems.

Chapter 1

Introduction

During the past two decades, many evolutions have triggered new era in computing. Among those are the rapid growth of the micro-electronics industry, the advances in communication technology, and the internet evolution. As distributed systems are the current model of computing; it is therefore not limited to desktop applications. Moreover, the aforementioned developments drew the attention for the emergence of ubiquitous computing and context awareness paradigms. The new paradigms utilise resources effectively to solve the new challenges in almost every aspect of our life and present a better mode of living. As a result, different applications emerged; one of these applications is the smart environment. Ubiquitous computing is considered important and has been regarded as the driving force behind these environments [1]. It supports the implementation of smart environments, as a broad space of invisible computational devices are used, and are graciously integrated with human users, so services are provided for the users without their intervention [2]. Context awareness is one of the drivers of ubiquitous computing paradigm and another key for providing prevalent mode of services and interaction in smart environments. A system is context-aware if it exploits the surrounding information to adapt its behaviour without explicit intervention from the users. The surrounding information should be captured, modelled and abstracted to make sense to the application. Hence, a well designed model is a key to access context information in any context-aware system, and a general context model is of interest since many applications can benefit from it [3]. Moreover, a generic context-aware framework implementing the general model is required to assist building these systems. Finally, a generic rule mechanism to form various context dependent rules is also essential for context interpretation and abstraction. All these issues are discussed in this thesis and the related enabling technologies are then presented. Existing work is outlined to give insight into the previous researches done in this field.

1.1 Ubiquitous Computing

Ubiquitous computing is the third era of computing, and also known as pervasive computing where one person benefiting from many computers. It is subsequent to mainframes (single computer for many persons), and personal computers (one computer is dedicated for one person) [4]. This new computing model uses the potential of information and computing technologies to implicitly and naturally aid people in their daily life. It is the age of calm technology, where computing devices resides out in the world with people. Mark Weiser [2, 5] introduced the idea of ubiquitous computing by describing the 21st century computers as; "the most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it". This vision, perceives that computing devices are vanished into the background, so people are freed to using them without thinking; they give attention to tasks rather than tools (computing devices) that perform the task.

Ubiquitous computing uses a combination of computing devices that have many forms and sizes, where each suits particular task. These computing devices are embedded in almost every object in the environment including cars, clothing, appliances, and various consumer goods; communicating through an interconnected network. They could be either mobile (wearable or portable) or fixed; for example tabs, pads, boards, etc. Ubiquitous computing employs systems that adapt and react with people, predict their behaviour and requirements. As pervasive systems may be geographically large, heterogeneous and distributed sensing is necessary to interact with people and monitor their activities in a scalable mode [6]. Finally, pervasive systems gather information from many sensors to interpret users' situation which is based on their action. Therefore, there should be solutions for access rights to this information to protect it from unauthorised access. The enabling factors for the revolution of ubiquitous computing are the following: -

- The development of low power consumption microelectronic technology, small, cheap and light weight; like small cameras, microphones, fingerprint sensors, tabs, sensor networks, etc.

- The existence of advanced, smart, high performance and new mobile devices such as Personal Digital Assistant (PDA), smart phones, smart papers, live boards, etc.
- Increase in processing speed and storage capacity.
- The evolution of the internet, and web technology.
- Advances in communication technology, with high data transfer rate at lower cost; e.g. fibre optics, wireless and mobile technology (GPRS, Bluetooth, WiFi, etc.).
- Standardisation of system components.
- Advances in the field of information security.

Ubiquitous computing offers much convenience to our lives, by freeing people from direct interaction with computers. However, ubiquitous computing applications are complicated and are difficult to realise, as there are many aspects that should be considered when building these systems, such as lack of information and others regarding ethical and social issues.

1.2 Context-Aware Computing

The world is based on the surrounding information that our human bodies are able to sense and interpret. People are aware of their environment; they interact with the outside world through their five senses. Through them, they receive information, and then the nervous system conveys the information to the brain to infer what this information could be about and finally responds (do an action). It is in a similar natural way context-aware systems should work. The notion of context and context awareness is introduced by many researchers, and observed in many applications since the end of the past century. Different aspects of context-aware systems have been studied and explored; these include context dimensions, modelling techniques, system architecture, reasoning techniques and their applications. Frameworks, middleware, and infrastructure for context-aware systems are also considered and constructed; although these approaches are still evolving. Security and privacy of context information is another important issue which inspires users' confidence if it is well handled.

"Context-awareness is an inherent feature of pervasive computing" [7]; it eases the interaction with computers, by automating the way of collecting information and

adapting system behaviour. The word 'context' is defined as any information that can be used to characterise the situation of an entity. A system is context-aware if it utilises this information to change its behaviour without user intervention [8]. The surrounding information refers to identity (identification of a user in a service environment and could include user profile), time (when context is generated and valid), and location of use (physical and electronic space [9]). Other pieces of surrounding information are activity (users expression and behaviour), and environment (nearby people, weather condition, etc). All these pieces of context information are the primitive contexts which are acquired by software and/or hardware sensors. For more relevance and reliable context information, some systems employ the scheme of sensor fusion (combination of multiple sensors [10]). The acquired information could be either static or dynamic, as these pieces of information change over time, with location, user's situation and other contexts. Since context information is dynamic, historical context would be of great importance; it predicts user's intention, current and future context. Historical context can be used with other information from internal or external sources to enrich the context-aware system and form a new context. Consequently, a large repository is required to store historical data. Recently advances in Cloud computing storage, Google health and HealthVault will solve the growing problems of data and shared storage.

Context representation provides a vocabulary to express context to be easily exploited, stored as knowledge base, reused, and enables reasoning. Several approaches to modelling context are introduced; each represents context information differently, with some approaches using hybrid models. These models include key-value, mark-up scheme, graphical model, object oriented, ontology based, and other application specific models. Reasoning is another feature of context-aware systems; it is about the abstraction of context to find meaningful information. Moreover, context-aware systems cover a wide area of applications [11], for example tour guides, reminder systems, context-aware mobile services, context-aware response to emergency, etc. Context-aware applications are divided into three different categories based on how they make use of context information. These categories are: information tagging for later retrieval and use, information and service presentation to the user, and automatic execution of a service [8].

The above discussion reveals the main elements of a context-aware system; these are sensing, modelling, interpretation, aggregation, storage/retrieval, management, and application (provide services), these are summarised as follows: -

- The sensing element represents a variety of sensors that acquire information from the environment to form primitive context information.
- The modelling element transforms primitive context information into machine readable format.
- The interpretation element is used for reasoning i.e. infer the abstracted context (meaningful context).
- The aggregation element collects and fuses several pieces of abstracted context data in order to enrich the context information.
- The storage/retrieval element is used either to store the historical context information and to form the context knowledge base, or to retrieve information from (external and internal) sources to enrich the context information.
- The application element takes the action and provides services.
- The management element governs and manages the whole system process.

1.3 Smart Environments

Smart environments serve people in their everyday live, at home, work and social life; functioning invisibly and unobtrusively in the background, freeing people from tedious and routine tasks. They integrate technology and services, interact with people and other entities to assist them and deliver a better quality of living. These environments interact with individuals through embedded devices to automatically acquire information from their surrounding, predict and process gathered information so they become meaningful. This information is then used to adapt the environment according to the user intention and needs by providing customised services without technical operation or user intervention.

Smart environment belongs to the field of ambient intelligence; it is an augmented spacious environment occupied with many sensors, actuators and computing devices [12]. These devices are embedded and integrated into a distributed computing system to sense the environment, and execute intelligent logic on computing devices to serve its inhabitants by actuators. These environments predict and automate inhabitants' actions [13]. The decision of the executed action should be precise and error free to

avoid correction, which can lead to performing the action manually. Suo et al. [14] have described three phases of smart spaces; these are individual space (a smart human-computer interactive space), open smart space (use of mobile devices, roaming with the users to discover smart space environment and making use of the resources in the space) and smart community (multiple smart spaces communicating and coordinating with each other).

The design of such environments need a combination of contemporary technologies; these include smart devices, wireless mobile communication, new computing paradigm like distributed computing, middleware, and scalable systems [15, 16]. Smart environments employ three groups of resources [17]; these are control resources, context resources, and interaction resources. The application type determines what resources are required to accomplish its goal. Control resources are the controllable network devices that are accountable for changing the environment, such as security locks, fire alarms, etc. Context resources are sensors and environment information that generates context information. Interaction resources are human-computer interaction devices such as mobile phones, PDAs, computers TV set etc. In brief, the following features of smart environments can be found: -

- They are physical spaces with various functions; they use large number of heterogeneous (static and mobile) devices and sensors, like laptops, smart phones, Personal Digital Assistants (PDA) (a handheld device that has the function of cellular phone, fax, provide Internet connectivity, management of personal information, run application software, etc. PDAs can interface, and synchronise with computer systems, which may require some optional accessories). Other examples are Global Positioning System (GPS), cameras, sensor networks, Radio-Frequency Identification (RFID) technology (a device for tracking which is used to implicitly acquire identity information), etc.
- The communication infrastructure which uses remote communication between devices (wired and wireless) is made through technologies like WiFi, Infrared (IR), Bluetooth, and Radio Frequency (RF).
- They cover wide range of environment types such as homes, offices, workplaces, classrooms, and vehicles.
- The interaction with these environments is natural as between people in everyday's life. It will, therefore, free people from attention to computers. The

communication can be accomplished by using voice, gesture, biometric, implanted sensors, etc; and it should be event driven to avoid delay experienced in a polling operation.

- They facilitate the collaboration between users and devices through agents or embedded devices.
- There is a continuous interaction, or automatic capture of events to adapt the environment, and dynamic capability to manage and process information changes.
- They have the ability to model captured information; put the information in a format to become useful and machine readable.
- They have the ability of controlling, reasoning and decision making.
- They are context-aware, and have some perceptual capabilities, i.e. the ability to predict and be aware of the surrounding situation using low-level details and abstracted information acquired from different sources, and provide services without user intervention.
- These environments are user-friendly, seamless interaction, and with good quality of service (QoS).
- They should employ information security; i.e. what information is collected from users through sensing and monitoring, and how to prevent unauthorised access.

Smart environments employ pervasive context-aware computing paradigms in their construction, in particular location aware systems. In these systems various ways of identifying location can be used. Sensing is the main function of these systems, which provides means to automatically acquire information about the location. Some examples to the most common location and identity awareness devices are: -

1. The Radio Frequency Identification (RFID) [18]: This technology employs tags or transponders (smart cards, labels, etc.) that are carried by users or attached to objects. The RFID reader (transceiver controlled by microprocessor or digital signal processor) handles the reading and writing from/into these tags. Then, a coiled antenna supplements the read/write processes at a distance; where the antenna size determines the read range. It differs from bar codes in using radio waves to read data from tags rather than optically scanning the label on barcodes. The tag is a circuit that reflects electromagnetic energy emitted continuously by

the reader, in order to exchange information. RFIDs come in many sizes and shapes to suit the applications. They have three types: passive, active, and semi-passive. Passive tags do not need batteries but need external electromagnetic field received from the reader to initiate a signal transmission. Active tags need batteries, and can transmit signals once an external source is identified. Active tags have built-in electronics including sensors, microprocessors and input/output ports, which allow them to be used for a variety of applications [19]. Semi-passive RFID tags use battery to monitor the environment, but rely on the reader to supply its power for broadcasting signals to the reader. RFID tags have three storage types; read-write, read-only and write once - read many.

2. Wireless sensor networks [20]: These networks made up of a very large number of small nodes. Each node is equipped with sensors, analogue to digital converter (ADC), processor (CPU), storage memory, power supply, and transceiver to send and receive data, as shown in Figure 1.1. Sensors transform the physical quantity from the environment into electric signals, which are then converted into digital (by the ADC) to be processed and stored in the memory. Sensors are powered with low power batteries, which are mostly not rechargeable. As shown in Figure 1.2, sensor networks consist of nodes which are scattered in a sensor field (some geographical area), in order to sense the environment. The area is divided into clusters, and nodes in each cluster communicate with a gateway (or sink) to collect the data from the nodes. Sensor networks work in one of two modes of operation, where either the nodes communicate directly with the gateway of the cluster, or communicate through chaining.

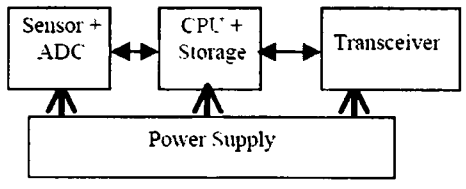


Figure 1.1: Architecture of a sensor node [20]

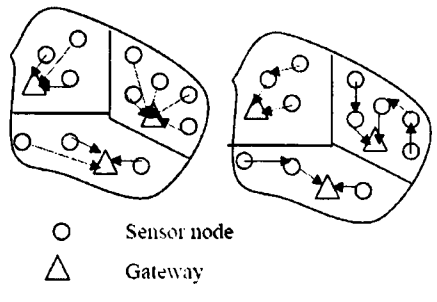


Figure 1.2: Two configurations of sensor networks [20]

3. Personal Digital Assistant (PDA) [21]: PDAs have small microprocessors, ROM memory to store the basic programmes, RAM memory to add users' programmes, and are powered by batteries. They have visual displays and audio capabilities. PDAs have different types of operating systems (OS): Palmtop, a special version of Microsoft Windows called Windows CE, Symbian, and Android. The Palmtop OS is used for handheld devices, the Windows CE is a general purpose OS for wireless and personal systems, and the Symbian OS is an open operating system for portable and handheld devices. Android OS is used for internet and mobile applications.
4. Global Positioning System (GPS) [22]: It is a navigation system based on a number of satellites placed into the orbit. A GPS system uses triangulation information to compute the user's location which can be made by measuring the distance to a number of satellites. In this case the time delay between the transmitted and the received signal is measured. Two dimensional location can be measured using the signal of three satellites, while three dimensional location can be calculated using four satellite signals.

1.4 Research Problem

From reviewing literature, it was found that context-aware systems are not widespread, where these systems are complex and very difficult to build. Many issues found with these systems; one of the issues related to these systems is the choice of the architecture used. It is found that each system use a different architecture style; however the layered architecture is the predominant one. The layered architecture has the benefits of hiding the low-level details, and supports separation between context acquisition and context use. It allows problem partitioning and supports enhancements and reuse. However, one of the disadvantages associated with such architecture is that not all systems are easily structured using this style. It does not actually solve problems of adaptability and restructuring. Besides, defining layers for some systems is not trivial as implementation can differ vastly from the model and it may be difficult to find the right levels of abstraction. This architecture style is usually found in networked and communication systems. Due to the complexity of context-aware systems, it is difficult to use this architecture style to support various applications as it lacks the ability to reconfigure, extensibility, and is difficult to standardise. It is to

say, there is an inherit problem with context-aware systems for not being similar. Also, there is no common pattern of system organisation; yet, each system is different and deals with different parameters and entities. Nonetheless, context-aware systems have common usual functions, which can be used recursively. Therefore, there is a need for a generic and extensible framework to support a variety of context-aware applications. The framework should facilitate the design of reusable and independent components that represent the common functions of any context-aware system, which can be assembled to construct a given application. The framework should simplify system maintainability, reliability, adaptability, extensibility, scalability, system development and application deployment rather than using a single monolithic structure.

1.5 Research Aim and Objectives

- **Research Aim**

The aim of this research is to propose a generic and extensible framework to support the design of a variety of context-aware systems. The framework should be made of standard components which represent the main functionalities of any context-aware system. These components should be reusable and could be extended to create new components based on abstracted ones. This would make the framework generic and extensible which enable application designers to build their own systems. It would also reduce the complexity of the design and simplify the implementation of a variety of context-aware applications.

- **Research Objectives**

Five objectives have been acknowledged to achieve the research aim. The objectives for this research are as below: -

1. To propose a general context model to abstract context primitives into a machine readable format. This model should consider dynamically changing context information such as historical data.
2. To design generic software architecture implementing the general context model provisions in pervasive computing.
3. To design an architecture deployment language (using XML code), to enable application developers identify the architecture components, the context in use

and also to specify a particular application on the network. An XML rule language is needed to identify rules used to infer abstracted context in the interpretation and reasoning processes.

4. To study and apply different design tools such as design patterns and frameworks, for the benefit of achieving the generic feature of the context-aware systems.
5. To apply the proposed model and the new framework to a typical smart environment in order to assess their functionalities and performance; also to discover failures and identify defects.

1.6 Research Questions

A number of questions have come up during the research process; these questions will be answered in the subsequent stages of the research process, these are: -

1. What is context-awareness? What are the main components of a context-aware system? Are there general context models and generic frameworks to construct these systems? Why do we need them, and what problems can be solved using these artefacts? How do the general model and the generic framework simplify the system design process? Also, how generic is the solution, and are these issues really applicable to any system?
2. How to make the context model general and how can we ensure that the framework is generic, extensible and scalable?
3. What is context history, why do we need it, and how can we add it to the proposed model?
4. What are the main categories of context-aware applications, and how can they be implemented?

1.7 Research Methodology

To understand and address questions related to the nature of the research problem the experimental research methodology is adopted which is primarily described and analysed by the positivist school [23]. This methodology aims at enabling researchers to follow a solid structure for defining a set of experimental steps which involve the definition layout, implementing, processing and analysing the results obtained by experimentation. This research undertakes the following activities to accomplish the

objectives and answer the research questions. Figure 1.3 shows the sequence of these activities.

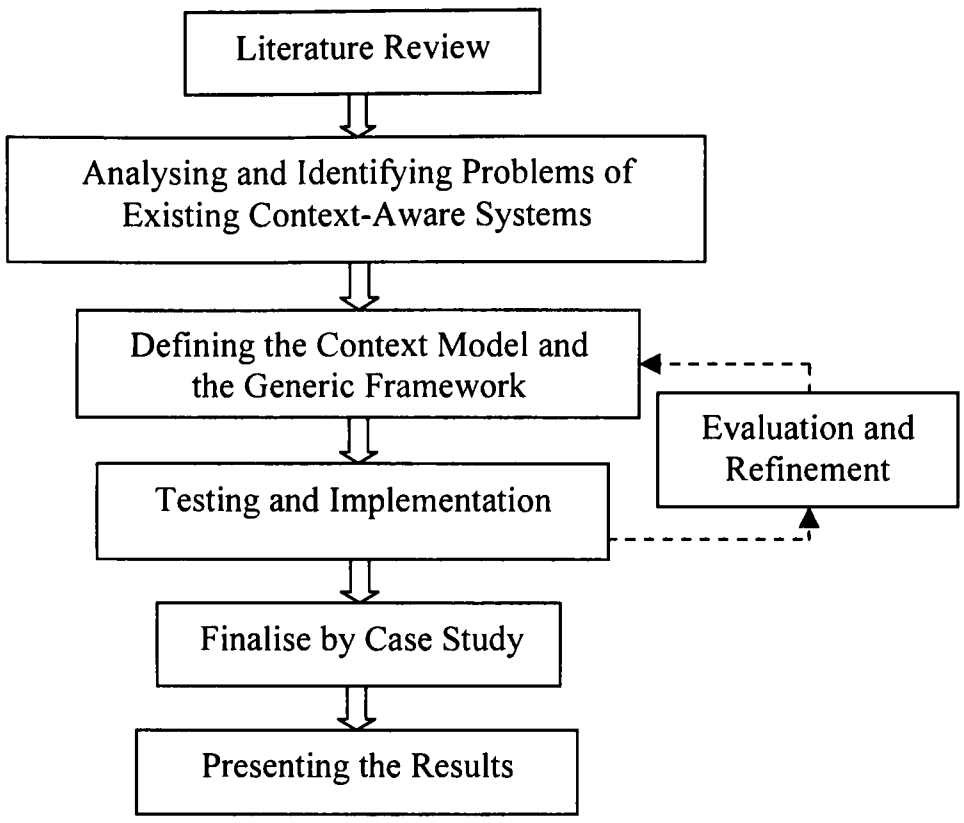


Figure 1.3: Research methodology

Activity 1: Literature Review

The first activity of the research is to conduct a review of previous related literature in order to acquire a thorough knowledge and understanding of pervasive context-aware systems. This will establish a more detailed definition of the research topic, the hypothetical framework and also identifying the core problem of existing context-aware systems. The experience learned from previous literature will prove useful during the development stage. Different types of context modelling techniques, context-aware system architectures and application categories are studied in details as they are essential in the domain of context-aware systems and smart environments. Also, a literature review has been conducted to gain a deep insight into existing software architecture styles in order to determine an architecture which best fits to the proposed system requirements. Finally, a number of researches regarding existing smart environments have been reviewed in chapters 2 and 3.

Activity 2: Analysing and Identifying Problems with Existing Context-Aware Systems

Problems pertaining to existing context-aware systems are identified in this activity. It started with studying the characteristics of context-aware systems and analysing the

way context information can be represented and stored. Attention then is given to find a general model for representing different pieces of context information. Afterwards a study of existing context-aware architectures has been conducted to analyse their functionalities in order to identify what building blocks constitute these architectures, and also to identify their strengths and deficiencies. An analysis of software architecture styles is carried out to find the most suitable style in designing the generic context-aware architecture. Application categories of context-aware systems are studied as well. From this analysis, a number of research questions and related problems are raised. As an outcome of the analysis, finding a solution to some of the existing problems and research questions was a big challenge.

Activity 3: Defining the Context Model and the Generic Framework

The contribution of the research is of three parts; firstly, a new and general context model is proposed and represented using a new XML based language. This simple model characterises context information as a function of context primitives. Secondly, an extensible and generic framework is proposed to represent the common functionalities (architecture components) of the context-aware systems which is followed by the design of an XML based deployment language for arranging the architecture components. Finally, an XML rule mechanism which is concerned with context interpretation is designed and implemented. A detailed design is discussed in chapter 4.

Activity 4: Testing and Implementation

In order to be confident with the proposed solution, its generic nature and functionality, the proposed framework should be illustrated and assessed. Therefore, a typical case study (smart classroom application) is picked to illustrate the proposed context-aware architecture. Moreover, a scenario is written to examine the generic framework performance and under different circumstances. The scenario describes a complex arrangement that deals with different context information aspects (environmental context information and context history, static or dynamic). The implemented application should also deals with different types of entities (human or non-human) such as persons and temperature. Object oriented programming language and other enabling technologies (Mark-up Languages, scripting language and web

services languages) will be used in the implementation. Detailed information is given in chapter 5.

Activity 5: Evaluation and Refinement

After studying problems related to context-aware systems, a preliminary idea to the design concept is proposed. It is based on the generic Pipe-and-Filter architecture. This concept is justified throughout studying and analysing existing context-aware architectures. Thereafter, an implementation for a given case study is set. Lessons learned from analysing the implemented case study are then used to find defects in the research concept and refine the proposed solution (generic architecture). Next, different case studies are examined carefully and critically evaluated to determine if further improvement and adjustments are necessary. This analysis resulted in adding missing building blocks to the designed architecture, refining the context model, and using an architecture style to govern the system and support multiple contexts. Afterwards, it was thought of building a framework that can be used by application developers to create the architectural building blocks. In this case creational design patterns are adopted to customise the building blocks of the context-aware system and adjust the design.

Finally, some criteria are used to evaluate the framework and justify our finding. This is accomplished by using the proposed framework building blocks to realise existing and well known context-aware systems. The selected systems are capable of using different context dimensions and application categories.

Activity 6: Finalise by Case Study

The smart classroom environment is finalised in this stage of the research. Such environment could use a networked system employing web services with advanced technologies including wireless devices and sensors (such as RFID) to achieve the required degree of intelligence. Then different contexts are used to test the functionality of the design.

Activity 7: Presenting the Results

The last activity comprised presenting the results through publications (three papers in national and international conferences), a poster, and finally writing up of a thesis to present the complete work to the University of Salford doctoral committee.

1.8 Contribution to Knowledge

During this PhD research, literature regarding context-aware systems has been reviewed, through which a number of issues have been found; one of these issues is the complexity of these systems. Accordingly, there is a need to propose a generic and extensible framework that minimises efforts required for building these systems. The contribution of this research is a new way for building context-aware systems, through building the generic and extensible framework that supports the design of a variety of these systems. During the design process a conceptual scheme has been set to show how different concepts collaborate to build the framework. The conceptual scheme included descriptions of a number of concepts that constitute the major elements of the new approach. These elements are a context classification, sensor terminology and a structure for describing the context model using a Context Description Language (CDL). Based on the developed context model, the generic framework that employs a Pipe-and-Filter architecture is built. This framework makes use of the reusable and independent components which represent the usual functions of context-aware systems. In this framework, six abstract and reusable building blocks are used. These are the acquisition, context representation, aggregation, data manipulation, interpretation, and application blocks. Context-aware Architecture Deployment Language (CADL) is used to put all these blocks together. CADL can also be used to represent different contexts which are distinguished using context identification code (CID). To govern the whole system operation and handle continuous context changes, the management block is employed. It assists in sequencing events and activates the whole system operation; see chapter 4.

The new framework depends on context model which arranges context information in a structured way in order to facilitate context reasoning and abstraction. Context history is also added to the model to enrich the context information. XML reasoning rules mechanism is proposed to support context abstraction. Therefore, different context related rules can be used and are distinguished using rule identification code (RID).

1.9 Thesis Layout

This thesis is organised as follows: -

- *Chapter 1* gives a brief introduction to the new computing technologies and smart environments. It also includes the problem domain, the research aim and objectives, research methodology, and the contribution to knowledge.
- *Chapter 2* depicts a brief review and analysis to context-aware systems, concentrates on context information dimensions. Context-aware systems considered context history are also reviewed to address related issues, as context information is dynamic, and changes over time. Context modelling is also studied and all modelling techniques were reviewed. A brief review to context-aware applications and applications in smart environments are also given.
- *Chapter 3* presents a literature review and analysis which concentrates on software architecture styles. Then challenges and issues of context-aware systems regarding their architectures, and the software elements that constitute these systems are given.
- *Chapter 4* demonstrates the research contribution. It gives details of the design development of a general context model, a new generic Pipe-and-Filter architecture and a rule mechanism. It also explains the languages used to assemble the architecture components and to set the interpretation rules. Moreover, different action types and applications categories will be given. Finally, scenario of a smart classroom is given as a case study, which considers all the relevant issues for the design progress.
- *Chapter 5* presents the system framework; it describes the realisation and deployment of the generic architecture using different creational design patterns. Considering the smart classroom scenario, different examples are given to test the framework functionality.
- *Chapter 6* adopts a set of heuristics to validate the proposed framework. Then two methods (quantitative and case study) are used to evaluate the framework. The quantitative method includes a comparison of some well known context-aware systems with the proposed framework. The case study method includes realising existing and well known systems using the framework building blocks.
- *Chapter 7* gives concluding remarks of the proposed framework, and discusses some possible directions for future research.

Chapter 2

Context Representation and Applications - Review

The notion of context and context awareness are introduced by many researchers, and observed in many applications since the end of the past century; although these systems are still at their early stages. This chapter presents a review of existing work related to context-aware systems to study their features and design. The background research covers challenges to context information dimensions: context primitives and context history. It also covers issues in context modelling and context-aware applications. At the end of the chapter, smart environment applications with preference to context-aware pervasive computing are reviewed and discussed in terms of their capability and performance in handling context information.

2.1 Introduction to Context and Context Awareness

Context-aware systems use the current status of users and other entities in the environment to adapt their behaviour according to the instantaneous requirements of the users and other objects that interact with them. Such systems need to define what constitutes the context (current status) of those entities and take actions accordingly. Researchers look for a seamless definition to context and context awareness, to assist them recognise the margins of context-aware computing, help application designers select context to use, structure context in applications, and choose what context-aware features to implement [8]. The following shows an overview to how researchers define context and context awareness.

The notion of context-aware computing was first introduced by Schilit et al. [9]; they defined context as the constantly changing environment. Then, Schmidt et al. [24, 25] simply defined context as that which surrounds and gives meaning to something. Schmidt et al. [26] also described context awareness as knowledge about

users and IT devices state, including surroundings, situation and location. According to Salber et al. [27] context is described as any environmental information that is relevant to the interaction between user application, and that can be sensed by the application. A more comprehensive and most usable definition to context and context awareness is given by Dey et al. [8]. This definition makes it easier for application developers to specify the context in application scenario; "context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". Their definition to context awareness is: "a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on user's task". Another definition to context is given by Gross et al. [28], which described context-awareness as a set of attributes (name, administrator, human members of context, location, artefact, applications, events, access control list of a context, and related contexts). They extend Dey and Abowd's [8] definition to context awareness by extending user's task to contain information about users whole work context. To conclude, it was found that Dey's definition is the closest in spirit to our work and realistic, therefore this definition is adopted in the context of this research. Also, the characteristics of context information are: dynamic, inconsistent, and dependable on application situation [29]. Context is the meaningful properties about that which surrounds and gives meaning to something else [30].

2.2 Aspects of Context Representation

An in-depth review to existing work is done in order to be aware of what constitutes context, and how context information are represented and used in different applications. In the following sections challenges to some of these aspects will be presented.

2.2.1 Characteristics of Context Information

To design a context-aware system, it is important to identify the characteristics of context information. These include the pieces of context information (context primitives), how context is described, classified and grouped. Many previous works show what constitutes a context; for example, Schilit et al. [9] for the first time listed three important parts of context (where you are, who you are with, and what resources

are nearby). While Ryan et al. [31] suggested context types of location, environment, identity and time. Dey et al. [8] gave a general definition to context, and found that the presence of other people in the room does not affect the user or the application for the purpose of this task. Therefore, it is not context; whereas user's location, identity and time are context. Context-aware applications look at who's, where's, when's and what's (that is, what the user is doing) of entities and use this information to determine why the situation is occurring. Therefore, location, identity, activity and time are important types of information to characterise context, i.e. they use 'activity' rather than 'environment' as presented by Ryan [31]. An operational definition to context is given by Dey [32] stating that context is all about the whole situation which is relevant to an application and its set of users. He couldn't enumerate which aspects of all situations are important as situation may change. He looked at two pieces of information: weather and the presence of other people, and use the definition to determine whether either one was context. Therefore, the weather does not, but presence of other people is context because it can be used to characterise user's situation, i.e. the environment is important. Abowd et al. [33] gave a basic definition to context by the five W's, these are; who (identity), what (activity), where (location), when (time), and why (understanding of why a person is doing a given activity). Salber [27] identified context attributes as location, identity, activity and state. Gross et al. [28] gave four main dimensions of context (location, identity, time and the environment or activity) that describes the artefacts and the physical location of the current situation; whereas Becker et al. [34] stated that three major criteria, i.e. identity, location and time can be used to access context information. They considered location as an important aspect that includes the position of entities and the spatial relation to other entities. Oh, et al. [35, 36] structured context as user context and system context. According to them, user context represents users' situation in terms of 5W1H. These are who (consists of user's identity, characteristics, and relationship), where (indoor or outdoor location), what (information of ubiSensor or ubiService being used by a user), when (time), how (body conditions or gestures of a user), and why (user's expression, intention, and emotion). Jang et al. [37] also represented user-centric context information in terms of 5W1H. Where, (Who) is a certain user, (Where) in a certain location, (When) at a certain time, (What) paying attention to certain object/services, (How) making a certain expression with physical signs, and

(Why) because of certain intention or emotion. Kunito et al. [38] focused on 3W's these are; when, where and what (environment or surrounding objects) instead of the full 5W1H suite presented by Oh. Bravo et al. [18, 39] use the 5W's (identity, time, location, activity, and why a user is carrying a task) to provide guidelines for context modelling. According to Roussaki et al. [7] context representation scheme was mainly concerned with location, identity and time. Finally, the common and the most widely used contexts are location and proximity [40].

In this respect, some existing works gave different ways to describe and discuss the features of context information and how different pieces of context can be originated and classified. For example, Henricksen et al. [41] described context as a piece of information which has a range of temporal characteristics, imperfect, has many alternative representations and highly interrelated. They categorised context as static or dynamic, sensed information versus information supplied by users. Henricksen et al. [42] also gave the properties of context information and discussed the important design issues when developing applications that rely on imperfect context information. They characterised four types of imperfect context information due to noise or sensor failure. These are unknown, ambiguous, imprecise and erroneous. Gwizdka et al. [43] made a distinction between internal and external context to describe user's state and environmental state. Gu [44] classified a wide range of contexts into two main categories; these are direct context and indirect context based on the means by which context is obtained. Direct context can be further classified into sensed context and defined context. Sensed context is obtained from physical sensors (e.g. curtain's status context sensed by curtain sensors), or from virtual sensors (e.g. a web service). Defined context is typically set by a user. Indirect context is obtained by interpreting direct context through aggregation and reasoning process, while aggregated context can be obtained by aggregating direct context. By using context reasoning engine, deduced context can be obtained and inferred from other types of context.

To conclude, context information has been described and classified with respect to either primitive (raw) contexts, or the abstracted (meaningful) context and its classifications. In Figure 2.1 we prepared a graph about primitive context information used by several existing works. Regarding context primitives, the majority of existing work considered identity, time, and location as the most important context

information. In general there are three schools to distinguish context primitives. The first considered activity as another important piece of context to be added to identity, location and time, the second considered environment as the most important one, whereas, the third school considered environment, activity and 'why' are important in addition to identity, location and time.

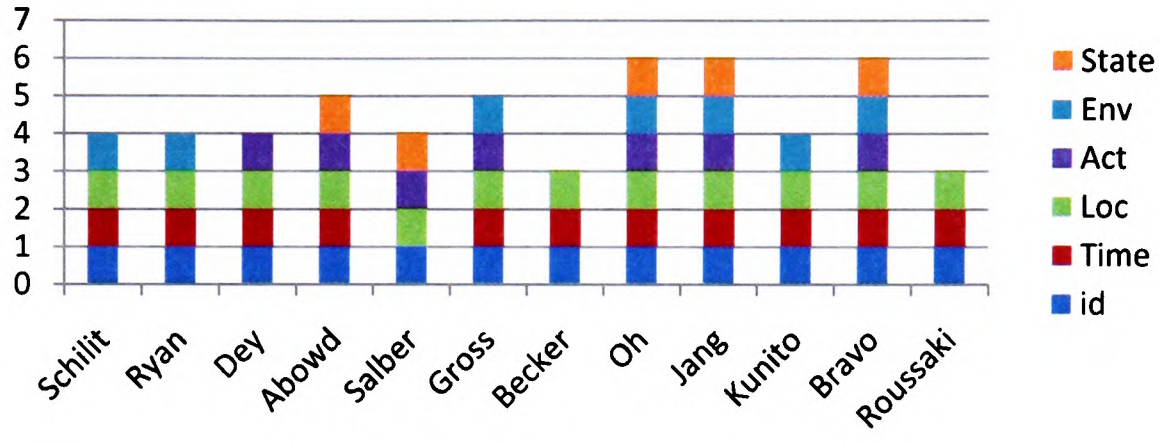


Figure 2.1: Existing work and primitive context

2.2.2 Context History

Context history is considered as an important feature for approximation of a given situation or environment. It evaluates users' behaviour to predict future context values and user actions, where current context depends on previous ones, and will frequently form part of context description. For example, person's disability context might change depending on time and health history and has an impact on current context and derived context. Historical context information represents volatile context, as it depends on time as well as past history of context information; so time stamp is used to create history. This section lists existing work that considers context history, shows its value in their context-aware systems design, and how to maintain and facilitate its query. For instance, Schmidt [24] considered historical data as an important feature for approximation of situation, where historical context can be obtained from changes in the feature space over time. According to Salber [27], context history is an important aspect; therefore, history of context information is maintained in the context Widgets and stored in a database. Korpipää et al. [45] reserved history space of each context type in a context database, where context history assists in producing high-level context. Beigl et al. [46] also considered history as an influence property, and used the Internal Situation Store (ISIS) to hold the context history and all information relevant to context-based enhancement of the communication and application.

Mantoro et al. [47] explained the importance of location history too, and how it is used in the context-aware system. They introduced a history database of events to store user identity and locations, and adopted a technique to control the growth of context history. Gu et al. [44] used a standalone context database and store each domain specific ontologies and context history of such domain in one logic database. Neelima et al. [48] also stored historical context of users and ontology in a database, which can be exploited for reasoning process. Zhang et al. [49] employed historical context stored in a database to support context reasoning as well. Oh et al. [35] used a database to store context history, which can be utilised as a clue to evaluate user's behaviour. They adopted a history based management technique to resolve user conflict and device conflict. Another approaches conducted by Al-Yasiri et al. [50] and Abdulahad et al. [51] intended to add context history to their context model, and insert this information to existing organisational information stored in LDAP server. Fahy et al. [52] gave the requirements that support context history. In their work a sever-based database is used to store context history, applications, and user data, in addition to inference rules, which can be manipulated using SQL. Kunito et al. [38] stored RFID detection history in a database. They adopted a history based recognition to compare between situations at different times to detect situations' change and their duration, and track of transitions to detect sequence, cycle or pattern. Baldauf et al. [53, 54] discussed the need to access context history to predict future context and to deal with sensor conflict, so a centralised high-resource storage is required. According to Chien et al. [55], historical context stored in a database can be used to build the application.

In brief, context history is an important element which is considered in all existing context-aware systems. It takes part during the implementation of context reasoning, predicting future context, and solving inconsistency in context information.

2.2.3 Context Modelling

In order to describe, exploit and store context information, a flexible vocabulary for context modelling is necessary. It simplifies the description of different context atoms and context instances in order to standardise context representation. Context atom generally refers to a specific piece of context-related information, where a single atom can be described as an entity with a couple of attributes; the two most obvious are

context type and context value. Each context type can have one or more context values. Context instance is a context value (characteristics) of certain type at certain circumstances. For example context instance is: Context (Environment: Temperature, 50 °C) [53]. Table 2.1 shows instances of context atoms.

Context type	Context Value	Context
Environment: Temperature	50 °C	Hot
Device : Placement	-	at hand

Table 2.1: Examples of context atoms [53]

Context modelling is important to provide a formal basis for representing and reasoning about context information, therefore it can be easily used and processed. Context model must support multiple representations of context and their relationships and at different level of abstraction [41]. Different efforts have been put in this area either by conducting a survey to show a classification of context modelling, or through studying the modelled aspects and presenting their general characteristics, and also by conducting analysis framework to context modelling. Other approaches discussed the advantages of modelling context information probabilistically [56]. A review of different aspects of context modelling is outlined in this section.

Schmohl et al. [57] carried out a survey discussing context abstraction and its modelling aspects, taxonomies of approaches to context modelling (theoretic and conceptual), and argued about different context models. Gu et al. [44], on the other hand, classified existing context models into three categories; these are application oriented, model oriented and ontology oriented. A more comprehensive research is conducted by Strang et al. [3] who presented a comparative study and distinguished the following context modelling techniques: -

- The Key-value model is easy to manage but it is suitable for simple structure of context modelling.
- The Mark-up scheme is specific and/or limited to small set of context aspects; it uses hierarchical data structures of mark-up tags with attributes and content.
- The Graphical model is simple but less formal compared to other methods, for example, the Unified Modelling Language (UML) diagram, or context extension to Object Role Modelling (ORM).

- The Object oriented model uses the full power of object orientation, by using objects to represent context information to encapsulate context processing and representation, so a well defined interface is required to access context.
- Logic based is a formal model and based on logic; it uses facts, expressions and rules to define context.
- Ontology based model represents concepts and relations between them. It enables context knowledge sharing and reuse to facilitate context reasoning and interoperability between applications.

Bettini et al. [58] discussed the requirements and characteristics of context modelling. These are heterogeneity and mobility, relationship and dependencies, imperfection, reasoning, usability of modelling formalism, and efficient context provisioning. They also conducted a survey investigating approaches to modelling and reasoning techniques for context information which are used by context-aware applications. Then, existing modelling techniques are evaluated; these are the early approaches like Key-value and Markup models, domain-focused models, a more expressive model like Object-Role based, Spatial model, and ontological approaches to modelling like OWL. Afterwards, they introduced a hierarchical hybrid approaches to integrate different formalisms and techniques to context modelling. Bolchini et al. [59] presented the general characteristics of the context model; these are type of formalism, level of formality, flexibility, variable context granularity, and valid context constraints.

On the other hand, many researchers adopted the expressiveness and modelling of context information. Schmidt et al. [24] proposed a structured working model for context. They used three general dimensions related to human factors, physical environment, and time, besides using sub-categories to hierarchically representing context feature space, see Figure 2.2.

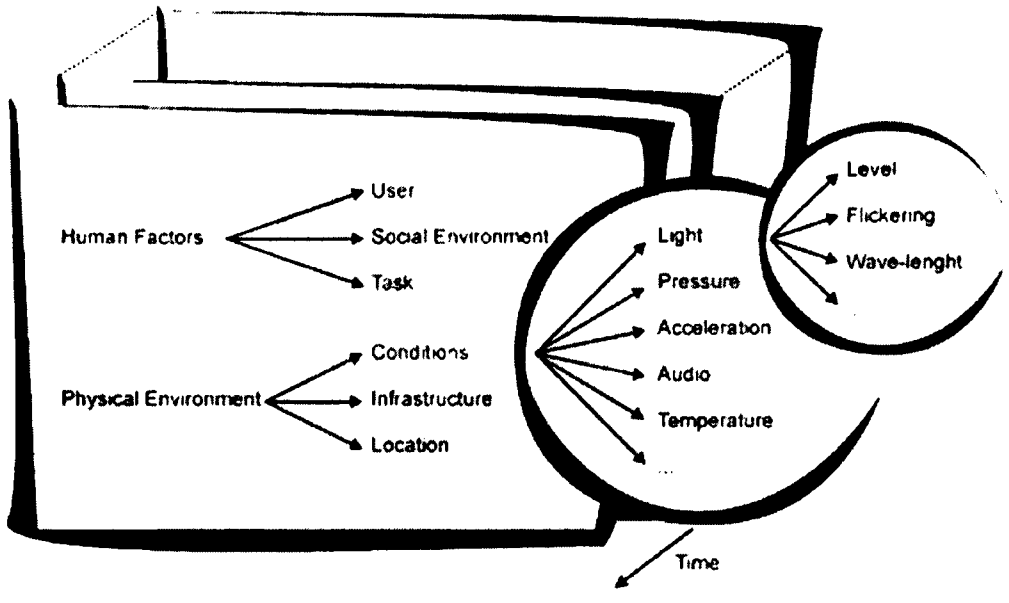


Figure 2.2: Context feature space [24]

Schmidt et al. [26] used the object oriented model to describe context through a three-dimensional space (environment, self, activity), see Figure 2.3. Sensors are represented as time dependent function that returns a scalar, vector, or symbolic value. Afterwards, the concept of cues is used to provide abstraction of physical and logical sensors. It is regarded as a function taking the values of a single sensor up to certain time and provides a symbolic or sub-symbolic output; the context is then derived from the available cues. They also described context by a set of two dimensional vectors, each consists of a symbolic value that describes the situation and a number indicating the certainty that a user or device is currently in this situation.

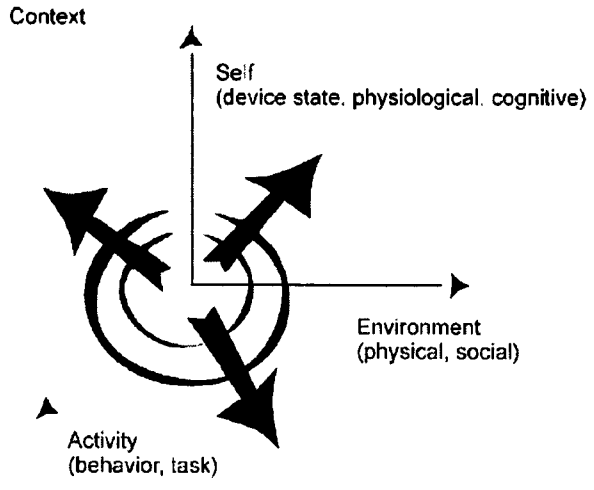


Figure 2.3: 3-D Context Model [26]

Henricksen et al. [41] gave a number of modelling concepts to represent, reason about, and deliver context. In their approach, context is structured around a set of entities their properties, relations and associations. They introduced a graphical notation to specify the context model, as shown in Figure 2.4. The graph has a number of nodes (representing entities such as person, device, and channel) which are

associated with their attributes; for example the person node is associated with name, activity and location.

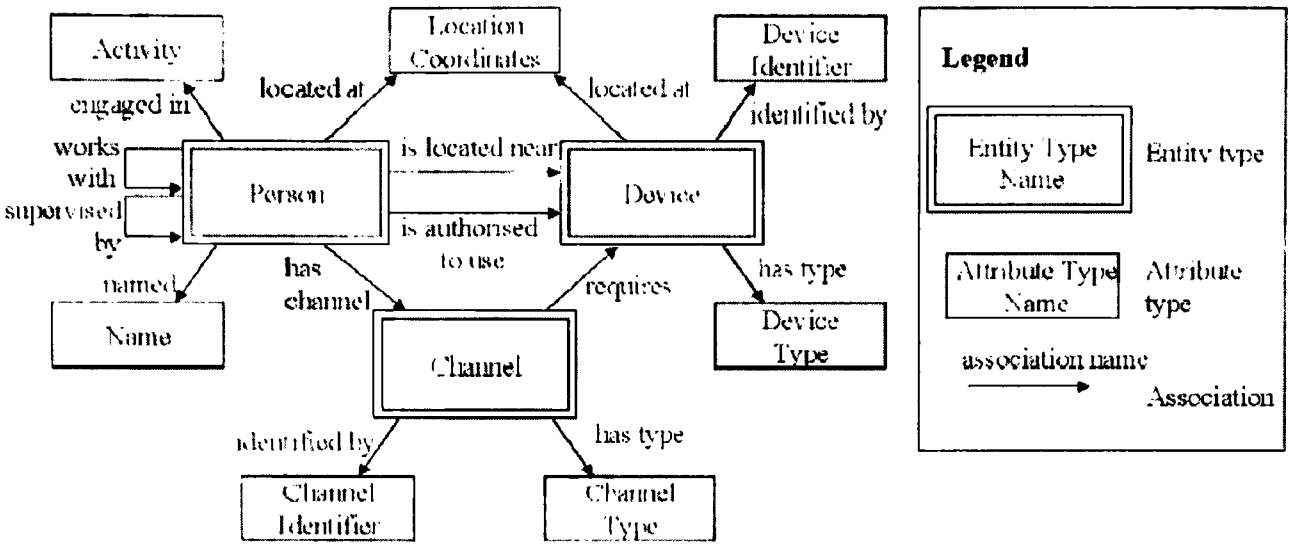


Figure 2.4: Example to modelling a scenario [41]

Logic base approach is another type of modelling which is adopted by Wang et al. [60]. They utilised expressions and conditions such as "and, or, between, >, <" to define complex conditions. The relational data model is used by Fahy et al. [52], where a database has been employed to store context and domain knowledge in the forms of rules and behaviours related to specific applications.

Ontology is adopted by many researchers to represent context information and their relations. Korpipää et al. [45] converted the unstructured raw context information into ontology representation by using RDF (Resource Description Framework) which facilitates context information reuse. Their ontology describes each context using six properties; these are type, value, confidence, source, timestamp and attributes; where context expression must contain at least type and value. Chen et al. [61] used OWL (Web Ontology Language) i.e. an ontology mark-up language that enables context sharing and context reasoning. They created a set of ontologies called COBRA-ONT for modelling context. The ontology defines concepts and relations for describing physical locations, time, people, software agents, mobile devices, and meeting event. Gu et al. [62] also proposed an ontology-based context model that leverages Semantic Web technology and OWL. Context is represented in first-order predicate calculus and the basic model has the form: "Predicate (subject, value)". In this representation, subject is a set of subject names (e.g. a person, location, or object), Predicate is a set of predicate names (e.g. is located in or has status), and value is a set of all values of subjects (e.g. living room, open, close, or empty). They extended the basic model to form a complex context or a set of contexts by combining the predicate and Boolean

algebra (union, intersection, and complement). They adopted a hierarchical approach by dividing the context ontology into a common high-level ontology and domain-specific ontology due to the difficult management and process of large amount of context knowledge in pervasive computing environments. Figure 2.5 shows a class hierarchy diagram of the ontology, in which the common ontology has 14 classes that describes the basic concepts (such as person, location, etc.) and six properties. It also shows the domain specific ontology (such as vehicle-domain, and home domain) which defines the details of the general ontology.

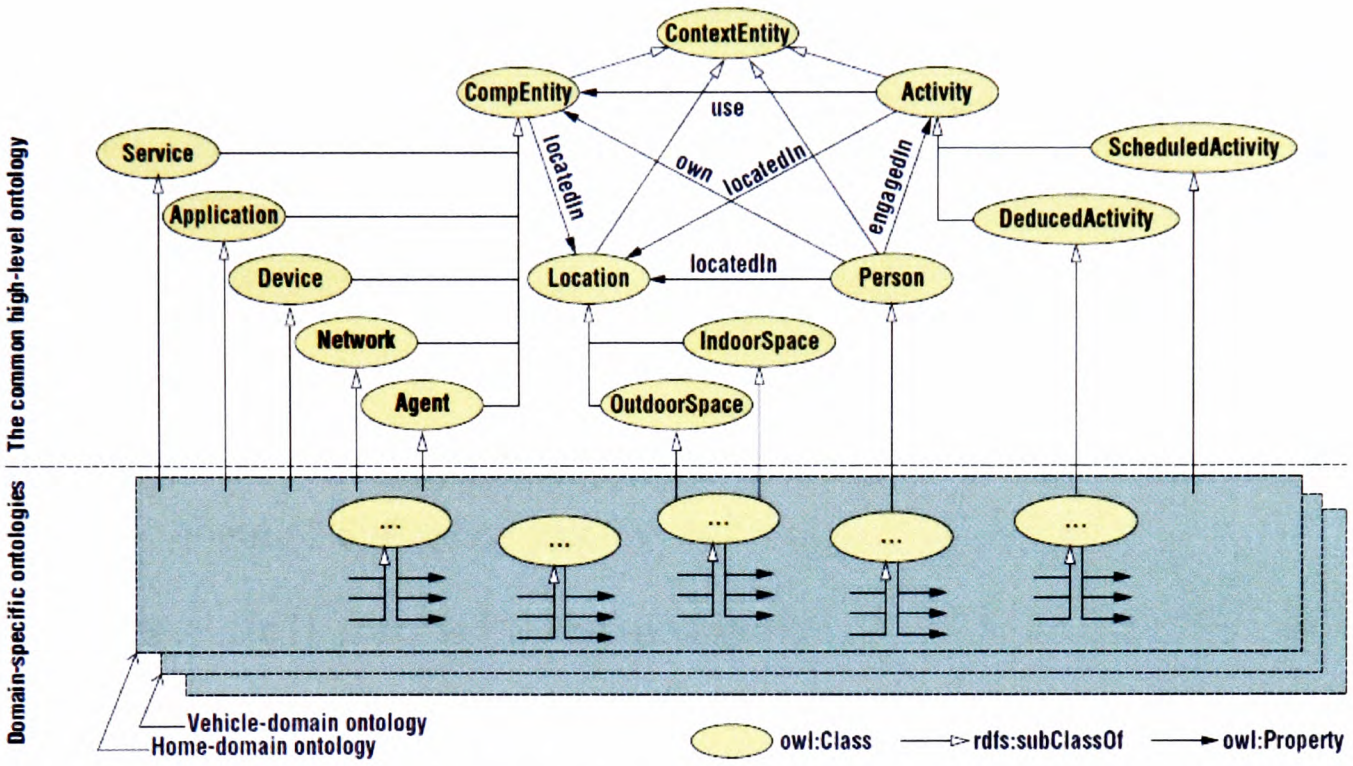
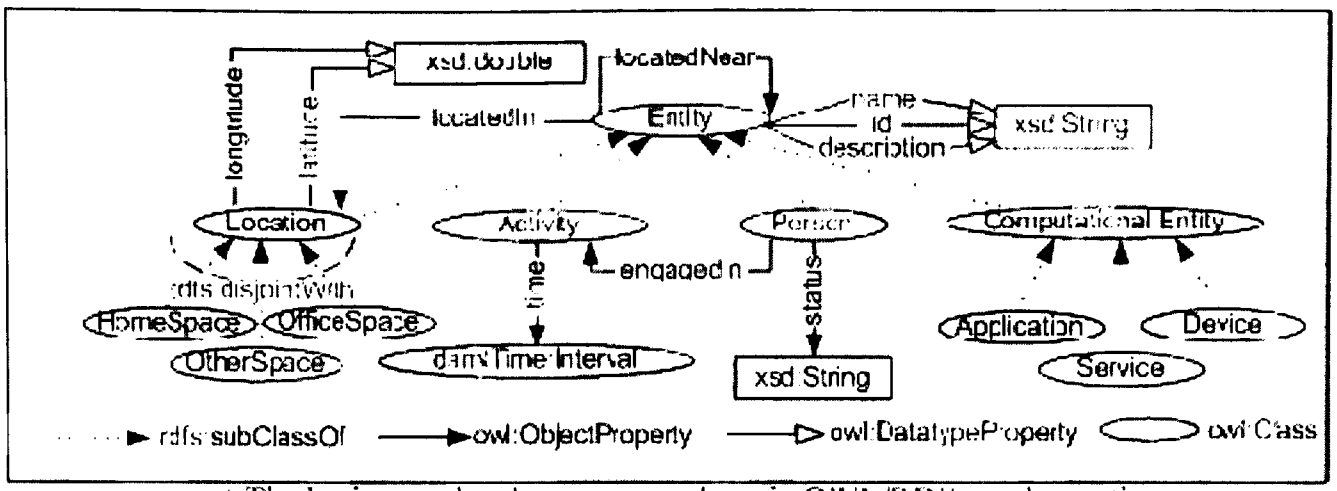


Figure 2.5: Class hierarchy diagram for context ontology [62]

Zhang et al. [49] utilised a Web Ontology based Context model (CONON) for context representation. The ontology for context entities is structured around describing objects (entities such as person, activity, etc.) which are linked with their attributes (e.g. user's status) or with other entity, see Figure 2.6. Roussaki et al. [7] integrate the maintenance, distribution and administrative facilities of a location-based context model with the semantic advantages of context ontology to develop a hybrid context representation scheme.



a) The basice top-level context ontology in OWL/RDF graph notation

```
<owl:Class rdf:ID="Entity">
</owl:Class>
<owl:DatatypeProperty rdf:ID="name">
 <rdfs:domain rdf:resource="#Entity"/>
 <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>

<owl:Class rdf:ID="Location">
 <rdfs:subClassOf rdf:resource="#Entity"/>
 <owl:equivalentClass>
   <owl:Class>
     <owl:unionOf rdf:parseType="Collection">
       <owl:Class rdf:about="#HomeSpace"/>
       <owl:Class rdf:about="#OfficeSpace"/>
       <owl:Class rdf:about="#OtherSpace"/>
     </owl:unionOf>
   </owl:Class>
 </owl:equivalentClass>
</owl:Class>
```

```
<owl:Class rdf:ID="Person">
 <rdfs:subClassOf rdf:resource="#Entity"/>
</owl:Class>
<owl:Class rdf:ID="Activity">
 <rdfs:subClassOf rdf:resource="#Entity"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="locatedIn">
 <rdf:type rdf:resource="TransitiveProperty" />
 < rdfs:domain rdf:resource="#Entity"/>
 < rdfs:range rdf:resource="#Location"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="locatedNear">
 <rdf:type rdf:resource="TransitiveProperty" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="longitude">
 <rdf:type rdf:resource="FunctionalProperty" />
 < rdfs:domain rdf:resource="GeoOnt:#Location"/>
 < rdfs:range rdf:resource=" &xsd:double"/>
</owl:DatatypeProperty>
```

b) A partial OWL/XML serialization

Figure 2.6: Context ontology and partial serialisation [49]

Miraoui et al. [63] proposed a service ontology for context modelling which is based on the concepts of services in pervasive computing systems. Neelima et al. [48] also used ontologies to model context dimensions, which are described through contextual element attributes. Chien et al. [55] proposed another approach which represent context by ontology (implemented using OWL) and used this ontology as a knowledge base. In their study, context information is divided into three classes; these are Sensor, Event, and Scenario contexts. Hwang et al. [64] described an ontology-based context model using F-logic and a rule set which contains reasoning, managing, judging function using contexts, services, environmental and situational information. Hong et al. [65] also designed an ontology-based context model called CALA-ONT, in which context information is expressed in first order logic and used OWL-DL to define Individuals, Properties, and Classes. CALA-ONT defines four top-level classes (person, places, activities, and computational entities), sub-classes, and also contains properties to describe the relations between individuals in top-level and its sub properties. However, the ontology didn't express all the objects and contexts; see Figure 2.7.

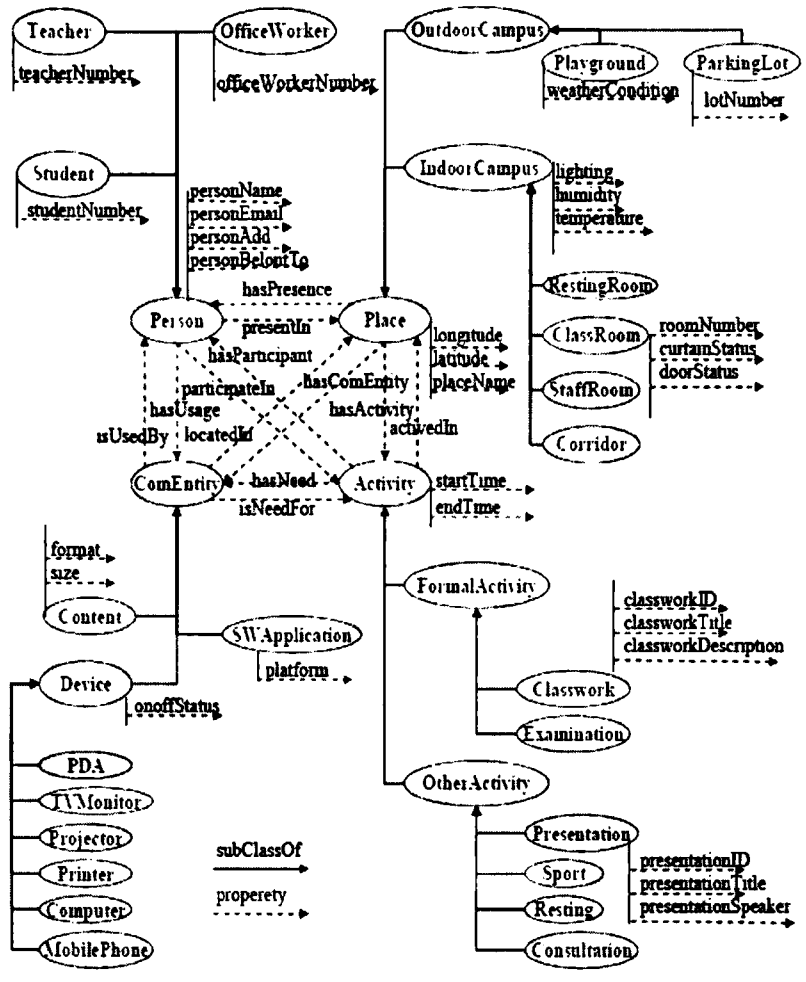
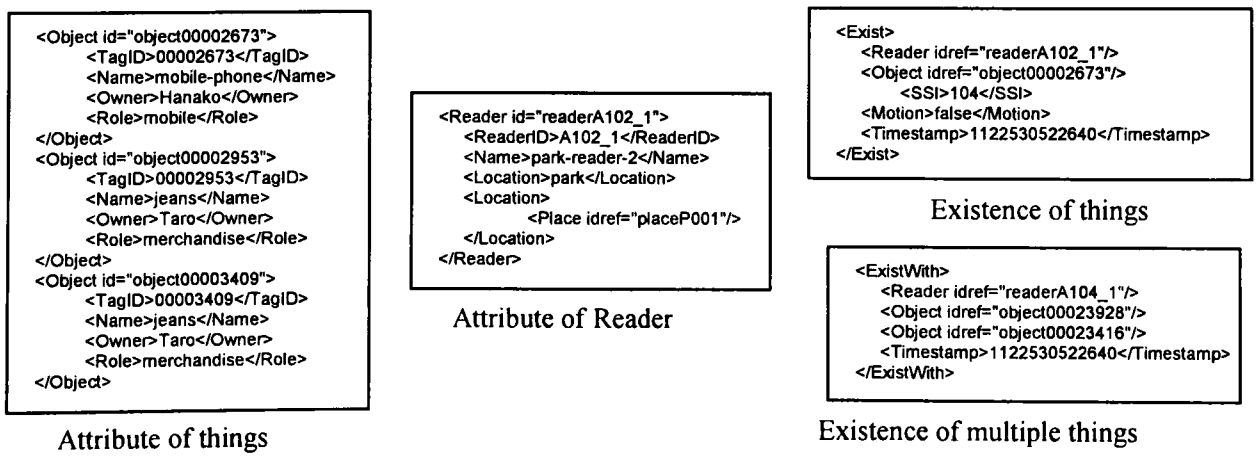


Figure 2.7: CALA-ONT model and its properties [65]

Regarding application oriented context modelling, Oh et al. [35, 36] proposed the unified context model, where the preliminary context generator plays an important role in converting feature extracted from physical sensors into formatted 5W1H. Kunito et al. [38] proposed an application specific model called Real World Model. Their model uses XML code in which the tags describe attributes of things (such as mobile phone, jeans, coats, etc.), attributes of reader (such as name, owner, etc.), and existence of things, see Figure 2.8. Finally, Choi [66] used the context reducer to handle context attributes collected from external and internal sources; they didn't give a specific model as each application has a different model.



```
<Object id="object00002673">
        <TagID>00002673</TagID>
        <Name>mobile-phone</Name>
        <Owner>Hanako</Owner>
        <Role>mobile</Role>
</Object>
<Object id="object00002953">
        <TagID>00002953</TagID>
        <Name>jeans</Name>
        <Owner>Taro</Owner>
        <Role>merchandise</Role>
</Object>
<Object id="object00003409">
        <TagID>00003409</TagID>
        <Name>jeans</Name>
        <Owner>Taro</Owner>
        <Role>merchandise</Role>
</Object>
```

Attribute of things

```
<Reader id="readerA102_1">
    <ReaderID>A102_1</ReaderID>
    <Name>park-reader-2</Name>
    <Location>park</Location>
    <Location>
            <Place idref="placeP001"/>
    </Location>
</Reader>
```

Attribute of Reader

```
<Exist>
    <Reader idref="readerA102_1"/>
    <Object idref="object00002673"/>
        <SSI>104</SSI>
    <Motion>false</Motion>
    <Timestamp>1122530522640</Timestamp>
</Exist>
```

Existence of things

```
<ExistWith>
    <Reader idref="readerA104_1"/>
    <Object idref="object00023928"/>
    <Object idref="object00023416"/>
    <Timestamp>1122530522640</Timestamp>
</ExistWith>
```

Existence of multiple things

Figure 2.8: Real World Model [38]

After reviewing the literature, it was found that different modelling techniques are distinguished, classified and used. The techniques used are: Key-value, Mark-up

scheme, Graphical, Logic based, Object oriented, Ontology based, and application oriented. However, there are other models in the literature that are not included in this thesis such as the Context Cube [67]. Each of these models has its own advantages and disadvantages, where some of these techniques support knowledge sharing and reasoning, yet they are complicated. Others are simple but have limited capabilities and are inflexible. Therefore, some existing systems use hybrid models. Table 2.2 illustrates a comparison of some existing modelling techniques to show their benefits and limitations.

The Modelling Technique	Pros	Cons
Key-value	Simple data structure. Easy to manage.	Inefficient in describing complex context information, and dynamic change of context. Does not support reasoning.
Logic-base	High degree of formality. Flexible.	Applicability to existing ubiquitous computing environment is a major issue.
Mark-up scheme	Can extend, and support dynamic and complex context information. Support dynamic context information.	Most existing models are either proprietary or limited to small set of contexts.
Graphical	High structure level.	Low level of formality.
Object oriented	Have the benefits of abstraction, encapsulation, inheritance, and reusability to support dynamic changes in context information. Scalable and simplify knowledge representation.	Need interface to access context information. Limited formality.
Ontology	High degree of formality. Support reasoning. Knowledge sharing and reuse. Flexible, and expressive.	Complex way of representing knowledge. Do not offer a complete description of context information. Require ontology engine to manage the ontology, which has high requirements on resources.

Table 2.2: A comparison to existing modelling techniques

2.3 Aspects of Context-Aware Applications

Context-aware systems employ computing devices that are aware of their environments to adapt their behaviour accordingly in the final stage. This section presents an overview to how context-aware systems make use of context information to change their state, showing ideas and categories of context-aware applications.

Schilit et al. [9] gave two orthogonal dimensions to context applications, taking into consideration whether the task is to get information or to execute a command and whether the task is executed manually or automatically. They presented four types of applications; these are proximate selection, automatic contextual reconfiguration, contextual command and context-triggered actions. Pascoe [68] developed taxonomy to identify the features of context-aware applications that are similar to schilit's classes of applications. These features are: contextual sensing, contextual adaptation, contextual resource discovery and contextual augmentation; the latter represents the ability to associate digital data with user's context. They stated that context-aware systems contain all four types of applications. Dey et al. [8] merged the ideas from Schilit and Pascoe's taxonomies and gave three application categories of context; these are presentation of information and services to a user, automatic execution of a service and tagging of context information for later retrieval. They made a comparison and applied these categories of context and context-aware features to different application systems. Bisgaard et al. [69] studied the above taxonomies [8, 9, 68] and took several application examples of mobile context-aware systems already developed; then applied Dey's application categories [8] to these systems. In their study, context-aware systems' examples were grouped with the type of context and context awareness.

Another form of classification to application types is given by Berens [70]. He categorised the fields of context-aware applications by location-aware, networked, and mobile devices. These categories are for remembering, helping, automating, and thinking. Remembering applications require sensing the environment via wireless networking and the output is mainly text based. Helping applications use the same technology with more complicated way of acting (for example using written text, light or vibrations); this necessitates hardware and more intelligent software. Automating applications need a more complicated way of acting; yet the output of these applications should have the ability to operate apparatus which need more intelligent software to recognise more signals and even predict user's requests. Thinking applications on the other hand use the same hardware as in automating applications but different software in order to handle wider range of signals. According to Konito et al. [38], three points to achieve services in ubiquitous computing environment is considered. Firstly, services should be provided to the user implicitly. Secondly,

service availability should be very flexible. Last point is to apply things as a way of triggering services. Cheng et al. [71] proposed a transactional model for context-aware applications. In their study context-aware applications are organised as a number of logic units (activities); each one representing an atomic operation such as turn on the air conditioner, make coffee, etc. Each unit may have a compensation module to handle error, exceptions and other abnormal cases. Their model formalises context-aware applications (or a group of closely related applications) in smart environments as a set of activities, where data and control flow between these activities. This method is not helpful for simple applications as it may only have one or two atomic functional units, and the control flows and dependencies between the units are very simple. In conclusion, we found that Dey's application categories cover all application types, realistic and are easy to implement.

2.4 Applications in Smart Environments

Smart environments are the result of pervasive computing and the availability of cheap, high speed and low-power computing technology. According to Mark Weiser [72] a smart environment is "a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network".

Many research works have been conducted in this area, mainly to develop the learning environment. Abowd [73] described classroom 2000 project, a capture system to support teachers and learners in traditional university environments. Jiang et al. [12] gave a generalised four characteristics of futuristic classroom; these are natural user interface, automatic capture of class events and experience, context awareness and proactive service, and collaborative work support. Shi et al. [74] described another approach which turns a physical classroom into a natural user interface for tele-education software. Teachers in this smart classroom can move freely using conventional teaching methods to teach local and remote students. In this respect, instructing local and remote students requires smaller workforce than separate on-campus and tele-education operations. The lecture is recorded as hypermedia courseware that is available for playback after class. Wang et al. [75] discussed different technologies that building smart spaces depend on, and focused on three key issues; these are explicit representation, context querying and context reasoning. They

developed a context infrastructure called Semantic Space to address these issues. Yau et al. [76] also presented a smart classroom environment that uses pervasive computing to enhance collaborative learning among college students. Then, Bravo et al. [77] used RFID technology as an identification process in the ubiquitous classroom to acquire a natural interaction. Another approach conducted by Back et al. [78] discussed the usability of ubiquitous computing in conference rooms.

The development of smart environment devices has been adopted by many researchers. Osbakk et al. [79] developed a BlueReminder device using off-the-shelf mobile information devices to bring ubiquitous computing to a wider audience and made them available at a larger scale. Lakas et al. [80] proposed an Active Classroom Probe (a classroom assessment tool) which can be used in an active and cooperative environment. The probe helps teachers to implement classroom assessments and analyse their results. El-Bishouty et al. [81] introduced a ubiquitous computing environment called PERsonalised Knowledge Awareness Map (PERKAM) that allows learners to share knowledge, interact, collaborate and exchange individual experiences. It provides learners with a knowledge awareness map to visualise the space of environmental objects that surround the learners, the educational materials space, and the peer helper space.

2.5 Summary

This chapter described the context information and how existing works identify context, context-awareness, and how different pieces of context information are categorised. This would help in making a decision on the essential pieces of context including context history. A review of different modelling techniques is then conducted to visualise how these techniques represent and relate context to support context abstraction and reasoning. Finally, context-aware application categories are studied to explain their applicability in smart environments. In conclusion, we found that hybrid modelling is the most appropriate to represent context information to overcome the limitations of each model.

Chapter 3

Context-Aware Architectures - Review and Analysis

The background research explores the past and present context-aware systems to obtain a deep insight into existing architectural design, and what software components constitute these systems. Standard software architecture styles applied to context-aware systems and other systems are discussed in this chapter. A focus on the Pipe-and-Filter architecture style is given, which will be adopted later in this research. Then, analysis and comparison to the existing context-aware systems are presented. Finally, context-aware systems' issues will be discussed.

3.1 Software Architecture Styles

Software architecture is a software engineering discipline that is centered on the idea of reducing complexity through abstraction and separation of concern [82]. The software architecture of a computing system is the structure of the system which comprises software components, their outside visible properties, and the interaction between them. In this respect different architectural styles and patterns are reviewed; the most familiar are Client-Server, Layered, Pipe-and-Filter, Event-based, Repositories, Model-View-Controller, Agent-based and Service-Oriented architectures. An overview of each style will be given in the following sections.

3.1.1 Client - Server Architecture

The Client-Server architecture [83] is a distributed system comprising both "Client" and "Server" software programs. In this architecture the "Client" initiates a service request to the "Server" which provides service to the request. Although the Client-Server can be used by programs on one computer, it could also be on a network interconnecting distributed programs across different locations (client and server are located on separate physical devices). The Client-Server software architecture has

many features for being message based, versatile and modular infrastructure. The architecture also improves usability, flexibility, interoperability and scalability. Examples for this architecture are email exchange, web access and database access.

3.1.2 Layered Architecture

The Layered architecture style [82] is organised hierarchically, each layer providing services to the layer above it and serving as a client to the layer below it. In some systems internal layers are hidden from others except the adjacent outer layer and connectors are protocols defined for communication between layers, see Figure 3.1. Layered fashion has the advantages of high level of abstraction, support enhancement and reuse. But, not all systems are easily structured, not reconfigurable, and it may be difficult to find the right level of abstraction. The most widely known example is the protocol stack in communication systems.

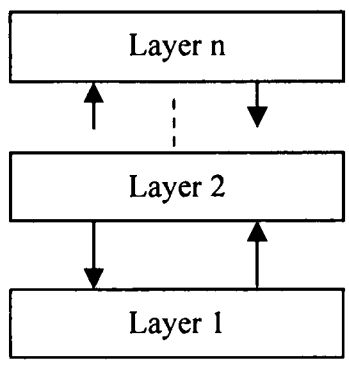


Figure 3.1: Layered architecture style [82]

3.1.3 Pipe-and-Filter Architecture

The Pipe-and-Filter architecture [82] consists of a chain of independent components known as filters. Each filter acts as a single processing component that transforms its input stream to an output stream based on a set of regulations. The connectors (pipes) of this style serve as conduits for the streams, transmitting output of one filter to inputs of another, see Figure 3.2. Filters do not know the identity of their upstream and downstream filters, and they should not share state with other filters. It has a number of advantages in allowing designers to understand the overall input/output behaviour of the system as a simple composition of the behaviours of the individual filters. This architecture style is reconfigurable, support reuse, easy to examine, maintain and extend, and support concurrent execution. Each filter can be employed as a separate task and probably executed in parallel with each other. This style has also disadvantages, where pipe-and-filter systems lead to batch organisation of

processing. The best-known examples of this style are programmes written in UNIX shell, parallel programming, and distributed systems.

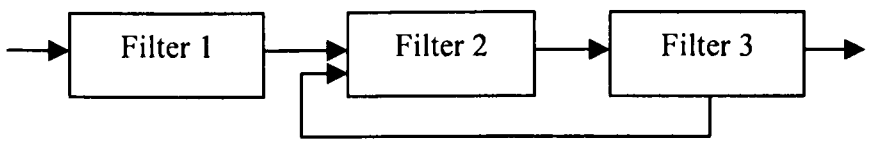


Figure 3.2: Pipe-and-Filter architecture style [82]

3.1.4 Event-Based Architecture

Event-based, implicit invocation architecture style [82] is structured around event handling. Components are modules with collection of procedures and events in their interface. The idea is that instead of invoking a procedure directly, a component can announce one or more events. Other components can register an interest in an event by associating a procedure with the event. When the event is announced, the system itself invokes all of the procedures that have been registered for the event. Thus an event announcement implicitly causes the invocation of procedures in other modules. This architecture has the advantages of strong support of reuse and eases system evolution, but it has drawbacks of pre-emptive control by system manager, exchange of data, and proving correctness is difficult. Examples of this architecture are operating systems, GUI, and tools such as editors and variable monitors register for a debugger's breakpoint events.

3.1.5 Repository Architecture

In Repository architecture [82] there are two major components; a central data structure (blackboard) and a collection of independent components operating on the central data store. The communications between these two components can vary among systems. This architecture has two main subcategories these are traditional database and blackboard which depend on the choice of control discipline. In other words, processes can be triggered by transactions (Database) or by the state of the blackboard. Figure 3.3 shows blackboard architecture which has three major parts; these are knowledge source (KS), blackboard data structure, and control. However there is no explicit representation to the control. It is applicable for applications requiring complex interactions of signal processing, such as speech and pattern recognition systems.

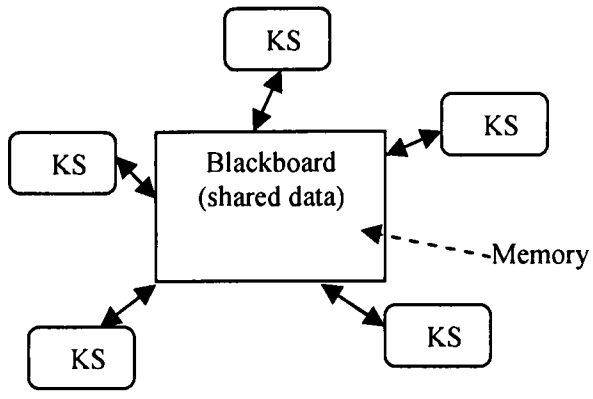


Figure 3.3: Blackboard architecture [82]

3.1.6 Model-View-Controller Architecture

The Model-View-Controller (MVC) architecture [84] style separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes, see Figure 3.4. The model represents the information of the application, the view for displaying all or a portion of the data and controller for handling events that affect the model or view(s), for example the web services. The MVC pattern introduces new levels of indirection and therefore increases the complexity of the solution slightly.

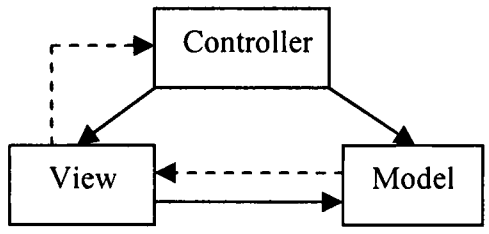


Figure 3.4: MVC architecture [84]

3.1.7 Agent Based Architecture

An agent is a software entity that is situated in some environment and has human's properties such as autonomy, reasoning, learning, knowledge level communication, and mobility, etc. [85] Agents work on behalf of individuals or as part of some wider problem solving initiative. Agent based architecture consists of agents that exchange information and services with other agents [48]. They have many advantages, for being naturally distributed, support modularity and scalability especially with multi-agent architecture, i.e. agents can be added or removed rather than adding new facility to a system. Agents have the capability of reconfiguring with new changes, and have the capability of carrying out tasks in parallel which accelerates the computation process. Finally, they can dynamically collaborate and share resources. Multi Agents architectures use agents that collaborate to carry out a task by sharing the workload

and cooperate to achieve their goals. An example of this architecture is CoBrA [61] that uses an intelligent agent (broker) central to it.

3.1.8 Service-Oriented Architecture

A Service-Oriented architecture [86] is a collection of services that communicate with each other. Its concept is built upon the older concepts of distributed computing and modular programming. The communication process can involve either simple data passing or it could involve two or more services managing some activity. Some means of connecting services to each other is needed. Service-Oriented Architecture aims at a loose coupling of services, so each interaction between services is independent of any other interaction. It is a design framework for realising rapid and low-cost system development and improving total system quality. Table 3.1 shows the benefits and limitations of the above architectures.

Architecture Style	Pros	Cons
Layered	High level of abstraction. Support enhancement and reuse.	Not reconfigurable. Not all systems easy structured. Difficult to find the right level of abstraction.
Pipe-and-Filter	Extensible, scalable and reconfigurable. Easy to examine and maintain. Guide for problem solving. Support decupling, reuse and concurrent execution.	Lead to batch organisation of processing. Interactive applications are difficult. Lowest common denominator on data processing (latency, parse and un-parse).
MVC	Separation between the presentation and application logic. Easy to edit and implement.	Increases the complexity of solution. Change in the model lead to change in view.
Repository	Flexibility of configuration and problem solving. Management of multiple level of abstraction.	No guide for problem solving. Congestion problem and need connection to blackboard at system build time.
Event based	Strong support for reuse.	Pre-emptive control by system manager. Event announcers do not know the effected components.
Centralised	Easy to implement. Simpler design.	Single point of failure and law fault tolerance. Congestion and B.W problem. Dedicated server, increase cost and limit usability.
Agent based	Natural distributed and support concurrent execution. Scalable and reconfigurable. Dynamic collaboration and share resources.	Managing agents is time intensive. High communication cost, and B.W problem. Not suitable for real time applications.
Service oriented	Scalable and loose coupling of services. Low cost system development.	Communication between applications becomes time and code intensive. Not for applications with high data transfer rate.

Table 3.1: A comparison to the software architecture styles

In conclusion, the Pipe-and-Filter architecture style is chosen because the style itself gives guidelines to problem solving, which better fits the design of context-aware systems. It has the benefits of high level abstraction, reconfigurable, scalable, extensible, easy to examine and maintain, support decoupling and reuse, and support concurrent execution.

3.2 Context-Aware Systems

Several research activities have been reported that consider the design of context-aware architectures, their implementation and evaluation [87]. Other research works [53, 40, 54, 88, 59, 89, 90, 91, 92, 93] conducted surveys of existing context-aware systems and presenting different architectures to analyse these systems and simplify their development. There is a variety of software architectural styles which are used in existing context-aware systems, such as layered, agent based, centralised server based, etc. The following sections discuss these systems according to their architectural styles. Some context-aware systems may use more than one style in their design; in this case these systems are grouped and discussed according to their dominant style.

3.2.1 Layered Context-Aware Systems

This section presents context aware systems that use the layered architecture in their design. Besides, many other multi-style systems use this approach in the internal components structure. For example, TEA-system [24] employs a layered architecture for multi-sensory context-awareness, see Figure 3.5. The lower layer is the Sensors layer, which represents a collection of hardware and software sensors. Next, the Cues, which provide an abstraction to software and hardware sensors. The Context layer then derives the abstracted context from the available Cues. Finally, the Scripting layer provides mechanisms to include context information in applications.

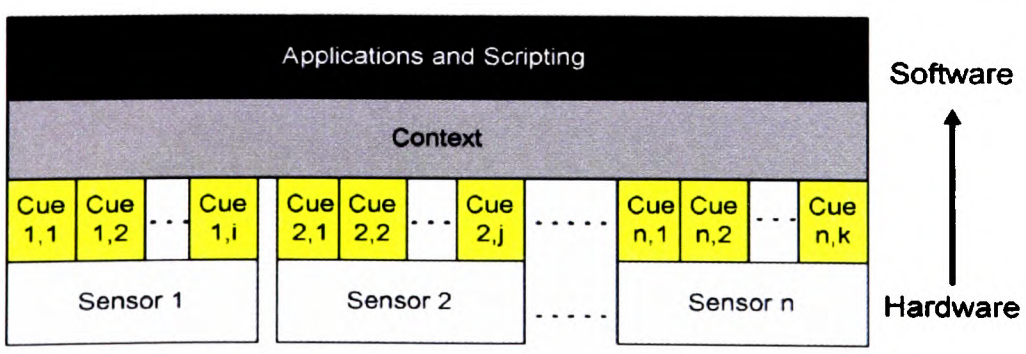


Figure 3.5: Layered architecture of TEA system [24]

Another system that uses the layered style is the Context Stack [49]. As shown in Figure 3.6, the architecture employs five-layers, these are: acquisition, representation, aggregation, interpretation, and utilisation. The architecture uses a database as a context knowledge base and storage of context history.

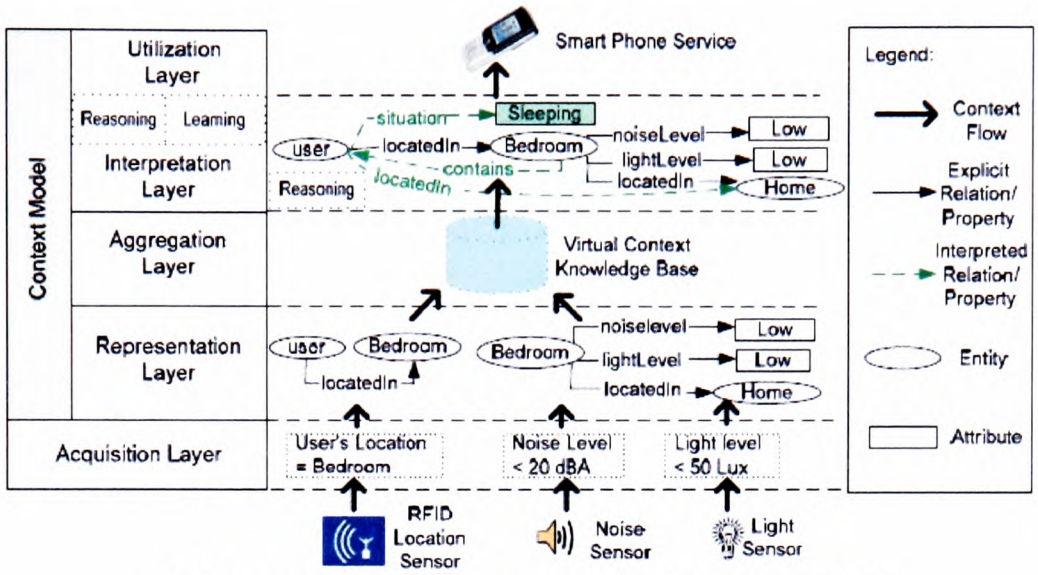


Figure 3.6: Context stack architecture [49]

The Real World Modelling (RWM) [38] is another layered architecture designed to achieve services in the ubiquitous computing environment. As shown in Figure 3.7, the architecture has four phases, phase-P1 for obtaining the real world information, phase-P2 for building the real world model, phase-P3 for service determination, and phase-P4 for service provisioning. Databases are also used as a knowledge base and to keep necessary context information and context history.

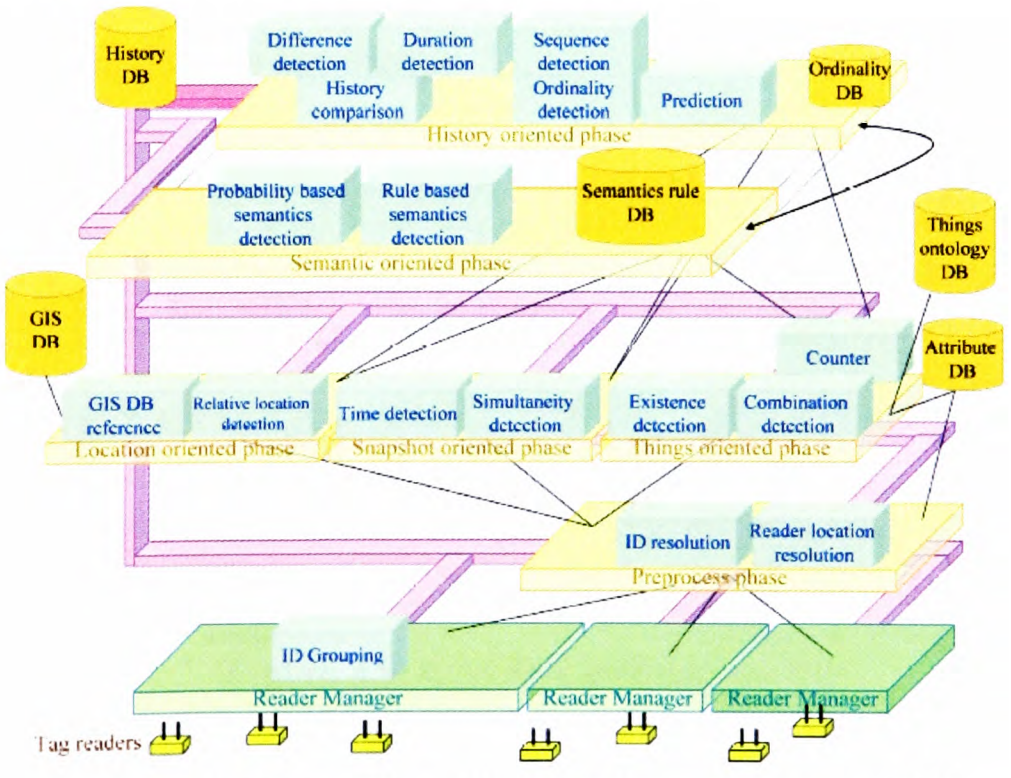


Figure 3.7: Real World Modelling architecture [38]

The General architecture [57] also uses a layered style which consists of five levels, see Figure 3.8. The lower level (lexical) abstracts sensor signals into context events.

Then, the syntactical/representation level translates these context events to atomic context information, which are reasoned about through the reasoning level for more sophisticated processing. The planning level evaluates, detects and schedules changes in context. Finally, the reaction to context changes is executed in the interaction level. Again a database is used as a persistent storage of context information.

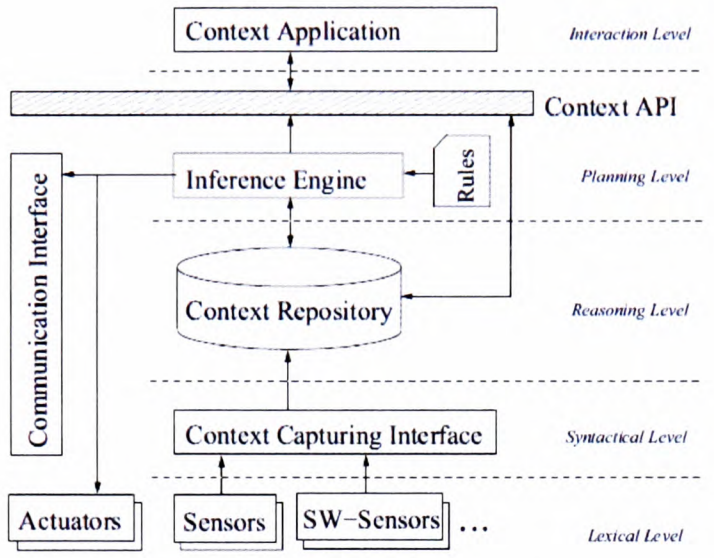


Figure 3.8: General architecture [57]

Another system is the 'Context-aware Architecture based on Database' (CADBA) [55] which uses the layered style, see Figure 3.9. CADBA imported the concept of physical and logical context independence. The architecture components are: devices, device manager, domain knowledge, context interpreter and context aggregator, context model, context management, context database, and service provider. Database is also used to store domain knowledge.

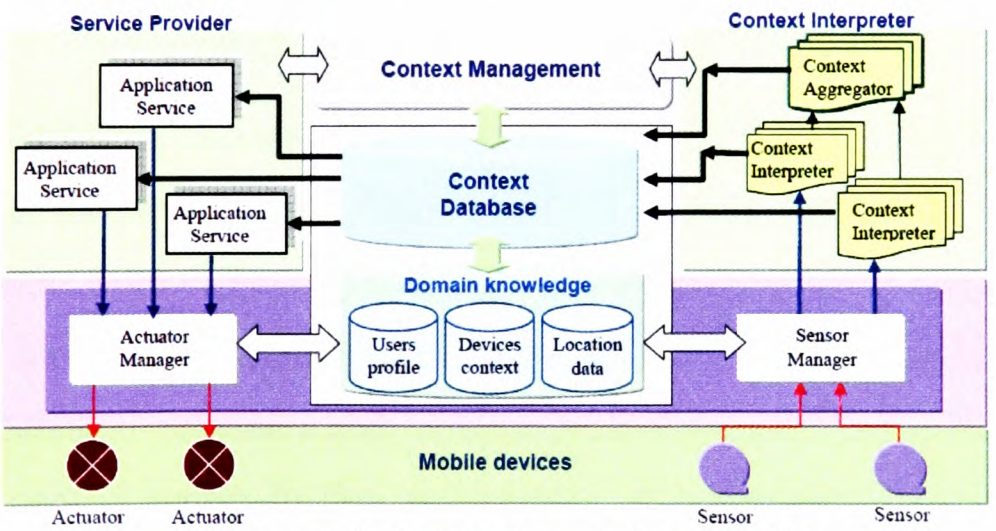


Figure 3.9: CADBA architecture [55]

Many other research works have used the layered style, such as in CAMUS [94] (a framework for network-based service robots). They proposed an architecture which uses five layers: context source and service, device access and control, context processing, context management, and Application. Also, the service architecture [95]

is proposed for integrating body sensor networks and social networks to provide users with the needed services at the right time i.e. improve reliability and real-time response. The architecture has four layers: device, access, control, and service layers.

3.2.2 Centralised, Server Based Context-Aware Systems

A number of systems use a centralised style (server based) as the predominant style, but internally the components may use other styles to communicate with each other. For example, the Context-Aware Sub-Structure (CASS) [52] is a server-based middleware that supports context-aware applications on hand-held and mobile devices. The architecture employs distributed sensor nodes. As shown in Figure 3.10, the middleware contains SensorListener, RuleEngine, ContextRetriever, and Interpreter. These components are in charge of listening to the updates from the sensor nodes, retrieve/store the data from/to a knowledge base to solve problems, and also perform context reasoning. CASS inference engine uses forward chaining technique.

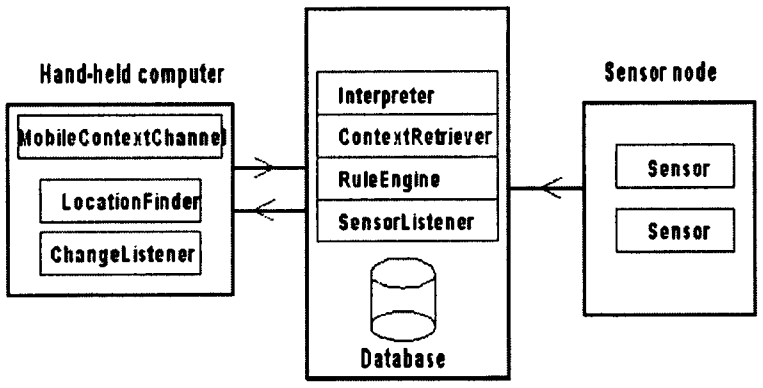


Figure 3.10: CASS architecture [52]

The Service-Oriented Context-Aware Middleware (SOCAM) [44] is also a server based reasoning architecture, see Figure 3.11. It consists of context providers for abstracting context from different resources, context interpreter for context reasoning, context-aware services for adapting the system, and service locating service for advertising the context providers and interpreters to the users. The context interpreter represents the central server to this architecture; it aggregates context information from the distributed context providers and offer it to the clients. Although the predominant SOCAM architecture is server based, its internal components are arranged in three layers. SOCAM supports ontology based reasoning (include RDF and OWL) and user defined reasoning.

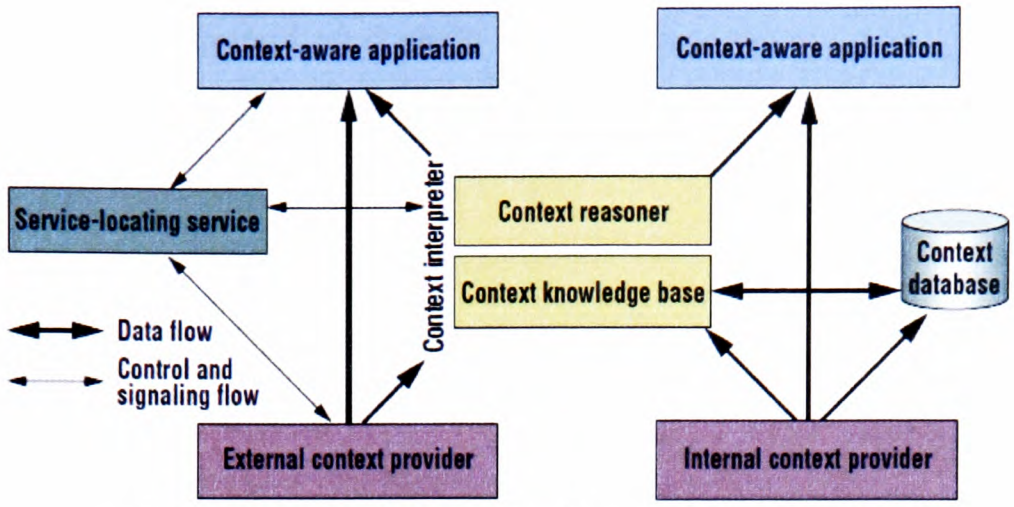


Figure 3.11: SOCAM architecture [62]

3.2.3 Agent Based Context-Aware Systems

Agent based architecture is used by many context-aware systems; however some use multiple independent agents to process the system's information and services. An example of agent based architecture is the Context Broker Architecture (CoBrA) [61, 96]. The architecture supports pervasive context-aware systems by exploiting Semantic Web technology, and uses OWL for modelling ontology of context for reasoning. Figure 3.12 shows the agent based architecture with a specialised server called the context broker (autonomous agent) central to it. The broker provides a context model that can be shared by devices, services and agents in the space. It also, acquires context information from resource-limited devices, support reasoning, resolve consistent knowledge, and solve privacy. The context broker has four functional components (Context Knowledge Base, Context Reasoning Engine, Context Acquisition Model, and the Policy Management Model) which interact in a layered manner.

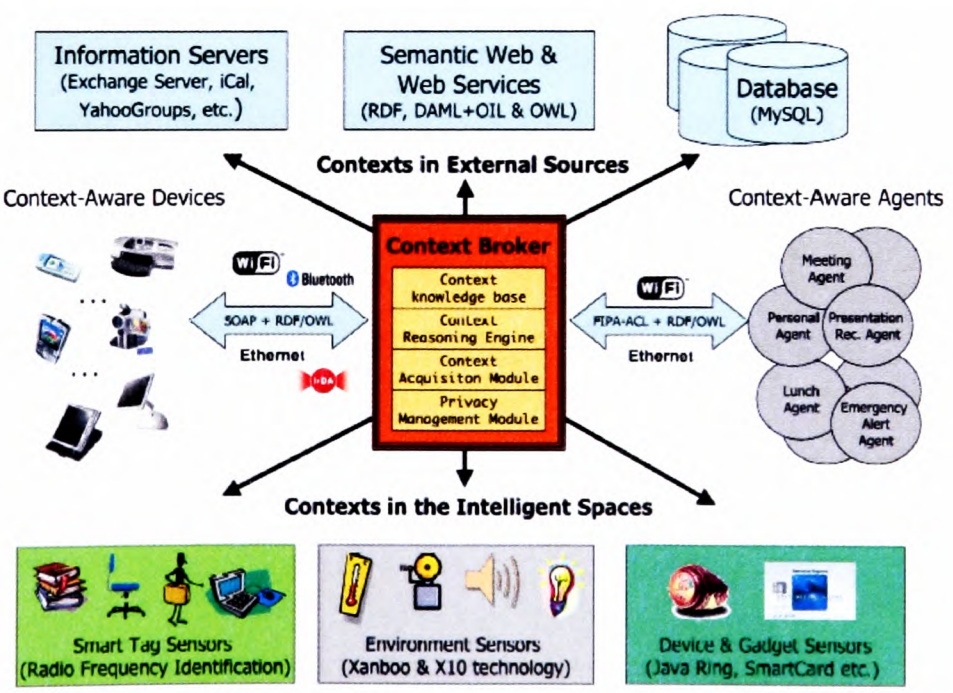


Figure 3.12: CoBrA architecture [61]

Another example of agent based architecture is the multi-agent based architecture [60] which uses multiple independent agents to represent several parts of the context-aware system. The architecture consists of three agents that communicate in a layered style, see Figure 3.13. These are: context collecting agent, context reasoning agent, and an information agent. The first agent collects user's information from devices. The second agent stores user profiles and defines executed commands according to different context. The last agent translates commands into actual actions.

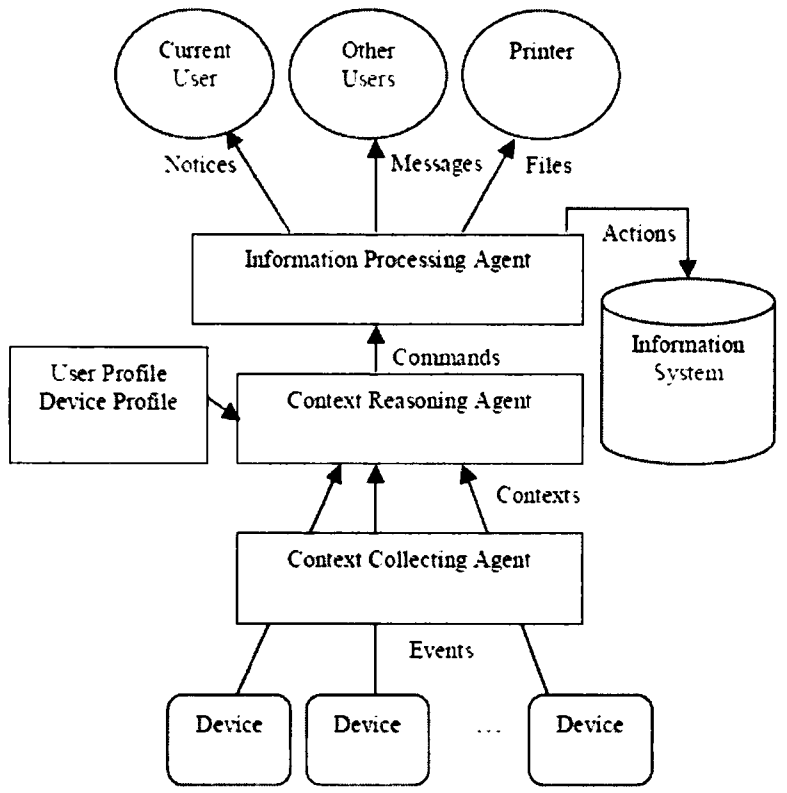


Figure 3.13: Multi-agent based architecture [60]

The Context-Aware Learning Architecture (CALA) [65] is also an agent based which supports user-centric ubiquitous learning services. It consists of personal agents, computing entities, physical sensors, activity agents and context-aware manager centring in a ubiquitous school space, refer to Figure 3.14. The context-aware manager agent acquires context information from other agents and shares a knowledge base. It consists of a context providing module, a context knowledge base, a context reasoning engine and a learning service coordination module that communicate in a layered style. The manager controls a context module using (OWL-DL) ontology for reasoning.

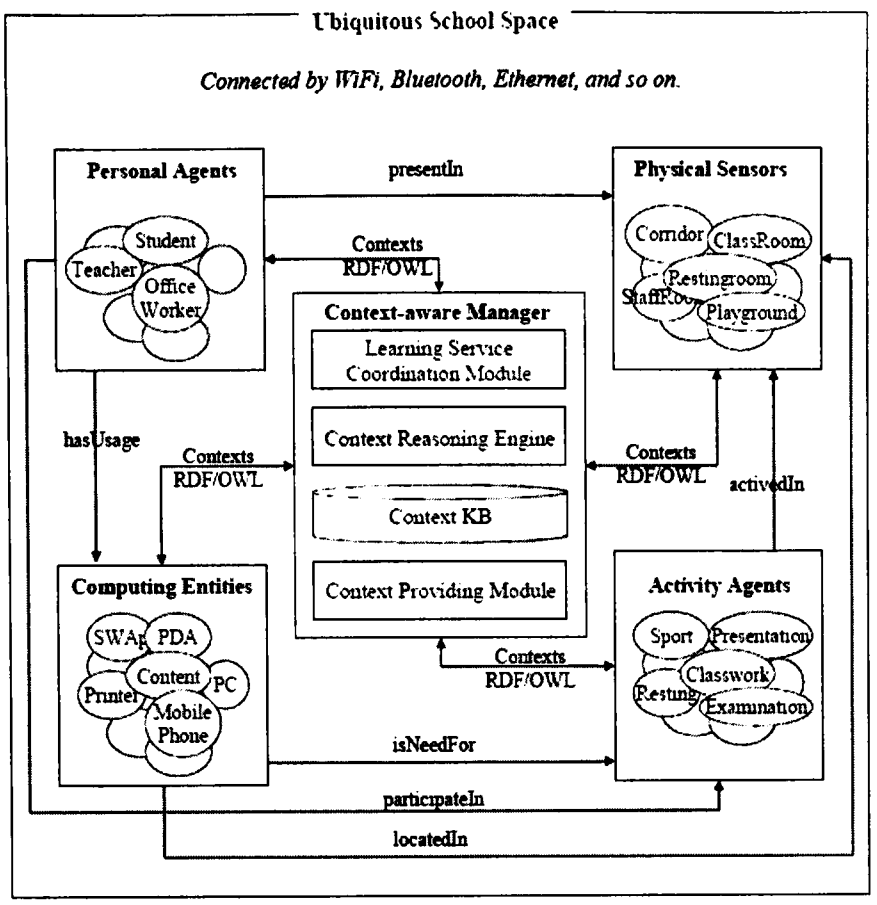


Figure 3.14: CALA architecture [65]

Another example is the Multi Agent, Service Resembling Architecture [48], refer to Figure 3.15. This architecture is used for generic location aware applications. The architecture consists of five components. First, the Context managing component; it is used for acquiring context and performing reasoning using ontology. Second, the Service identifying component; it has the responsibility of identifying application and firing services. Then the Sensor zone which uses sensors to acquire and represent context using ontology. The last two components are Repository and Intelligent artefacts.

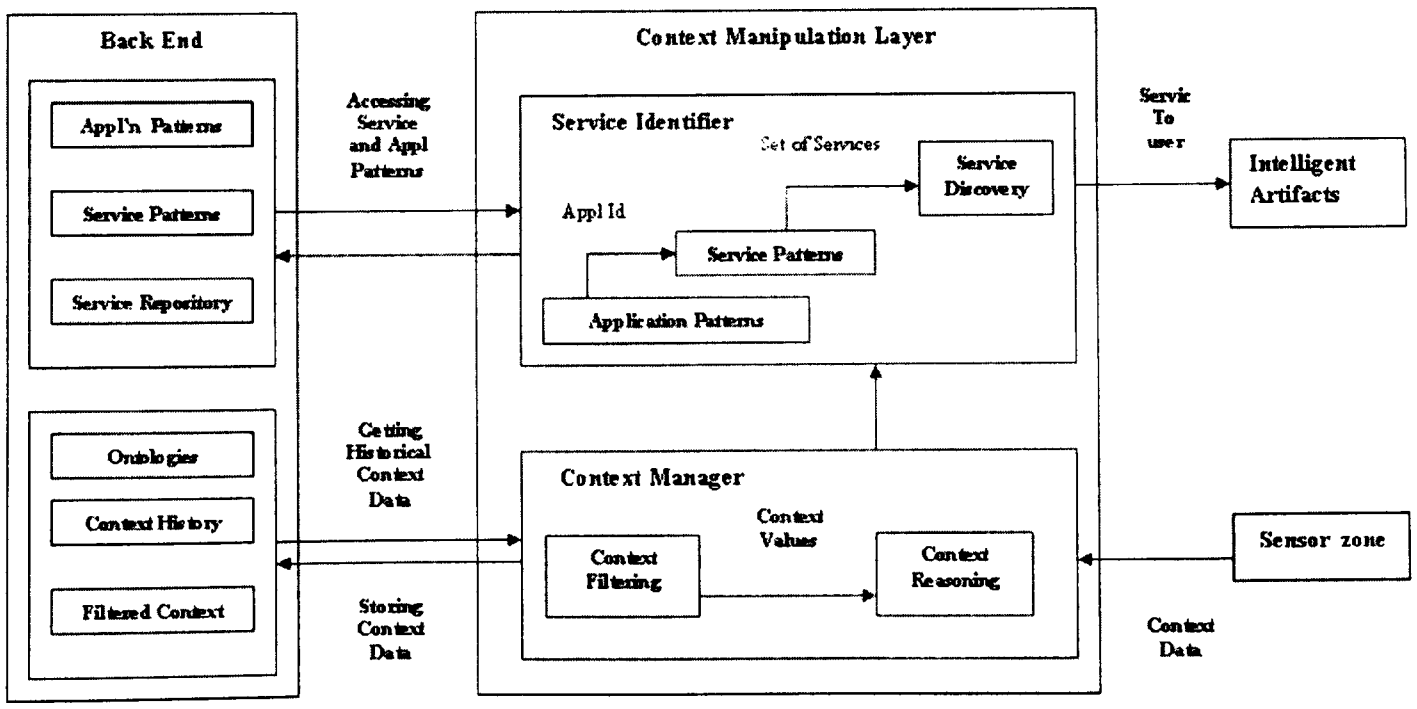


Figure 3.15: Multi agent service resembling architecture [48]

Finally, A-CoBrA [97] merges the proposals of both the General architecture [57] mentioned earlier and CoBrA architecture. The architecture employs number of agents; these are: Sensors and Actuator agents, Activity Context Broker agent, and Activity context-aware agents. All these agents communicate in a layered style. Again, a database is used as a context repository, see Figure 3.16.

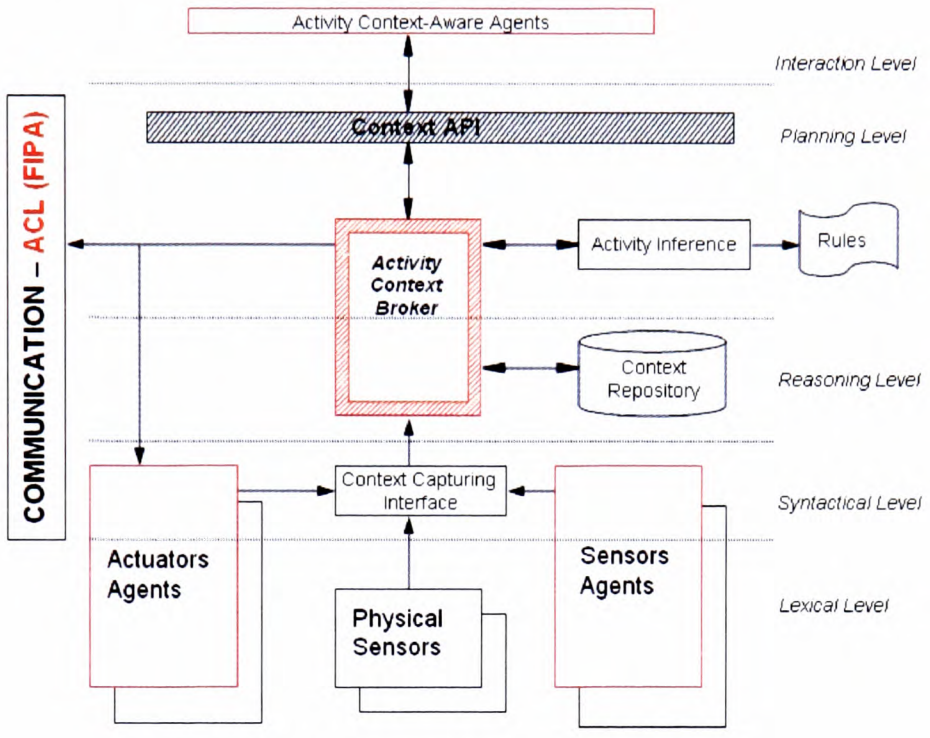


Figure 3.16: A-CoBrA architecture [97]

3.2.4 Other Context-Aware Systems

Many other styles are used by a number of context-aware systems, such as peer-to-peer, MVC, and blackboard architectures, etc. For example, the Context Toolkit [98, 99, 100] consists of three main components: Widgets, Aggregator, and Interpreters, see Figure 3.17. The toolkit relies on the concept of context widgets which can be defined as software components that provide applications with access to context information. They hide the complexity of actual sensors, abstract context information, and provide customisable building blocks of context sensing. The aggregators are meta-widgets which have the ability to aggregate real world context information. The interpreters abstract low-level context into higher level information. Although the toolkit uses peer-to-peer style, Widget's underlying components interact in a layered manner. The toolkit stores context history in a database.

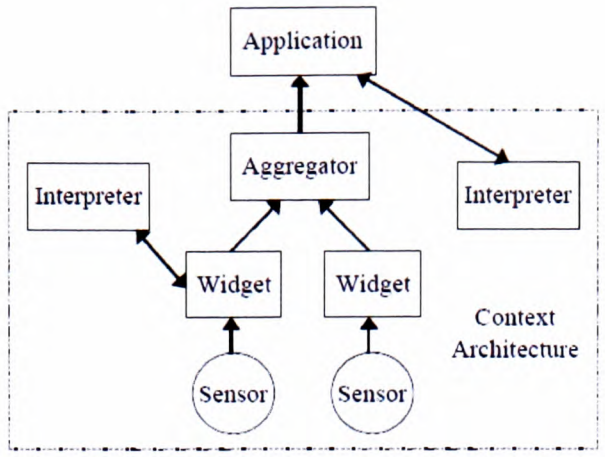


Figure 3.17: Context toolkit architecture [99]

The Context Management Framework (CMF) [45] uses a different style in their design (the blackboard-based approach). It is composed of a blackboard based context manager, resource server, context recognition service, and application (refer to Figure 3.18). During communication, the context manager functions as a central server, whereas other entities (except security) act as clients. The resource servers send collected context information to the blackboard to be processed. The resource server works with the recognition service (which abstracts context) to convert raw data into context ontology. The security module detects the consistency of the received context and check trustworthiness of context incoming from the environment. CMF uses fuzzy logic to reason about high-level concepts.

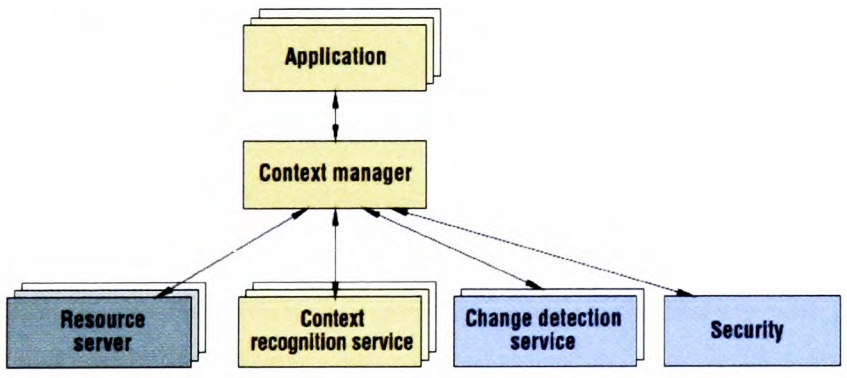


Figure 3.18: CMF architecture [45]

CORTEX [101] is a middleware based on the notion of sentient object model (an entity that consumes and produces events), see Figure 3.19. In this architecture the sensors produce software events in reaction to real world situations; the actuators consume the processed events and accordingly change the state of the real world using hardware devices. CORTEX middleware is based on a component framework to provide services to the sentient objects. These services are for discovery, communication, and changing resource allocation. The inference engine uses rules for reasoning based on C Language Integrated Production System (CLIPS). The architecture maintains context history in a database. The CORTEX architecture has an

event model that allows for dissemination of events in an ad hoc network environment.

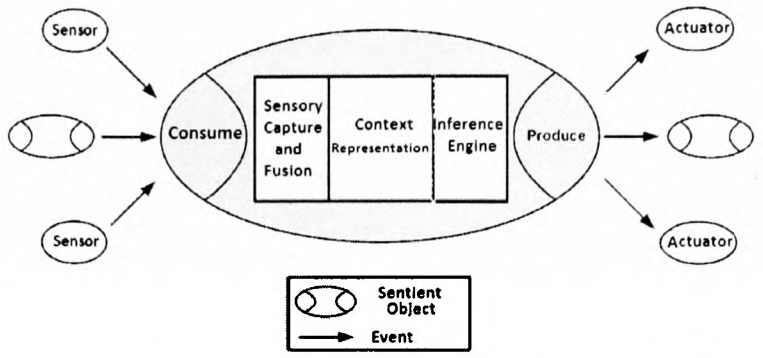


Figure 3.19: Sentient object model [102]

ubi-UCAM [35] is a unified context-aware application model for ubiquitous computing environment, which provides user-centric services for multiple users. It has two independent and reusable modules: ubiSensors and ubiService. The ubiSensor consists of physical sensor, feature extraction module, preliminary context generator, and self configuration manager. The ubiService consists of Self Configuration Manager, Context Integrator, Context Manager, Interpreter, and Service Provider. The internal components for each of these two modules communicate in a layered style, see Figure 3.20.

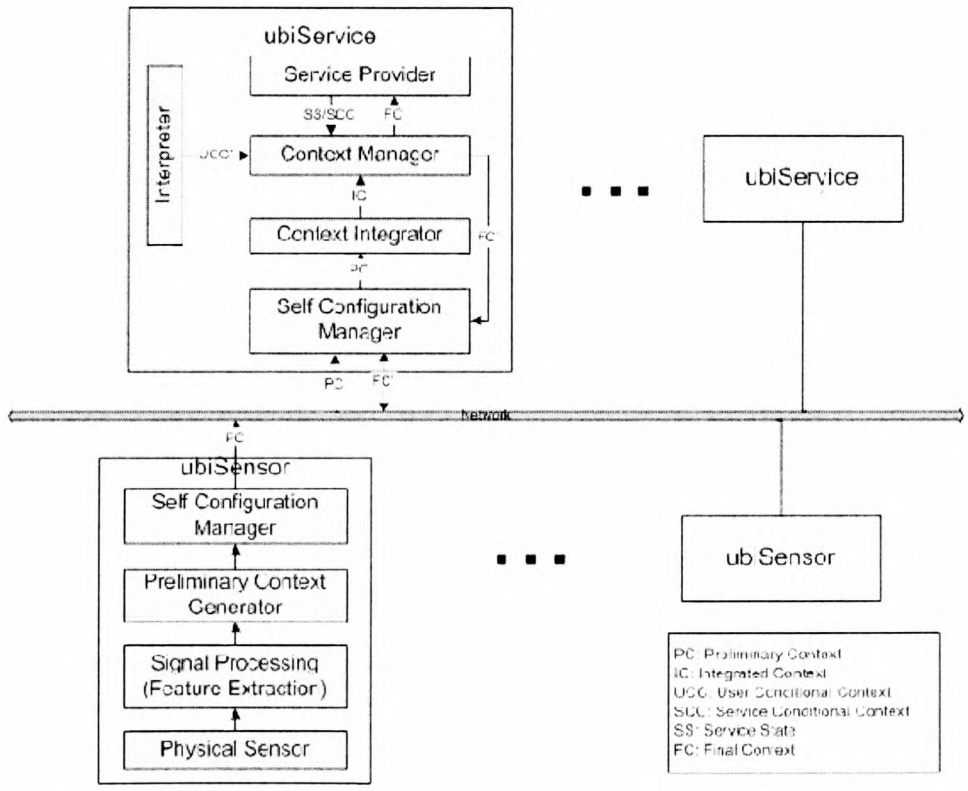


Figure 3.20: ubi-UCAM architecture [35]

WCAM (Watcher, Controller, Action, and Model) [66] architecture uses the MVC style. It is composed of four modules, which reduces complexity by separation of concerns. These components are: (1) the Watcher which uses physical and logical sensors to sense the external environment, (2) the Controller which bridges context and services, (3) the Action to represent a set of services and (4) the Model for

managing the whole system. Each of the four modules is composed of a number of elements that internally communicate in a layered style, see Figure 3.21.

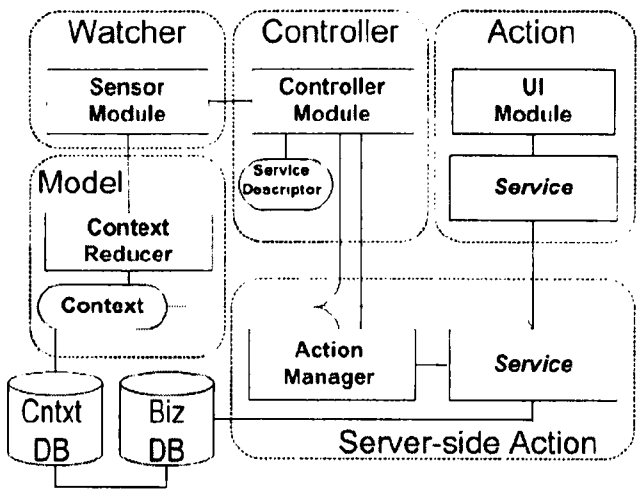


Figure 3.21: WCAM architecture [66]

3.3 Elements of Context-Aware Architecture – Review

The following sections present an in-depth review and description of the main elements (functional components) that are normally found in context-aware systems. The rationale behind this description is to become aware of what elements constitute these systems, how these systems work, and what problems and gaps they might have. This information will be useful in identifying the design requirements of the generic framework which we are proposing in the next stages. The new proposed framework can be used in the development of a variety of context-aware systems with a wide range of applications.

3.3.1 Sensory and Context Abstraction

Context-aware systems gather context information from heterogeneous sources. These systems exploit different types of distributed sensors to acquire raw data from the surrounding environment. The acquired data can then be represented in a low level of abstraction, so that it can be easily processed and used in the application level. This section focuses on sensor types and sensory functional components of various context-aware systems.

Various types are distinguished and different names are given to the sensors of context-aware systems. For example, Schmidt et al. [24, 26, 103] distinguished between two kinds of sensors; physical sensors (electronic hardware components that measure physical parameters in the environment) and logical sensors (all information gathered from the host of awareness component); e.g. GSM cell and current time.

They also gave a brief overview of available sensor technologies and the context information that can be obtained with them. These are; Optical/Vision (photo-diode, color sensor, IR, UV-sensor), Audio (cheap microphone), Motion (mercury switches, angular sensors and accelerometers), Location (GPS, GSM cell, active batch, radio beacons), etc. Henricken et al. [41] used hardware and software sensors to acquire context information. Indulska et al. [6] classified sensors to three types: physical, virtual, and logical; where logical sensor is a combination of physical and virtual sensors. For example, location information can be physical, virtual or logical. Physical information are gathered by physical sensors; e.g. GPS device. Virtual information can be acquired from calendar application which contains person's location, a travel-booking system, or from email which contain location information. Logical information can be provided by a physical or virtual sensor with information from other resource to infer the physical location of an entity. Finally, there are three categories of context acquisition, these are direct access to hardware sensors, through middleware infrastructure, and context server [104, 54].

Context-aware systems have components that collect and abstract context information. Some of these systems have dedicated components to handle different pieces of context information, while others are tightly coupled with other components. For example; the TEA-system [24, 103] uses Sensors layer to measure the physical parameters in the environment. The *Cues* which represent the second layer provide an abstraction of physical and logical sensors. The Cues also make changes of hardware (sensors) transparent to the next layer (context recognition layer) and reduce the amount of data provided by the sensors. The context *Widget* [98] is another context sensing block that abstracts the context information; it hides the complexity of the actual sensors from the application. It is a reusable and customisable software component that mediates between the user and application to access the context information from their operating environments. The *resource servers* in CMF [45] are also sensing components that acquire raw data from sensors (microphone, accelerometers, light channels, temperature, humidity, touch sensors, GPS, internet and other networked resources) and aggregate them to semantic information before sending them to context manager at a determined intervals. The *Context Acquisition Module* in CoBrA [61] is a library of procedures that forms a middle-ware abstraction for context acquisition. It acquires context from sources and its task is similar to the

context Widgets which shields the low-level sensing implementation from high level applications. SOCAM [44] uses independent components (external and internal *context providers*). They abstract low-level sensing implementation (obtained from heterogeneous internal or external sources) to high-level manipulation (useful contexts) and convert them into OWL representations to be shared by other components. Context information can be sensed by physical sensors (e.g. curtain sensors used to sense curtain's status) or virtual sensors (e.g. web service). On the other hand, CASS [52] uses distributed *sensor nodes* (could be mobile or static) to listen to context updates. In CORTEX [101] the *sentient object model* interacts with their environment using sensors and actuators. It can abstract context from the environment using incoming events from sensors and other sentient objects, where sensors produce software events in reaction to a real word stimulus. ubi-UCAM [35] uses various kinds of *ubiSensors* to represent user's situation. *ubiSensor* consists of physical sensors, feature extraction model, preliminary context generator, and self configuration manager. The preliminary context generator converts features extracted from physical sensors into formatted 5W1H context, and the self configuration manager multicasts preliminary context to the ubiService. The *Context Acquisition Layer* in Context Stack [49] also acquires raw context from various ubiquitous context sensors to be passed to the upper layers. In the RWM architecture [38], the real world information is detected in *phase-P1* using RFID tags, cameras, GPS, and other sensors. Another example, is the Multi-agent based architecture [60] in which the *Context Collecting Agent* is responsible for collecting users event information from various mobile or stationary devices used in daily life. Context information is translated into some entries (such as device id, data type, value, and date time) to be presented to the context reasoning agent. The *Watcher* in WCAM [66] consists of sensors (physical and logical) that sense values from internal or external environment to provide logical views of sensors. The *Context Providing Module* in CALA [65] obtain different pieces of context information from various sensors through the personal agent, activity agent, and computing entity. The *lexical level* of the general architecture [57] also uses sensors (including software components) to acquire raw environmental data, which is refined to context information through the context capturing interface. The *Context Managing Agent* in Multi Agent Service Reassembling Architecture [48] acquires data from the Sensor Zone, where sensors

are polled by an intelligent agent. Then the context information is distinguished and modelled. The *Device layer* in CADBA [55] framework represents the physical equipment, devices, and drivers used by the context-aware system to acquire context information. Finally, the *Sensors agent* in A-CoBrA [97] employs interfaces for capturing context information.

In conclusion existing context-aware systems either use fixed or reusable software/hardware components to abstract context information. Examples of reusable sensing components that abstract context information are: Cues, Widgets, Context Providers, Sensor nodes, Watcher, and Sensors agents. The devices that communicate with the Context Collecting Agent (CCA) are also reusable components that collect context information which could be mobile or stationary.

3.3.2 Context Modelling

Context modelling is the process of representing context information into machine readable format using relationships between the sensed information to be easily used, processed and interpreted. This section presents an overview of the modelling components and/or associated modelling techniques for various context-aware systems. For example the *preliminary context integrator* in ubi-UCAM [35] converts features extracted from physical sensors into a formatted 5W1H context. *Phase-P2* of RWM architecture [38] builds the real world model, which describes various situations, events, and states. The *syntactical/representation level* of the General architecture [57] translates context events to atomic context information, where the raw data is abstracted into discrete data structures so it can be processed by software. On the other hand, many systems use ontology for modelling context information. For instance, the *resource server and context recognition services* in CMF [45] use ontology and fuzzy logic to represent and transform raw data to high level context, which is stored in the context manager as a knowledge base. CoBrA [61] uses COBRA-ONT (expressed in OWL) for modelling context, which is stored as a knowledge base to support reasoning. SOCAM [44] also uses ontology for modelling context; it converts the abstracted contexts (formed by the context providers) to OWL representation to be shared by other components. The Context Stack [49] uses web ontology based context model (CONON). In this system, the *Representation Layer* is an abstraction layer that converts sensor data into semantic information. CALA [65] is

another example who uses ontology (CALA-ONT) which is formed by the *context providing module*. Here the context information is expressed in first order predicate logic and the context model is written using OWL-DL. In Multi Agent Service Reassembling architecture [48] the '*context management agent*' filters context dimensions (time, location, identity, and activity) into a suitable profile to distinguish context and represent it by ontology. CADBA [55] uses the *resource layer* to represent the domain knowledge which can be described by ontology semantic network.

To conclude, context representation is an important component of the context-aware system. It provides a vocabulary to maintain, express and relates context to support context reasoning.

3.3.3 Aggregation

As raw pieces of context information are sensed and modelled, they have to be enriched to create a reliable context. User profile, preferences and context history, etc. are significant information that enriches the context-aware system. These pieces of information can be aggregated and fused at some stage during information processing to obtain higher level of abstraction. Context aggregators are responsible for augmenting existing information about a specific entity to be utilised in the reasoning process. This section presents approaches that have the concept or the functionality of context aggregation. For instance TEA-system [24] uses the sensor fusion to obtain contexts that cannot be derived from single sensor. The principle of Cues is based on the fusion of multiple simple sensors in the awareness component and the association of patterns in the sensory data with specific context. The context Toolkit [99] also uses aggregator to collect context information of real world entities, where the servers are used to aggregate context information. In CORTEX [101] the *Sensory Capture* component performs sensor fusion to solve uncertainty of sensor data and make higher-level context from multi-modal data sources. The *Aggregation Layer* in the Context stack [49] is another example that aggregates and relates context from distributed context sensors to form a centralised knowledge base by using relational database, thus simplifying context query and interpretation. On the other hand, the *Context Integrator* in ubi-UCAM [35] periodically collects preliminary context from various ubiSensors, classify context (to each element of 5W1H) and uses the fusion

method to create integrated context. The *Context Reducer* of WCAM [66] gathers context attributes from external or internal resources such as database, and variables. Then the *context attribute collector* gathers context attribute values from the system and manages the storage of context attributes periodically. Another example is the *Context aggregator* in CADBA [55], which gathers low-level context information acquired from sensors to form a higher level context. In this system context is divided into classes: Sensor context, Event context, and Scenario context.

In brief, aggregators are used to collect contexts in order to enrich and solve consistency of context information. These components could be either explicitly or implicitly shown in the context-aware architectures.

3.3.4 Context Reasoning

A vital functional component of the context-aware system is the reasoning component. It is used to derive high-level context information from low-level context (from the surrounding environment) and aggregated context using appropriate reasoning rules, ontologies, etc. This section presents an overview to the reasoning components of various context-ware systems and/or the reasoning mechanisms. For example, the *Context Layer* of TEA-system [24] is used to compute current context from available cues, by using simple if-then rules or other AI methods. In the context toolkit [98], the *interpreter* component is responsible for abstracting raw data to higher level context information either by using a simple lookup table or a complex AI- based inferences. Another example is the *Context Reasoning Engine* in CoBrA [61]. It is a reactive inference engine to reason about stored knowledge using a set of ontologies and heuristic knowledge; it also detects and solves inconsistency of information. SOCAM [44] architecture also uses *Context Interpreter* that provides logic reasoning services to process context information. It consists of Reasoning Engines and Context Knowledge Base. The Context Reasoning Engines uses ontology reasoning (RDF schema and OWL-Lite) and (user defined) inference rules to infer deduced context, resolve conflicts and maintain the consistency of the context knowledge base. The *RuleEngine* in CASS [52] employs a stored knowledge base that contains rules used by the Inference engine. The *RuleEngine* uses the forward chain method to infer over context in the database which is based on the context conditions and their goals. In CORTEX [101] the *Inference Engine* is the brain of the sentient

objects; it includes intelligence and control logic which realises the mechanism of decision-making. On the other hand, the *Context Inference Engine* within ubi-UCAM [35] infers the "Why" context among 5W1H. The inference is based on context transition (infers context by observing change of other context) and complex fusion. Again the *Interpretation Layer* in Context Stack [49] controls the reasoning rules and/or the learning mechanisms to deduce high-level implicit knowledge from low level explicit context information. The reasoning mechanisms use first order logic and ontology. In the RWM architecture [38], Layer 3 (Phase-P2) interprets the real world situation from data about time, location, and objects. Layer 4 performs a more abstract interpretation including history and semantic. The *Context Reasoning Agent* in the Multi-agent based architecture [60] receives real time events from the collecting agent. Then it uses SQL grammar with logic base and device profile to generate corresponding commands according to different contexts. The Context Reasoning Agent could be mobile or stationary. In WCAM [66], the *Model* includes the Inference engine which uses rules to infer high level context from the gathered context attributes; whereas the *Context reasoning engine* in CALA [65] interprets the context information using ontology and rule based reasoning. In contrast, The *Reasoning module* of the Multi Agent Service Reassembling Architecture [48] performs the reasoning of context information using ontologies. Rule based reasoning is also used to identify the right application using location context. The *planning level* of the General architecture [57] uses rules to derive context independently by the inference engine. The inference engine then triggers the actuators that are affected by context updates. The *Context layer* in CADBA [55] processes and generates context, where the context interpreters interpret structures of raw data from sensors and represent information as law level context. Finally, the *Activity Context Broker* in A-CoBrA [97] is responsible for reasoning about context knowledge in context repository.

In summary, reasoning is an important function to infer the meaningful context in context-aware systems. The reasoning process can be carried out using rules such as AI rule base, forward chain, heuristic knowledge, SQL grammar, application specific, or ontology. Different naming are given to reasoning components, i.e., context layer, reasoning engine, reasoning agent, planning level, inference engine, ruleEngine, interpreter, and interpretation layer.

3.3.5 Application

The application component id concerned with the action (adapting the context-aware system behaviour) which is driven by context changes. The following are some examples of application components and application concepts for different context-aware systems. TEA-system [24] uses the *Scripting Layer* which provides mechanisms to include context information in the application using semantics; these are, entering a context, leaving a context, and while in the context. CMF [45] employs the client's *Application* component in order to transparently gain higher-level context from the manager and deliver responses to the clients in an event based manner. In SOCAM system [44], the *Context-aware services* component uses different abstracted levels of context to adapt the system behaviour. Different services which are advertised by the service locating services component can be accessed by the users and applications. The *ubiServices* in ubi-UCAM [35] also provides application services according to the user's needs by recognising context. Then the Service Providers within the ubiServices manage and operate services after receiving information about the service execution. In the sentient object model [105], the *producer interface* is responsible for producing events to adapt current context via some hardware devices. The Context Stack [49] uses the *Utilization Layer* to hosts the context-aware services, which abstracted information to customise the system behaviour according to the current situation (it is responsible for predicting context and triggering certain actions). On the other hand, *Phase-P4* of the RWM architecture [38] is responsible for many tasks, these are: identifying information about service provisioning, resolve device connection, and evaluating services. The *Information Processing Agent* in Multi-agent based architecture [60] is another example of the application component. This agent translates the received commands into actual actions (either simple or complex). The *Action* in WCAM [66] represents a set of both traditional and context-aware services. Subsequently, the Action manager coordinates the context adapter within the Action to support traditional services with context-aware information. In General architecture [57], the *interaction level* is responsible for context changes through interacting with users and other hosts. Again, the Multi Agent Service Reassembling Architecture [48] employs the '*Service Identifying Agent*' to specify the appropriate application and the related services. Moreover, the

application layer in CADBA [55] builds and executes the application according to the context and the related history.

In summary, different services or actions are provided for different systems through the application functional components. These actions could be either an interaction with users or trigger and adapt the environment to provide services.

3.3.6 Management and Service Discovery

Different functional components of the context-aware system should collaborate to process the context information and adapt the system behaviour. The context management, is the process of organising multiple components of context-aware system and also responsible for context sharing to create the final illustration to the users. The management component is part of some context-aware systems, which will be presented in this section. For example, the Widget's *Discoverer* [98] is used for the resource discovery (locate components that are relevant to a given application's functionality); whereas the *Context manager* in CMF [45] is a centralised server for managing a blackboard, stores context and delivers responses and changes to the clients using different mechanisms. On the other hand, the intelligent Broker in CoBrA [61] maintains and manages a shared contextual model, where various policies are used by the *policy management module* to handle the accessing and changing of context information. In SOCAM [44], the *Service Locating Service* provides a mechanism that enables users or applications to access services announced by context providers and interpreters. The *Context Manager* within ubiService [35] is responsible for matching integrated context with an appropriate service to be executed. *Phase-P3* of the RWM architecture [38] is also concerned with service detection and coordination. WCAM's *Controller* [66] bridges events, contexts and services from other components of the architecture according to the model. Important issues of this controller are scalability and dynamic configuration. Another example is the *Learning Services* in CALA [65], which coordinates and provides learning services based on context information through a user defined learning support rule. On the contrary, the *planning level* of the General Architecture [57] detects and evaluates context, makes plans and schedules context reactions. Finally, the *Activity Context-Aware agents* in A-CoBrA [97] coordinate the sensor and actuator agents to assist the interaction with users.

In summary, the management components are used in a number of the context-aware systems. However, within each system this component has different function and task; it is accountable for governing the system, locating or specifying the services.

3.3.7 Knowledge Base

Many context-aware systems have dedicated context repositories to store context history, user profile and other entity related information. This information could be exploited during context processing. Storing and retrieving this context information is necessary to enrich context; it can also be used to derive new context values or predict future context. As an example, the context toolkit [98] stores historical data in distributed databases which could be retrieved by interpreters, but it is harder to maintain and manage. Widget uses two mechanisms to store context in local cash and persistent store. On the other hand, the CMF [45] uses the blackboard (*context manager*) so components post data to this common space to be available to other components. The *Context Knowledge Base* in CoBrA [61] serves as a persistent storage for context information which is represented by semantic web languages (OWL and RDF). Stored knowledge can be queried by components using a set of APIs. It also contains ontologies and a set of heuristics associated with a smart space. The *Context Knowledge Base* in SOCAM [44] also contains a set of APIs so other components can modify context knowledge in the context database. CASS [52] again uses a database for persistent data storage to store context, application, user data and domain knowledge as context awareness rules and behaviours of specific applications. In this system the *Context Retriever* is responsible for retrieving context from a database through the services of the interpreter. In ubi-UCAM [35], the context database keeps necessary context, related information, and context history to support the generation of final context. This operation is controlled by the *Context Manager*. In the Context Stack [49], the *Aggregation Layer* has the task of storing context history and relations with sensed context. This can form a centralised context base and supports the interpretation which is based on current and past context. The RWM architecture [38] also stores user profile, context history and semantics in a database which can be retrieved at some stages. For example, *phase P1* (layer 2) has the task of retrieving id information from database since RFID has a limited capacity. CALA [65] also uses a *Context Knowledge Base* to maintain and share context knowledge on

behalf of the personal agent, which is restricted by the resources and the computing entity. The Multi Agent Service Reassembling Architecture [48] again uses a database to store ontology, context and historical data, which can be accessed using the *Context Managing Agent*. The *Service Manager* accesses the repository to identify application patterns and atomic services of the subtasks. The *storage layer* in CADBA framework [55] stores and queries the context information (such as user profile, device context, and location data) in the domain knowledge.

In brief, nearly all existing systems store context information (including context history), user profile, ontologies in a dedicated database. So, these pieces of information can be shared and retrieved at some stage during information processing.

3.4 Analysis of Context-Aware Systems with Respect to Architecture Styles

After introducing various context-aware systems and studying their architecture aspects, an analysis will be given to evaluate their architecture style and to be aware of what building blocks constitute these systems. From the review it was found that the majority of these systems are application-specific and restricted to specific scenarios. Most of them used fixed components; such components are difficult to change or extend to cope with new requirements requiring new contexts. They use different software architecture styles in their design; some also use multiple styles. Typical styles used in such systems are layered, agent-based, and service-based, although other styles are also used. For example, a number of systems use the layered architecture such as TEA-system [24], Context Stack [49], RWM architecture [38], General architecture [87], CADBA [55], the conceptual layered architectural framework [54], CAMUS [94], and the service architecture [95]. Also, the context toolkit [98] although uses the concept of context widgets that cooperate in peer-to-peer, the Widget's underlying components interact in a layered manner. Another system that uses the layered architecture internally is the ubi-UCAM system [35], which is built around a framework that has two main independent and reusable components (ubiSensor and ubiServices) providing user centric services to multiple users. Some systems use Agent-based style such as CoBrA [61], although the Broker internal components communicate in a layered manner. The multi-agent architecture [64] also uses agent-base architecture; however the agents communicate in a layered

manner. The Multi Agent Service Reassembling architecture [48] uses an agent-based and service-based architecture. CALA [65] uses an agent-based and centralised architecture, with a context-aware manager agent central to it. Another research work that uses this style is the agent-based mobile service architecture [106]. A number of systems use centralised or server-based architectures; CASS [52] and SOCAM [44] are two examples. Although SOCAM used the centralised architecture, the internal components communicate in a layered manner. Many systems adopt other architectural styles; for example, the context toolkit [98] uses peer-to-peer, while CMF [45] uses a blackboard style. Further styles are also employed in some systems; for example CORTEX [101] is based on the sentient object model, which consists of three main components and designed for context-aware applications in ad-hoc mobile environments. The communication between these sentient objects based on events. The WCAM [66] adopts the MVC style, using four independent components; some of these components still use the layered architecture.

A comparison between various context-aware architectures based on some categories of styles has been conducted by many researchers. For example, Bolchini et al. [59] divided context-aware architecture styles into centralised and distributed. While, Miraoui et al. [90] used three styles to differentiate between context-aware systems; these are Client/Server, peer to peer, and hybrid. According to Lee et al. [107] two main representation styles can be differentiated; these are centralised (Context Server) and decentralised (peer to peer). Their advantages and disadvantages have also been given.

From the above analysis and discussion, it was found that the layered architecture is the predominant one; it has the benefit of hiding the low-level details, and supporting separation between context acquisition and context use. Due to the complexity of context-aware systems it is difficult to use this style to support different applications as this architecture lacks the ability to reconfigure, extend and it is difficult to standardise. Other systems using a server based or centralised broker architecture has the benefit of creating thin clients, thus freeing the clients from processing resource-intensive tasks. Nonetheless, such an architecture style has the drawback of a centralised solution which causes a single point of failure as all other components communicate directly with the centralised module. This contradicts with the nature of pervasive systems which are distributed having independent devices. The centralised approach requires a

dedicated server which increases its cost and limits its usability. Bandwidth problem may arise if too many clients (demanding computation) are connected to one sever. Finally, the agent-based solution may not be suitable for real-time applications where response time must be deterministic. The agent based architecture used by some systems has the drawbacks of high communication cost, and requires high time response due to the competition for bandwidth, especially when several agents send data at the same time.

3.5 Analysis of Context-Aware Systems with Respect to Context Representation

This section presents analysis concerning context representation and modelling. It was found that TEA-System uses an object oriented model. It encapsulates context information into objects (the Cues) which allows data abstraction, reusability and controlled access to information. Object oriented approaches require the implementation of an infrastructure to support context operations. The Toolkit uses the attribute value tuple which is a simple model that has poor support to reasoning and knowledge sharing in comparison to other modelling tools such as ontology. Also, the toolkit does not use environmental information in the modelling. Some context-aware systems use ontology in their context modelling, which enables context reasoning, knowledge sharing and reuse. However, it is complicated when used with generic and extensible applications as it cannot handle changing information in a scalable manner (complicated to build general ontology). As an example; CoBrA, SOCAM, Context stack, CALA, Multi-agent service reassembling architecture, CADBA, and A-CoBrA use ontology to represent concepts and relations between them. SOCAM uses OWL and divides their ontology into upper general ontology and domain-specific ontologies. CALA uses OWL to represent a learning environment. The Context stack also uses a web ontology based model (CONON) to facilitate context representation, semantic data sharing and reasoning. On the other hand CORTEX and CASS uses the relational data model, while ubi-UCAM uses application oriented model.

In summary, we believe that existing context models range from simple to sophisticated; each one has its benefits and limitations in supporting a given application. Therefore, a hybrid model can solve the limitations of these models and

increase consistency of the systems. Also, the model should exploit the environmental information and become flexible with various applications.

3.6 Analysis of Context-Aware Systems with Respect to Context History and Knowledge Base

Context history is considered in the design of nearly every context-aware system, where context information is dynamic and changes over time. However, employing history in some of these systems is not clear. Cues considered history as it is a function taking sensor values over time and providing an output. The Context toolkit, CASS, CORTEX, and CADBA store context history in a database. CoBrA, SOCAM, CALA, A-CoBrA, and Multi-Agent Service Reassembling uses database as a knowledge base to store context history and ontology that support knowledge sharing and context reasoning. CASS uses database to store context information, its history, and rules used by the inference engine and the applications. The ubi-UCAM and WCAM uses database to store context history to support the generation of final context and context related to services. The RWM architecture is another example that uses database to involve history and semantic. On the other hand, the Context Stack supports knowledge reuse and reasoning using ontology and context knowledge base through adopting a relational database. The Multi Agent architecture uses database server to store messages, context and reference information for estimating locations. CADBA uses context database to store system context and historical context information.

We believe that context history is an important element of context information; therefore it should be well integrated and used in the context representation. Existing context-aware systems use either distributed or central databases to keep context history. Distributed storage is harder to maintain and manage, while the central store has the drawback of single point of failure. Therefore, using a backup database server can overcome the last problem. Both approaches use dedicated databases other than the organisational databases attached to these systems. In my opinion, context information and context history can be attached to the organisational database rather than a dedicated store to avoid redundancy, inconsistency of information and reduce data overhead as context information is growing with time.

3.7 Comparison of Context-Aware Systems

After demonstrating various context-aware systems and studying their different aspects, a number of criteria are used to compare these systems. The comparison is made according to the architectural style, the main architecture components, context representation, reuse, and context history. Tables 3.2 and 3.3 summarise the features of context-aware systems, and as listed below: -

- Context Modelling: in this category context model can take one of the values: key-value, mark-up scheme, graphical model, object oriented, logic based, and ontology based model, see to Table 3.2.
- Context History: it is considered an important feature and will often form part of the context description of all existing systems although it is not exploited properly see Table 3.2.
- Reuse: two forms of reuse are utilised in this discussion, these are: context reuse and component reuse. Context reuse indicates that systems may use ontology, or knowledge base to support knowledge sharing and history. Component reuse refers to the reusability of components to ease system structuring, reconfiguring, and maintainability, see Table 3.2.
- Architectural Style: the architectural styles used in this comparison are: Layered, Agent based, Server based, blackboard, peer-to-peer, centralised, and MVC. Peer-to-Peer is used by distributed computing networks, where each computing entity has equivalent capabilities and responsibilities; each can functions either as a client or as a server. Centralised architecture is used by some systems for distributed applications; this may involve multiple processes which depend on one central process to serialise all events. Normally, the central unit is responsible for the management, Table 3.3 summarises this study.
- Architecture Functional Components: typical functional components used for context-aware systems are sensing, reasoning, aggregation, modelling, application, management and services, and context repository. However different names were given by each system. Table 3.3 shows the main functional components that each system supports.

From Table 3.2 it was found that each system uses different context primitives which are sensed from the environment. All the systems consider history as an important aspect of context information which can be reused in subsequent states.

Many existing work used ontology to model context information. Also, the majority of the systems have reusable components (mainly for the sensory component). Finally, we found that the context toolkit and SOCAM systems are the most well known systems which are cited and referenced by many researches. They are closest in spirit that gives us guidelines to inspire the new proposed framework. Therefore, these systems will be used in the evaluation of the framework.

Context-aware systems	Context primitives	Context history	Reuse	Context Model
TEA-System		√	Context Component	Object oriented
Toolkit	4W	√	Context Component	Attribute value tuples
CMF	4W1H	√	Context	Ontology RDF
CoBrA		√	Context Component	OWL
SOCAM	5W1H	√	Context Component	OWL
CASS		√	Context Component	Relational data model
CORTEX		√	Context Component	Relational data model
Ubi-UCAM	5W1H	√	Context Component	Application oriented, unified model
Context Stack	4W	√	Context	Object-oriented, OWL
RWM	3W	√	Context	Application oriented, RWM
Multi Agent Architecture	3W	√	Context Component	Logic base
WCAM		√	Context Component	Application oriented
CALA	4W1H	√	Context Component	OWL & RDF
General Architecture		√	Context Component	Data structure
Multi Agent Service Reassembling Architecture	4W	√	Context	Ontology
CADBA	3W1H	√	Context	OWL
A-CoBrA		√	Context Component	OWL

Table 3.2: Comparison to context-aware systems-context representation

Context-aware systems	Architecture								
	Style	Components - Main Functionalities							
		Sensing	Aggregation	Reasoning	Modelling	Application	Management/ Services locating	Store/ Retrieve	Use of context Repository
TEA-System	Layered	Cues		Context layer		Scripting layer			Tuple space
Toolkit	Peer-to-Peer *	Widgets	Aggregator	Interpreter			Discoverer		DB
CMF	Blackboard	Resource server			Context recognition services	Application	Change detection	Context manager	DB
CoBrA	Agent based	Context acquisition		Context reasoning engine			Policy management	Context knowledgebase	DB knowledge base
SOCAM	Server based *	Context provider		Context Interpreter		context-aware services	Service locating services		DB knowledge base
CASS	Server based	Sensor Listener		Rule Engine				Context Retriever	DB knowledge base
CORTEX	Sentient object model	Consumer	Sensory capture	Inference engine	Context representation	Producer			DB
Ubi-UCAM	Distributed * Framework	ubiSensor	Context integrator	Inference engine	Preliminary context generator	Service provider	Context manager	Context preprocessor	DB
Context Stack	Layered	Acquisition	Aggregation	Interpretation	Representation	Utilisation			DB knowledge base
RWM	Layered	real world information phase		Semantic recognition	Real world model phase	Service provision	Service determination		DB
Multi agent architecture	Multi Agent *	Context collecting agent		Context reasoning agent		Information processing agent			DB
WCAM	MVC	Watcher		Model		Action	Controller		DB
CALA	Agent based	Context provider		Context reasoning engine	Context manager		Learning services		knowledge base
General Architecture	Layered	Lexical level		Reasoning level	Syntactical level	Interaction level	Planning level		Context Repository
Multi Agent Service Reassembling Arch.	Multi Agent	Sensor Zone		Context reasoning	Context manager	Application pattern	Service identifying agent		DB Ontology, services
CADBA	Layered	Device layer		Context layer		Application layer		Storage layer	DB for domain knowledge
A-CoBrA	Agent base *	Sensors agents		Activity context Broker	Context capturing Interface	Actuator agent	Activity context-aware agents	relevant context retrieval	DB knowledge base

Table 3.3: Architecture comparison of context-aware systems. Where * refers to multi-style architecture with the predominant style

3.8 Context-Aware Issues

As an outcome of reviewing literature, studying and analysing the characteristics of various context-aware systems the following issues are drawn: -

1. Issues Related to Context Modelling

Context modelling has an important role in information processing of context-aware systems. Therefore, context models should be general, well organised, and adaptable in order to support context abstraction, reasoning, and context reuse. In this respect problems related to context modelling are identified as follows: -

a. Context information such as identity, time, location, activity, environment and other derived context are not fully exploited in the existing context-aware systems. Also, it lacks the possibility to cast off unused context. Moreover, there is no clear definition to context history which describes its role and use in context modelling.
b. Existing models have limited capabilities in context representation and abstraction (rigid representation to abstracted context), and restricted support to context reasoning. Also, some of these models require a large knowledge base for reasoning.
c. Complex representation to context, especially with application specific systems and some ontology based modelling. Ontology based model represents concepts and relations between them; it has the advantages of enabling context knowledge sharing and reuse to enable context reasoning and interoperability between applications. However, existing ontologies have the drawbacks of lacking generality (cannot be used for a range of applications). It is a sophisticated concept of knowledge representation and reasoning. It may be imprecise or incorrect for certain applications and also cannot handle changing information in a scalable manner. Ontologies require storage mechanisms and engines for managing, with high requirements on resources; so, editing and updating existing ontologies are complex tasks.

2. Issues Related to Context-Aware Architectures

By reviewing various context-aware systems and studying their architectures, it was found that most existing context-aware systems implement the layered architecture style in their design. This style has the advantages of high level of abstraction in

design (allows problem partitioning) and support enhancements and reuse. However, it has its disadvantages too, for not all systems are easily structured using this style. Besides, defining layers for some systems is not trivial with implementation can differ vastly from the model, and it may be difficult to find the right level of abstraction. Using layered architectures with context-aware systems makes it difficult to support extensibility and reconfiguration (i.e. updating or removing services according to context). There is an inherit problem with context-aware systems, where they are not similar as the case with network systems. Moreover, there is no common pattern of system organisation, for each system is different and deals with different parameters and entities. In general, most existing architectures do not handle dynamic changes in context or application. Nonetheless, context-aware systems have common usual functions which can be used recursively (Table 3.3).

3.9 Summary

Advanced techniques care for designing reusable and independent modules which can be assembled to ease system maintainability, reliability, adaptability, system development and simplify application deployment rather than using a single monolithic program. After investigating existing context-aware systems and considering context-aware issues, we aim at designing a generic context-aware architecture that can be used by a variety of systems. It is to note that context dimensions, context history, and modelling techniques of existing systems are studied and analysed in chapter 2; whereas the functional components of context-aware systems and their architecture styles are studied and analysed in this chapter in order to find the best fit to the proposed solution. From the discussion given in this chapter, and considering the requirements of the existing systems, we concluded that a generic and extensible context-aware framework using the Pipe-and-Filter architecture style is needed. This framework can simplify the construction of these systems, so context-aware applications can be widespread.

Chapter 4

Research Approach and Concepts

Despite the development made in ubiquitous computing and context-aware systems, these fields are still evolving, and their applications are still limited. Throughout reviewing literature much experience has been gained, along with getting through many existing problems to find answers to these questions. These problems were concerned with the conceptual issues in this field and others related to the architecture of context-aware systems. This chapter provides an outline to our findings and presents the research contribution, which will be represented by a conceptual scheme. The scheme includes classification of context information, some definitions and terminologies used, the proposed context-aware architecture, and finally a scenario to demonstrate a case study of these systems in a smart environment. All these subjects will be discussed in this chapter with the proposed design of the context-aware framework.

4.1 The Conceptual Scheme of The Proposed Framework

A conceptual scheme can be used to help in the communication of the main ideas behind the design process. In this research a new framework for developing context-aware smart environments will be proposed. The new framework helps in solving problems related to the context-aware architecture and the complexity of designing these systems as mentioned in the research problems in chapter 1. Figure 4.1 shows a conceptual scheme (similar to mind map) which is employed in the design of the proposed framework. This Figure illustrates how different concepts collaborate to realise the architecture.

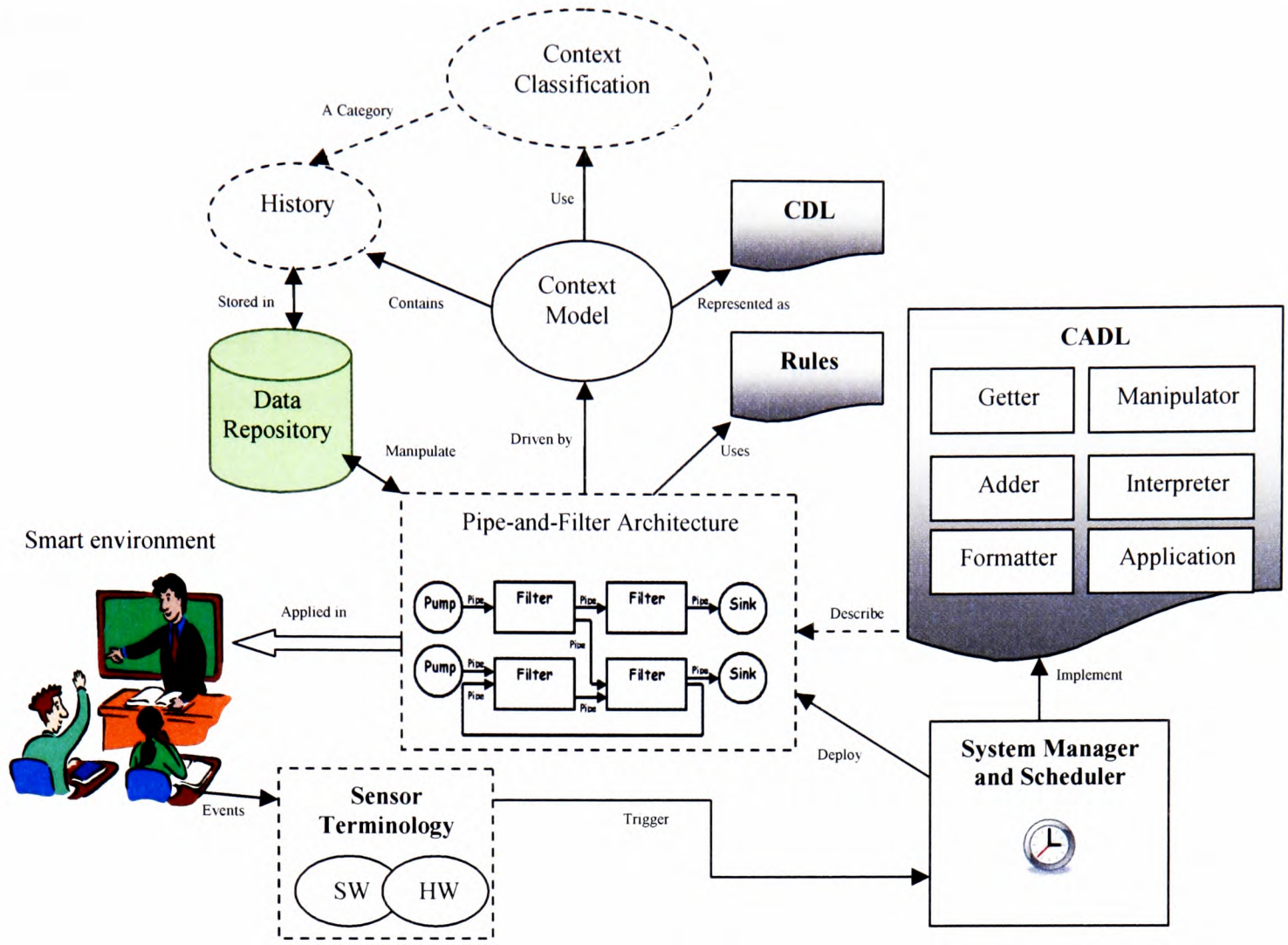


Figure 4.1: GECAF framework components and relationships between them

Firstly, context information is classified into different classes and categories. Then using ontology we show the relations between these classes and categories, and how each context can be derived and used. This classification with the definition of some unclear terms are utilised in specifying which context type to use, the structure of the context model and the structure of the Context Description Language (CDL code). Context history is one of the context categories that is identified in this chapter to better understand and use in the context-aware system design; where context history is kept in a data repository as a knowledge base. Afterwards, the main building blocks (components) of the context-aware system were identified and adopted in the Pipe-and-Filter architecture to put them together in different arrangements (depending on a given application). As the interpreter has an important role in any context-aware system for reasoning about context information, an XML Rule Language has been developed to be used by application developers. Consequently, a Context-aware Architecture Deployment Language (CADL) has been designed; it is written in XML code to describe the architecture's components. The sensor's terminology defines and discusses important terms related to various sensor types; it has an important role in the proposed design as it characterises context sources and describes how different

pieces of context information are acquired. To govern the context-aware system operation, a 'system manager and scheduler' has been proposed. It is used to schedule events triggered by the software and hardware sensors. It also manages deploying and implements the context-aware system using descriptions from the CADL code. Finally, the proposed framework is implemented in a smart environment scenario as a case study using different application's categories. In the following sections we are going to describe the elements of the framework.

4.2 Context Representation

This section demonstrates a classification to context information, and shows the relations between them. The rationale behind this classification is to be aware of what constitutes a context, what are the relations between different types of context in order to simplify context reasoning. Moreover, definitions and terminologies are presented to explain some useful terms.

4.2.1 Context Classification and Definitions

Several approaches have defined and given classification to context information and its different terms. To better understand what constitutes a context and how it is used, we present a new classification to context information which shows how each context can be derived. Ontology to describe the relations and associations between these categories is also used, see Figures 4.2 and 4.3. Context information can be classified into present, historical, and predicted (or futuristic) context; these categories can be further classified as demonstrated below: -

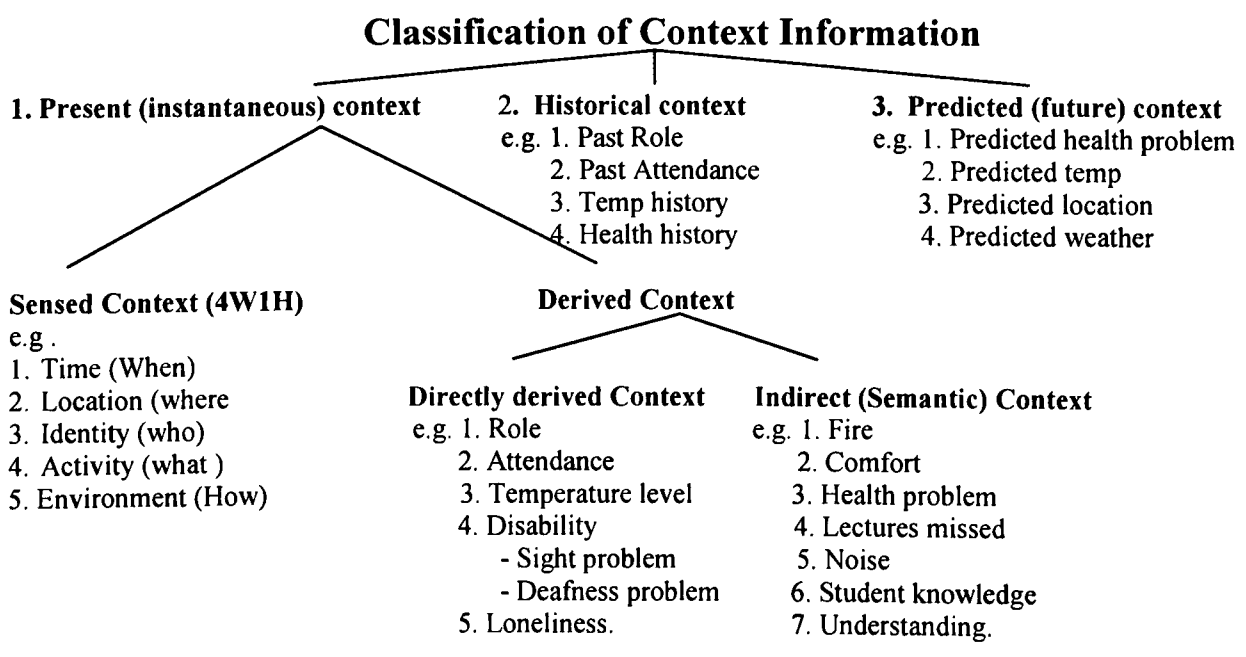


Figure 4.2: Context classification

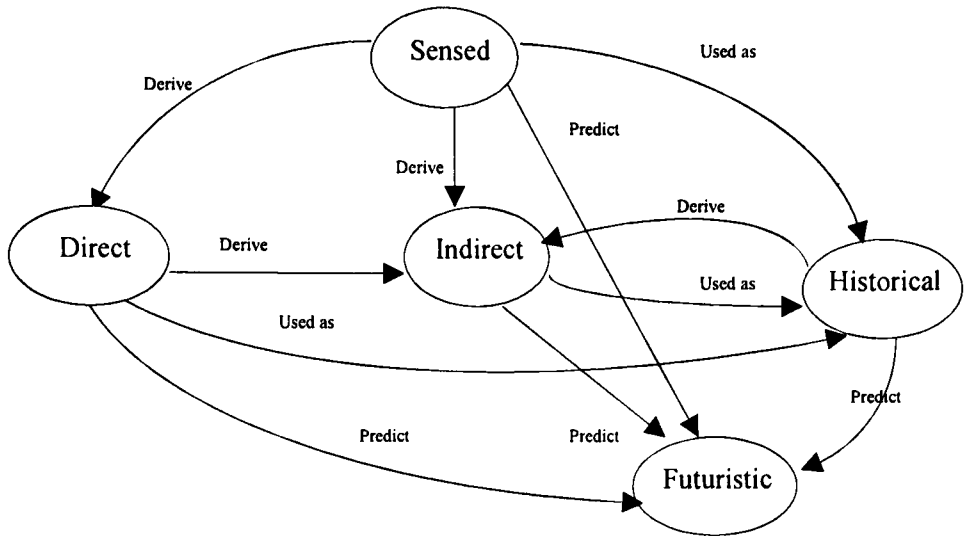


Figure 4.3: Ontology to represent the relations between context categories

1. *Present* context: is the instantaneous context or the current situation that characterises an entity; it can be either sensed (primitive) or derived context.

   - *Sensed* contexts (4W1H) represent different pieces of raw context information which can be sensed by hardware, software, or both sensors (see section 4.2.2). These are the identity (who), location (where), time (when), activity (What) and environment (How).
   - *Derived* context can be direct or indirect (semantic context). *Directly* derived context can be imitated from primitive context, with entity characteristic i.e. information retrieved from internal or external databases. *Indirect* context can be inferred by abstracting different pieces of context information.

2. *Historical* context: is considered as an important feature for approximation of a given situation or environment. Context changes with time; it has an impact on context-aware systems and can be used to derive other contexts. In this respect, some context information is considered dynamic. Referring to Figure 4.1 we can conclude that historical context is a time variant entity. To better understand and use context history, a comprehensive definition is proposed: -

   'The past context information or status of an entity that affects context-aware systems behaviour (i.e. the system adapts its behaviour according to the changing status of an entity rather than its absolute value). Contextual information is a time variant entity, therefore context history represents the order and time of events which may be used to accurately predict the action of an entity or the intention of a user in current or future state'

3. *Predicted* context (futuristic context): is the anticipated context which could be derived from entity characteristics, present and historical context. This context can be derived and stored to take action in subsequent status.

All the above mentioned categories of context could be either static or dynamic depending on context information and the application, i.e. primitive contexts like, time, location, activity, and environment could be static or dynamic. On the other hand context information in some situations may have different interpretation. For example, a person's role in a store environment is a static value and does not change (shoppers, supplier, shop assistant, etc.). However, a person's role in a university environment may change with time, location, or in conjunction with other sensed information (virtual or sensed). In this case, a person could be a postgraduate student or teaching assistant when time, location environment and activity change. Another example is within a hospital environment, where a person's role my change from physician to patient depending on persons health situation or activity. Temperature context can also be changed with time, location and environment, etc. In the literature there is less emphasis on the context history; hence from the study of this thesis it can be concluded that this element will be highlighted in the implementation chapter.

4.2.2 Sensor Terminology

Context information is captured from physical environment as well as electronic world using different types of sensors. These sensors can take different names depending on how context information is sensed. From reviewing the literature, one can quickly realise that sensor terminology is not standardised, and the existing works use dissimilar terms to define sensor types. To solve the confusion in using sensors terms, we introduce a general naming convention as shown below: -

1. *Hardware* sensors: are sensing (physical) devices such as RFID tags, Camera, GPS and temperature sensors that measure the physical parameters (raw context information) from the environment.
2. *Software* sensors: are the logical data acquired from host devices (system's internal or external databases like LDAP [108] or system files such as system calendar, a stored picture or stored address). The term 'software' is used when contextual information is stored using software means.

3. *Hybrid* sensors: combine the information from both sensor types (hardware and software) to derive a new piece of context. For example, person's identity can be detected by combining a profile captured using a camera with a picture retrieved from a database.

In reality, to apply the above terminologies, context can be acquired from two sources these are: hardware and software sensors. However, the information acquired from hybrid sensors is derived by combining the information from both hardware and software sensors through aggregation, and interpretation processes.

4.2.3 Context Modelling

A general context model is important to simplify context representation, so that context-aware applications can be built easily. To represent context information and consider all context dimensions (context primitives, context history, etc.), a general and simple model has been proposed. This formal model characterises context information as a function of four W's and one H (or 4W1H); these are Who, Where, When, What and How; refer to equation 4.1 below.

$$\text{Context} = \text{Entity (ID, Time, Location, Activity, Environment)} \quad \ldots\ldots (4.1)$$

These five parameters represent the primitive pieces of context information (Identity, Time, Location, Activity or Task and How the Environment is). Therefore, abstracted context information can be derived from these 4W1H and other terms (such as context value and context history). Depending on the context in use, each of these parameters could be assigned an applicable value or denote not applicable. As time (When) context could be a series of time periods, it is used to order events and identify when information is detected (to be considered as historical data afterwards). To illustrate this concept, context information primitives are represented using a Context Description Language (CDL) which is XML language used for wrapping the elements of the formal context model. Each context primitive is modelled by a mark-up tag with attributes and values as shown in example 4.1.

Example 4.1: CDL code to represent 4W1H

```
<contextPremitive name="context_name" value="context_value">
    <ID>identification code</ID>                          <! --- Who  --->
    <Time>Time and date of occurrence</Time>              <! --- When --->
    <Location>location</Location>                         <! --- Where --->
    <Activity>what activity</Activity>                    <! --- What  --->
    <Environment>existence of others</Environment>        <! ---  How  --->
</contextPremitive>
```

Where,

Name: refers to context type such as temperature, attendance and role.

Value: raw information collected from sensors which depend on context type such as degree Celsius or m/sec, or derived context such as attendance status and person's role.

ID: is an identification code attached to an entity to reference the characteristics of given pieces of context.

Time: is a date/time-value describing when the context is sensed.

Location: refers to where context information is detected.

Activity: refers to context activity and its current situation, e.g. walking, running or sleeping.

Environment: refers to the surrounding conditions such as existence of other people, resources in the room or weather conditions.

The proposed model is simple since context is represented in a general form (employ all context terms) using a fixed number of tags to define each context primitive parameters (4W1H). Also, it is flexible as each of these parameters may or may not exist (can take a 'null' value). Since context information is dynamic and changes over time, history is added to the model. Besides, context value is added as it represents the actual (abstracted or meaningful) context. Example 4.2 illustrates the proposed model in CDL code with history and value. Context Identification Code (CID) is also added to reference a particular context during information processing.

Example 4.2: CDL code with history

```
<Context CID="c1">
   <contextPremitive name="Role" value="student">
      <ID>@103</ID>
      <Time>1 April 2011 9:30 Am</Time>
      <Location>Classroom N233</Location>
      <Activity>Sitting</Activity>
      <Environment>Occupied</Environment>
   </contextPremitive>
   <History time="1 March 2009 9:30 Am">Lecturer</History>
</Context>
```

Indirectly derived context is also represented in XML as given in example 4.3. This document gathers different abstracted contexts to be further enriched or reasoned about to derive an indirect context.

Example 4.3: Indirect formatted context in CDL

```
<context name="Lecture type", value="Recap">
    <context1 name="Role">
        <c1 time="10">Lecturer</c1>
    </context1>
    <context2 name="Attendance">
        <c1 time="10">21</c1>
    </context2>
</context>
```

4.3 The Generic Framework Design

Due to the inherited complexity of context-aware systems and bearing in mind the analysis given in the previous chapter, it is argued that using an open architecture is important for designing such systems. It handles diverse applications, supports reuse, and considers adaptability and extensibility, so developers can adopt it to customise their design when building their own systems.

4.3.1 The Generic Pipe-and-Filter Architecture

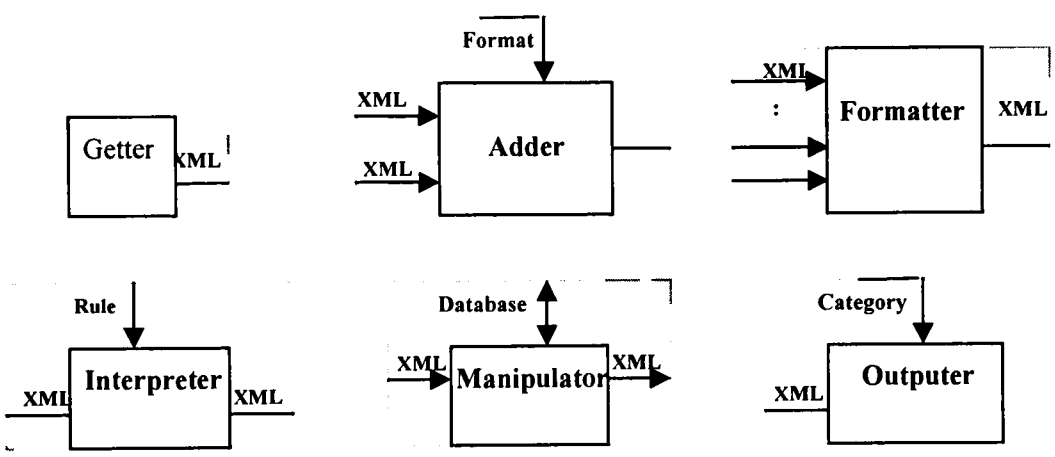
To design an open architecture, different software architecture styles have been reviewed, followed by studying and analysing the usual functional modules of existing architectures. All context dimensions (primitive contexts, user's profile and context history) for context representation are also considered. In view of these requirements and considering the main components of the context-aware systems used in the analysis, a generic framework has been proposed. The framework provides reusable building blocks to simplify system integration and minimises development efforts. The framework employs the Pipe-and-Filter architecture style, in which context information (with machine readable format) is passed through a chain of components (known as filters or processes) to be processed and transformed in a variety of ways. These filters have functional cohesion and can be used in different contexts. When systems are constructed, a number of filters may be joined together and configured seamlessly to suit the implemented system requirements. This is similar to the UNIX pipe where a number of tools may be strung together to form a larger functionality. Conceptually, the framework can be implemented using six major building blocks (filters) to build a context-aware architecture. These building blocks perform context acquisition (getting), modelling (formatting), enriching (aggregating), reasoning (interpreting), storing/retrieving (manipulating), and processing (application) context information. In the proposed architecture the pipes (XML

documents) are joined to glue the components together, where the output document of one filter becomes the input to another. This gives a smart solution which makes the system easily maintained or modified by eliminating, changing or adding new filters. The framework allows developers (through a deployment language based on XML) to design flexible, extensible, and adaptable systems by means of reusable filters that are independent of each other. New context and services can be easily added by using and rearranging various filters due to the architecture adaptability. Developers may choose from a pool of existing filters to perform a specific process or they may build new ones and use them in the pipe. Accordingly, when a new filter is developed it must conform to one of the general types in order to be used in the pipe.

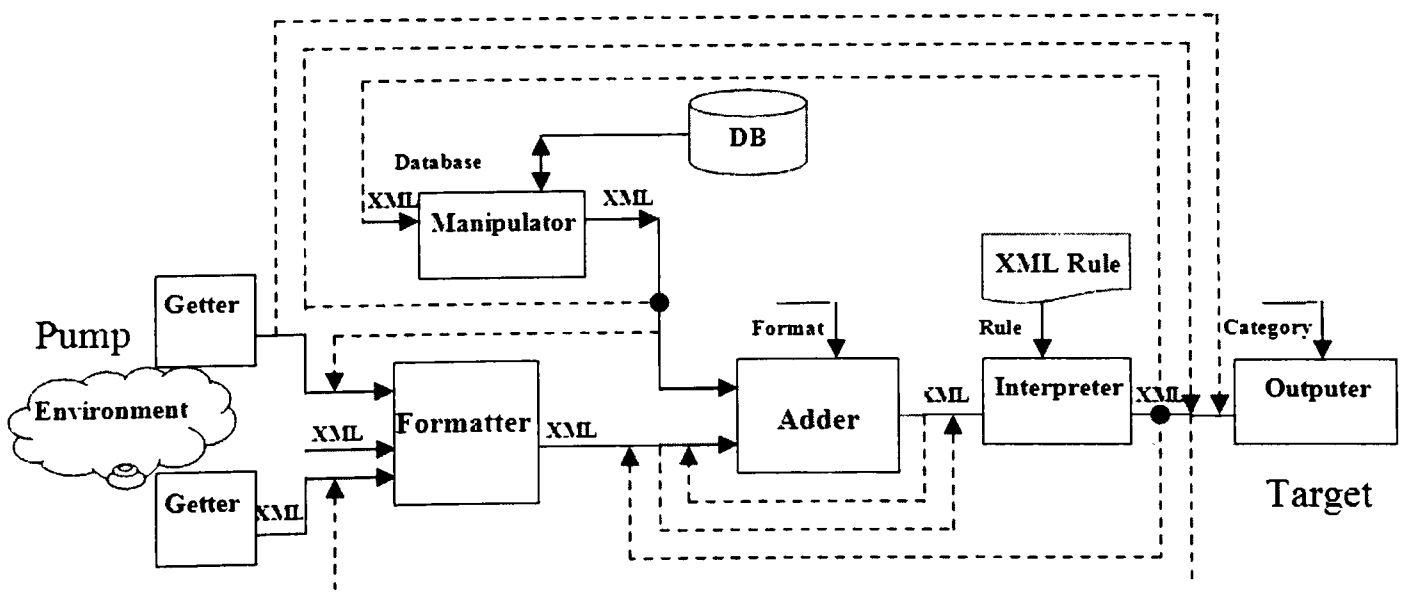
General filter types of the proposed framework are presented in Table 4.1. Figure 4.4 (a) shows the major abstract filters, whereas Figure 4.4 (b) shows a possible arrangement of the architecture filters in context-aware systems. Figure 4.4 (b) shows that the environment acts as a pump (producer), while the sink (or consumer) is a data target or the action (or application) of the context-aware system. The "*Getter*" filter detects and retrieves primitive context information (ID, location, time, activity, and environment) from different dimensions, either physically from the surrounding environment or logically from the internal system. The *"Formatter"* filter takes primitive or indirectly derived pieces of context information and place them in a machine readable model i.e. represent context information using CDL code (described in section 4.2.3). The *"Adder"* filter is used to aggregate context information depending on a given format. It can either gathers derived with formatted context information, collects several pieces of retrieved context information or links primitive context information with historical context (from database) and other data resources. This will enrich the context information to become more accurate and practical. These pieces of context information will be advertised in a machine readable model (using CDL code) to be processed in the next stages. Example 4.2 illustrates the modelled context information with the augmented historical data.

Filter type	Operation	Description
Getter	Context acquisition and retrieving.	Extract primitive context information from the environment using different types of sensors.
Formatter	Modelling or Formatting.	Model context information in a formal XML code, using the CDL language.
Adder	Context aggregation	Augment the retrieved or interpreted context information to the formatted XML code.
Interpreter	Reasoning and Processing	Reason about context using a given XML rule language (specified by the application designer).
Manipulator	Storing and retrieving context information	Store context for later retrieval as a historical context. This is achieved by specifying the path of data, i.e. URL, file name, etc.
Outputer	Adapting context-aware environment	Outputer filter specify the application type (automatic, tagging, or presentation) and use the abstracted context to change the environment.

Table 4.1: General filter types



(a) Main architecture filters



(b) A possible arrangement of the architecture filters

Figure 4.4: Pipe-and-Filter architecture Filters

The *"Interpreter"* filter is used to reason about a given context either directly from context primitives or indirectly from the derived context. Directly derived context information can be reasoned about by interpreting formatted primitive context acquired from hardware and software sensors. In this case the output represents the abstracted context value; see example 4.2. On the other hand, indirectly derived context uses primitive, derived and aggregated context information to infer a meaningful context. The Interpreter utilises an XML rule language (see section 5.3), which arranges the interpretation rules and the relation between different contexts in an XML document. The interpretation rule mechanism benefits from mathematical, logical, conditional, and other functions. These rules are retrieved during runtime and arranged in Reverse Polish Notation (RPN). The rule mechanism is implemented using the Interpreter design pattern [109]. The *"Manipulator"* filter has the role of either storing context information for later retrieval, or retrieving stored context information (historical context or others) from internal or external databases. The *"Outputer"* filter is used to change system behaviour according to the final abstracted context (performs an action). Outputer applications are of three categories: automatic, tagging, or presetting. Besides, different application types can be implemented, these are: hardware, software or distributed application such as SOAP [110]. Finally, the deployment of these filters along with the allocated services can be realised through an application developer interface which is set in an XML document (CADL code, see section 4.3.4).

4.3.2 Basic Guidelines for Arranging the Architecture Components

The key characteristic of this architecture is being customisable for the targeted environment. Like any architecture using the Pipe-and-Filter style, there are certain guidelines that should be followed when using and arranging its filters to design a context-aware system; these guidelines are: -

1. The *"Getters"* should be used as starting filters to acquire primitive context information from the surrounding environment or from the internal system.
2. The input to the *"Formatter"* is one or more primitives, derived or indirectly derived context produced by the Getter, Manipulator or Interpreter filters.
3. The *"Adder"* takes a single piece of context (either from Getters, Manipulator, Interpreter or even from an Adder itself) to be aggregated in an XML document.

4. The *"Interpreter"* takes an XML formatted context (Formatter output) or aggregated context (Adder output) to produce a single piece of context (abstracted context).
5. The *"Manipulator"* can either be used to store interpreted context, or retrieve context information (context history, or any other information) from internal or external sources (databases) to be fed to the Adder or the Formatter.
6. The *"Outputer"* takes the abstracted context from the Interpreter or the Getter to adapt the system's behaviour (takes an action).

4.3.3 Application Types

Context-aware systems use context to provide relevant information and/or services to the user. According to Dey et al. [8], context-aware applications have three different categories, these are: presentation of information and services to a user, automatic execution of a service, and tagging of context information for later retrieval. The proposed framework which employs the Pipe-and-Filter architecture has a number of filters; where the 'Outputer' filter represents the final stage of the architecture. It has the role of providing services (take an action according to context changes), which are related to the specified application. In this filter all the application categories proposed by Dey [8] have been considered. Moreover, two types of actions are identified, these are: -

1. *Hardware* action: is an action associated with the output devices (actuators) which are connected to the context-aware systems. These devices (local or remote) can change their states according to a context value. This value could be either *'discrete'* (e.g. on/off to change door status), or *'continuous'* (e.g. a given voltage to change device temperature, speed, etc). In either case, a number of parameters should be specified; these are device address, output value, data type (discrete or continuous) and transmission type (serial USB port, Blue-tooth, IR, Parallel port, etc.). These parameters can be specified by the application developer in the CADL code.
2. *Software* action: is an action where the software alters its status according to context changes. The output value could be a '*discrete*' value (e.g. to switch-on or shut-down a screen), '*continuous*' (e.g. to represents speaker volume, font size, etc.), or string (e.g. a web site address, email address, etc.).

As stated earlier, there are three application categories: tagging, presentation, and automatic execution. The first type monitors system's changes without changing the environment, while the second presents information/services to the user, which could be either software or hardware action. The third type takes an immediate action according to the context, which could be either software or hardware and depends on the application.

To develop distributed applications different protocols and languages can be employed to facilitate accessing web services. Remote Procedure Calls (RPC), SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), UDDI (Universal Description Discovery and Integration) [111], message queues, middleware, etc. are the de-facto protocols for communicating between application components in a service-oriented distributed application. SOAP is the widely used protocol in most application examples. Therefore, explicit support to SOAP has been included in the framework (as a software action) and as an example of delivering distributed applications. A brief overview of the SOAP protocol is described in appendix A.

4.3.4 Context-aware Architecture Deployment Language (CADL)

To build an open and generic framework, a user language is necessary to assemble the architecture filters and specify the types of services. It should be an easy-to-learn syntax and usable by application designers, in addition of being extensible. A simple user interface language is of great importance to build a user specific application and abstract the context-aware system formation. This language would greatly lessen the amount of code that needs to be maintained and reduce the maintenance cost. In this respect, a new mark-up language is developed to realise and assemble the context-aware architecture filters. Context-aware Architecture Deployment Language (CADL) is the new developed user interface language based on XML; its code can be written by the application developers to build their specific context-aware system. This CADL code should conform to a given XML Schema (see appendix B). According to this language the application developer can specify the architecture filters (processes as used in the CADL code) to be assembled, the contexts in use, the interpretation rules, the application type and category, in addition to the address and value type used by the output functions. This language is extensible as various contexts can be

included; it also has generic elements *"Element"* which can be used to set different parameters for different applications. The CADL code also differentiates between different events fired by different sensors and associate a CID with the relevant event. Example 4.4 illustrates the skeleton of the CADL language. More information about CADL structure will be given in chapter 5. There are other description languages that can be tailored to other specifications. For example, Siegemund [112] used a higher level description language called Smart-Its Context Language (SICL); however it is used to facilitate the development of context-aware services and applications.

Example 4.4: The skeleton of the CADL code, highlighting the main elements of the code.

```
<?xml version="1.0"?>
<Filters xmlns:an="http://www.w3.org/2001/XMLSchema-instance"
    an:noNamespaceSchemaLocation="CADLSchema.xsd">
    <events>
        <event id="an_event_id" CID="context_id">
            <description>a_given context</description>
        </event>
              :
    </events>
    <process type="main_filter_name" CID="context_id">
        <class name="concrete_class" address="associated_address"
            context="context_value" RID="rule_ID">
            <source no="file_number" type="file_type">source_file_name</source>
                :
            <target type="file_type">target_file_name</target>
            <Element name="element_name" value="element_value"
             type="element_type" fileType="file_type">source_file_name</Element>
                :
        </class>
            :
    </process>
        :
</Filters>
```

4.3.5 System Manager and Scheduler

Context-aware systems are responsible for acquiring, processing, managing, and distributing context information with respect to the applications' requirements and services. To govern the whole system process, an event driven system manager is essential to sequence events fired from heterogeneous and distributed sensors (such as RFID readers, system timer, temperature sensors, etc.) and manages the context-aware system. In reference to Figure 4.1, we are going to explain in details the system manager and scheduler. As shown in Figure 4.5, the System Manager has a central First-In-First-Out (FIFO) queue to sequence contexts in the order of their occurrence. As an event occurs, the primitive pieces of context information (4W1H) are acquired from the attached sensors; then abstracted and attached with an Event Identification

code (EID) and finally saved in a queue. The EID is a unique identification code attached to each event, and is associated with context primitives. To specify the context in use, the application developers CADL code can map each event with a Context Identification Code (CID). The system manager has also a timer that issues an event every x seconds, to read the queue head and proceed with the Pipe-and-Filter sequence according to the CADL code.

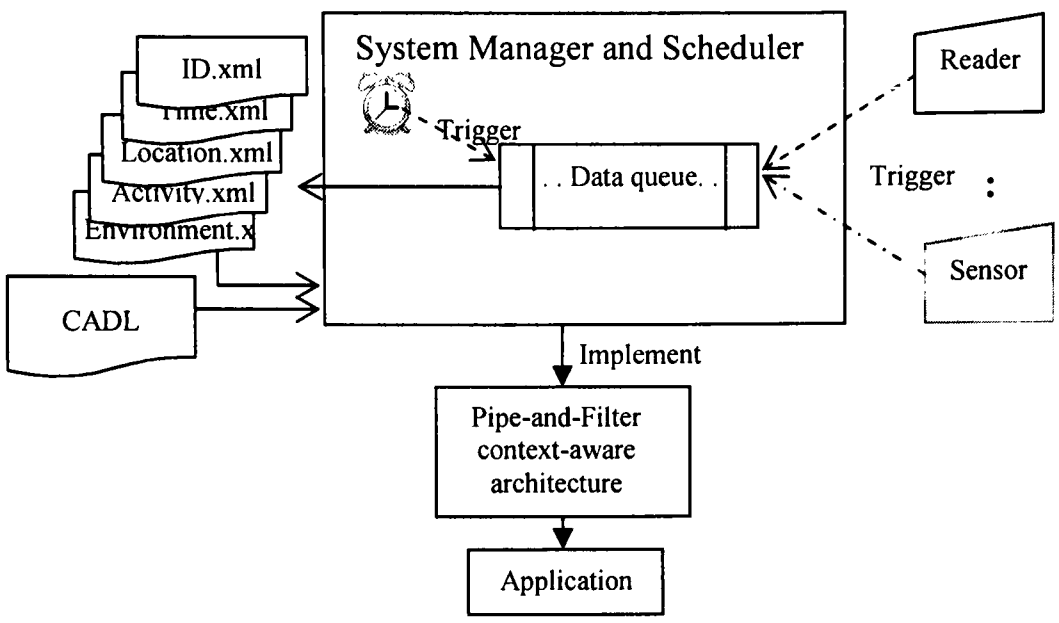


Figure 4.5: System Manager and scheduler

4.4 Framework Realisation in Smart Environments – Classroom Scenario

As an illustration for a detailed scenario that interprets the generic feature of the new proposal, we use the smart classroom scenario to validate the proposed framework. To illustrate the framework main concepts, a smart classroom scenario has been chosen as a case study. A context-aware classroom is a smart environment equipped with many context-aware innovations. Such innovations comprise networked computers, electronic boards, intelligent displays, etc. Consider a student who has particular needs because of personal circumstances such as short sighted and hearing difficulties. Such a student requires that all fonts of delivered materials to be adjusted (enlarged) and voice amplified. These particular requirements are only needed when this student is present in the classroom and when a lecture is delivered. Presence of other students may place other requirements on the environment; for example how many students are currently present but were absent from previous lectures. This smart environment has to detect time, identity, role of the classroom inhabitants and their history, and then interpret the information as context to adapt the behaviour of the target application. Target applications could be a projector intensity setting, size of

presentation slides, speaker volume or even web browser software (running on individual PCs) displaying other information related to the delivered lecture. Bill (a fictitious student) has found such an environment particularly useful to aid his learning considering his personal problem (short sighted and hearing disability).

It is nearly 8.45 O'clock on Monday morning when Bill arrives at the University building and intends to attend his first lecture of the day. As Bill enters the classroom, his picture together with attendance time is displayed on a smart e-board hanged on the side wall. When Bill sits down, his personal computer (PC) displays the student's web page, and the font size is adjusted according to his need. Several messages are displayed telling him about his homework, history of attended lectures, e-mail, etc. Few minutes later, most of the students become inside the classroom, sitting in front of their PCs. As Professor Smith (a fictitious lecturer) enters the classroom, and becomes near the front e-board, a recap of the last lecture is displayed on the e-board, because students' attendance was less than 20% in the preceding lecture. The lecturer's voice amplifier is also adjusted according to students' needs. The PC displays of all students are switched to the lecture webpage viewing the lecture notes, student e-book and other links. The above scenario will be referred to in the next chapters to demonstrate the implementation of the framework.

4.5 Summary

In this chapter, a new way for building context-aware systems is proposed. The new approach is based on a generic and extensible framework which employs the Pipe-and-Filter style to arrange the architecture building blocks. The framework is customisable and can be used to instantiate the architecture building blocks of various context-aware systems. Two languages have been proposed; one for the deployment of the architecture building blocks and another for the XML rules (used in the reasoning process). These languages assist application developers to customise their own systems. Finally, a scenario for a smart classroom is given; it illustrates the requirements of smart environments, summarises the use of context dimensions, and demonstrates various applications. This scenario will be realised in the next chapter through the implementation of the extensible framework.

Chapter 5

GECAF Framework - Implementation

This chapter describes the extensible framework implementation including the employed (Pipe-and-Filter) architecture and its main filters in use, their creation and association. A description of rule mechanism for abstracting context information is also given. Moreover, a demonstration to customise the main building blocks to a particular application is presented, followed by the implementation of the system manager and scheduler. Then, the implementation of the case study for the smart classroom is given.

5.1 The Pipe-and-Filter Components

A Framework is a set of collaborating classes that form a reusable design for a particular class of software [113]. We devised a context-aware framework as an instance of the Pipe-and-Filter architectural style. This style is mainly concerned with stream transformation that the functional behaviour of the system can be derived compositionally from the constituent filters' behaviour. Each filter in the Pipe-and-Filter architecture reads a stream of data from its input, do some sort of transformation on it, and produces a stream of data on its output [82]. The connectors (pipes) of this style serve as conduits for the streams, which are represented by XML documents in the architecture. The architecture is implemented using six abstract filter types that perform acquisition (*Getter*), enriching and aggregation (*Adder*), modelling (*Formatter*), reasoning (*Interpreter*), storing and retrieving context information (*Manipulator*), and Application (*Outputer*). The realisation of the architecture's abstract filters involves identifying classes at different levels of abstraction and interfaces, and establishes relationships among the classes and inheritance. An overview of these filters and their implementation details are given in the following sections. To create the framework components object-oriented software is built, in

which different design patterns are used. This software is written in java, and other enabling software technologies, i.e. XML, DOM, XML Schema, and HTML with a scripting language such as PHP. SOAP protocol is employed to access Web Services in distributed applications. A database server to store users' profile, context information, historical context and other information is then used.

5.1.1. Getter

The Getters are the first filters in the context-aware architecture, as they are accountable for acquiring primitive context information from distributed heterogeneous sensors. The UML diagram shown in Figure 5.1 represents the prototype of the *'Getter'* filter, in which *'GetContext'* is an abstract class that implements the *'Getter'* interface. The concrete classes *'ID'*, *'Time'*, *'Location'*, *'Activity'*, *'Environment'* and *'Value'* can then be used to instantiate different types of Getters objects. The Getter interface defines a generic method *'getContext'*, so each concrete class can tailor the method to read sensor's data and then place the low level abstracted context in an XML document. Example 5.1 illustrates the target XML document of the *'ID'* Getter.

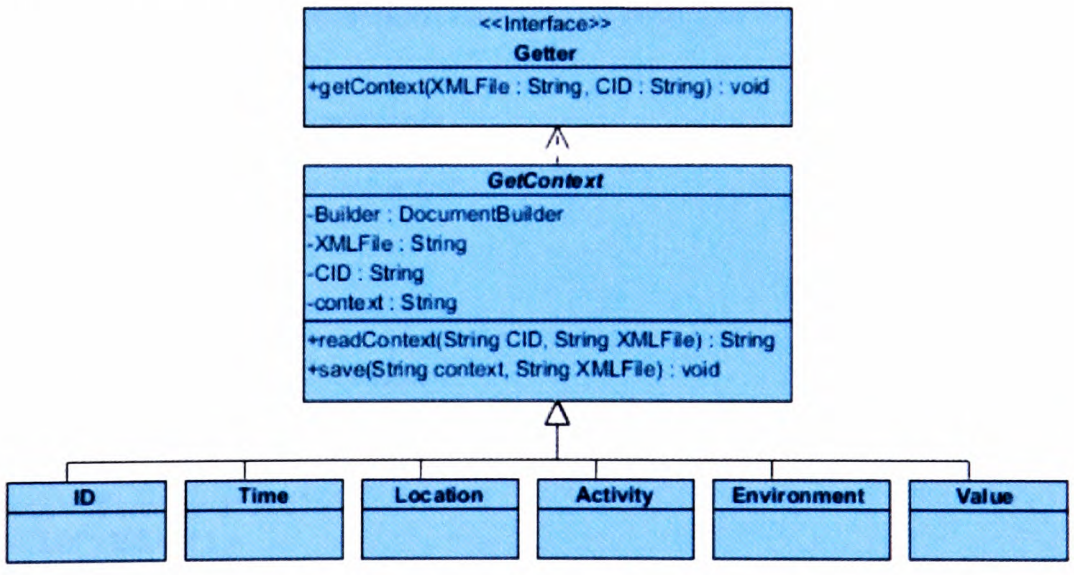


Figure 5.1: Getter filter of the GECAF framework

Example 5.1: The target XML document of the *'ID'* Getter.

```
<?xml version="1.0" ?>
<context>
   <a1 time= "10">@103</a1>
</context>
```

5.1.2. Formatter

The Formatter is used to model the context information in a machine readable format in order to support context reasoning. Figure 5.2 shows a UML diagram representing

'FormatContext' (an abstract class implements the *'Formatter'* interface), which defines a generic method *'FormatToXML'*. The concrete class *'FormatParameter'* tailors the generic method to get primitive contexts from the Getter's outputs (the pipes or XML documents) then transforms the modelled context information into a target XML document (CDL code); see example 5.2. The concrete class *'FormatIndirectContext'* arranges different pieces of primitives and/or abstracted context information (direct or indirect) in a given format; see example 5.3.

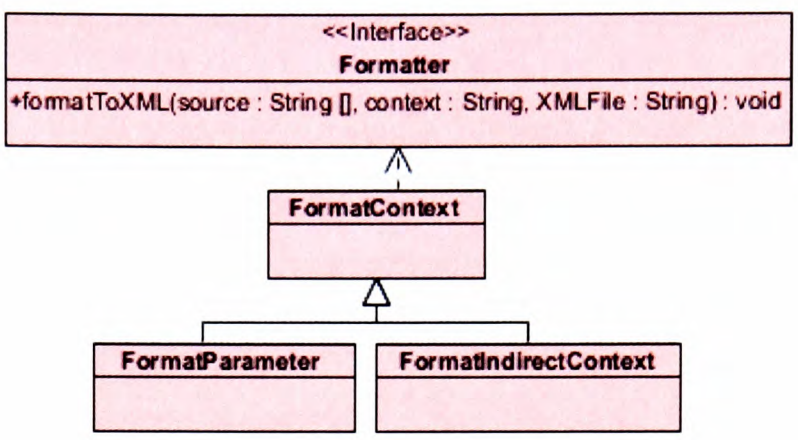


Figure 5.2: Formatter filter of the GECAF framework

Example 5.2: Formatted primitive context

```
<?xml version="1.0" ?>
<Context CID="c1">
    <contextPrimitive name="personAttendance" value="Present">
        <ID>@103</ID>
        <Time>10</Time>
        <Location>classroom 245</Location>
        <Activity>Sitting</Activity>
        <Environment>Occupied</Environment>
        <Value>a_Value</Value>
    </contextPrimitive>
</Context>
```

Example 5.3: Formatted indirect context

```
<?xml version="1.0" ?>
<Context name="Fire" value="ON">
    <context1 name="Temperature">
        <c1 time="10">50</c1>
    </context1>
    <context2 name="Humidity">
        <c1 time="10">0.4</c1>
    </context2>
    <context2 name="Smoke">
        <c1 time="10">Yes</c1>
    </context2>
</Context>
```

5.1.3. Adder

The Adder is used to enrich context information by aggregating context information from different sources. The UML diagram shown in Figure 5.3 represents the Adder filter, in which *'AddContext'* is an abstract class that implements the *'Adder'*

interface. The Adder interface defines a generic method *'addContext'*. The concrete class *'AddValue'* which is a type of adder, tailors the generic method to add the abstracted context value to the formatted primitive contexts and as illustrated in example 5.4. The *'AddHistory'* is another concrete class which is used to add retrieved context history to the formatted primitive contexts; see example 5.5. The '*aggregateContext'* is also a concrete class which is basically used to aggregate abstracted context with the indirectly formatted context information; see example 5.6. Different applications may use different adder classes or define new ones.

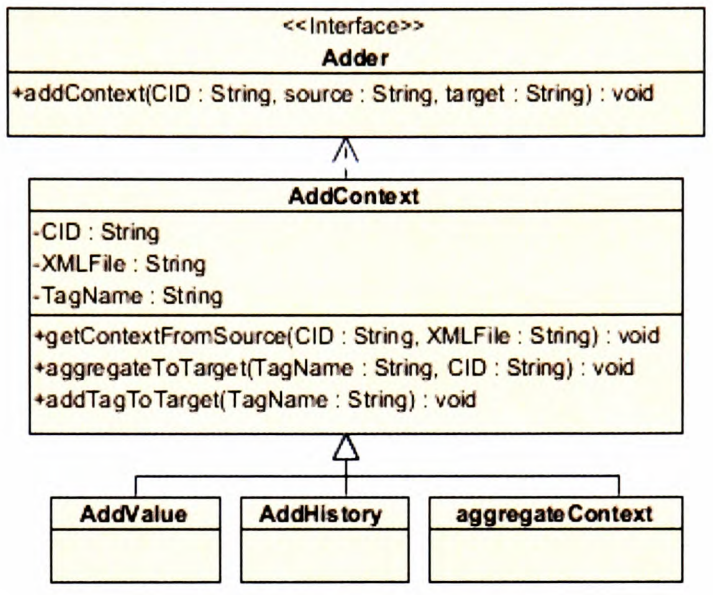


Figure 5.3: Adder filter of the GECAF framework

Example 5.4: Add value to formatted context

```
<?xml version="1.0" ?>
<Context CID="c1">
    <contextPrimitive name="Role" value=" Lecturer">
        <ID>@103</ID>
        <Time>10</Time>
        <Location>classroom 245</Location>
        <Activity>Sitting</Activity>
        <Environment>Occupied</Environment>
    </contextPrimitive>
</Context>
```

Example 5.5: Add history to formatted context

```
<?xml version="1.0" ?>
<Context CID="c1">
    <contextPrimitive name="Role" value="Lecturer">
        <ID>@103</ID>
        <Time>10</Time>
        <Location>classroom 245</Location>
        <Activity>Sitting</Activity>
        <Environment>Occupied</Environment>
    </contextPrimitive>
    <History time="9">Student</History>
</Context>
```

Example 5.6: Aggregate Context information

```
<?xml version="1.0" ?>
<Context name="TypeOfLecture">
```

```
    <context1 name="Role">
        <c1 time="10">Lecturer</c1>
    </context1>
    <context2 name="Attendance">
        <c1 time="10">0.15</c1>
    </context2>
</Context>
```

5.1.4. Interpreter

The Interpreter filter is responsible for context reasoning by making use of an XML rule language set by the application developer (to be discussed in section 5.2). Figure 5.4 shows the Interpreter's UML diagram, in which an abstract class *'InterpretContext'* implements the *'Interpreter'* interface. The *'Role'*, *'TypeOfLecture'*, *'FontSize'*, *'Fire'* and '*Temperature*' are some examples of concrete classes for instantiating interpreters' objects. The Interpreter interface defines a generic method *'ImplementRule'*, which is used to retrieve either formatted primitive contexts, or indirectly formatted contexts from source XML documents. Then according to the associated XML rule language they reason about the abstracted context, and finally save the results to a target XML document; see example 5.7.

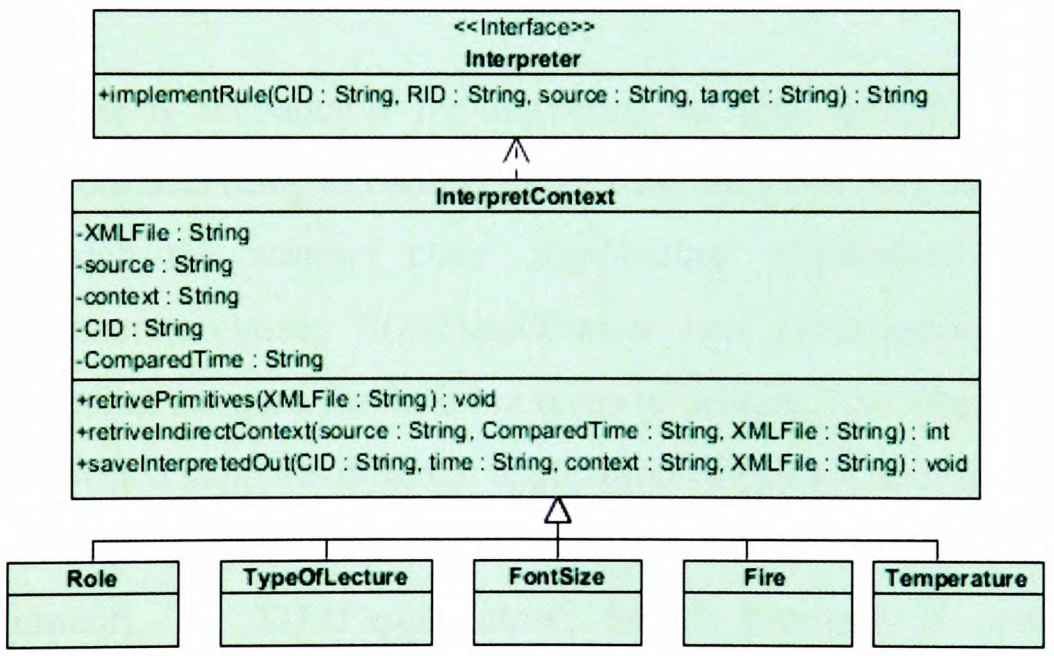


Figure 5.4: Interpreter filter of the GECAF framework

Example 5.7: The target XML document of the Interpreter

```
<?xml version="1.0" ?>
<context>
    <a1 time="10">Review</a1>
</context>
```

5.1.5. Manipulator

The manipulator filter is responsible for storing and retrieving context information into/from internal or external data sources such as a database server. The UML

diagram in Figure 5.5 shows the abstract class *'ManipulateContext'* which implements the *'Manipulator'* interface. The concrete class *'Store'* stores context history, while the concrete class *'Retrieve'* retrieves historical context information, user profile, or other information from a database server by tailoring the generic method *'Manipulate'* within the Manipulator interface. More specialised manipulator filters may be added by developers to build their applications.

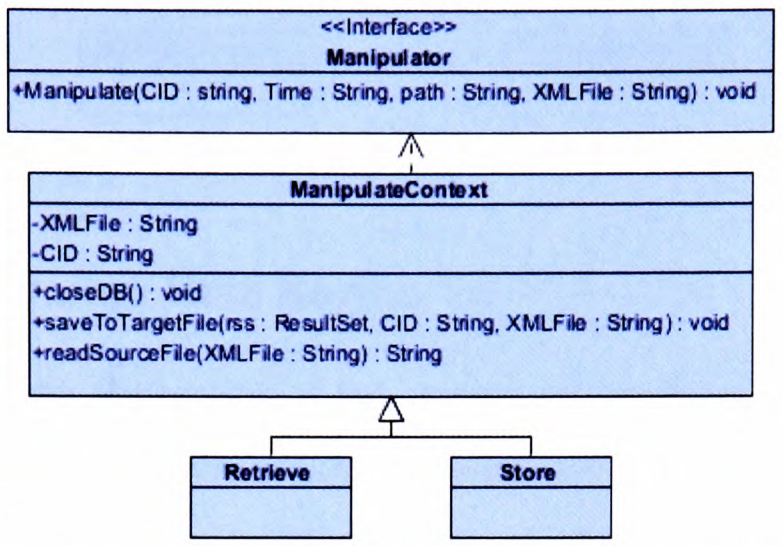


Figure 5.5: Manipulator filter of the GECAF framework

5.1.6. Outputer

The Outputer filter is accountable for delivering the final action of an application (adapt its behaviour according to context in use). Figure 5.6 shows the Outputer UML diagram, in which the abstract class *'Application'* implements the *'Outputer'* interface. The concrete classes *'SOAPapplication' and 'LocalApplication'* are some application types that are used for different types of actions. The *'Output'* Method is a generic method which supports different application categories (automatic, tagging, or presentation). These categories can be set through the application developer interface (CADL document). In *'SOAPapplication'*, SOAP protocol is used to access distributed applications on remote or local server (such as IIS PHP server). In this case the application address, the name of the remote procedure and its parameters should be specified by the CADL document. The context-aware system then runs a specified application according to the returned result.

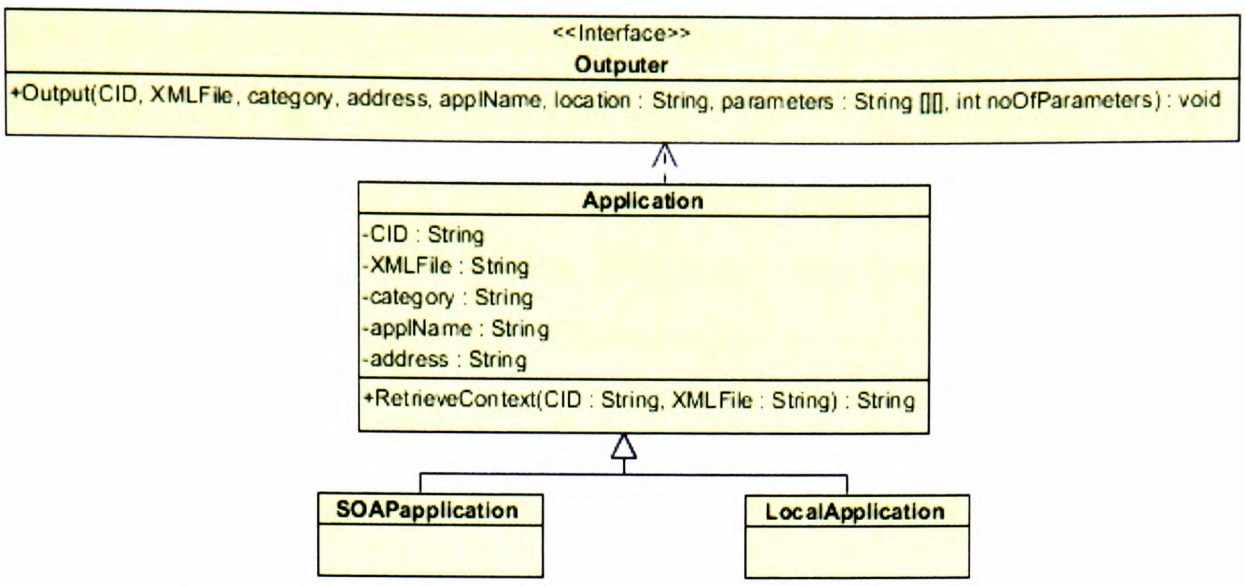


Figure 5.6: Outputer filter of the GECAF framework

5.2 The Interpretation Rule Engine

It is well known that the abstraction of the context information is implemented using the Interpreter filter. This section presents a technique for reasoning and processing context information to bring it closer to the level of abstraction. The interpreter filter uses an XML document (written by an application developer) to express rules used in the reasoning process. This XML document describes the reasoning rules (mathematical, logical or other expressions) in a structural way, so they can be retrieved in Reverse Polish Notation (RPN). The rules that the interpretation rule engine supports with descriptions of these rules are shown in Table 5.1. The XML Schema which describes the legal structure of the XML rule document is given in appendix C.

Rule	Description
AND	Logical AND (∩)
OR	Logical OR (∪)
NOT	Logical NOT
GT	Conditional operator: Greater Than (>)
GE	Conditional operator: Greater or Equal (≥)
LT	Conditional operator: Less Than (<)
LE	Conditional operator: Less or Equal (≤)
Equal	Conditional operator: equality (=)
Plus	Mathematical operator: Addition (+)
Minus	Mathematical operator: Subtraction (—)
Times	Mathematical operator: Multiplication (.)
Divide	Mathematical operator: Division (÷)
subStr	Sub of a string subStr (aString, Start, NoOfCharacters)

Table 5.1: The interpreter's rules

Example 5.8 shows the XML rule code which describes expression (given in equation 5.2) for interpreting the 'TypeOfLecture' context. This expression shows that if students' attendance in the last lecture is below or equals to 20%, person's role is 'Lecturer' which is represented by the abstracted role context (context1) and the environment is occupied then the returned value will be 'Review', otherwise 'NOReview'.

Example 5.8: The Rule expression and the XML rule document for 'TypeOfLecture' context.

$$\text{TypeOfLecture} = (\text{Role}=\text{'Lecturer'}) \cap (\text{PastAttendee} \leq 20\%) \cap (\text{Environment}=\text{Occupied}) \quad (5.2)$$

```
<?xml version="1.0" ?>
<rule xmlns:an="http://www.w3.org/2001/XMLSchema-instance"
   an:noNamespaceSchemaLocation="ruleSchema.xsd ">
   <context name="typeOfLecture" RID="2">
      <operation name="And">
         <parameter type="complex">
            <operation name="And">
               <parameter type="complex">
                  <operation name="LE">
                        <parameter type="simple" source="external" value="p3.xml" />
                        <parameter type="simple" source="fixed" value="20" />
                        <returnValue>
                              <case value="true" return="true" />
                              <case value="false" return="false" />
                        </returnValue>
                  </operation>
               </parameter>
               <parameter type="complex">
                  <operation name="equal">
                     <parameter type="simple" source="internal" value="context1" />
                     <parameter type="simple" source="fixed" value="Lecturer" />
                     <returnValue>
                        <case value="true" return="true" />
                        <case value="false" return="false" />
                     </returnValue>
                  </operation>
               </parameter>
               <returnValue>
                  <case value="true" return="true" />
                  <case value="false" return="false" />
               </returnValue>
            </operation>
         </parameter>
         <parameter type="complex">
            <operation name="equal">
               <parameter type="simple" source="external" value="G4.xml" />
               <parameter type="simple" source="fixed" value="Occupied" />
               <returnValue>
                  <case value="true" return="true" />
                  <case value="false" return="false" />
               </returnValue>
            </operation>
         </parameter>
         <returnValue>
            <case value="true" return="Review" />
            <case value="false" return="NOReview" />
         </returnValue>
      </operation>
   </context>
</rule>
```

The following steps show how to represent an expression that contains mathematical, logical, conditional operations, and special functions using the XML rule document. It typically includes a number of constraints when writing the XML rule document and is as given below: -

1. The XML rule document has a root element called *'rule'*.
2. The root element has many child *'context'* elements representing a number of rules for each abstracted context. The *'context'* element has two attributes representing context name and an associated rule identification code (RID) which is used to specify the interpreted rule.
3. Within each *'context'* element there is a hierarchy of structured elements representing the expression syntax tree of an operation; where each *'operation'* element has *'name'* attribute, number of *'parameter'* elements and *'returnValue'* element associated with the results of the operation.
4. Each parameter type can be either *'complex'* or *'simple'*, denoted by the parameter *'type'* attribute. Complex parameter contains recursive operations, while simple parameter has an associated value. The association may be achieved via mark-up within the XML document itself, or via some external means. Simple type can take either a *'fixed'* value (given in the XML document), *'external'* (read from another XML document), or *'internal'* (calculated within the interpreter filter at run time). The internal values can be either primitive contexts value (ID, time, location, activity, environment, and value) or indirect context (context1... contextn).

The mechanism used for associating and reading XML rule document within the interpreter filter is based on the Interpreter design pattern [109]. The Interpreter pattern describes how to define a grammar for simple languages, representing sentences in the language, and interpret these sentences; see appendix D. The basic idea of using the interpreter pattern in the XML rule engine is to have a class for each expression (terminal or non-terminal). The abstract syntax tree (tree representation of the abstracted expression taken from the XML rule document) is an instance of the composition pattern and is used to evaluate or interpret the expression. Figure 5.7 shows the UML diagram representing the XML rule, and the following steps show the XML rule implementation and its classes: -

1. A number of classes that are inherited from the super class *'Expression'* are used; these represent the mathematical, logical, conditional and special functions (e.g. And, Plus, LE, etc.) in addition to simple expression which could be the result of evaluating other complex expression.
2. An XML rule document which represent the expression to be interpreted.
3. An *'expressionEvaluator'* class is used to read the expression from the XML rule document and then evaluate the expression; where the retrieved expression is represented in RPN.
4. The abstract method *'Interpret( )'* within the Expression abstract class is responsible for interpreting an expression or the subparts (parameters) of an expression.

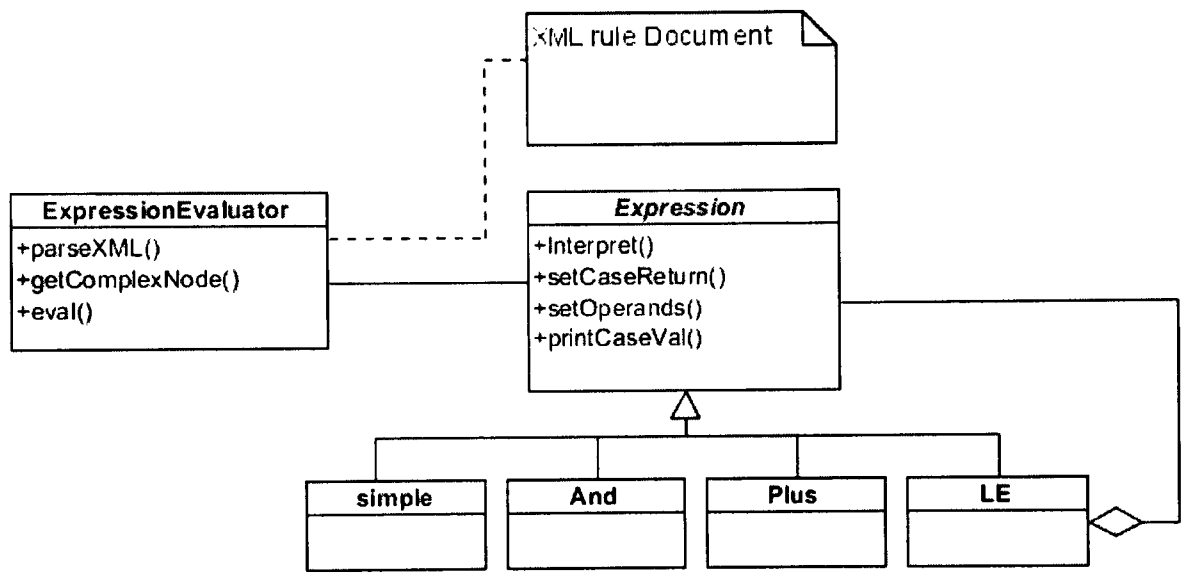


Figure 5.7: The XML rule engine using the interpreter design pattern

5.3 Deployment of The GECAF Filters

The deployment process of the GECAF filters explained in chapter 4 involves interpreting the CADL document which contains a description of the filters arrangement, source and targets of these filters, application types, etc. Throughout the deployment process, reflection and generics (java features) are employed to instantiate the filters' objects. Design pattern is also used for abstracting the creational process of the architecture's filters. All these techniques will be discussed in the following sections.

5.3.1 Instantiating The Framework Filters

Reflection and generics are two features of java that play an important role in making the framework extensible and generic; they are part of the instantiation process of the

framework filters. These concepts will be discussed in this section, and the related codes used for building the framework are given in appendix E.

Reflection is a java feature that enables programme to introspect (examine) itself, so class information could be found at run time without knowing the methods and classes at compile time [114]. In other words reflection enables the developers to specify new classes to be added to the system without having to compile the whole system. Consequently, reflection can make the software flexible as the information (e.g. class name) does not need to exist at compile time and it will become available after the deployment of the application.

The new framework uses a fixed number of abstracted classes representing the main filter types such as 'GetContext', 'FormatContext', 'InterpretContext', etc. However, concrete classes are changeable and are specified by the application designers (at run time) through the CADL code. This CADL code holds information to describe the concrete filter names (class names), and other parameters loaded during parsing to create runtime data structures. Reflection is used to instantiate objects of concrete classes during runtime while parsing the CADL document. Therefore, there is no constraint on filter's type set in the CADL code, and many applications can be easily adapted for customised solutions. In this case reflection is used to create new instances of concrete classes. This is accomplished by calling the '*newInstance*' method to instantiate an object returned by '*forName*' whose class name (retrieved from CADL) is not known at compile time. Once the concrete object is instantiated, the associated method (e.g. *getContext(..)*) of an object can be called at run time; see example E.1 (appendix E).

Generics is another important feature of java; it is the ability to write generic code which is independent of an application. Throughout the implementation, this built-in feature of java has been used; so one piece of code ('Creator' class) is employed to build a generic creator for the filters which is irrespective of how many and what type of filters are instantiated. Therefore, different filter types can be specified at run time after retrieving their names from the CADL code; see example E.2 (appendix E).

5.3.2 Using Creational Pattern for Abstracting The Creation Process

Creational design patterns abstract the instantiation process by making the system independent of how its object is created, composed, and represented. A class

creational pattern uses inheritance to vary the class that is instantiated, whereas an object creational pattern will delegate the instantiation to another object [109]. A creational design pattern has been used to build flexible, customisable and reusable object-oriented design and to avoid redesign or at least minimises it. The following section presents a description of the use of a creational design pattern (Factory method) in the creation of the architecture's filters. However, there are other methods which can be used for the creation of objects such as the Abstract factory, the Builder, and the Prototype [82].

5.3.2.1 Factory Method Design Pattern

The Factory Method [109] defines an interface for creating an object, but let the subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses. It allows the creation of an application framework without the need to bind application-specific classes into the code. By providing hooks for subclasses, it allows the extension of the original object model and the creation of new classes that fit into the framework. Hence, the factory method pattern supports coding to an interface to handle future change.

Figure 5.8 shows the Factory Method UML diagram in which the *'Creator'* hides the creation and instantiation of the *'Product'* from the client. This is an advantage to the client as they are now insulated from future modifications, so the *'Creator'* will look after all of their creation needs and allowing decoupling. Furthermore, as the *'Creator'* and the *'Product'* conform to an interface known by the client, the client doesn't need to know about the concrete implementations of either. As shown in Figure 5.8 the *'Creator'* provides an interface (factoryMethod), which is used by the subclasses (ConcreateCreator) to create the objects. Other methods in the abstract Creator can operate on the created Products.

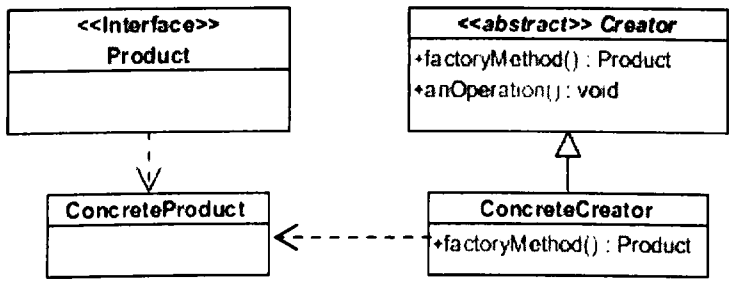


Figure 5.8: Factory method design pattern

Figure 5.9 shows a class diagram of the framework. In this diagram, the *'Creator'* is an abstract class provides an interface (*'create'* method) to create the objects. The

concrete classes *'GetterCreator'*, *'FormatterCreator'*, *'InterpreterCreator'*, *'AdderCreator'*, *'ApplicationCreator'*, and *'RepositoryManipulatorCreator'* are used to instantiate the architecture classes. Each of these classes uses reflection to instantiate the concrete filters of the architecture. For example, the *'GetterCreator'* is used to create objects of 'ID', 'Time', 'Location', 'Activity', 'Environment', and 'Value'; see example E.3 (appendix E).

5.4 Event Triggered System Initiation

"An event can be defined as a signal to the program that something has happened". [115] Event sources can be external action such as detecting the presence of a person or internal such as a timer lapse. The program can choose to ignore or respond to an event. Events are processed by registered listener objects. In order to be used as a listener, a class must implement a listener interface. Therefore, when an event occurs, the object on which the event occurred (source) sends the event to all registered listeners interested in that event by invoking the appropriate method of the listener object. An event object is passed as an argument to the method. The Listener object can then take an action based on that event or can ignore it.The System Manager and Scheduler of the architecture (described in section 4.3.5) is responsible for managing and sequencing the context-aware system events which are fired by various types of sensors (refer to Figure 4.5). The following points describe the mechanism of the System Manager and Scheduler: -

- The class 'Manager' which represents the system manager creates and starts the events' listener. It also instantiates an object of a queue. Sensors' events (e.g. RFID reader and other sensors) keep a combination of their values attached with an event identification code (EID) in the queue. The event handler reads and schedules these events which are processed in the order they are received. Multiple processes serve the queue, and each process can progress at a different speed. Sensors' events which are accountable for delivering primitive context information (4W1H) to the queue might be processed at a point of time different from timer events that read queue's head every x seconds. Therefore, there is a difference between the time that one event is taken off the queue head and the others accumulated at the queue end.

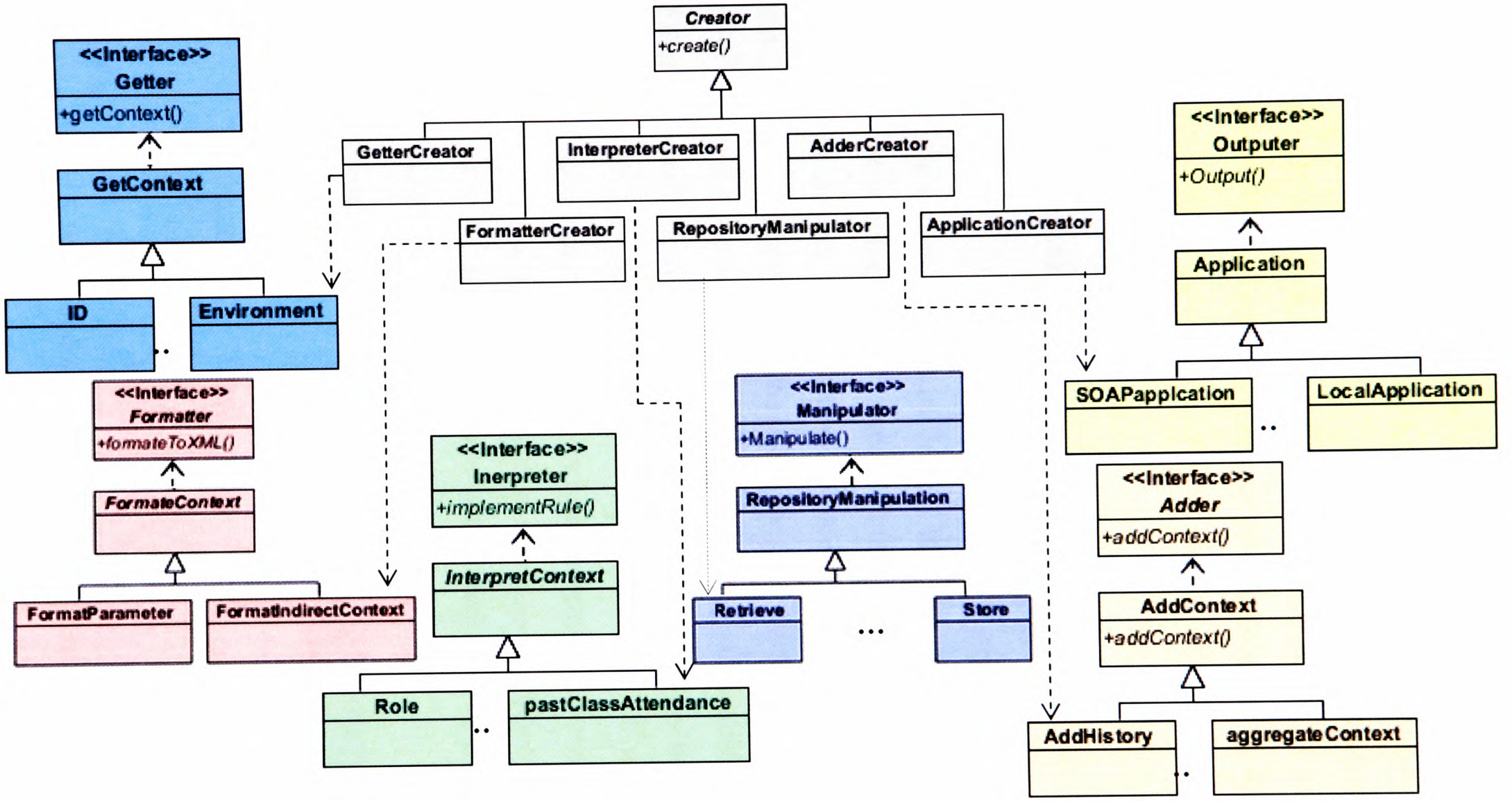


Figure 5.9: A simplified UML diagram for the Generic Framework

- When the timer fires an event, the action listener reads the queue's head. Then, if entries were found, it will be saved in six XML documents (ID, Time, Location, Activity, Environment, and Value) and the manager continues listening to events triggered by the sensors. Thereafter, the system manager reads the application developer CADL code, makes a mapping between EID and CID, and continues in the Pipe-and-Filter architecture stream. As the context-aware system proceeds in its processes (filters), the abstracted context value will be found to make changes to the environment accordingly.

5.5 The Conceptual Process for Using The Pipe-and-Filter Framework

A Framework is a system that sets the logical flow of the applications and gives a general way for structuring them. A framework to design and develop context-aware systems has been created; so application developers who are aware of context-aware systems' main functional blocks (filters) can use it by following a set of instructions to realise their own systems. They can also use the employed architectural style (Pipe-and-Filter) to benefit from all its advantages, modularity, abstraction, low coupling, and high cohesion (all the properties that the style aims at). The framework also achieves faster development by using proven patterns and reusable components (filters). The framework provides a pool of ready-to-use filters (see component diagram in Figure 5.10), and enables users to build new ones by inheriting from the interfaces and methods and adding them to the pool.

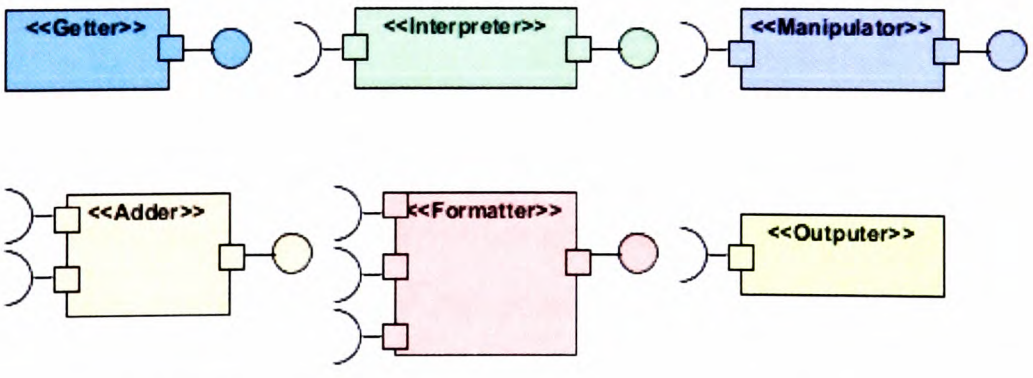


Figure 5.10: Component diagram shows general filter types

The following is a list of instructions that can help application developers to construct and customise their own systems: -

Stage 1. Define the meaningful (abstracted) context used in the design process e.g. context is 'Role', then specifies the elements of the context model to represent the useful context, e.g. Context= f (ID, Time, Location…).

Stage 2. Identify the major filter types used for building a context-aware system, such as Getters (e.g. ID, Time, Location, etc.), Formatter (e.g. FormatParameter, and FormatIndirectContext), Interpreter (e.g. Role, LectureType), etc. Then draw the filter diagram. If new filters are required, create them by implementing the interfaces or inheriting from the abstracted classes in the framework.

Stage 3. Write the CADL code (XML deployment language) which sets up the flow of the architecture, establishes which communicates with what, and define the flow of information and actions. This CADL code which defines the structure of an application should conform to the CADL's XML schema (see appendix B).

Stage 4. Write the XML rule code and associate a RID with each context. The XML rule code should conform to the XML schema given in appendix C.

Stage 5. Design an application (for example 'Smart Classroom' website), then select and build the required hardware and the associated interface.

The following points represent rules that should be followed when writing the CADL code; the skeleton of the code is given in example 4.4: -

1. The CADL document has a root element called '*Filter*'.
2. The root element has many children ('*process*' elements) which represent different filter types. This process has an associated context identification code (CID) to specify the interpreted context and an optional '*category*' attribute used with the application filter.
3. Within each '*process*' element there is a hierarchy of structured elements representing the concrete class, source and target, and other elements. The '*class*' element has different attributes depending on the concrete class in use. These attributes could be '*name*' representing class name, '*context*' representing context name, '*RID*' representing Rule identification code, or '*address*' representing application address.
4. Each '*class*' element could have a number of children: '*source*', '*target*', and/or a number of '*Element*' children used by the application filter.

5. The '*source*' element has a text content that represents the source file name; it also has two optional attributes, these are '*type*' attribute that specifies file type (e.g. xml, html, etc.) and '*no*' attribute that represents the source file number.
6. The '*target*' element has a text content that represents the target file name, and an optional attribute '*type*' specifying the file type.
7. The '*Element*' is a generic element which can be of any number; each element could have a number of attributes like '*name*', '*value*', '*type*', and '*fileType*'.
8. The CADL code has a child element '*events*'. This element includes different events and a mapping between their RID (Rule identification code) and CID.

Figure 5.11 shows a component diagram with filter arrangement representing person's '*Role*' context; the CADL code is given in appendix F.

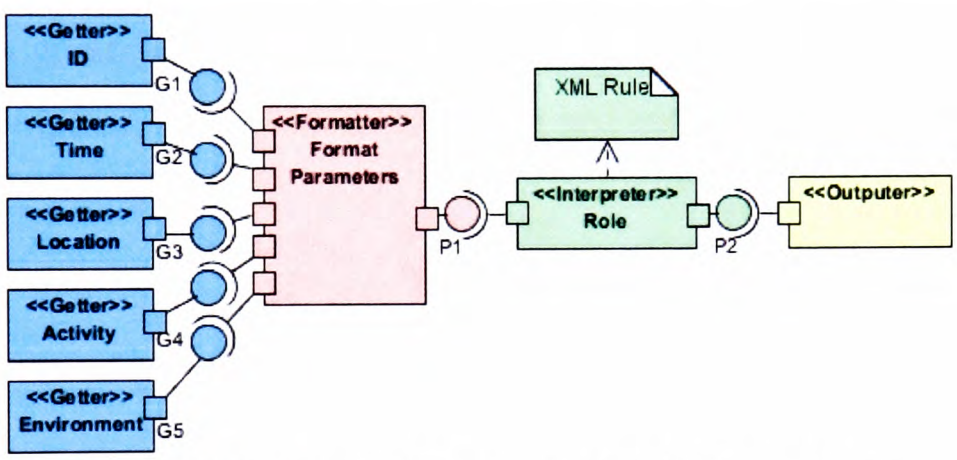


Figure 5.11: Filter diagram for the Role context

5.6 Case Study in a Smart Classroom Scenario

To conceive the smart classroom scenario (refer to section 4.4), networked computers, RFID tags and readers, and simulated sensors have been utilised; where, we have specifically implemented an RFID in the case study to represent the Getter that reads the student ID from the environment. A number of meaningful contexts which are adopted in the scenario are also represented using the framework components. Using the design instructions given in section 5.5 the following steps should be followed: -

Stage 1. Specify the required (primitive and the meaningful) contexts. For example, primitive contexts are: 'ID', 'Time', 'Location' and 'Environment'. Abstracted contexts are: 'Role', 'Student attendance history', 'Lecture type' and 'Font size'. 'Lecture type' is an indirect context that can be inferred using three contexts: Person's Role (Lecturer), Students' Attendance (e.g. 20%), and Environment (Occupied). Person's Role is a function of the context primitives ID, Time, and

Location. Past students' attendance can be inferred from students' attendance history. Classification to context information for the classroom environment is given in Figure 5.12.

Stage 2. To realise these contexts the required filters have to be identified as given below. Then we use the filter diagram to build the architecture, see Figure 5.13.

1. Getters are used to acquire the primitive contexts: ID, Time, Location, and Environment.
2. Formatters are of two types: format the acquired parameters, and the indirect context (attendance history, Role and Environment).
3. Interpreters used are: person's role, lecture type and font size.
4. Manipulator is used to retrieve attendance history from data repository.
5. Outputers are accountable for taking an action by displaying a classroom web page with a given lecture. In this case application category has to be specified with each Outputer, which could be tagging and automatic in this case study.

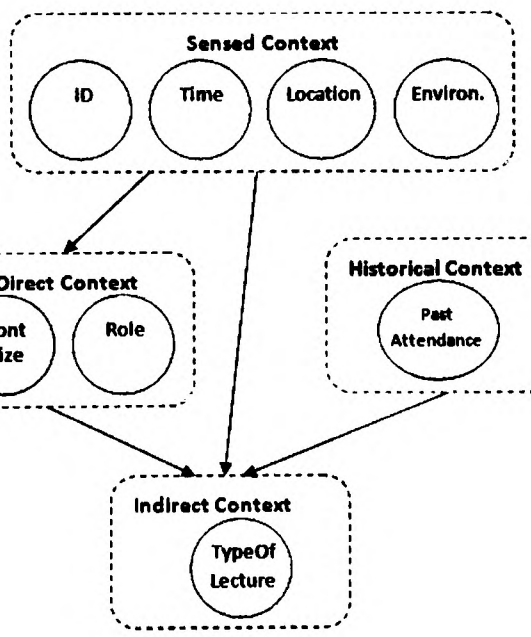


Figure 5.12: Classification to context information in classroom environment

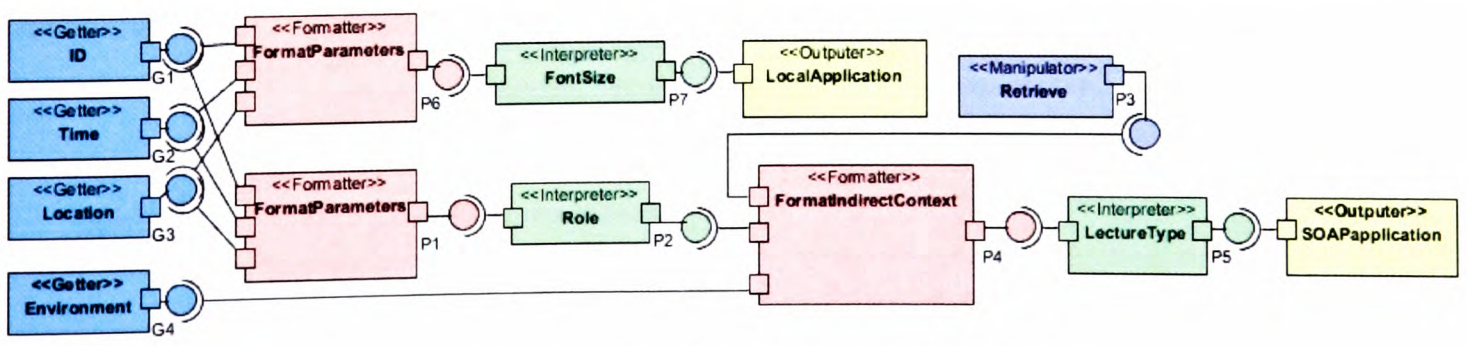


Figure 5.13: Filter diagram for the lecture type and font size contexts

Stage 3. A CADL code is written to deploy the architecture filters which conform to the XML Schema set in the design, refer to appendix G.

Stage 4. This stage is concerned with writing the XML rules for the interpretation process including all the abstracted contexts specified in stage 1 (refer to appendix H); the XML rule code should conform to the rule XML Schema. An example to the rule expression of 'TypeOfLecture' context is given by: -

TypeOfLecture = (Role = 'Lecturer') ∩ (PastAttendance≥'20%') ∩ (Environment = 'Occupied')

Stage 5. In this stage and for demonstration purposes, a website has been designed as part of the smart classroom application using HTML code and PHP; refer to appendix I. Then, the required sensors have been selected (e.g. RFID reader and tags, simulated sensors) with the associated software interface. All remote procedures for the SOAP application are also written using SOAP and PHP. Also, a database server is installed to store all the required information to this application. Figure 5.14 shows a smart classroom web site page. Figure 5.15 shows a visualisation of classroom with the designed web page; where the lecture font size changes with the existence of the special need student 'Bill' as given in the scenario (refer to section 4.4). The Figure also illustrates the RFID functionality of the system in the smart classroom scenario.

Figure 5.14: Smart classroom web page

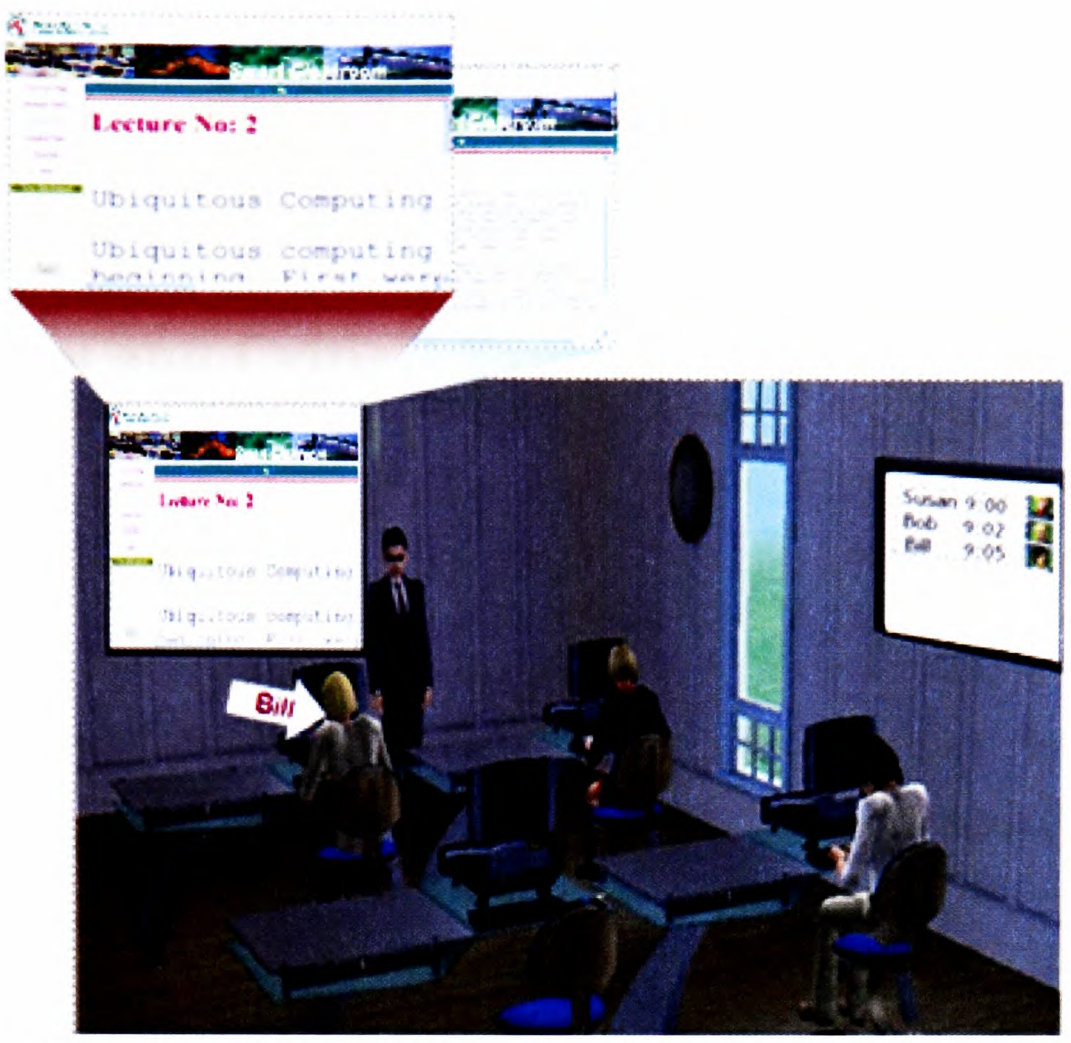


Figure 5.15: A visualisation of the smart classroom with the designed website

5.7 Summary

This chapter presented the implementation of the new framework, which adopts the Pipe-and-Filter architecture style. First, the framework filters were created and then the factory method was employed as a creational design pattern to create the filter's objects. Afterwards, the System Manager and Scheduler building block is built to schedule and capture the environmental contexts and also to manage the whole system operation. Event driven programming has also been used for the users' interaction. Finally, a set of instructions to customise the architecture filters and a smart classroom case study are given.

Chapter 6

Validation and Critical Evaluation

Since context-aware systems are still developing and there is no standard prototype to build them; there should be some criteria to evaluate and test the new approaches in order to show their applicability to different scenarios. To evaluate the GECAF frameworks, a set of heuristics have been utilised which test systems' performance. Then different techniques have been used to validate the new system. Firstly, we have used a quantitative method through literature survey and comparison to identify commonality and differences between existing context-aware systems' design. Secondly, a case study and scenario based experiments were used to demonstrate the use of the framework in the implementation and testing of various context-aware systems with different scenarios. These techniques will be discussed in this chapter.

6.1 Criteria for Evaluating the GECAF Framework

The evaluation criteria represent the systems features that can be used to make a decision on the performance of a given system. It also shows the limitations and drawbacks of the system design. Some research was needed in order to decide what criteria can be used for evaluating context-aware systems. To evaluate context-aware systems, Oh et al. [87] adopted an approach using a set of heuristics, these are: separation of concerns, flexibility and openness, scalability, support for reuse, support during development, debugging and deployment, explanations and accountability, security and privacy, reliability, match between context-aware system and real world, manual override, and reconfiguration and management. Another approach that studies the requirements of context-aware systems is presented by Fahy et al. [52]. They gave a number of requirements for designing context-aware systems. These requirements are: supporting large number of applications, provision to context history, support for interpretation, higher level of context abstraction, event based in detecting context

changes, and extensibility. According to Fahy, context-aware systems should also support transparent use of distributed context sources, and the separation between context use and application. Again, Choi [66] gave ten requirements for the context-aware systems, these are: distributed system, heterogeneity, mobility, dynamic adaptability, context modelling, extensibility, scalability, configurability, portability, and support for restructuring. Another study which considered the requirements of context-aware system is made by Bratskas et al. [116]. These requirements are: modularity/Plug-ability, privacy/security, decision-making, tolerance to failure, support distribution, technology used for context modelling and communication, and mobility.

As a conclusion, we have adopted the evaluation criteria and the requirements of existing works, then create a modified one to evaluate the new framework. The following heuristics summarise and group the reviewed work: -

1. Reusability: Context-aware systems have reusable building blocks that support data acquisition, aggregation, context modelling, reasoning, storing/retrieving, taking action, and managing. These modules are independent (support separation of concern) and can be used in different scenarios.
2. Support for Context history: Context-aware systems should consider context history as an important term in context modelling.
3. Scalability: The architecture should be scalable, i.e. applicable in small systems and also in large scale applications with large number of users.
4. Extensibility: As context-aware systems are not similar, their architecture should be extensible, so new building blocks can either be added or extended from the abstracted ones.
5. Adaptability: The architecture should support adaptability, i.e. easy to reconfigure with new changes in context during run time.
6. Event Based: The system should be event based to avoid delay of polling operation.
7. Reliability: The context-aware system must have tolerance to failure and have the ability to recover.

According to the above heuristics the framework can be evaluated and as given in the following sections.

6.1.1 Framework Features

The GECAF framework has a number of features as listed below: -

1. The framework shows a new way of building context-aware systems. The framework has support for decoupling and reusability in terms of context reuse and component reuse, where the framework utilises a number of reusable components that can be employed in building various context-aware systems. It is also customisable and support re-configurability, where the framework components can be used in different scenarios as the Pipe-and-Filter style is employed.
2. The GECAF framework is extensible and can be evolved, where new concrete building blocks (concrete filters) can be added to the framework. These concrete filters can use all the features of the abstracted ones. Moreover, it is considered generic as the abstracted filter types represent the usual building blocks of context-aware systems.
3. The framework considers context history as important information and the effective context, which is used to enrich the context information and predict user's behaviour in current and future state. Therefore, it is incorporated as part of the proposed model.
4. The GECAF framework uses both structured and object oriented approaches for context modelling. Therefore using multiple modelling techniques can overcome the limitations in each of these techniques.
5. The employed architecture is reconfigurable during run time; these changes are made according to the interpretation rules which can be customised or set by the application designers. A reconfigurable architecture improves the system's extensibility by allowing the system to change at run time.
6. The framework is event driven, where the System Manager (a main building block of the framework) is event driven, which can be triggered by the environmental sensors.
7. The framework supports context information enrichment as it has the aggregator and the Manipulator building blocks to retrieve context information from different resources to enrich the context information.
8. The system is reliable as it uses a generic rule mechanism for rule processing which gives the same reasoning outcome if the same context occurs.

6.2 Validating The GECAF Framework

The validation of the new approach involves analysis of various existing context-aware systems, modelling, and experimentation by case study. In this research we adopted two methods to validate the generic nature of the proposed architecture which will be discussed in the following sections. Firstly, a quantitative method has been used; it reviews and analyses various context-aware systems to study and examine their design and behaviours. Secondly, the GECAF framework is used in implementing various case studies or realising existing systems by employing the framework building blocks.

6.2.1 Validation Using a Quantitative Method

This method involves reviewing existing context-aware systems, and then thoroughly examines the requirements of these systems to analyse their functionalities. From the analysis given in chapter 3, different items are classified and arranged, see Table 6.1. The table demonstrates an idea about the architectural styles of each system. It also shows the main building blocks that constitute various context-aware systems (sensing, modelling, reasoning, storing/retrieving, aggregating, application, and managing services). It is to mention that components given in the comparison were found to be the only components supported by various context-aware systems which already exist.

The table shows that existing context-aware systems use some or all of these functional components, which could either be reusable or tightly coupled (implicitly involved) with other components. From this comparison an analyses for the necessity of using all these functional components to build a generic context-aware system was carried out, as all these components can be put in various arrangements. The table also shows that existing systems use different modelling techniques to represent and relate the context information, where ontology based is the most used one. In reference to section 4.2, about the context history we found that all the existing work considers history as an important category of context information, therefore it was considered and used as part of the model. The histogram in Figure 6.1 illustrates the main components for various context-aware systems with the number of systems employing these components. Figure 6.2 shows a pie chart presenting the contribution of each style to the total architectural styles used by the reviewed context-aware systems. The

chart shows that the layered architecture is the dominant one with 35% of the systems using this style. However, with this style not all systems are easily built [82] and it is not easy to find the right level of abstraction, so it is difficult to be used with complex context-aware systems. Therefore, a generic framework cannot be built using this architecture style, and as explained before.

Context-Aware Systems	Architecture									Context history	Reuse	Context Model
	Style	Functional Components										
		Sensing	Reasoning	Aggregation	Context Modelling	Application	Managing & Services	Data Manipulation	Context Repository			
TEA-System	Layered	√	√	√		√				√	Context Component	Object oriented
Toolkit	Peer-to-Peer *	√	√	√			√		√	√	Context Component	Attribute value tuples
CMF	Blackboard	√			√	√	√	√	√	√	Context	Ontology RDF
CoBrA	Agent based	√	√				√	√	√	√	Context Component	OWL
SOCAM	Server based *	√	√			√	√		√	√	Context Component	OWL
CASS	Server based	√	√					√	√	√	Context Component	Relational data model
CORTEX	Sentient object model	√	√	√		√		√		√	Context Component	Relational data model
Ubi-UCAM	Distributed * Framework	√	√	√	√	√	√	√	√	√	Context Component	Application oriented, unified model
Context Stack	Layered	√	√	√	√				√	√	Context	OWL
RWM	Layered	√	√		√		√		√	√	Context	Application oriented, RWM
Multi Agent Architecture	Multi Agent *	√	√			√			√	√	Context Component	Logic base
WCAM	MVC	√	√	√		√	√			√	Context Component	Application oriented
CALA	Centralised	√	√		√		√		√	√	Context Component	OWL & RDF
General Architecture	Layered	√	√		√	√	√		√	√	Context Component	Data structure
Multi Agent Service Reassembling Architecture	Multi Agent	√	√		√		√		√	√	Context	Ontology
CADBA	Layered	√	√	√	√	√	√	√	√	√	Context	OWL
A-Cobra	Multi Agent *	√	√		√	√	√	√	√	√	Context Component	OWL

Table 6.1: Architectural comparison for different context-aware systems

Where * refers to multi-style architecture with the predominant style.

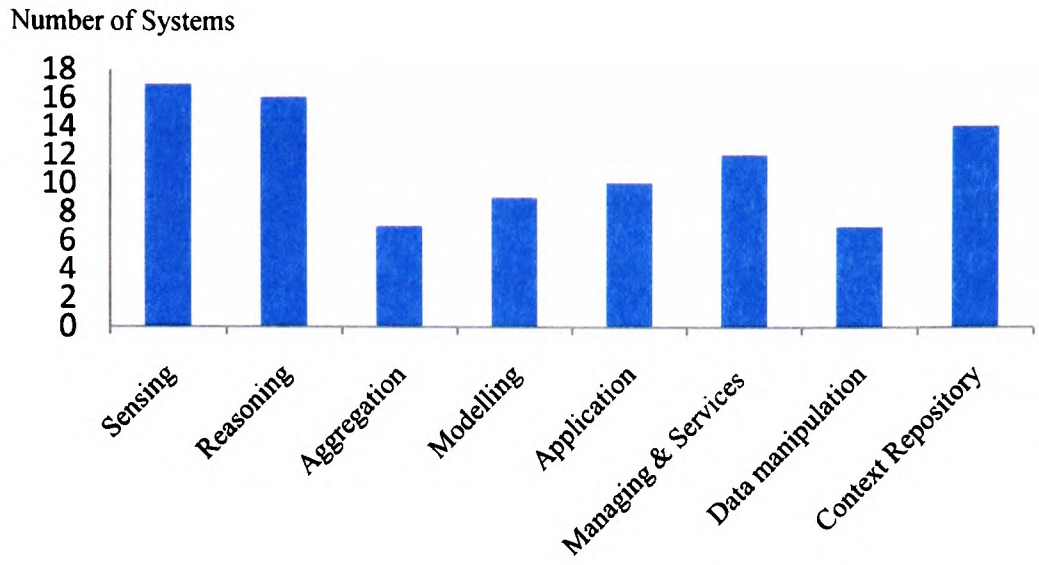


Figure 6.1: Architecture components used by the context-aware systems

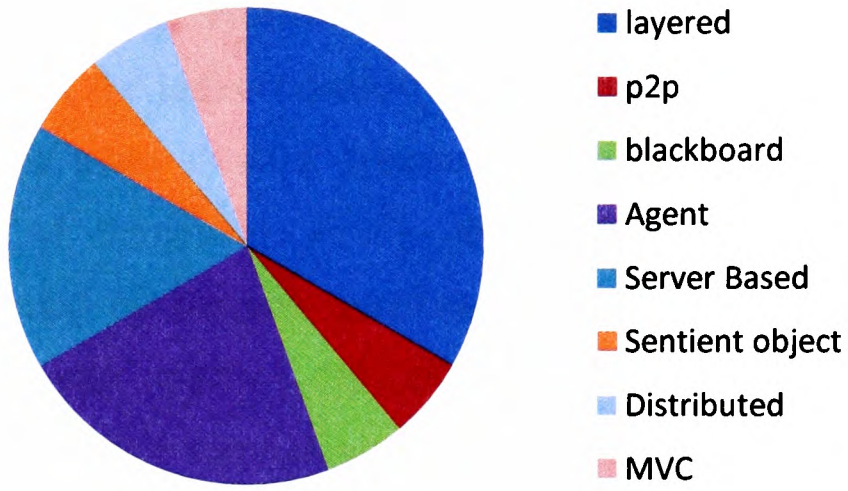


Figure 6.2: Architecture styles of the reviewed context-aware systems

To match up the functionality of the GECAF framework with the existing context-aware systems, the following points are concluded: -

1. The abstracted filters of the new framework represent the reusable building blocks which have common functionalities with the above systems. As shown in Table 6.1 all context-aware systems employ the sensing building block which gathers context information from the environment. Likewise our framework uses 'Getters' to gather primitive context information from the environment. For example, an ID getter can be used to acquire user ID, and a Value getter can be used to acquire discrete context information (e.g. Temperature, Voltage, etc). The table also shows that the majority of the existing systems utilise a reasoning component which is essential for abstracting the context information. In the GECAF framework this component is known as the 'Interpreter' filter which

employs a rule mechanism for reasoning. The interpretation rules can be set by the application developers through the XML rule code (whose schema is defined in appendix C). As shown in Table 6.1, some systems use the aggregators to enrich the context-aware system; however, this functionality is implicitly involved in other systems. In the GECAF framework this component is known as the 'Adder' filter. It is used to aggregate the context information to enrich the context-aware system by adding historical context or other information retrieved from internal or external sources using the 'Manipulator' filter; where the later component is part of some architectures functionality. Context representation is considered by all existing systems. However, 50% of the existing systems use modelling components while others considered it as part of the reasoning components (especially with systems using ontology). To compare, the modelling component is known as the 'Formatter' filter; it is used to represent and relate the primitive contexts or the indirect context information. Application component is again employed by many existing systems, which is known as the 'Outputer' filter in the GECAF framework. Finally, the functionality of management and services is supported by the new framework through the use of the 'System Manager and Scheduler' and the CADL code.

2. The GECAF framework's components can be used in different arrangements to build various systems, as it employs the Pipe-and-Filter style to arrange these components. This comparison reveals that the framework employ all the existing systems' functionalities, so various systems can be built with these building blocks. Therefore, we believe that the framework is generic. It is also extensible, because we can either use the existing concrete filters or create new ones by inheriting the features and properties of the abstracted filters.
3. The comparison also shows that context information can be reused and shared. The developed context model considers context history as an important feature, which can be shared by many contexts. Context history is important for the enrichment and abstraction of context information which can be used as a knowledge base.
4. Table 6.1 shows that different context models have been employed by various systems, with the majority using the ontological one. Concerning the new system, the context model employs a structured technique to represent the context

information. Object oriented method is also used to represent context information in different level of abstraction. We believe that multiple techniques are better to be used in order to overcome the weaknesses of each of these methods.

In brief, and according to the above discussion this approach is viable as it is generic and extensible. Therefore, the framework helps application designers to build their own systems without the need to go into every detail about the specifications of the design.

6.2.2 Validation Using Case Study

In an attempt to verify the finding of this study and examine the applicability of the proposed framework, the new frameworks' building blocks have been employed to rebuild some existing systems. Two well known systems have been selected; these are the context toolkit [100] and SOCAM [44]. The idea of choosing these systems is due to being widely cited.

6.2.2.1 Case Study for The Context Toolkit

The context toolkit has five main components: BaseObject (for communicating context with application), widgets, interpreters, aggregators, and discoverers. The context toolkit relies on the concept of the context widget (a software component for data handling that mediates between the acquired environmental data and its use) which can be shared by many applications. It acquires and abstracts context information from the environment using sensors, and also provides services to the applications. Each widget has a 'state' (attributes queried by applications) and a set of behaviours or 'callbacks' to make changes in the environment.

Using the new framework, the 'Presence' widget which senses the presence of a person in a room can be built. It provides callback notifications to the application when a new person arrives, or when a person leaves. In this case the attributes of location, time, and identity are used; where location attribute is constant. A 'Meeting' widget is another example that is built on top of the 'Presence' widget, where meeting is represented by the presence of more than one person in a room. Equations 6.1 and 6.2 present the interpreting expressions for Presence and Meeting contexts; whereas Figure 6.1 and Figure 6.3 show the simplified representation of both Presence and Meeting contexts using the Pipe-and-Filter architecture components. The Presence

context is interpreted using a combination of the primitive contexts (location, time, and identity), while the Meeting context is interpreted using indirectly derived contexts. The XML rules used by the interpreters are given in appendix J.

$$\text{Presence} = (\text{ID} = \text{an\_ID}) \cap (\text{Time} = \text{a\_Time}) \cap (\text{Location} = \text{a\_room}) \quad (6.1)$$

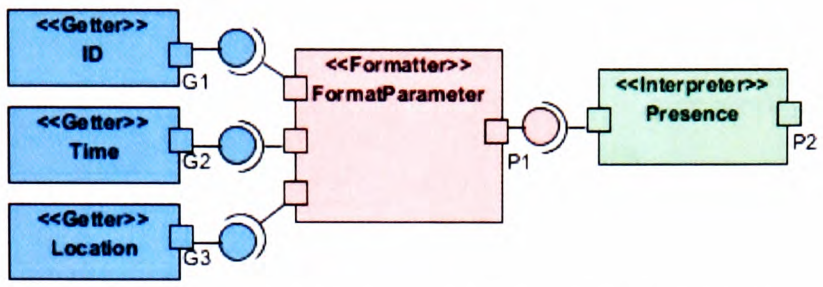


Figure 6.3: Filter diagram representing Presence context

The Meeting context is built on top of the presence context, by using an aggregator to aggregate the presence of people. The XML rule for interpreting Presence and Meeting contexts is given in appendix J.

$$\text{Meeting} = (\text{Presence} = \text{YPresence}) \cap (\text{attendeeNo} > 1) \quad (6.2)$$

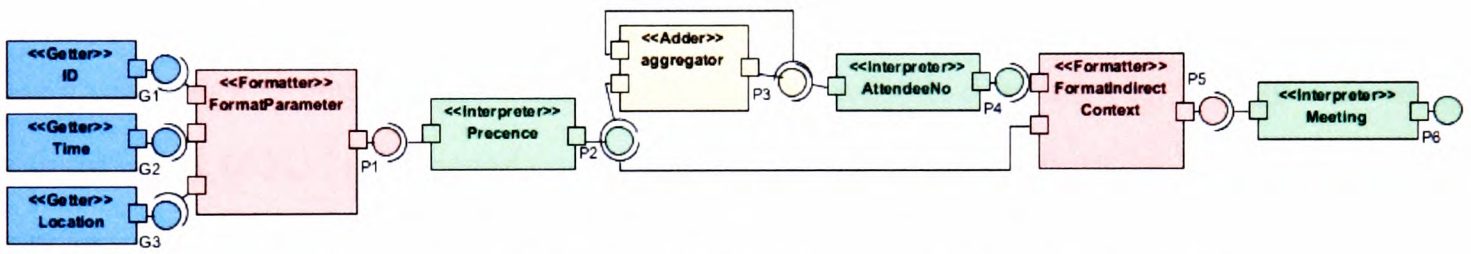


Figure 6.4: Filter diagram representing Meeting context

The In/Out board and the context-aware Mailing List are two applications for the context toolkit, see details in appendix K. According to the Pipe-and-Filter architecture the In/Out board applications can be implemented using the following filters: Getters (ID, Time, Location), Formatter (FormatParameter), Manipulators (Store, Retrieve), Adders (AddValue, AddHistory), Interpreters (Presence and In/Out status), and Outputer (SOAP_Application), see Figure 6.5. In this diagram the Getters acquire ID, Time and Location information from the environment as the occupant is detected. The Occupant's name is retrieved from a database server using SOAP application, where application category is tagging. ID and Time parameters are formatted, then the In/Out status history is retrieved and added to enrich these information. These pieces of context information are interpreted to find the In/Out status. Accordingly, the In/Out board information, or the mail list information are changed according to the status and occupant name, where application category is

automatic. The rule expression for the In/Out status interpreter is given in equation (6.3). The CADL code for In/Out Board is given in appendix L.

In/Out status = (Presence ≠ Presence_ history) . . . . . (6.3)

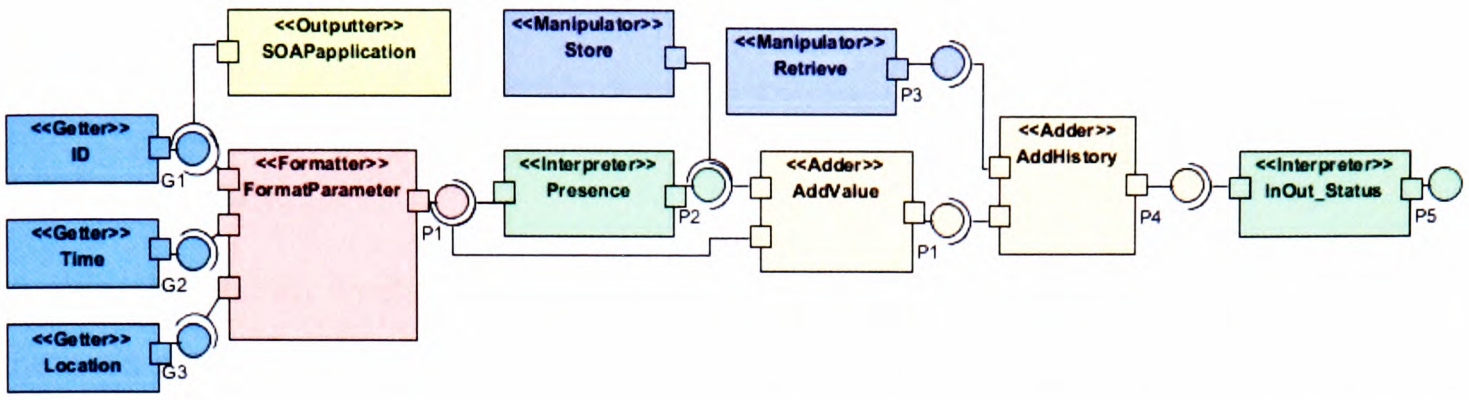


Figure 6.5: In/Out Board and Mailing list implementation using the new framework

6.2.2.2 Case Study for The SOCAM System

SOCAM architecture consists of: context providers, context interpreter, context-aware services, and service locating services. SOCAM developed a number of context providers to build an application. For example, an indoor location provider is used to track person's location at room granularity, which depends on RFID tag and transponders [62]. Then, based on number of persons, their identities, time, and user's context, they can derive the room activities (such as breakfast, lunch, dinner, birthdayParty). The representation of '*LocatedIn*' and '*hasActivity*' using the new framework are given in Figures 6.6, and 6.7 respectively. The rule expression for '*LocatedIn*' depends on ID and location (see equation 6.3), while the rule expression for '*hasActivity*' depends on time, location, activity and the number of attendee (see equation 6.4). The CADL code for *'hasActivity'* is given in appendix M. It is to mention that the details of SOCAM architecture implementations are not available. Appendix N illustrates the XML rule for interpreting *'hasActivity'* context.

LocatedIn = (ID = an_ID) ∩ (Location = a_room) . . . . . (6.3)

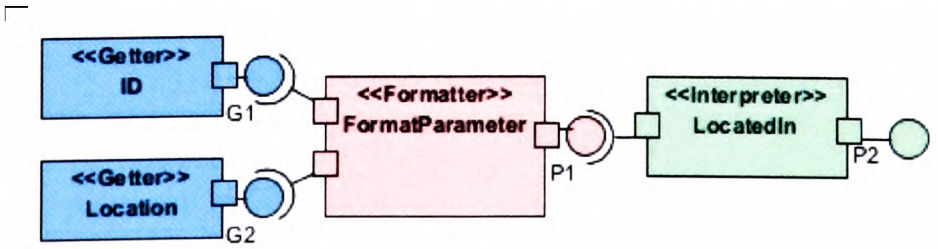


Figure 6.6: Filter diagram representing LocatedIn context

hasActivity=(Time=a_Time)∩(Activity=an_Activity)∩(Location=a_Room)∩(AttendeeNo>N). . .(6.4)

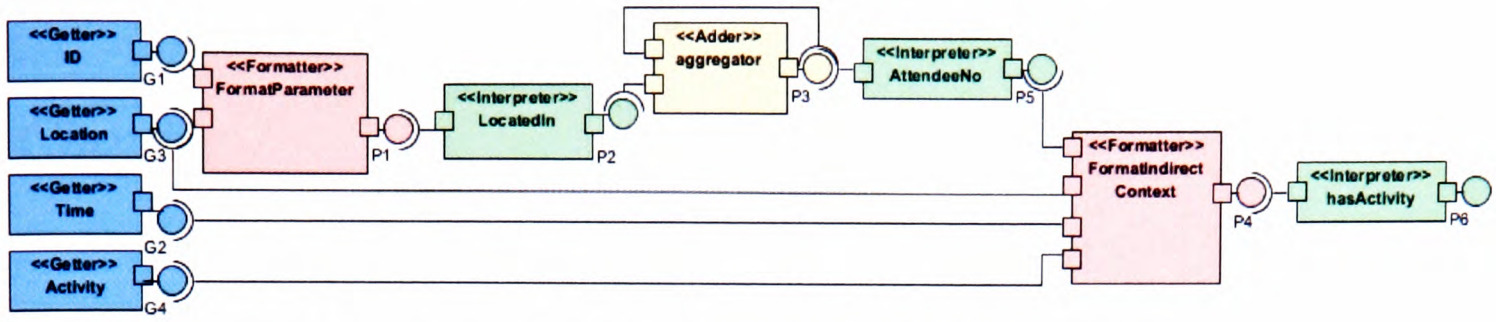


Figure 6.7: Filter diagram representing hasActivity

6.3 Summary

In this chapter an evaluation technique is presented using a set of heuristics. Two methods are employed to validate the findings, these are: quantitative based and case study. In the first method a literature survey is conducted then analysed and compared with the new framework. The second method (using case study) includes implementing different scenarios and case studies of the existing and well known systems. The implementation of existing systems can justify the applicability of the proposed design in terms of the main functionality. However, this technique only validates the applicability of the new framework. It was not possible to conduct a comparison between the performance of the two approaches as it was not possible to acquire the original implementation of the toolkit and SOCAM systems. Therefore, it was found that quantitative method (comparison by literature survey) is the most helpful for the justification.

Chapter 7

Conclusions and Future Work

This thesis has described an extensible and generic framework through adopting an architecture style that supports designing, and developing context-aware applications. A comprehensive conclusion to the study along with a discussion of limitations and future research is presented in the following sections.

7.1 Conclusions

The major conclusions are summarised in this section. We believe that the new framework is extensible and generic; it simplifies system integration, improves maintainability, adaptability, reusability, support decoupling of components, and reduces the development effort. This enables application designers to create a variety of context-aware systems. In my opinion, the aforementioned features are based on the following grounds: -

1. The framework uses the Pipe-and-Filter architecture style, which employs a number of reusable and independent filters. These filters can be used in different arrangements to build a variety of applications; therefore it is adaptable and reconfigurable. This style also enables filter reuse in the design, and hence information reuse.
2. The main filter types of the framework represent the usual functionalities of any context-aware system as demonstrated in the comparison given in Table 6.1. New concrete filters can be built by inheriting all the features and properties of their parents (generic filter types). Therefore, the framework is extensible and generic.
3. The deployment language (CADL) can be used to put the architecture filters in different arrangements; therefore, the framework is customisable. The schema of the CADL code shows that it contains generic elements, which means that

different parameters can be used when establishing a new filter type. Therefore, the CADL code supports the framework extensibility.

4. The use of several software design patterns in the deployment of the filters (such as the Factory method creational design pattern) enables the creation and reuse of filter types in the design process. Interpreter design pattern is used to build the rule engine of the framework. Moreover, the rules used in the reasoning process can be extended, where new rules can be added to the rule engine. Therefore, the framework can be extended.
5. The use of some features such as generics and reflection in implementing the framework enables the addition of new concrete filter types by inheriting all the properties and features of the abstracted filters. Moreover, these design tools made the framework extensible.
6. Two modified criteria to evaluate the framework are used. Then two methods are adopted to justify the findings; these methods proof the completeness and the applicability of the GECAF framework.

7.2 Framework Limitations

The GECAF Framework has some limitations and drawbacks; these are mainly concerned with the following: -

1. The framework is not distributed; it does not support distributed environment.
2. As we assume that storing information in a secure repository is provided by external systems, GECAF framework did not implement any security functionality. End-users' privacy is an important issue that should be considered, especially when using their private information such as bank account, assets, biometric information, health history, company information, etc. This information should be stored in a secure place to avoid theft. Moreover, there should be an access control to the information as well as displaying the information should be well controlled.

7.3 Future Work

Future work in the subject area is outlined below: -

- The proposed framework is implemented as a desktop application. However, the intention is to build a middleware to solve the scalability issue of the design,

where context-aware systems are distributed and scalable. Middleware is a software layer that mediates the interaction between the operating system and an application or between different applications in a network. It provides a common programming abstraction as it hides the low level programming details. It has many requirements, like heterogeneity, support for mobility, scalability, support for privacy, tolerance to failure, and ease of deployment and configuration [117].

- Ontology is the current trend for knowledge representation (representing, relating, and reasoning context information); yet, existing ontology based approaches does not support generic applications. The future plan of this study is to build a common ontology inference rules to be used with the XML rule language. A future extension could also aim to develop the XML rule language and add more operations for accurate reasoning process.
- The framework uses SOAP as a type of application to exchange information across the web. To make the web services self-describing, WSDL can be used as another web services application in the design. WSDL describes network services and the way to access them regardless of the protocol used.
- To enhance and simplify the deployment process of the proposed framework, a graphical editor can be used to build the architecture filters. As a result, application providers can specify the primitive and abstracted context; then put a pattern to their architecture and the interpretation rules.
- As context-aware systems exploit users' information to adapt their behaviour, these systems are liable to security threats. Therefore it is important to consider information security and privacy in their design.
- Current trends are focused on applications concerned with the interactions between the vehicle and its occupants [118], and on the applications of context-aware computing to real-life domains, both home based and clinical [119]. In this respect, GECAF framework can be employed to develop applications in the healthcare and automotive trends.

PUBLICATIONS

Publication outputs during the past Four years are:

1. A. Abdulahad and A. Al-Yasiri. "Integrating Context Information into LDAP" *9th Annual Postgraduate Symposium on the Convergence of Telecommunicating, networking and broadcasting*, Liverpool, UK, 23-24 June, 2008.

2. A. Abdulahad and A. Al-Yasiri. "A Generic architecture implementing a Formal Context model for Context-Aware Systems" *10th Annual Postgraduate Symposium on the Convergence of Telecommunicating, networking and broadcasting*, Liverpool, UK, 22-23 June 2009.

3. A. Sabagh and A. Al-Yasiri. "Smart Classroom" *A poster to, Telecommunication and Networking Research Centre*, Salford University, Sept. 2010.

4. A. Sabagh and A. Al-Yasiri. "An Extensible Framework for Context-Aware Smart Environments" in *the 24th International Conference on Architecture of Computing Systems,* Lake Como, Italy, 22-25 Feb. 2011.

REFERENCES

[1] H. Henn. *"Past, present, future"* in J. Burkhardt, et al. (Eds.). Pervasive Computing: Technology and Architecture of Mobile Internet Applications. Boston, London: Addison-Wesley, ISBN: 0-201-72215-1, 2002.

[2] M. Weiser. "The Computer for the 21st Century". *Scientific American*, vol. 26, no. 3, pp. 94-104, 1991.

[3] T. Strang and C. L-Popien, "A context modelling survey" *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004, The 6th International Conference on Ubiquitous Computing*. Nottingham, Sept. 2004, pp. 33-40.

[4] M. Weiser, Ubiquitous computing. (March 2008), Available: http://sandbox.xerox.com/ubicomp/

[5] D.L. Becta, (May 2011) *Emerging Technologies for Learning*. Chapter 6, vol. 2, 2007, Available: http://www.terena.org/mail-archives/schoolnet/pdf6c94NcEJLX.pdf

[6] J. Indulska and P. Sutton. "Location management in Pervasive Systems" *Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing*, Adelaide, Australia, 2003.

[7] I. Roussaki, et al. "Hybrid context modeling: A location-based scheme using ontologies" in *Proceedings of the 4th annual IEEE international conference on Pervasive Computing and Communications Workshops, (PERCOMW'06)*, Pisa, Italy, 2006.

[8] A. Dey and G. Abowd. "Towards a better understanding of context and context-awareness" in *Proceedings of Workshop on the What, Who, Where, When and How of Context-Awareness*, Hague, Netherlands, April, 2000.

[9] B. Schilit, et al. "Context-aware computing applications" in *Proceedings of the 1st International Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, Dec. 1994, pp. 85-90.

[10] D. Salber, et al. "Designing for Ubiquitous Computing: A Case Study in Context Sensing", *Georgia Institute of Technology, GVU Technical Report; GIT-GVU-99-29*, 1999.

[11] S. Loke. *Context-Aware Pervasive Systems: The Architecture of a New Breed of Applications*. Monash University, Australia: Auerbach Publications Taylor & Francis Group, ISBN-10:0-8493-7255-0, 2006.

[12] C. Jiang, et al. "Classroom in the Era of Ubiquitous Computing - Smart Classroom" *IEEE International Conference on Wireless LANs PANs and Home Networks (ICWLHN2001), Singapore*, 5-7 Dec. 2001, pp. 14-26.

[13] G.M. Youngblood, et al. "Automation intelligence for the smart environment" in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, 2005, pp. 1513-1514.

[14] Y. Suo, et al. "Open Smart Classroom: Extensible and Scalable Learning System in Smart Space using Web Service Technology". *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 6, pp. 428-439, June, 2009.

[15] S.K. Das and D.J. Cook. "Designing and modeling smart environments" in *Proceeding of International Symposium on Wireless, Mobile and Multimedia Networks (WOWMOM)*, Niagara-Falls, Buffalo, NY, 2006, pp. 490-494.

[16] C.A.N. Guerra and C.F.S. Correa Da Silva. "A middleware for smart environments" in *Proceedings of British Society for Studies of Artificial Intelligence and Simulation of Behaviour*, Scotland, 2008.

[17] Z. Salvador, et al. "Smart Environment Application Architecture" in *Proceeding of the 2nd International Conference on Pervasive Computing Technologies for Healthcare, PervasiveHealth 2008.* Tampere, Finland, 2008, pp. 308-309.

[18] J. Bravo, et al. "Modelling Contexts by RFID-Sensor Fusion" *Forth IEEE Annual Conference on Pervasive Computing and Communications Workshops - PerCom Workshops*, Pisa, Italy, 2006.

[19] RFID Tags: Passive Tags or Active Tags, (January 2011), Available: http://www.buzzle.com/articles/rfid-tags-passive-tags-or-active-tags.html

[20] M. Aboelaze and F. Aloul. "Current and Future Trends in Sensor Networks: A Survey" in *Proceedings on Wireless and Optical Communications Networks,* Dubai, UAE, 2005, pp. 551-555.

[21] A. Klemetti, (July 2011) PDA Operating Systems. Available: http://www.tml.tkk.fi/Studies/Tik-111.590/2001s/papers/aarne_klemetti.pdf

[22] What is GPS?, (July 2011), Available: http://www8.garmin.com/aboutGPS/

[23] D. Charoenruk, (Dec. 2009) Communication Research Methodologies: Qualitative and Quantitative Methodology. Available: http://utcc2.utcc.ac.th/localuser/amsar/PDF/Documents49/quantitative_and_qualitative_methodologies.pdf

[24] A. Schmidt, et al. "There is more to context than location", *Computers and Graphics Journal,* vol. 33, no. 6, pp. 893-902, 1999.

[25] A. Schmidt. "Implicit Human Computer Interaction through Context". *Personal Technologies*, vol. 4, no. 2&3, pp. 191-199, 2000.

[26] A. Schmidt, et al. "Advanced Interaction in Context" *1th International Symposium on Handheld and Ubiquitous Computing (HUC99)*, Karlsruhe, Germany, 1999 and Lecture notes in computer science; 1707, ISBN 3-540-66550-1, Springer, 1999.

[27] D. Salber, "Context-Awareness and multimodality" in *Proceedings of the first workshop on multimodal user interfaces*, Grenoble, France, 2000.

[28] T. Gross and M. Specht, "Awareness in context-aware information systems" in *Proceedings of the Mensch und Computer - 1. Fachübergreifende Konferenz, Bad Honnef*, Bonn Germany, 2001, pp. 173-182.

[29] R. Kernchen, et al. "Context-Awareness in MobiLife" *15th IST Mobile and Wireless Communications Summit,* Mykonos, Greece, June, 2006.

[30] F. Azouaou and C. Desmoulins, "Using and modelling context with ontology in e-learning: the case of teacher's personal annotation" in *Proceedings of International Workshop on Applications of Semantic Web Technologies for E-Learning*, Dublin, Ireland, 2006.

[31] N. Ryan, et al. "Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant". *Computer Applications in* Archaeology, 1997.

[32] A.K. Dey. "Understanding and Using Context". *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4-7, 2001.

[33] G. Abowd and E. Mynatt. "Charting Past, Present, and Future Research in Ubiquitous Computing". *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 1, pp. 29-58, 2000.

[34] C. Becker and D. Niclas. "Where do spatial context-models end and where do ontologies start? A proposal of a combined approach" in *Proceedings of the 1st*

International Workshop on Advanced Context Modelling, Reasoning and Management UbiComp 2004, Nottingham, Sept. 2004.

[35] Y. Oh, et al. "ubi-UCAM 2.0: A Unified Context-aware Application Model for Ubiquitous Computing Environments" *first Korea/Japan Joint Workshop on Ubiquitous Computing & Networking Systems 2005 (ubiCNS2005)*, 2005.

[36] Y. Oh, et al. "User-centric Integration of Contexts for a Unified Context-aware Application Model", *ubiPCMM*, pp. 9-16, 2005.

[37] S. Jang and W. Woo. "Unified context representing user-centric context: Who, where, when, what, how and why" *ubiComp Workshop (ubiPCMM)*, 2005, pp. 26-34.

[38] G. Kunito, et al. "Architecture for Providing Services in the Ubiquitous Computing Environment" *International Conference on Distributed Computing Systems*, Lisboa, Portugal, 2006.

[39] J. Bravo, et al. "Ubiquitous Computing in the Classroom: An approach through Identification Process". *Journal of Universal Computer Science*, vol. 11, no. 9, pp. 1494-1504, 2005.

[40] K.E. Kajaer. "A Survey of Context-Aware Middleware" in *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*, 2007, pp. 148-155.

[41] K. Henricksen, et al. "Modelling Context Information in Pervasive Computing Systems" in *Proceedings Pervasive 2002*, Zurich, Aug. 2002.

[42] K. Henricksen and J. Indulska. "Modelling and using imperfect context information" in *Proceedings of the second IEEE Annual conference on pervasive computing and communications workshops (PERCOMVV'04)*, 2004.

[43] J. Gwizdka. "What's in the Context?" *Computer Human Interaction 2000 (CHI2000) - Workshop, The What, Who, Where, Why and How of Context-Awareness,* The Hague, Netherlands, 2000.

[44] T. GU, et al. "An ontology-based context model in intelligent environments" in *Proceedings of Communication Networks and Distributed Systems Modelling and Simulation Conference*, San Diego, California, USA, 2004.

[45] P. Korpipaa, et al. "Managing Context Information in Mobile Devices". *IEEE Pervasive Computing*, vol. 2, no. 3, pp. 42-51, 2003.

[46] M. Beigl, et al. "AwareCon: Situation Aware Context Communication" in *Proceedings of Ubicomp 2003,* Seattle, USA, Oct. 12-15, 2003.

[47] T. Mantoro and C.W. Johnson. " Location History in a Low-cost Context Awareness Environment" *Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing, ACSW 2003*, Adelaide, Australia, Australian Computer Science Communications, vol. 25, no. 6, pp. 153-158, 2003.

[48] A.V. Neelima, et al. "A Multi Agent, Service Reassembling Architecture for Context-Aware Systems". *International Journal of Recent Trends in Engineering*, vol. 1, no. 1, pp. 563-567, 2009.

[49] D. Zhang, et al. "Enabling Context-Aware Smart Home with Semantic Web Technologies". *International Journal of Human-friendly Welfare Robotic Systems*, 2005.

[50] A. Al-Yasiri and N. Linge. "A Lightweight Architecture for Context Aware Applications Using LDAP" *2nd International Conference on Computer Science & Information Systems*, Athens, Greece, 19-21 June, 2006.

[51] A. Abdulahad and A. Al-Yasiri. "Integrating Context Information into LDAP" *9th annual postgraduate symposium in PGNet 2008*, Liverpool, 23-24 June 2008.

[52] P. Fahy and S. Clarke, "CASS: Middleware for Mobile Context-Aware Applications", *Workshop on Context Awareness at MobiSys 2004*, Boston, 2004.

[53] M. Baldauf and S. Dustdar, "A Survey on Context-Aware Systems", TUV-1841-2004-24, Nov. 2004.

[54] M. Baldauf, et al. "A survey on context-aware systems". *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263-277, 2007.

[55] B.C. Chien, et al. "CADBA: A Context-aware Architecture based on Context Database for Mobile Computing" *International Workshop on Pervasive Media, in the Sixth International Conference on Ubiquitous Intelligence and Computing, Brisbane, Australia*, 7-9 July, 2009, pp. 367-372.

[56] J. Hong and J. Landay. "An infrastructure approach to context-aware computing". *Human-Computer Interaction*, vol. 16, no. 2 & 3, 2001.

[57] R. Schmohl and U. Baumgarten. "Context-aware Computing: A Survey Preparing a Generalized Approach" in *Proceedings of the international Multi-Conference of Engineers and Computer Scientists*, IMECS 2008, *Hong Kong*, 19-21 March 2008, vol. 1.

[58] C. Bettini, et al. "A survey of context modelling and reasoning techniques". *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161-180, April 2010.

[59] C. Bolchini, et al. "A Data-oriented Survey of Context Models". *ACM SIGMOD Record*, vol. 36, no. 4, pp. 19-26, 2007.

[60] C. Wang and X. Wang. "Multi-agent Based Architecture of Context-aware Systems", *International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, Seoul, Korea, 26-28 April, 2007.

[61] H. Chen, et al. "An Intelligent Broker for Context-Aware Systems" in *Adjunct Proceedings of Ubicomp, 2003 Seattle, Washington, USA*, 2003.

[62] T. GU, et al. "Toward an OSGi-Based Infrastructure for Context-Aware Applications". *IEEE Pervasive Computing*, pp. 66 -74, Oct-Dec, 2004.

[63] M. Miraoui, et al. "Context Modeling and Context-Aware Service Adaptation for Pervasive Computing Systems". *International Journal of Computer and Information Science and Engineering (IJCISE)*, vol. 2, no. 3, pp. 148-157, 2008.

[64] Z. Hwang, et al. "A Context Model by Ontology and Rule for Offering the User-Centric Services in Ubiquitous Computing" in *Proceedings of the 2007 International Conference on Convergence Information Technology*, 2007, pp. 77-82.

[65] M. Hong and D. Cho, "Ontology Context Model for Context-Aware Learning Service in Ubiquitous Learning Environments". *International Journal of Computers*, vol. 2, no. 3, pp. 193-200, 2008.

[66] J. Choi, "Software Architecture for Extensible Context-Aware Systems" in *Proceedings of the 2008 International Conference on Convergence and Hybrid Information Technology*, 2008, vol. 0, pp. 811-816.

[67] L. Harvel, et al. "Context Cube: Flexible and Effective Manipulation of sensed Context Data" in *Proceedings of the second International Conference on Pervasive computing (Pervasive 2004)*, Vienna, Austria, 21-23 April, 2004, LNCS of Springer, vol. 3001, pp. 51-68.

[68] J. Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers" in *Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, 1998, pp. 92-99.

[69] J. Bisgaard, et al. (December 2004) How is Context and Context-awareness Defined and Applied? A Survey of Context-awareness. Aalborg University 2004. Available: http:/www.csconsult.dk/rap/inf7_con.pdf

[70] M. Berens, "Sketching a Possible Field for Context-Aware Applications" in *Proceedings of the 2nd Twente Student Conference on Information Technology*, 2005, pp. 198-202.

[71] N. Cheng, et al. "A model for context-aware applications". *International Journal of Pervasive Computing & Communications*, vol. 4, no. 4, pp. 428–439, 2009.

[72] A. Chen, et al. "*Smart Rooms*" in D.J. Cook, S.K. Das and A.Y. Zomaya (Eds). Smart Environments: Technologies, Protocols and Applications. Hoboken, New Jersey: John Wiley and Sons, ISBN: 978-0-471-54448-7, 2005.

[73] G.D. Abowd, "Classroom 2000: An Experiment with the Instrumentation of a Living Educational Environment", *IBM Systems Journal*, vol. 38, no. 4, pp. 508-530, 1999.

[74] Y. Shi, et al. "The smart classroom: Merging technologies for seamless teleeducation". *IEEE Pervasive Computing*, vol. 2, no. 2, pp. 47-55, 2003.

[75] X. Wang, et al. "Semantic Space: An Infrastructure for Smart Spaces". *IEEE Pervasive Computing*, vol. 3, no. 3, pp. 32-39, 2004.

[76] S.S. Yau, et al. "Smart classroom: enhancing collaborative learning using pervasive computing technology" in *Proceedings of the 6th WFEO World Congress on Engineering Education and the 2nd ASEE International Colloquium on Engineering Education (ASEE '03)*, Nashville, Tenn, USA, June 2003, pp. 13633–13642.

[77] J. Bravo, et al. "Ubiquitous Computing in the classroom: An Approach through identification process". *Journal of Universal Computer Science*, vol. 11, no. 9, pp. 1494-1504, 2005.

[78] M. Back, et al. "Usable Ubiquitous Computing in Next-Generation Conference Rooms: Design, Evaluation, and Architecture" in *Proceeding, UbiComp 2006 Sept, workshop*, 2006.

[79] P. Osbakk and E. Rydgren. "Ubiquitous Computing for the Public" in *Proceedings of pervasive 2005 workshop of pervasive Mobile Interaction Devices (PERMID 2005)*, 2005, pp. 56-89.

[80] A. Lakas, et al. "ACP: An Interactive Classroom Response System for Active Learning Environment" in *Proceeding of the 2006 International Conference on Communications and Mobile Computing*, 2006, pp. 1301-1306.

[81] M. El-Bishouty, et al. "PERKAM: Personalized knowledge awareness map for computer supported ubiquitous learning". *Educational Technology and Society*, vol. 10, no. 3, pp. 122-134, 2007.

[82] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, New Jersey: Prentice Hall, 1996.

[83] Client-Server Architecture, (July 2009), Available: http://en.wikipedia.org/wiki/Client-server

[84] Model-View-Controller, (May 2009), Available: http://msdn.microsoft.com/en-us/library/ff649643.aspx

[85] M. Wooldridge, "Agent-based computing". *Interoperable communication Networks (Baltzer Science Publisher BV)*, vol. 1, no. 1, pp. 71-97, 1998.

[86] S. Guner. "*Architectural approach and methodologies of Service Oriented Architecture*" Master thesis, Technical University Hamburg, Harburg, Germany, 2005.

[87] Y. Oh, et al. "Designing, Developing and Evaluating Context-Aware Systems" International Conference on Multimedia and Ubiquitous Engineering (MUE'07), 2007, pp. 1158-1163.

[88] A. Singh and M. Conway "Survey of Context aware Framework – Analysis and Criticism" University of North Carolina at Chapel Hill Information Technology Service, Technical Report, 2006.

[89] A. Devaraju, et al. "A Context Gathering Framework for Context-Aware Mobile Solutions" *ACM International Conference on Mobile Technology, Applications and Systems,* Singapore, 10-12 Sept, 2007.

[90] M. Miraoui, et al. "Architectural Survey of Context-Aware Systems in Pervasive Computing Environment". *Ubiquitous Computing and Communication Journal*, vol. 3, no. 3, 2008.

[91] J. Hong, et al. "Context-aware systems: A literature review and classification". *Expert Systems with Applications*, vol. 36, no. 4, pp. 8509-8522, 2009.

[92] A. Saeed and T. Waheed, "An extensive survey of context-aware middleware architectures" *Electro/Information Technology (EIT), 2010 IEEE International Conference*, 2010, pp. 1-6.

[93] P. Bellavista, et al. "A Survey of Context Data Distribution for Mobile Ubiquitous Systems". *Accepted in ACM Computing Surveys (CSUR)*, ACM Press, expected to appear in vol. 45, no. 1, pp. 1-49, Mar. 2013.

[94] C. Hong, et al. "The Context-Awareness for the Network-based Intelligent Robot Services". *Advances in Service Robotics*, Ho Seok Ahn (Ed.), InTech Publishing, 2008.

[95] M.C. Domingo. "A context-aware service architecture for the integration of body sensor networks and social networks through the IP multimedia subsystem". *Barcelona Tech University, Barcelona, Spain, IEEE Communication magazine*, vol. 49, no. 1, pp. 102-108, 2011.

[96] H. Chen, et al. "Semantic Web in the Context Broker Architecture" in *Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom04)*, Orlando, FL, March 2004.

[97] C. De Gatti, et al. "An agent-based architecture for knowledge management in context-aware business processes" in *Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design*, 2010, pp. 318-323.

[98] D. Salber, et al. "The Context Toolkit: Aiding the Development of Context-Enabled Applications" in *Proceedings of the 1999 ACM Conference on Human Factors in Computer Systems (CHI'99),* Pittsburgh, PA, ACM Press, 15-20 May, 1999, pp. 434-441.

[99] A.K. Dey and G.D. Abowd. "The Context Toolkit: Aiding the Development of Context-Aware Applications" in *Workshop on Software Engineering for Wearable and Pervasive Computing,* Pittsburgh, 2000.

[100] K.A. Dey. *Providing Architectural Support for Building Context-Aware Applications*. Thesis (PhD), College of Computing, Georgia Institute of Technology, 2000.

[101] H.A. Duran-Limon, et al. "Context-Aware Middleware for Pervasive and Ad Hoc Environments" *MPAC '04 Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, New York, NY, USA, ACM Press, 2004, pp. 107-110.

[102] H.A. Duran-Limon, et al. "Context-Aware Middleware for Pervasive and Mobile Ad Hoc Environments", Department of Computing Science, Tecnológico de Monterrey (ITESM), Campus Guadalajara, México and Computing Department, Lancaster University, Bailrigg, Lancaster LA1 4YR, UK, 2004.

[103] A. Schmidt and K. Van Laerhoven. "How to build smart appliances?". *IEEE Personal Communications*, vol. 8, no. 4, pp. 66-71, 2001.

[104] H.L. Chen. *An intelligent broker architecture for pervasive context- aware systems*. Dissertation (PhD), University of Maryland, 2004.

[105] G. Biegel and V. Cahill A "Framework for Developing Mobile, Context-aware Applications" in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (PERCOM'04)*, USA, 2004.

[106] O.B. Kwon. "Modeling and generating context-aware agent-based applications with amended colored Petri nets". *Expert Systems with Application,* vol. 27, no. 4, pp. 609-621, 2004.

[107] S. Lee, et al. "A study on Issues in Context-Aware Systems Based on a Survey and Service Scenarios" *Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, SNPD '09, 10th ACIS International Conference*, Daegu, Koria, 2009, pp. 8-13.

[108] G. Carter. *LDAP System Administration.* USA: O'Reilly and Associates Inc, 2003.

[109] E. Gamma, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1994.

[110] J. Snell, et al. *Programming Web Services with SOAP*. USA: O'Reilly & Associates Inc, 2002.

[111] S. Seely, and K. Shrkey, *SOAP Cross Platform Web Service Development Using XML*. London: Prentice Hall PTR, 2001.

[112] F. Siegemund. "A Context-Aware Communication Platform for Smart Objects" in *Proceedings of 2nd International Conference on Pervasive Computing (PERVASIVE)*, Springer, 2004, pp. 69-86.

[113] R.E. Johnson and B. Foote "Designing Reusable Classes". *Journal of Object-Oriented Programming*, vol. 1, no. 2, pp. 22-35, 1988.

[114] I.R. Forman and N. Forman. *Java Reflection in Action*. Greenwich, USA: Manning Publications, 2005.

[115] Y.D. Liang, *Introduction to Java programming: core version*. 5th ed. Upper Saddle River, N. J.: Prentice Hall, 2005.

[116] P. Bratskas, et al. "An evaluation of the state of the art in context-aware architectures" *Sixteenth International Conference on Information Systems Development (ISD 2007)* (Galway, Ireland), Springer, Verlag, 2007, pp. 1-12.

[117] K. Henricksen, et al. "Middleware for distributed context-aware systems" in *Proceedings of On the Move to Meaningful Internet Systems*, LNCS 3760, 2005, pp. 846-863.

[118] A. Schmidt, et al. "Automotive Pervasive Computing". *Pervasive computing, IEEE*, vol. 10, no. 3, pp. 12-13, March 2011.

[119] A.K. Dey and D. Estrin. "Pervasive Healthcare 2010: Two Perspectives". *Pervasive computing, IEEE*, vol. 10, no. 3, pp. 8-11, March 2011.

BIBLIOGRAPHY

[1] B. McLaughlin. *Java & XML*. Sebastopol. CA Cambridge: O'Reilly, 2001.

[2] B. Oh, et al. "U-Learning Framework for U-Classroom". *IADIS International Conference*, 2006, pp. 367-370.

[3] C. Chatfield and R. Hexel. "User Identity and Ubiquitous Computing: User Selected Pseudonyms" in *Proeedings of UbiComp*. Tokyo, Japan, Sept. 2005.

[4] C. Ciavarella and F. Paterno. "The design of a handheld, location-aware guide for indoor environments". *Personal Ubiquitous Computing*, vol. 8, pp. 82-91, 2004.

[5] C. Shin and W. Woo. "Conflict Resolution Method utilizing Context History for Context-Aware Applications" in *Proceedings of ECHISE'05, held in Conjunction with PERVASIVE'05*, Cognitive Science Research Paper - University of Sussex CSRP 2005, ISSU 577, pp. 105-110.

[6] D. Ayed, et al. "A data model for context-aware deployment of component-based applications onto distributed systems". *Component-oriented approaches to context-aware systems Workshop ECOOP'04*, Oslo, Norway, June 2004.

[7] D. Flanagan. *Java in a nutshell*. Beijing Franham: O'Reilly, 2005.

[8] D. Kang, et al. "A Context Aware System for Personalized Services using Wearable Biological Signal Sensors" *International Conference on control, Automation and systems*, in COEX Seoul, Korea, 14-17 Oct. 2008.

[9] D. Salber, et al. "Designing For Ubiquitous Computing: A Case Study in Context Sensing". Georgia Institute of Technology, GVU Technical Report; GIT-GVU-99-29, 1999.

[10] D.H. Wilson, et al. "Using Context History for Data Collection in the Home" in *Proceedings of ECHISE'05, held in Conjunction with PERVASIVE'05*, Munich, Germany, 11 May, 2005.

[11] E. Castro. *HTML 4 for the World Wide Web*. Berkeley, Calif, Great Britain: Peachpit Press, 2000.

[12] G.T. Huang. "China's clever classroom". *Technology Review*, vol. 107, no. 5, pp. 26, June 2004.

[13] G. Thomson, et al. "Situation determination with distributed context histories" in *Proceedings of ECHISE'05, held in Conjunction with PERVASIVE'05*, Munich, Germany, 11 May, 2005.

[14] H. Gu et al. "SLAP: a Location-aware Software Infrastructure for Smart Space" in *Proceedings of the 3rd IEEE workshop on Software Technologies for Future Embedded and Ubiquitous Systems(SEUS'05)*, IEEE CS Press, Seattle, USA, 2005, pp. 35-38.

[15] H. Si, et al. "A Stochastic Approach for Creating Context-Aware Services based on Context Histories in Smart Home" in *Proceeding of the 3rd International Conference on Pervasive Computing (Pervasive2005), (ECHISE2005)*, Munich, Germany, May 2005, pp. 37-41.

[16] H. Yuan, et al. "Design of a Novel Indoor Location Sensing Mechanism using IEEE 802.11e WLANs", *The Seventh Annual PostGraduate Symposium in Telecommunications and Network System (PGNet)*, Liverpool, UK, 26-27 June, 2006.

[17] I. F. Akyildiz, et al. "A Survey on Sensor Networks". *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102-114, 2002.

[18] J. Wu, et al. "Building a Context-Aware Smart Exhibition Environment", *International Conference on Wireless Communications, Networking and Mobile Computing, WiCom*, 2007, pp. 2177-2180.

[19] J. Hightower and G. Borriello. "A survey and taxonomy of location systems for ubiquitous computing" *Computer*, vol. 34, no. 8, pp. 57-66, Aug. 2001.

[20] J. Ma, et al. "Towards a smart world and ubiquitous intelligence: A walkthrough from smart things to smart hyperspaces and UbicKids". *International Journal of Pervasive Computing and Communications*, vol.1 no.1, pp.53-68, 2005.

[21] K. Khido. "Context-Aware Systems for Mobile and Ubiquitous Networks" in *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)*, 23-29 April 2006.

[22] K. Rahman and F. Stajano. "An Architecture for Interactive Context-Aware Applications". *IEEE Computer Society*. pp. 73-80, 2007.

[23] L.M. Ni, et al. "LANDMARC: Indoor Location Sensing Using Active RFID". *Wireless Networks*, vol. 10, pp. 701-710, 2004.

[24] M. Kaenampornpan and E. O'Neill "Integrating History and Activity Theory in Context Aware System Design" in *Proceedings of the W8 ECHISE 2005 - 1st International Workshop on Exploiting Context Histories in Smart Environments, Pervasive 2005*, Munich, Germany, 8-13 May, 2005.

[25] R.C.A. Da Rocha, et al. "Middleware for ubiquitous context-awareness" in *Proceedings of the 6th international workshop on Middleware for pervasive and ad-hoc computing, Leuven*, Belgium, 2008, pp. 43-48.

[26] R. Englander. *Java and SOAP*. Sebastopol, Calif, Cambridge: O'Reilly 2002.

[27] R. Lafore. *Object-oriented programming in C++*. 4th ed. Indianapolis, Ind.: Sams; Hemel Hempstead: Prentice Hall [distributor], 2001.

[28] R. Lerdorf, et al. *Programming PHP*. Beijing Franham: O'Reilly, 2006.

[29] S. Voida and E. D. Mynatt. "Context Histories, Activities, and Abstractions: Ubiquitous Computing Support for Individual and Collaborative Work" in *Proceedings of ECHISE'05, held in Conjunction with ERVASIVE'05,* Munich, Germany, 11 May, 2005.

[30] T.J. Teorey. *Database modeling & design.* 3rd ed., San Francisco: Morgan Kaufmann, 1999.

[31] W. Edwards. "Discovery Systems in Ubiquitous Computing". *IEEE Pervasive Computing*, vol. 5, no. 2, pp. 70-77, April 2006.

[32] W. Heinzelman, et al. "Middleware to Support Sensor Network Applications". *IEEE Network Magazine*. Special Issue, MiLAN, Jan. 2004.

[33] Y.D. Liang. *Introduction to: JAVA Programming Core Version.* 5th ed. Person Prentice Hall, 2005.

[34] Y. Kawahara, et al. "Design and Implementation of a Sensor Network Node for Ubiquitous Computing Environment" in *Proceedings of IEEE Semiannual Vehicular Technology Conference (VTC-Fall)*, Orland, FL, USA, Oct. 2003.

APPENDICES

A. SOAP (Simple Object Access Protocol)

SOAP is a simple extensible protocol based on XML standards; it is used with HTTP messages to communicate between application level objects among multiple components with different languages running on different vendor platforms. SOAP transaction begins with an application making a call to a remote procedure or RPC (Remote Procedure Call) style. The SOAP client script encodes the procedure request as an XML payload and sends it over the transport protocol to a server script. The request is parsed by the server and passed to a local method which sends a reply to the client encoded in XML. Then the client parses the response and passes the result to the original function.

SOAP message consists of an envelope containing optional header and mandatory body, see Figure A.1. The header contains blocks of information relevant to how the message is to be processed (including routing and delivery setting, authentication, and transaction contexts). The body carries the main payload of the message to be delivered and processed, and it may also contain fault element to indicate error at the server. This information can be the remote requested method (i.e. method name and parameters name, type and value) [110].

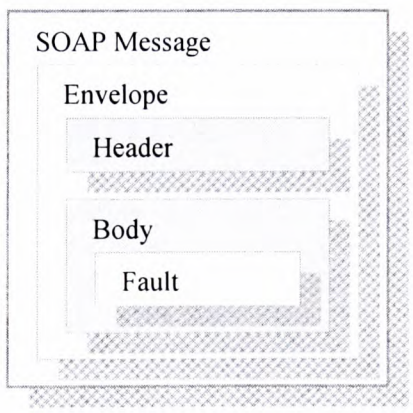


Figure A.1: SOAP message

In the proposed architecture SOAP has been used to call a procedure on the remote server. This type of web service application is specified through the deployment language (CADL). Example (A.1) below, illustrates a simple RPC-style SOAP message. In RPC-style message, messages come in pairs; the request (client sends function call information to the server), and the response (server sends return value(s) back to the client). In this example, the requested method is *"String database (String context)"* with context parameter value as *"Review"* to be sent to a server. The returned parameter in a response

message is *"filename"* with a value of *"lecture1.htm"*, and function namespace is defined as: "http://localhost/soap/soap_database_server.php".

The XML syntax for expressing a SOAP message is based on the "http://www.w3.org/2001/12/soap-envelope" namespace. The XML namespace identifier points to an XML schema that defines the structure of what SOAP message looks like. An encoding style, which is a set of rules, is used to define exactly how applications on different platforms share information, even though they may not have common data types or representations.

Example (A.1): RPC-style SOAP message

SOAP request contains details of a call to be made, as given in the following XML code: -

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
        soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body>
        <m:database xmlns:m="http://localhost/soap/soap_database_server.php" >
            <m:context xsi:type="xsd:string">Review</m:context>
        </m:database>
    </soap:Body>
</soap:Envelope>
```

SOAP response will be very similar in structure to the request; it contains the result of the call and as given below: -

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
        soap:encodingStyle= "http://www.w3.org/2001/12/soap-encoding">
    <soap:Body>
        <m:databaseResponse xmlns:m="http://localhost/soap/soap_database_server.php">
            <m:filename xsi:type="xsd:string">lecture1.htm</m:filename>
        </m:databaseResponse>
    </soap:Body>
</soap:Envelope>
```

B. XML Schema (CADLSchema.xsd) for the CADL Code.

```
<?xml version="1.0" ?>
<an:schema xmlns:an="http://www.w3.org/2001/XMLSchema">
<an:element name="Filters">
  <an:complexType>
    <an:sequence>
      <an:element name="events" minOccurs="1" maxOccurs="1">
        <an:complexType>
              <an:sequence>
               <an:element name="event" minOccurs="1" maxOccurs="unbounded">
            <an:complexType>
              <an:sequence>
                 <an:element name="description" type="an:string" />
              </an:sequence>
              <an:attribute name="id" type="an:string" default="" />
              <an:attribute name="CID" type="an:string" default="" />
            </an:complexType>
          </an:element>
        </an:sequence>
      </an:complexType>
    </an:element>
    <an:element name="process" maxOccurs="unbounded">
      <an:complexType>
        <an:sequence>
          <an:element name="class" minOccurs="1" maxOccurs="6">
           <an:complexType>
             <an:sequence>
               <an:element name="source" minOccurs="0" maxOccurs="6">
                 <an:complexType>
                   <an:simpleContent>
                      <an:extension base="an:string">
                       <an:attribute name="no" type="an:string" default=""/>
                       <an:attribute name="type" type="an:string" default="xml"/>
                      </an:extension>
                    </an:simpleContent>
                  </an:complexType>
                </an:element>
                <an:element name="target" minOccurs="0" maxOccurs="1">
                 <an:complexType>
                   <an:simpleContent>
                      <an:extension base="an:string">
                       <an:attribute name="type"type="an:string"default="xml"/>
                      </an:extension>
                    </an:simpleContent>
                  </an:complexType>
                </an:element>
                <an:element name="Element" minOccurs="0" maxOccurs="10">
                  <an:complexType>
                     <an:simpleContent>
                      <an:extension base="an:string">
                       <an:attribute name="name" type="an:string" use="required"/>
                        <an:attribute name="value" type="an:string" />
                        <an:attribute name="type" type="an:string" />
                        <an:attribute name="fileType" type="an:string"  default="xml"/>
                       </an:extension>
                     </an:simpleContent>
                  </an:complexType>
                </an:element>
              </an:sequence>
              <an:attribute name="name" type="an:string" use="required" />
              <an:attribute name="context" type="an:string" />
              <an:attribute name="address" type="an:string" />
               <an:attribute name="time" type="an:string" />
               <an:attribute name="RID" type="an:string" />
            </an:complexType>
          </an:element>
        </an:sequence>
        <an:attribute name="type" type="an:string" />
```

```
            <an:attribute name="CID" type="an:string" />
            <an:attribute name="category" type="an:string" />
          </an:complexType>
        </an:element>
      </an:sequence>
    </an:complexType>
</an:element>
</an:schema>
```

C. The Rule Schema 'ruleSchema.xsd' for the XML Rule Document.

```
<?xml version="1.0"?>
<an:schema xmlns:an="http://www.w3.org/2001/XMLSchema">
  <!--definition of 'operation' element-->
  <an:complexType name="opType">
    <an:sequence>
      <an:element name="parameter" minOccurs="0" maxOccurs="unbounded">
        <an:complexType>
          <an:sequence>
            <an:element name="operation" minOccurs="0" maxOccurs="unbounded" />
          </an:sequence>
          <an:attribute name="type">
            <an:simpleType>
              <an:restriction base="an:string">
                <an:enumeration value="simple" />
                <an:enumeration value="complex" />
              </an:restriction>
            </an:simpleType>
          </an:attribute>
          <an:attribute name="source">
            <an:simpleType>
              <an:restriction base="an:string">
                <an:enumeration value="internal" />
                <an:enumeration value="external" />
                <an:enumeration value="fixed" />
              </an:restriction>
            </an:simpleType>
          </an:attribute>
          <an:attribute name="value" type="an:string" use="optional" />
        </an:complexType>
      </an:element>
      <an:element name="returnValue" minOccurs="1" maxOccurs="1">
        <an:complexType>
          <an:sequence>
            <an:element name="case" minOccurs="1" maxOccurs="unbounded">
              <an:complexType>
                <an:simpleContent>
                  <an:extension base="an:string">
                    <an:attribute name="value" type="an:string" default="" />
                    <an:attribute name="return" type="an:string" default="" />
                  </an:extension>
                </an:simpleContent>
              </an:complexType>
            </an:element>
          </an:sequence>
        </an:complexType>
      </an:element>
    </an:sequence>
    <an:attribute name="name">
      <an:simpleType>
        <an:restriction base="an:string">
          <an:enumeration value="And" />
          <an:enumeration value="OR" />
          <an:enumeration value="NOT" />
          <an:enumeration value="GT" />
          <an:enumeration value="GE" />
          <an:enumeration value="LT" />
          <an:enumeration value="LE" />
          <an:enumeration value="egual" />
          <an:enumeration value="Plus" />
          <an:enumeration value="Minus" />
          <an:enumeration value="Times" />
          <an:enumeration value="Divide" />
          <an:enumeration value="subStr" />
        </an:restriction>
```

```
    </an:simpleType>
  </an:attribute>
 </an:complexType>

 <!—Start of schema elements-->
 <an:element name="rule">
   <an:complexType>
     <an:sequence>
       <an:element name="context" minOccurs="1" maxOccurs="unbounded">
         <an:complexType>
           <an:sequence>
             <an:element name="operation" type="opType" />
           </an:sequence>
           <an:attribute name="name" type="an:string" use="required" />
           <an:attribute name="RID" type="an:string" use="required" />
         </an:complexType>
       </an:element>
     </an:sequence>
   </an:complexType>
  </an:element>
</an:schema>
```

D. Interpreter Design Pattern

The Interpreter pattern describes how to define a grammar for simple languages, representing sentences in the language, and interpret these sentences. Figure D.1 shows the UML diagram for the Interpreter design pattern. The interpreter pattern uses an Expression abstract class and a terminal or non-terminal classes. The Client class is accountable for building the syntax tree of arithmetic, logical, or other expression.

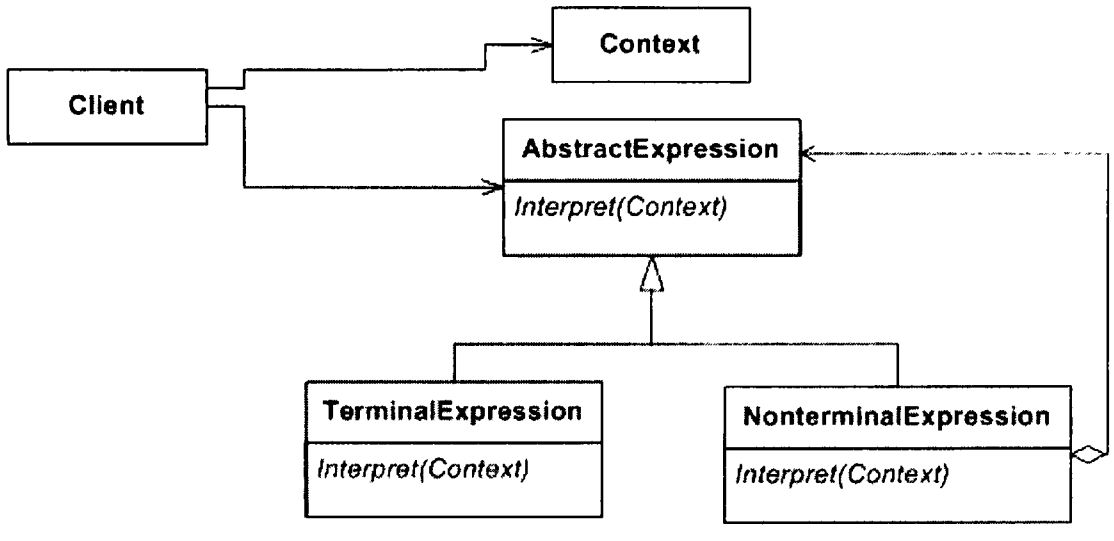


Figure D.1: Interpreter design pattern

E. Examples of the Deployment Process of the Architecture

Example E.1: Using reflection to instantiate framework objects of concrete classes inherited from the super class 'GetContext'. Where 'concreteClassName' is a string retrieved from the CADL code.

```
GetContext concreteObject = null;
try{
        concreteObject=(GetContext)Class.forName(concreteClassName).newInstance();
        concreteObject.getContext(target, Eid);

        }catch (InstantiationException e) {
              e.printStackTrace();
        } catch (IllegalAccessException e) {
                    e.printStackTrace();
        } catch (ClassNotFoundException e) {
                    e.printStackTrace();
        }
```

Example E.2: Using Generics to create different filters' types of the architecture.

Here 'E' represents a generic filter object such as 'Getter', 'Adder', 'Formatter', etc. while 'filterName' represents the concrete filter type such as 'ID', 'FormatParameter', 'AddHistory', etc.

```
public abstract class Creator <E> {
    String filterName;

    public Creator(){}
    public Creator(String filterName){
          this.filterName = filterName;
    }

    public abstract E create(String filterName);
}
```

Example E.3: The concrete class 'GetterCreator' using reflection feature.

```
public class GetterCreator extends Creator <GetContext> {

   public GetterCreator(){}
   public GetterCreator(String getterName){
          super (getterName);
   }

   public GetContext create(String getterName){

          GetContext  a_getter=null;
          try {
                    a_getter=(GetContext)(Class.forName(getterName)).newInstance();
                    } catch (InstantiationException e) {
                              e.printStackTrace();
                    } catch (IllegalAccessException e) {
                              e.printStackTrace();
                    } catch (ClassNotFoundException e) {
                              e.printStackTrace();
                    }
                    return a_getter;
     }
}
```

F. CADL code to representing 'Role' context.

```
<?xml version="1.0" ?>
<Filters xmlns:an="http://www.w3.org/2001/XMLSchema-instance"
          an:noNamespaceSchemaLocation="CADLSchema.xsd">
    <events>
       <event id="a11" CID="a1">
          <description>Type of lecture</description>
       </event>
    </events>
    <process type="Get" CID="a1">
       <class name="ID">
          <source type="xml">sensor1</source>
          <target type="xml">G1</target>
       </class>
          <class name="Time">
                <source type="xml">sensor2</source>
          <target type="xml">G2</target>
       </class>
       <class name="Location">
          <source type="xml">sensor3</source>
          <target type="xml">G3</target>
       </class>
       <class name="Activity">
          <source type="xml">sensor4</source>
          <target type="xml">G4</target>
       </class>
       <class name="Environment">
          <source type="xml">sensor5</source>
          <target type="xml">G5</target>
       </class>
       <class name="Value">
          <source type="xml">sensor6</source>
          <target type="xml">G6</target>
       </class>
    </process>
    <process type="Format" CID="a1">
       <class name="FormatParameter" context="Role">
          <source no="1" type="xml">G1</source>
          <source no="2" type="xml">G2</source>
          <source no="3" type="xml">G3</source>
          <source no="4" type="xml">G4</source>
          <source no="5" type="xml">G5</source>
          <source no="6" type="xml">G6</source>
          <target type="xml">p1</target>
       </class>
       </process>
    <process type="Interpret" CID="a1">
       <class name="Role" RID="1">
          <source type="xml">p1</source>
          <target type="xml">p2</target>
       </class>
    </process>
    <process type="Application" category="automatic" CID="a1">
       <class name="SOAPapplication" address=
             "http://localhost/smartclassroom/sclassroom1.php">
          <Element name="functionName" value="database" />
          <Element name="location" value=
                   "http://127.0.0.1/soap/soap_database_server.php"/>
          <Element name="parameter" value="path" type="STRING"
                   fileType="xml">p2</Element>
       </class>
    </process>
</Filters>
```

G. Smart Classroom CADL Code.

```
<?xml version="1.0" ?>
<Filters xmlns:an="http://www.w3.org/2001/XMLSchema-instance"
    an:noNamespaceSchemaLocation="CADLSchema.xsd">
    <events>
        <event id="a11" CID="a1">
            <description>Type of lecture</description>
        </event>
        <event id="a22" CID="a2">
            <description>Lecture font size</description>
         </event>
    </events>
    <process type="Get" CID="a1">
        <class name="ID">
             <source type="xml">sensor1</source>
             <target type="xml">G1</target>
         </class>
        <class name="Time">
             <source type="xml">sensor2</source>
             <target type="xml">G2</target>
         </class>
        <class name="Location">
             <source type="xml">sensor3</source>
             <target type="xml">G3</target>
         </class>
        <class name="Environment">
             <source type="xml">sensor5</source>
             <target type="xml">G4</target>
        </class>
     </process>
    <process type="Format" CID="a1">
        <class name="FormatParameter" context="Role">
              <source no="1" type="xml">G1</source>
              <source no="2" type="xml">G2</source>
              <source no="3" type="xml">G3</source>
              <target type="xml">p1</target>
       </class>
</process>
<process type="Interpret" CID="a1">
    <class name="Role" RID="1">
        <source type="xml">p1</source>
        <target type="xml">p2</target>
    </class>
 </process>
<process type="Manipulate" CID="a1">
    <class name="Retrieve" time="10">
        <source>attendance</source>
        <target type="xml">p3</target>
    </class>
</process>
<process type="Format" CID="a1">
    <class name="FormatIndirectContext" context="typeOfLecture">
         <source no="1" type="xml">p2</source>
         <source no="2" type="xml">p3</source>
         <source no="3" type="xml">G4</source>
         <target type="xml">p4</target>
    </class>
</process>
<process type="Interpret" CID="a1">
    <class name="typeOfLecture" RID="2">
        <source type="xml">p4</source>
        <target type="xml">p5</target>
    </class>
</process>
<process type="Application" category="automatic" CID="a1">
    <class name="SOAPapplication" address="http://localhost/smartclassroom/
            sclassroom1.php">
        <Element name="functionName" value="database" />
        <Element name="location" value="http://127.0.0.1/soap/soap_database_server.php" />
```

```
        <Element name="parameter" value="path" type="STRING" fileType="xml">p5</Element>
    </class>
</process>
<process type="Get" CID="a2">
    <class name="ID">
        <source type="xml">sensor1</source>
        <target type="xml">G1</target>
    </class>
    <class name="Time">
        <source type="xml">sensor2</source>
        <target type="xml">G2</target>
    </class>
    <class name="Location">
        <source type="xml">sensor3</source>
        <target type="xml">G3</target>
    </class>
</process>
<process type="Format" CID="a2">
    <class name="FormatParameter" context="personFont">
        <source no="1" type="xml">G1</source>
        <source no="2" type="xml">G2</source>
        <source no="3" type="xml">G3</source>
        <target type="xml">p6</target>
    </class>
</process>
<process type="Interpret" CID="a2">
    <class name="personFont" RID="3">
        <source type="xml">p6</source>
        <target type="xml">p7</target>
    </class>
</process>
<process type="Application" category="tagging" CID="a2">
    <class name="SOAPapplication" address="">
        <Element name="functionName" value="font" />
        <Element name="location" value="http://127.0.0.1/soap/soap_font_server.php" />
        <Element name="parameter" value="font" type="STRING" fileType="xml">p7</Element>
        </class>
    </process>
</Filters>
```

H. The Smart Classroom XML Rule Code.

```
<?xml version="1.0" ?>

<rule xmlns:an="http://www.w3.org/2001/XMLSchema-instance"
    an:noNamespaceSchemaLocation="ruleSchema.xsd ">

    <context name="typeOfLecture" RID="2">
        <operation name="And">
            <parameter type="complex">
                <operation name="And">
                    <parameter type="complex">
                        <operation name="LE">
                            <parameter type="simple" source="external" value="p3.xml" />
                            <parameter type="simple" source="fixed" value="20" />
                            <returnValue>
                                <case value="true" return="true" />
                                <case value="false" return="false" />
                            </returnValue>
                        </operation>
                    </parameter>
                    <parameter type="complex">
                        <operation name="equal">
                            <parameter type="simple" source="internal" value="context1" />
                            <parameter type="simple" source="fixed" value="Lecturer" />
                            <returnValue>
                                <case value="true" return="true" />
                                <case value="false" return="false" />
                            </returnValue>
                        </operation>
                    </parameter>
                    <returnValue>
                        <case value="true" return="true" />
                        <case value="false" return="false" />
                    </returnValue>
                </operation>
            </parameter>
            <parameter type="complex">
                <operation name="equal">
                    <parameter type="simple" source="external" value="G4.xml" />
                    <parameter type="simple" source="fixed" value="Occupied" />
                    <returnValue>
                        <case value="true" return="true" />
                        <case value="false" return="false" />
                    </returnValue>
                </operation>
            </parameter>
            <returnValue>
                <case value="true" return="Review" />
                <case value="false" return="NOReview" />
            </returnValue>
        </operation>
    </context>

</rule>
```

I. HTML and PHP Codes.

1. Smart classroom website

```
<html>
    <head>
        <title>university smart classroom</title>
        <style type="text/css"> <!-- internal style sheet -->

            div {position:absolute}
            th.a {width:140; height=25; text-align:center; valign=top;font-size=12;}
            a:link{background:white; color:"#A0A0A0"; text-decoration:none;}
            a:visited{background:white; color:"#990099"; text-decoration:none;}
            a:hover{background:"#A8A8A8"; color:"606060"; text-decoration:none;}
            a:active{background:white; color:"#3366FF"; text-decoration:none;}
            <!-- set color, font-size, and font-style -->
                p{ color:White;
                  font-size:18pt;
                  font-style:normal;
                 }
        </style>
    </head>

    <body bgcolor="#e0e0e1"left:5px; top:60px; width:957px; background-color:C71585; color:white;
        text-align:center; overflow:auto;">
        <div style="left:90px; top:0px; width:830px; height:585px; background-color:white;       color:red;
            text-align:left; overflow:auto;">

            <?php
                //read the information related to context from out.txt
                //e.g. lecture note to be displayed 'lecture1.php'
                $txtfile = fopen("c:/database.txt", 'r');
                $fData = fread($txtfile, 50);
                 fclose($txtfile);
            ?>

            <h> <!-- salford university logo --><img src="ex\logo.bmp"></h>
            <h>  <!-- smartclasspic1 logo -->
                <img  style="left:90px; top:0px; width:830px; height:80px;" src="ex\smartclasspic2.bmp">
            </h>
        </div>
        <!-- Table for the following links: University Page, Students Names, Time Table, Lecturer Notes,
            Tutorials, and Mail-->
        <div style="left:90px; top:145px; width:135px; height:500px; background-color:white;
            color:red; overflow:auto;">
            <table>
              <tr><th class=a><a HREF="http://www.salford.ac.uk">University Page</a><br>
                    <hr></th></tr>
              <tr><th class=a><a HREF="names.htm">Students Names</a><br><hr></th></tr>
              <tr><th class=a><A HREF="time.htm"> Time Table </a><br><hr></th></tr>
              <tr><th class=a><A HREF="lecture1.php">Lecturer Notes</a><br><hr></th></tr>
              <tr><th class=a><A HREF="tutorial.htm">  Tutorials  </a><br><hr></th></tr>
              <tr><th class=a><A HREF="https://webmail.salford.ac.uk/horde/imp/">   Mail  </a>
                    <br><hr></th></tr>
              <!-- text area to input data -->
              <tr><th bgcolor="#808000"><font size=2 color=FFFFFF>Your Blackboard</th></tr>
              <tr><th><form action="action.php" method="post">
                        <textarea name="name" Rows=10 cols=12></textarea><br>
                        <input color=#909090 type="submit" value="Send"/>
                    </form></th></tr>
            </table>
        </div>

        <!--Line saparator between links and lecture notes -->
        <!-- Table for lecture note titl and the lecture notes -->
        <div style="left:225px; top:145px; width:688px; height:390px; color:white; ">
            <table>
            <!-- lecture note title -->
              <tr><th bgcolor="#CC0066"><font size=5 color=FFFFFF>Lecture Note</th></tr>
```

```
            <tr><th><iframe id='iframe1' src="<?php print $fData;?>" width=688
                height=400></iframe></th></tr>
          </table>
        </div>
    </body>
</html>
```

2. Lecture 1

```
<html>
    <?php
        //read context ID from a text file 'ID.txt', e.g. ID='@001' to find the font size
        $txtfile = fopen("c:/font.txt", 'r');
        $font= fread($txtfile, 50);
        fclose($txtfile);
    ?>
    <head>
        <title>smart classroom</title>
        <!--style sheet to set color, font style and size-->
        <style type="text/css">
          p{
            color:#000080;
            font-style:normal;
            font-size:<?php print $font; ?>
          }
        </style>

        <!-- refresh the page every 10 sec. -->
        <META http-equiv="REFRESH" content="10">
    </head>

    <BODY BGCOLOR="#FFFFF0">
        <!-- Lecture note title -->
        <h2><font color=C71585>Lecture No: 1</h2>
            <!-- set the color font a, b, c, or d -->
            <!-- these information to be retreived from external style sheet -->
            <?php $colorFont='b';?>
            <hr>
            <pre>
            <!-- Lecture notes starts with the title, and then followed by the lecture-->
            <p><font color=#000080>
                Context and Context-Awareness

                What is Context?

                'context is any information that can be used to characterise the situation of an entity. An
                entity is a person, place, or object that is considered relevant to the interaction between a
                user and an application, including the user and application themselves'
                    (Dey)

                    <font color=#660066>
                What is Context-Awareness?

                   'the use of context to provide task-relevant information
                   and/or services to a user, wherever they may be'  (Dey)
            </p>
    </BODY>
</html>
```

3. PHP Code – soap_database_server

```
<?php
 // Pull in the NuSOAP code
include("lib/nusoap.php");

// Create the server instance
$srv = new soap_server();
```

```
// Register the method to expose
$srv->register("database");

// The method connect to database to aquire a data from a table
function database($context0)
{
   $con = mysql_connect("localhost:3306/contexthistory", "root", "gecaf");
   if (!$con) {
        die('Could not connect: ' . mysql_error());
   }
   //database to connect to is 'contexthistory'
   mysql_select_db("contexthistory", $con);

   //table name is 'contextfiles' it has the following colomns (CID, context, filename)
   $query='SELECT * FROM contextfiles WHERE context='."'" . $context0 ."'";
   $result = mysql_query($query);

   //get the content of path colomn
   while($row = mysql_fetch_array($result)){
        $a=$row['filename'];
   }
   return $a;
   mysql_close($con);
}

// Use the request to (try to) invoke the service
$HTTP_RAW_POST_DATA = isset($HTTP_RAW_POST_DATA) ? $HTTP_RAW_POST_DATA : "";
$srv->service($HTTP_RAW_POST_DATA);
?>
```

4. PHP Code – SOAP_font_server

```
<?php
// Pull in the NuSOAP code
include("lib/nusoap.php");

// Create the server instance
$srv = new soap_server();

// Register the method to expose
$srv->register("font");

// The method connect to database to aquire a data from a table
function font($context0)
{
   $con = mysql_connect("localhost:3306/contexthistory", "root", "gecaf");
   if (!$con) {
    die('Could not connect: ' . mysql_error());
   }

   // database to connect to is 'contexthistory'
   mysql_select_db("contexthistory", $con);

   // table name is 'profile' it has the following colomns (ID, font, name, email)
   $query='SELECT * FROM profile WHERE ID='."'" . $context0 ."'";
   $result = mysql_query($query);

   // get the content of font colomn
   while($row = mysql_fetch_array($result)){
    $a=$row['font'];
   }
   return $a;
   mysql_close($con);
}

// Use the request to (try to) invoke the service
$HTTP_RAW_POST_DATA = isset($HTTP_RAW_POST_DATA) ? $HTTP_RAW_POST_DATA : "";
$srv->service($HTTP_RAW_POST_DATA);
?>
```

J. The XML rule for interpreting Presence and Meeting contexts

```
<?xml version="1.0" ?>
<rule>
  <context name="Presence" RID="1">
    <operation name="And">
      <parameter type="complex">
        <operation name="And">
          <parameter type="complex">
            <operation name="equal">
              <parameter type="simple" source="internal" value="Time" />
              <parameter type="simple" source="fixed" value="aTime" />
              <returnValue>
                <case value="true" return="true" />
                <case value="false" return="false" />
              </returnValue>
            </operation>
          </parameter>
          <parameter type="complex">
            <operation name="equal">
              <parameter type="simple" source="internal" value="Location" />
              <parameter type="simple" source="fixed" value="aLocation" />
              <returnValue>
                <case value="true" return="true" />
                <case value="false" return="false" />
              </returnValue>
            </operation>
          </parameter>
          <returnValue>
            <case value="true" return="true" />
            <case value="false" return="false" />
          </returnValue>
        </operation>
      </parameter>
      <parameter>
        <operation name="subStr">
          <parameter type="simple" source="internal" value="ID" />
          <parameter type="simple" source="fixed" value="0" />
          <parameter type="simple" source="fixed" value="3" />
          <returnValue>
            <case value="true" return="true" />
            <case value="false" return="false" />
          </returnValue>
        </operation>
      </parameter>
      <returnValue>
        <case value="true" return="Ypresent" />
        <case value="false" return="Npresent" />
      </returnValue>
    </operation>
  </context>

  <context name="Meeting" RID="2">
    <operation name="And">
      <parameter type="complex">
        <operation name="equal">
          <parameter type="simple" source="internal" value="context1" />
          <parameter type="simple" source="fixed" value=" Ypresent" />
          <returnValue>
            <case value="true" return="true" />
            <case value="false" return="false" />
          </returnValue>
        </operation>
      </parameter>
      <parameter type="complex">
        <operation name="GT">
          <parameter type="simple" source="internal" value="context2" />
          <parameter type="simple" source="fixed" value="1" />
          <returnValue>
```

```
            <case value="true" return="true" />
            <case value="false" return="false" />
          </returnValue>
        </operation>
      </parameter>
      <returnValue>
        <case value="true" return="Ymeeting" />
        <case value="false" return="Nmeeing" />
      </returnValue>
    </operation>
  </context>
</rule>
```

K. The Context Toolkit Applications

Figure K.1 shows the components used by the context toolkit to build two applications [99]. These applications are In/Out board and Mailing list; where both applications use the presence widget. In the first application the user explicitly docks an 'iButton' sensor. The In/out board displays in/out status (green/red dot) of building occupation and also the time when the occupant enter or left the building. The widget uses the sensors event to detect user's status while getting in or out. This application can also use the radio frequency-based indoor location system sensor to automatically detect the presence of a user. Both technologies return the id information to indicate the detection of a user. Then an interpreter is used to convert id information into user's name. The given application detects the widget's status and update the In/Out board display. The second application is the context-aware mailing list, in which email messages are delivered only to the members of a research group who are present at a given location (e.g. a given building). In this application the same widget of the In/Out board is used, where the widget detects the presence of group members in a building. Then the application adds every user's name to the mailing list using a standard mail program.

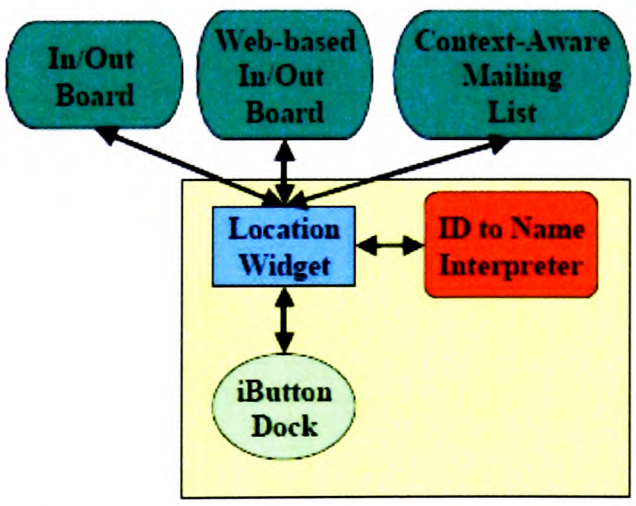


Figure K.1: The context toolkit In/Out Board and Mailing List applications using iButton docking sensor

L. The CADL Code for In/Out Status Application for the Context Toolkit

```
<?xml version="1.0" ?>
<Filters xmlns:an="http://www.w3.org/2001/XMLSchema-instance"
    an:noNamespaceSchemaLocation="CADLSchema.xsd">
    <events>
        <event id="a11" CID="a1">
            <description>In Out Board Application</description>
        </event>
    </events>
    <process type="Get" CID="a1">
        <class name="ID">
            <source type="xml">sensor1</source>
            <target type="xml">G1</target>
        </class>
        <class name="Time">
            <source type="xml">sensor2</source>
            <target type="xml">G2</target>
        </class>
    </process>
    <process type="Application" category="automatic" CID="a1">
        <class name="SOAPapplication" address=" ">
            <Element name="functionName" value="database" />
            <Element name="location" value="http://127.0.0.1/soap/soap_database_server.php"/>
            <Element name="parameter" value="path" type="STRING" fileType="xml">G1</Element>
        </class>
    </process>
    <process type="Format" CID="a1">
        <class name="FormatParameter" context="presence">
            <source no="1" type="xml">G1</source>
            <source no="2" type="xml">G2</source>
            <target type="xml">p1</target>
        </class>
    </process>
    <process type="Interpret" CID="a1">
        <class name="presence" RID="1">
            <source type="xml">p1</source>
            <target type="xml">p2</target>
        </class>
    </process>
    <process type="Manipulate" CID="a1">
        <class name="Store">
            <source type="xml">p2</source>
            <target>presence</target>
        </class>
    </process>
    <process type="Manipulate" CID="a1">
        <class name="Retrieve" time="10">
            <source>presence</source>
            <target type="xml">p3</target>
        </class>
    </process>
    <process type="Format" CID="a1">
        <class name="FormatIndirectContext" context="InOut_Status">
            <source no="1" type="xml">p2</source>
            <source no="2" type="xml">p3</source>
            <target type="xml">p4</target>
        </class>
    </process>
    <process type="Interpret" CID="a1">
        <class name="InOut_Status" RID="2">
            <source type="xml">p4</source>
            <target type="xml">p5</target>
        </class>
    </process>
</Filters>
```

M. The CADL Code for 'hasActivity' Application for SOCAM System

```
<?xml version="1.0" ?>
<Filters xmlns:an="http://www.w3.org/2001/XMLSchema-instance"
   an:noNamespaceSchemaLocation="CADLSchema.xsd">

   <events>
      <event id="a11" CID="a1">
         <description>An Activity Application</description>
      </event>
   </events>
   <process type="Get" CID="a1">
      <class name="ID">
         <source type="xml">sensor1</source>
         <target type="xml">G1</target>
      </class>
      <class name="Time">
         <source type="xml">sensor2</source>
         <target type="xml">G2</target>
      </class>
      <class name="Location">
         <source type="xml">sensor3</source>
         <target type="xml">G3</target>
      </class>
      <class name="Activity">
         <source type="xml">sensor4</source>
         <target type="xml">G4</target>
      </class>
   </process>

   <process type="Format" CID="a1">
      <class name="FormatParameter" context="LocatedIn">
         <source no="1" type="xml">G1</source>
         <source no="2" type="xml">G3</source>
         <target type="xml">p1</target>
       </class>
   </process>

   <process type="Interpret" CID="a1">
      <class name="LocatedIn" RID="1">
         <source type="xml">p1</source>
         <target type="xml">p2</target>
      </class>
   </process>

   <process type="Add" CID="a1">
      <class name="aggregate">
         <source type="xml">p2</source>
         <target type="xml">p3</target>
      </class>
   </process>

   <process type="Interpret" CID="a1">
      <class name="Attendance" RID="2">
         <source type="xml">p3</source>
         <target type="xml">p5</target>
      </class>
   </process>

   <process type="Format" CID="a1">
      <class name="FormatIndirectContext" context="hasActivity">
         <source no="1" type="xml">p5</source>
         <source no="2" type="xml">G2</source>
         <source no="3" type="xml">G3</source>
         <source no="4" type="xml">G4</source>
         <target type="xml">p4</target>
      </class>
```

```
    </process>

    <process type="Interpret" CID="a1">
        <class name="hasActivity" RID="3">
            <source type="xml">p4</source>
            <target type="xml">p6</target>
        </class>
    </process>

</Filters>
```

N. The XML rule for interpreting 'hasActivity' context

```
<?xml version="1.0" ?>
<rule>
  <context name="hasActivity" RID="1">
    <operation name="And">
      <parameter type="complex">
        <operation name="And">
          <parameter type="complex">
            <operation name="And">
              <parameter type="complex">
                <operation name="equal">
                  <parameter type="simple" source="internal" value="Time" />
                  <parameter type="simple" source="fixed" value="aTime" />
                  <returnValue>
                    <case value="true" return="true" />
                    <case value="false" return="false" />
                  </returnValue>
                </operation>
              </parameter>
              <parameter type="complex">
                <operation name="equal">
                  <parameter type="simple" source="internal" value="Location" />
                  <parameter type="simple" source="fixed" value="aRoom" />
                  <returnValue>
                    <case value="true" return="true" />
                    <case value="false" return="false" />
                  </returnValue>
                </operation>
              </parameter>
              <returnValue>
                <case value="true" return="true" />
                <case value="false" return="false" />
              </returnValue>
            </operation>
          </parameter>
          <parameter type="complex">
            <operation name="equal">
              <parameter type="simple" source="internal" value="Activity" />
              <parameter type="simple" source="fixed" value="anActivity" />
              <returnValue>
                <case value="true" return="true" />
                <case value="false" return="false" />
              </returnValue>
            </operation>
          </parameter>
          <returnValue>
            <case value="true" return="true" />
            <case value="false" return="false" />
          </returnValue>
        </operation>
      </parameter>
      <parameter type="complex">
        <operation name="GT">
          <parameter type="simple" source="external" value="p5.xml" />
          <parameter type="simple" source="fixed" value="2" />
          <returnValue>
            <case value="true" return="true" />
            <case value="false" return="false" />
          </returnValue>
        </operation>
      </parameter>
      <returnValue>
        <case value="true" return="anActivity" />
        <case value="false" return="NoActivity" />
      </returnValue>
    </operation>
  </context>
</rule>
```