
# *Extending CACI:*

## *Implementing Support Functions for the Context Producer and Designing a Security Framework*

Thesis for a Masters Degree in Telematics from the University of Twente, The Netherlands

By:
Tania Tariq

Graduation Committee

***University of Twente***
Dr. ir. M. J. van Sinderen
Dr. ir. Tom Broens

**February 2009**

# *Abstract*

Ubiquitous computing has gained considerable popularity in recent years due to the proliferation of mobile devices like laptops, mobiles phones and pocket computers. An emerging field in this regard is providing users with context-aware applications that are able to adapt themselves to their situations. The benefit of context-aware applications like smart homes is that they are able to sense information about the user's environment and react automatically based on the input that they receive. This allows the application designer to provide users with newer and better services.

Unfortunately the distributed nature of participating entities when combined with the requirements for gathering/sensing, storing, advertising and accessing contextual information had lead many developers to believe that a broad deployment of context-aware applications requires a supporting infrastructure. A novel proposal of such a supportive platform is the Context Aware Component Infrastructure (CACI) which aims to assist application developers by abstracting from the details of where and how the context aware information is acquired. The developer only needs to indicate what type of information he/she requires and CACI will do the rest. In CACI terminology sources that provide the context information are called 'context producers' whereas applications that use this context information are called 'context consumers'. Since CACI is still a work-in-progress, in this thesis we present our work on two outstanding issues related to CACI.

Our first contribution was in the design and implementation of support functions for CACI context producers. At the outset of this project CACI only provided support functions to context consumers. The second contribution is in developing a security framework for CACI that meets our requirements. The challenge to security for CACI lies not in the security algorithms or security platforms, a rich plethora of which already exist, rather in the selection of an optimal framework. It is widely believed that security must be an integral part of system design and not a later add-on. In this regard we have identified the different architectural components which will play a part in enabling security and specified their functional roles. As CACI edge closer towards reality, security becomes an important concern since any vulnerability in the system will limit its practical use. In order for CACI to succeed, it must safeguard the contextual information by providing properties like privacy, trust and availability.

# Table of Contents

## CHAPTER 4 *Support Functions for the CACI Context Producer*



## CHAPTER 5 *Securing CACI*



## CHAPTER 6 *A Security Framework for CACI*

## CHAPTER 7 *Related Work*



## CHAPTER 8 *Conclusions and Future Work*

# CHAPTER 1 *Introduction*

The arrival of mobile communication has brought with it a wave of change in the computing industry. This includes development of new types of applications which leverage on certain key features of mobile communication. Mobile and ubiquitous computing have heightened users' expectations with respect to the accessibility of information [1, 2], with users feeling that they should be able to access information at any time and at any place.

The current trend in this direction has been to take the provision of information to the users one step further by providing them with contextual information. The intent is to move from the traditional explicit information request model to a new model where information is supplied based on the context. Although context-aware computing came to the limelight in the 1990s, 'context' is a well known concept. Well known because since time immemorial humans have used the context of situations and circumstance to enrich their comprehension and experiences. Humans are 'context-aware' without even realizing it [1, 4], and unconsciously acquire large amounts of additional information, or context, from the situation and the environment around them.

A great deal of research is being carried out to replicate this kind of 'awareness' in mobile applications. However a pre-requisite for such application development is that context and context-awareness have to be formally defined. Researchers have put forth various definitions of context-awareness, for example as "*a term that describes the ability of the computer to sense and act upon information about its environment, such as location, time, temperature or user identity*" [19]. It is noteworthy that existing context-aware applications can be described as monolithic entities which take in input from users, context from various sources and processes it to produce an output. These can generally be classified into either [1, 3]:

- Office and meeting tools
- Tourist guides
- Context-aware fieldwork tools, or

◆ Memory aids.

## *1.1 Context-Aware Applications*

A central challenge in the development of context-aware applications is (a) the collection and (b) the manipulation of contextual information. As an evolving research area there are as yet no standards for data interpretation and manipulation. Context-aware applications have traditionally been built using ad hoc processes that limit the development to specific application domains and, as such, restrict the expansion and reuse across disciplines [1]. The need for reuse and increased flexibility has resulted in research focusing on providing a solution to this problem. Reusability and flexibility are quite important because the field of computer science evolves very rapidly and applications must be reusable and flexible for them to remain relevant [21].

Context-aware applications have traditionally been developed using one of a number of available techniques, for example, tight coupling and sensor abstractions [1]. In tightly coupled applications the sensors which were used to generate context were hardwired into the application. As such it was extremely difficult and highly labour intensive to reuse the application with a new set of sensors. The next generation of context-aware applications were 'sensor abstracted' applications. These applications, unlike the tightly coupled applications before them, allowed freedom from the limitations of hardwired sensors by using daemons or servers to abstract from the sensors. Abstraction produced a new overhead, i.e. the application needs to deal with a different interface for every sensor (server) that it had to communicate with. Despite the advantage of abstracting from the sensors, these applications were not proactive with regards to the collection of context.

### 1.1.1 Middleware and CACI

The use of abstractions led application developers to consider the benefits of middleware for context-aware application development. The use of middleware allows the application developer to abstract from the underlying complexities of a computing environment and can generally be classified into:

◆ Object-oriented middleware

◆ Event-based middleware

◆ Reflective middleware

◆ Message-oriented middleware, or

◆ Component middleware.

The work presented in this thesis has been carried out in the context of the CACI project [22]

which has conducted research to assess the viability of **component based middleware** for context-aware applications.

"*Component middleware is defined as a class of middleware that enables reusable services to be composed, configured, and installed to create applications rapidly and robustly*" [16].

Such middleware allow the applications freedom from platform dependency, delineate the boundaries of components and provide well defined interfaces for the interaction of various components.

CACI has defined two types of basic entities. The first known as context *consumers* retrieve and use context information, and the second known as context *producers* create and offer context information. A context consumer is usually a context-aware application while a context producer is usually a sensor. To summarize, CACI utilizes component middleware to provide context-aware applications with key support functions such as the binding between context-producers and context-consumers. Binding [37] refers to the association of a context consumer to a producer. It is through this association that the consumer receives context information.

## *1.2 Motivation and Objectives*

The CACI model utilizes component middleware for the collection and manipulation of context-information. Its goal is to simplify application development by providing applications with access to supporting infrastructure. This infrastructure provides binding between context consumers and context producers based on the consumers' requirements, as specified in a CACI specific descriptive language called CBDL (CACI Binding Descriptive Language).

At the start of this project the binding facility offered by CACI was incomplete as it did not provide services for context producers. The existing prototype lacked the capability to dynamically add context producers. Furthermore, it did not have any features which allowed context producers to advertise their services. The prototype needed to be extended so that it could support binding of such context producers based on their requirements and offerings respectively.

While binding and binding related functions lie at the heart of the CACI project, there were also a number of non-functional requirements such as generality, extensibility, performance and security that needed to be addressed. To date CACI had been designed such that most of these non-functional requirements were achieved to a reasonable extent. However security and issues related to security were not handled as yet. Security is necessary for context applications because the information used for providing context-awareness may be sensitive in nature. Experience has shown that security must be an integral part of system design and not a later add-on.

Since CACI has been designed as a light weight middleware for mobile battery operated devices the main challenge in securing CACI lies not in the development of new security algorithms or protocols (a rich plethora of which already exist) but in the development of a framework suitable for mobile resource constrained devices. The main obstacle to securing mobile wireless environments is that the security mechanisms need to be robust while being lightweight and energy efficient [24]. This is because mobile devices are generally very constrained in terms of available resources such as battery life and processing power.

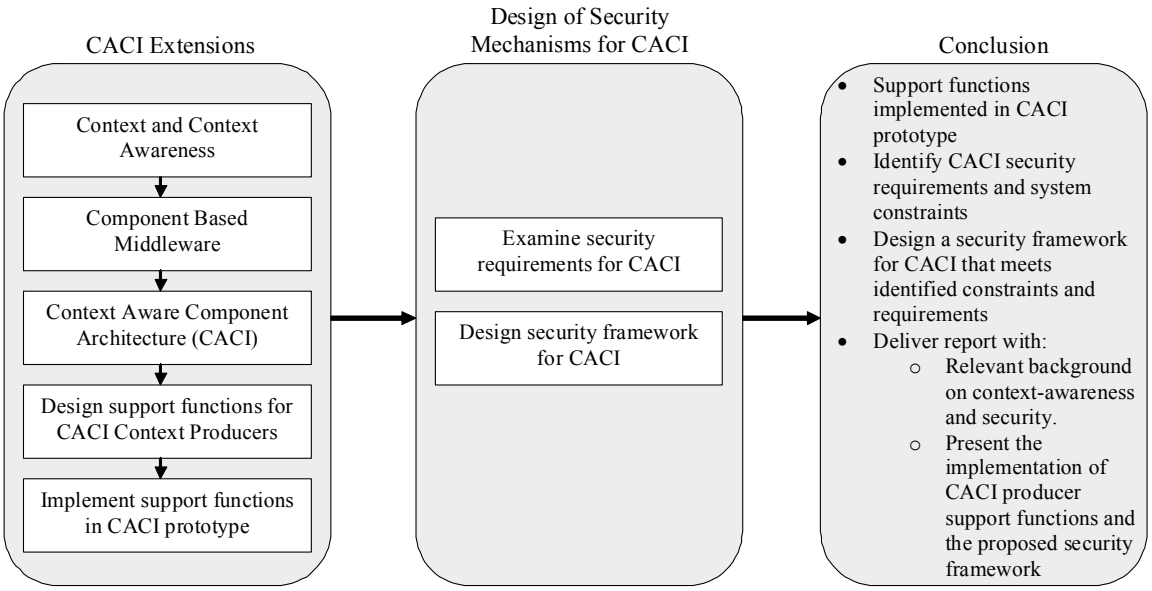
### 1.2.1 Approach

Summarizing, we have two main objectives in this work. The first is to design and implement an extension to CACI for providing services for context producers and the second objective is to design a security framework for CACI that is both robust and lightweight. At the conclusion of this project we should have:

- Researched context, context-awareness and CACI.
- Extended the CACI model to include support for context producers.
- Incorporated the extensions for the context producer into the CACI prototype.
- Identified CACI's security requirements and any related system constraints.
- Designed a suitable security framework for CACI that meets the above constraints.

A more diagrammatic representation of the structure of this project is illustrated in Figure 1.1.

**FIGURE 1.1. Project Structure**



## 1.3 Structure

We have structured our work in two sections.

*Section 1: CACI Context Producer*

Chapter 2 provides the background information for context-awareness and context-aware applications. This chapter includes a comparison of context-aware applications to date and highlights the advantages of component based context-aware applications.

Chapter 3 gives an overview of middleware and more specifically component based middleware. Specifically, it also gives an overview of CACI where we highlight the need for a producer component.

Chapter 4 highlights the design for the support functions for the context producer component and its integration with CACI.

*Section 2:Design of security architecture for CACI*

Chapter 5 gives a brief overview of security topics that are of relevance when designing a security framework for a distributed system like CACI.

Chapter 6 discusses the security requirements for CACI with the help of a use case. We also highlight the system constraints imposed on our design and discuss various approaches for securing CACI.

Chapter 7 gives a brief discussion of various related technologies.

Chapter 8 provides a concluding discussion on the project.

CHAPTER 2

# *Context and Context Awareness*

Although speech is the central feature of our interpersonal interaction, it is not the only tool that we use for communicating with our fellow human beings. We, as humans, have evolved a complex set of tools which we use to gather information in any given situation. When interacting with others we retrieve situational information from our environment, tone, inflection, body language etc. All of this additional information serves to better inform us about our situation. Application developers are working to emulate this awareness in computing.

Traditionally, computer applications have worked on an 'explicit information request' model i.e. where the applications have been provided with specific inputs in response to specific user requests. However, a move is being made gradually towards a more implicit model where applications will provide contextual information to users without them having to ask explicitly for it. This field of research is called context-awareness. The main premise of research in this domain is to create context-aware applications that provide the user with information based on the application's awareness of the user's environment.

## *2.1 Understanding Context and Context-Awareness*

Since researchers have typically defined context based on the contextual requirements of the application being developed they have overlooked the need to explain the concept of context in a more general sense. The following are some of the definitions of context and context awareness that have been put forth by various researchers.

◆ Schilit and Theimer, researchers at the Xerox Pao Alto Research Center, define context aware software as software that "*adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time*" [18]. Their definition of context and context awareness was made with the PARCTAB application in mind. Refer to section 2.3.2 for a overview of PARCTAB.

The authors identify three important aspects of context, which are (a) where you are (b) who you are with and (c) what resources are nearby. They also explained that context can extend to include factors such as lighting, noise level, network connectivity etc.

◆ Ryan, Pascoe and Morse, from the University of Kent at Canterbury, define context-awareness as "*a term that describes the ability of the computer to sense and act upon information about its environment, such as location, time, temperature or user identity*" [19].

◆ Dey and Abowd, researchers at Georgia Tech, defined context aware computing as "*an area of research where the human computer interface leverages knowledge of the user's context. Context includes, but is not limited to, information the user is attending to, emotional state, focus of attention, location and orientation, date and time of day, objects and people in the user's environment*" [20].

It is clear that the afore mentioned definitions of context loose clarity outside of the scope of the applications for which they were intended. This was realized by Dey and Abowd [4] who proposed the following definition:

*"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."*

Furthermore, Dey and Abowd defined context-awareness as:

*"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."*

This is our definition of choice when referring to context and context-awareness for the remainder of this report.

## *2.2 Evolution of Context-Aware Applications*

Context-aware applications have undergone an evolutionary process with advancements in the complimentary fields of mobile and wireless communications and the decrease in cost of sensors. The first generation of context-aware applications were developed with tight coupling between the application and the sensors [1]. As such the sensors which these applications relied on for contextual information were typically hardwired into the application. Consequently these applications were both quite complex to create and highly inflexible.

The subsequent generation of context-aware applications tried to solve the problems of the first generation by utilizing sensor abstractions [1]. Thus these applications did not deal with the sensors directly but interacted with a server or a daemon which was responsible for that

sensor. This generation of applications therefore, had more flexibility than the first. However inspite of an increase in flexibility, the task of creating an separate interface on a server for each sensor was still quite cumbersome.

The use of middleware is a well known means of addressing the issue of heterogeneity in a distributed computing environment. Middleware too has undergone an evolutionary process which has resulted in a number of middleware, each better suited to a specific situation. Middleware provide different levels of abstractions which provide network, platform and machine independence. Middleware such as CORBA, and JAVA provided application developers with a number of interfaces. The use of such interfaces frees the application developers from network or platform details, allowing them to focus on features such as the storage of context and notifications in the changes of context etc. This has also decreased the complexity of context-aware applications. Since providing flexibility and scalability are central to the design of middleware; their use in the development of context-aware applications has also increased their flexibility and scalability. Researchers are also looking towards creating middleware which are specifically tailored to handle the needs of context-aware applications.

## *2.3 Context Aware Applications*

The following are brief introductions to some first and second generation context aware applications:

### 2.3.1 The Active Badge System

The Olivetti Research Lab created the Active Badge System in 1992. This system provides a mechanism for locating people within a building with the help of special badges. These badges emit unique infra-red signals periodically which are picked up by sensors placed at various locations within the building. The location of personal can be pinpointed based on the readings of these sensors. The active badge system is one of the earliest context-aware applications [1, 3, 6, 10].

### 2.3.2 The PARCTAB System

This system was created by the Xerox Palo Alto Research center in 1993. The central goal behind this system is to evaluate the viability of mobile computers in an office environment. To this end, the PARCTAB system provides its users with a handheld mobile computer that interacts with workstation-based applications wirelessly via infrared transceivers [3, 7, 10].

### 2.3.3 Cyberguide

A group of undergraduate students at Georgia Tech developed several prototypes of the Cyberguide between 1995 and 1997. The Cyberguide aims to provide context-aware tour guide information to visitors at the GVU (Graphics, Visualization and Usability) Center at

Georgia Tech. Each of the visitors are provided with a handheld device which displays a map of the laboratory. Various areas of interest and information related to these areas are indicated on the maps. As the visitors moved about the lab, the mobile device provides them with information relevant to their current position. Incidentally, 'tour-guide' applications have since become the most commonly available form of context-aware applications [1, 3, 8, 10].

### 2.3.4 CoolTown

In 2001 Hewlett Packard Laboratories presented the idea of a ‘web-enabled world’ in which every device is connected to the Internet. In the CoolTown project, every person or device is represented as a web page that is consistently updated to reflect changes in the user's environment. Users are provided with updated information about their environment with the help of sensors and infra-red badges [9, 10].

## *2.4 Context Aware Middleware*

The following are explanation of some of the context-aware middleware available to application developers. These middleware have been used to build various context-aware applications with limited success.

### 2.4.1 The Context Toolkit

The Context Toolkit was proposed by Dey and Abowd in 1999. This toolkit aims to provide developers with an architecture that will support the development of context aware applications. The central focus of the toolkit is to separate the context acquisition process from the use of context within an application. To this end, the toolkit provides three abstractions, called widgets, servers and interpreters, which deal with the acquisition and delivery of context. [1,3]

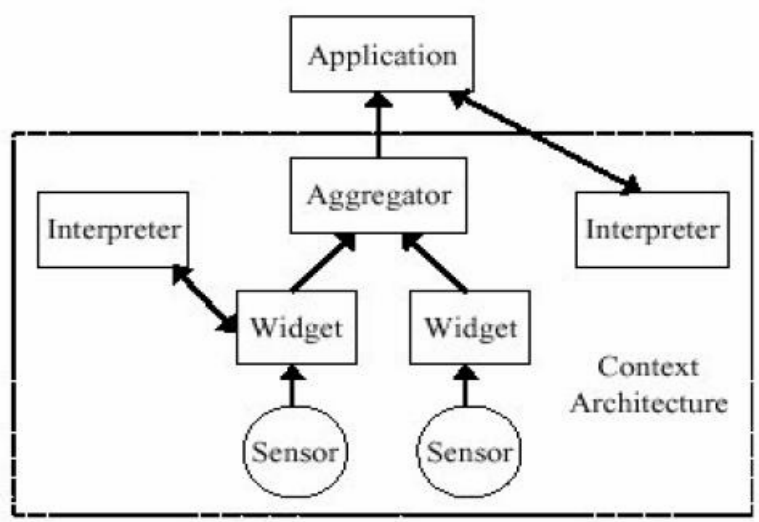
◆ **Context widgets** are software components that provide applications with access to context sensed from their operating environment. They free applications from the context acquisition process by hiding the complexity of the sensors from applications. Each widget encapsulates state and a set of event callbacks. The state is comprised of contextual information that applications can exploit via polling or subscribing. Callbacks represent the types of events that the widget can use to notify subscribing applications. The widget also maintains contextual state allowing other components to retrieve historical context information. [10]

◆ **Context servers** are used to collect the entire context about a particular entity, such as a person. The context server is responsible for subscribing to every widget of interest and acts as a proxy to the application, collecting information for that particularly entity. The context server can be seen as a compound widget. As such, it has attributes and callbacks, it can be subscribed to and polled, and its history can be retrieved. [10]

◆ **Context interpreters** are responsible for implementing the interpretation of context information. They transform between different representation formats or merge different context information to provide new representations. [10].

The above mentioned abstractions are implemented as a set of Java objects. Please refer to Figure 2.1 for a diagrammatic representation of the Context Toolkit architecture.

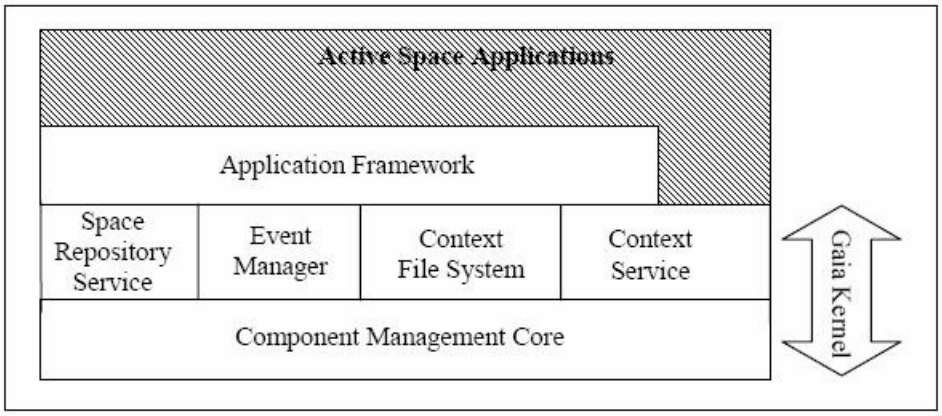
**FIGURE 2.1. Context Toolkit Architecture**




### 2.4.2 Gaia

The Gaia project was developed at the University of Illinois in 2002. It is designed to provide tools for the creation of context-aware applications in smart environments such as smart homes and offices. It provides a set of core services in addition to a framework for the construction of distributed context-aware applications [11].

FIGURE 2.2. **Gaia Architecture**




The Gaia architecture is comprised of the Gaia kernel, the Gaia Application framework, and the applications (see Figure 2.2). The tools required for the development of active space environment applications are provided by the Gaia Application Framework. All other applications and non-kernel services which are running in the active space are housed in the active space application layer.

The Gaia kernel represents the minimum functionality that is required to activate a physical space. A physical space is defined as "*a geographic region with limited and well defined physical boundaries, containing physical objects, heterogeneous networked devices, and users performing a range of activities*" [12]. The Component Management Core and four basic services, namely the Event Manager Service, the Context Service, the Space Repository Service and the Context File System, make up the kernel. The component core and its supporting services are implemented with the help of middleware platforms. Currently, Gaia has been implemented in CORBA.

### 2.4.3 A Service Oriented Context Aware Middleware (SOCAM)

This middleware was proposed by researchers at the National University of Singapore in 2004. The central aim of this middleware is to provide an architecture for the rapid prototyping of context-aware applications. SOCAM converts different physical spaces where context is acquired into a semantic space where context-aware applications can share and access the context easily.

The SOCAM architecture is comprised of the following key components [13]:

◆ **Context Providers** are responsible for the acquisition of context from a variety of internal and external sources. Furthermore, the acquired context informations is converted to OWL so that they are accessible by other service components as well.

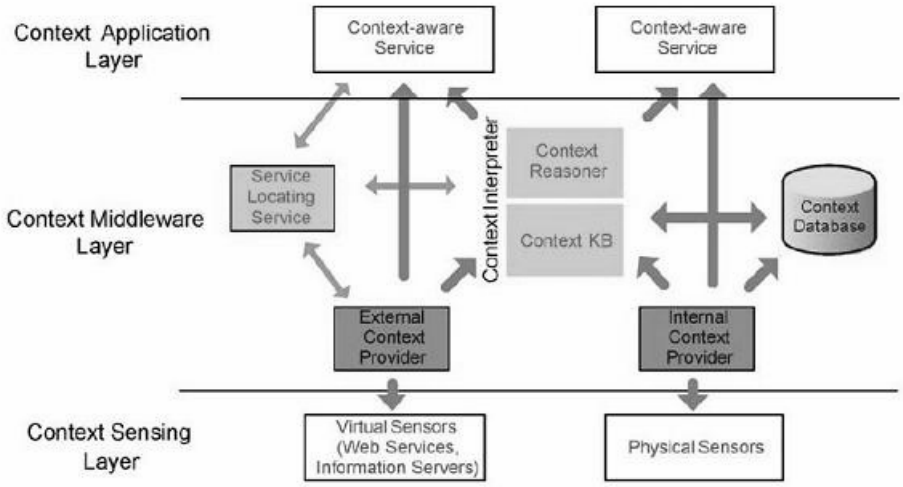
◆ **Context Interpreters** are responsible for providing the logic that is necessary to process the context information.

◆ **Context Databases** are databases of context ontologies and past contexts

◆ The **Service Locating Service** is responsible for providing a mechanism which allows users and context-aware applications to locate the various services which are advertising context.

The SOCAM components have been implemented in JAVA. Please refer to Figure 2.3 [13] for a diagrammatic representation of the SOCAM architecture.

**FIGURE 2.3. SOCAM Architecture**




### 2.4.4 The PACE Middleware

The PACE middleware was presented by Henrickson et. al. in 2005 as a result of research conducted under the PACE project. The main components of this middleware are: the context management system, the preference management system, the programming toolkit and the messaging framework.

The context management system, as the name suggests, is responsible for the management of context-information. In addition to performing query evaluation, this system is responsible for the aggregation and storage of context. The context management system consists of a number of repositories, each of which is maintains a catalog containing the fact type and situation definitions of context models.

The preference management system builds on the functionality of the context management system. The main role of the preference management system is to provide storage of user preference information and the evaluation of preferences to determine which application

actions are preferred by the user in the current context. The preference information of various applications is stored in one or more preference repositories.

The programming toolkit is responsible for providing the tools necessary to facilitate the interaction between the application components and the context and preference management systems. Finally, the messaging framework facilitates the remote communication between the components of context-aware systems.

### 2.4.5 The Java Context Awareness Framework (JCAF)

The Java Context Awareness Framework was proposed in 2005 by Jacob Bardram from the Center of Pervasive Computing at the University of Aarhus, Denmark. The central aim behind this proposal is to provide programmers with a simple framework that can be used in the development and deployment of context-aware applications. The framework provides a Java API that can be used to develop specialized context-aware applications. It is designed to support different implementations.

JCAF is comprised of two key components: a *Context-awareness Runtime Infrastructure* and a *Context-awareness Programming Framework.* The central design features of the runtime environment include providing the programmer with services that can be used in a distributed and cooperative manner. It also aims to provide the user with an infrastructure that is geared to react to the changes that are inherent to context applications. The provision of security has also been taken into account be the runtime infrastructure. The central aim of the programming framework is to provide the application developer with tools that make is easy to design and develop context-aware applications.

## *2.5 Assessing Context-Aware Applications and Middleware*

In order to fully appreciate the motivation behind this research project, it is beneficial to look at the key features of the above mentioned context aware applications. Karren Henricksen [11] has provided the following as guideline for the requirements for middleware for context-aware applications:

◆ **Support for hertrogeneity:** This is a key feature of middleware. Middleware must be able to provide support for legacy systems and a number of different underlying technologies and languages

◆ **Scalability:** This a very important feature for all development projects including middleware. The middleware must scale well if the number of variables increase. For example the middleware is expected to perform well if the number of sensors increase from 10 to 100.

◆ **Support for privacy:** The privacy of the information being relayed by the context-aware

application should not be compromised.
◆ **Ease of deployment:** Finally, the middleware should be easy to deploy by both experts and non experts.

Please refer to [11] for a detailed explanation of the above mentioned features.

◆ **Binding:** We have also added binding as one more requirement for the comparison of middleware. Binding [37] refers to the association of a context consumer to a producer. It is through this association that the consumer receives context information. Binding is a very crucial part of the context consumer and producer interaction. The application developer should be unaware of how the actual binding is created and maintained.

**TABLE 2.1. Comparison of the various Context-Aware Applications and Middleware**

| | Context Tool Kit | Gaia | SOCAM | PACE Middleware | JCAF |
|---|---|---|---|---|---|
| Support for Heterogeneity | X | X | X | X | X |
| Scalability | | X | X | | X |
| Support for Privacy | | X | | | X |
| Ease of Deployment and Configuration | | X | X | X | X |
| Binding | | | | | |

X - indicates features that are present in a specific middleware.

Table 2.1 looks at the features supported by the middleware from section 2.4. We can see that application developers have so far not focused on features that facilitates the creation of bindings between context consumers and context producers. This failure has been addressed by the Awareness team [22] with the development of the Context-Awareness Component Infrastructure (CACI). Please refer to Chapter 3 for a more detailed explanation of CACI.

CHAPTER 3

# The CACI Middleware

Distributed computing, leveraging on the pervasiveness of networked environments such as LANs and WANs, aims to enable interaction with services in a location independent manner and without concern for the underlying environment. This goal, however, is hard to realize in practice because of the heterogeneous nature of the underlying environments. The only scalable way to solve this problem is to provide developers with sufficient degrees of independence from the platforms, languages, machines and networks [14]. This independence is provide by a type of software known as middleware.

Middleware can be defined as "*a distributed software layer or platform that abstracts over the complexity and heterogeneity of the underlying distributed environment with its multitude of network technologies, machine architectures, operating systems, and programming languages*" [15].

As distributed applications have become more widespread, so too has middleware. Different types of middleware have been developed to provide support to different distributed environments. Middleware can typically be classified into object-oriented middleware, event-based middleware, reflective middleware, message-oriented middleware, and component middleware.

## *3.1 Component Based Middleware*

"*Component middleware is a class of middleware that enables reusable services to be composed, configured, and installed to create applications rapidly and robustly*" [16].

It has the following key features:

◆ It creates a virtual boundary around application component implementations allowing interaction only through well-defined interfaces.

◆ It defines standard container mechanisms that are needed to execute components in generic component servers.

◆ It specifies the infrastructure to assemble, package, and deploy components throughout a distributed environment.

When using component middleware the actual business logic of an application is implemented in the application's components. These components are housed within 'containers' and interact with the help of ports, interfaces, and connection points. The containers provide an execution environment for similar components. Finally, these containers and components can communicate via a middleware bus, and reuse common middleware services.

Next we present an overview of the OSGi component framework which forms the basis of the design and implementation of CACI.

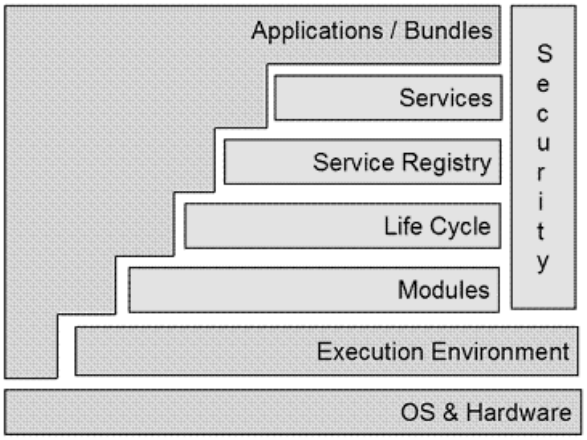
## *3.2 Open Service Gateway initiative (OSGi)*

The OSGi alliance is an open standards organization that has provided developers with a service oriented component based environment. It provides the specifications for a component framework and offers a standardized approach to manage software life cycles. The main aim of this specification was to provide software developers with a tool that would assist them not only in application development but, more importantly, also provide them with a standardized way to integrate existing components. The OSGi specification has been in use for a number of years now. Several implementation of the specification are available, for instance, Oscar, Knopplerfish, Equinox and Osxa.

### 3.2.1 OSGi Framework

OSGi applications are composed of various small reusable components that combine to offer a given functionality. This OSGi framework is responsible for handling the life cycle management of applications and components. It is a Java based technology distributed into bundles, which are basically applications packaged as standard Java Archive (JAR) files. Figure 3.1 shows a diagrammatic representation of the OSGi architecture.

FIGURE 3.1. **OSGi Framework**




As can be seen in Figure 3.1, the framework is divided into a number of layers.

◆ The execution environment is the specification of the Java environment.

◆ The modules layer contains the definitions of the class loading policies. The class loading model in OSGi based on Java but with some additions.

◆ The life cycle layer handles the life cycle management of bundles. The framework provides for the installation, the starting and stopping, updates and un-installations of bundles via this layer.

◆ The service registry is responsible for providing the model for the sharing of objects between bundles. It also defines a number of events which organize the arrival and departure of services.

◆ Security in the OSGi specifications is based on the Java and the Java 2 security model. This layer is responsible for enforcing the security requirements on the bundle at every other layer.

The OSGi alliance has designed a services layer on top of the framework. This layer provides many services which are specified via a Java interface. These services are implemented via applications and registered with the service registry. Once a service has been registered with the registry, clients of the service can find it. The following are examples of services being offered in release 4 of OSGi.

◆ Framework Services: These services are related to the functioning of the framework.

◆ System Services: Services that are required in almost every system have been implemented as system services.

◆ Protocol Services: In OSGi provisions have been made for the mapping of external protocols onto OSGi service. All such services are being offered under the protocol services grouping.

In addition to the above mentioned services a number of miscellaneous services have also been specified [23]. At this point, CACI has been tested on the Oscar and Knopflerfish OSGi implementations both on PCs and windows mobile devices.

## *3.3 Context Aware Component Interface (CACI)*

A typical context aware system has two types of entities. The first known as context *consumers* retrieve and use context information, and the second known as context *producers* create and offer context information. A context consumer is usually a context aware application while a context producer is usually a sensor. In order to exchange information, the context consumer must be bound to the context producer at run time.

Context-aware application have benefited from the creation of middleware which provide infrastructural support for the discovery and exchange of context information [11]. There is however no standardized way to support the creation of bindings between context producers and context consumers. These bindings are being created in existing applications by an explicit programming effort on the part of the application developer. The Context Aware Component Infrastructure (CACI) [22] aims to provide an infrastructure that will alleviate this problem. To this end, CACI shifts the responsibility of the establishment and creation of these bindings to the support infrastructure thereby freeing the application developer. It also provides application programmers a descriptive binding language that can be used to define context requirements on a high level of abstraction.

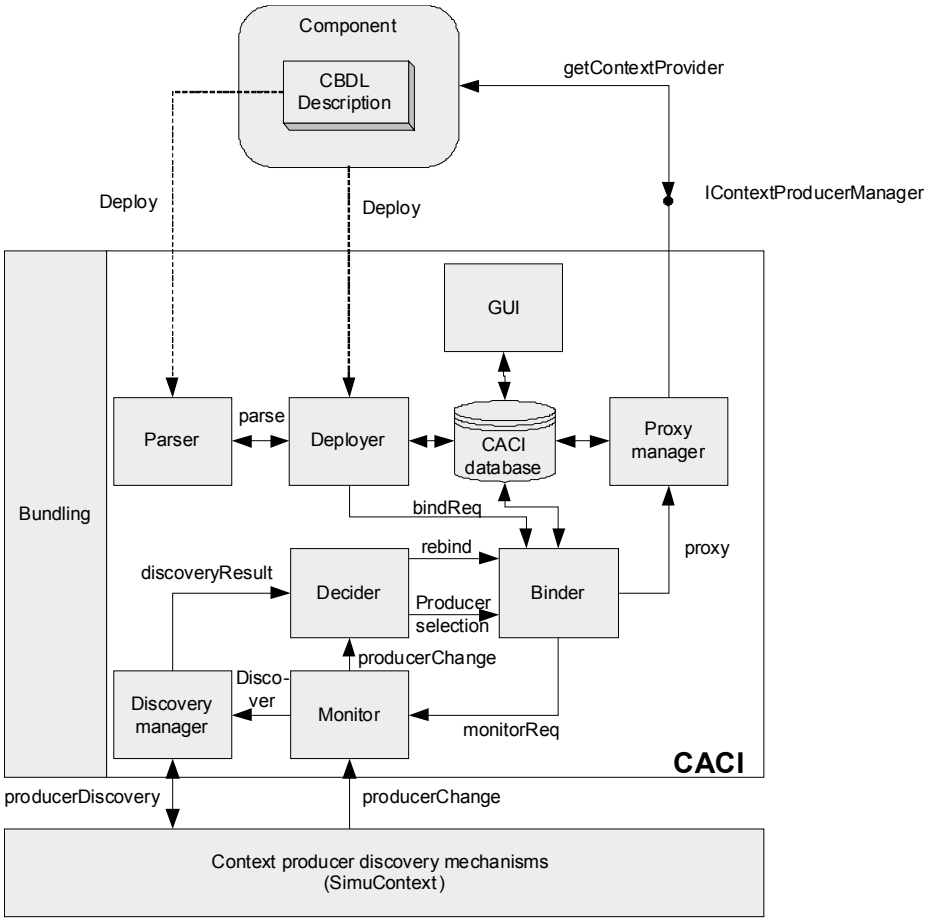
### 3.3.1 CACI Design and Implementation

CACI has therefore been defined as a client-side support infrastructure. It has been designed with a component oriented approach whereby both consumers and producers have been represented by components. These components have in-turn been defined inside a container with two ports namely, application ports and context ports. The application port is responsible for handling the functionalities being required or offered by the component whereas a context port is responsible for the context requirements or offerings of a component.

The context requirements for a component have been described with the help of a declarative language. This is an XML based language and is called **C**ACI **B**inding **D**escription **L**anguage (CBDL). CBDL allows application programmers to specify context requirements at a high level of abstraction. This in turn reduces the effort required on the part of the application programmer and helps in the development and maintenance of context aware applications. Refer to [38] for a detailed discussion on CBDL.

CACI has been based on the OSGi component framework and is bundled as a standard OSGi component. It has been implemented such that a virtual container is created within the OSGi container. This virtual container interacts with the deployed components to establish the binding. As a result it is lightweight and can be deployed on different implementation of OSGi. The binding mechanism lies at the heart of CACI. Refer to Figure 3.2 for a diagrammatic representation of the functional blocks of the CACI binding mechanism.

**FIGURE 3.2. CACI Functional Blocks**




The following is a step by step explanation of the consumer binding mechanism, as implemented in CACI.

◆ The deployer intercepts the component that is being deployed.

◆ Once the component has been deployed, the parser can extract the CBDL description from it. When the CBDL context request has been extracted and parsed, it is stored in the CACI database.

◆ The deployer then dispatches this request to the binder. The binder plays a key role in the consumer binding operation in CACI. Upon receipt of the request, the binder forwards it to the monitor. The monitor deploys a probe in the underlying discovery network. This is done so that the monitor can be informed of any changes. It is notified in the event of a producer's disappearance or the arrival of new context producers. In the former case a new discovery process is initiated.

◆ If the request is being sent to the monitor for the first time, it forwards the request to the discovery manager. The discovery manager conducts a search on the underlying discovery network for a context producer. Possible results of this search are sent to the decider which is responsible for selecting an appropriate context producer.

◆ Once the appropriate producer has been selected the binder creates a proxy. The context requesting component is "bound" to the proxy via a proxy manager. The proxy is in turn bound to the real context producer.

◆ The proxy manager is responsible for handling the binding between the requesting component and the proxy.

CACI does not support context producers. A simulation function is currently being used to simulate the presence of context producers. One of the central goals of this research project is to provide support for context-producers in CACI. Once the support for context-producers has been added, CACI will be able to react to context advertisements by context producers. The following table rates CACI against the guidelines discussed in the previous chapter.

**TABLE 3.1. CACI Features**

| Middleware Features | CACI |
|---|---|
| **Support for Heterogeneity** | X |
| **Scalability** | X |
| **Support for Secure communication** | |
| **Ease of Deployment and Configuration** | X |
| **Binding** | X |

CHAPTER 4

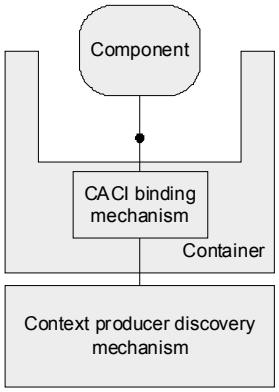
# *Support Functions for the CACI Context Producer*

This chapter will discuss the design of the CACI context producer which has been successfully implemented in the CACI prototype. The actual code is a significant part of the project but can not be shown here.

## *4.1 The CACI Binding Mechanism for Context Producers*

Since CACI is based on the OSGi component framework, the OSGi container has been extended to include a virtual container which represents CACI. This container interacts with the various components via the *IContextProducerManager* interface. The interface is responsible for the operations pertaining to the retrieval of and establishment with context producers. All underlying services which are required by CACI are deployed within the OSGi container as standard components. Please refer to Figure 4.1 for a diagrammatic representation of the high level implementation architecture of CACI.

**FIGURE 4.1. Implementation Architecture of CACI**



The CACI binding mechanism assumes the presence of a generic discovery mechanism. The current implementation of CACI uses the SimuContext repository [26] as the context producer discovery mechanism. SimuContext offers a library to simulate context producers. Developers can specify the characteristics of a producer that they wish to emulate via the SimuContext interfaces. Once a context producer has been created and registered via SimuContext it can be discovered by an application.

As explained in the previous chapter, CACI currently only provides support for binding requests initiated by context consumers. This limits the support that CACI can provide to applications developers who create both context consumers and developers.

## *4.2 Design Decisions*

One of the main functional requirement of CACI was that CACI should establish a binding between a contest consumer and a producer based on the requirements specified by the context consumer developer and the offerings specified by the context producer developer. Furthermore, the end user should remain unaware of the creation of such a binding.

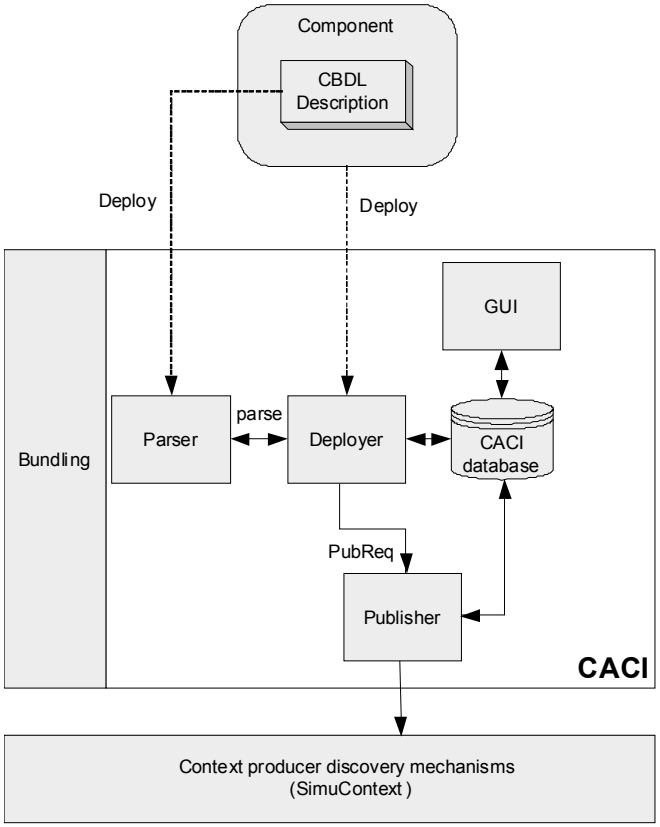
As the CACI consumer binding component has already been developed, any design decisions taken to develop the producer advertisement component have to be compatible with CACI's existing architecture. Therefore, the CACI producer advertisement architecture closely follows the CACI consumer binding architecture.

Furthermore, a central design strategy of the CACI developers has been to ensure scalability and flexibility. To this end, adapters have been used where possible. The use of adapters makes it possible to plug-in new functions simply by ensuring that the new functions have compatible adapters. Adapters were, therefore, used in our design where possible.

## *4.3 CACI Producer Advertisement Architecture*

After initialization, CACI examines the component to check for the presence of a CBDL description. If a CBDL description does not exist the component is ignored by CACI. If a CBDL description does exist CACI parses this description and extracts the binding or publishing requests. The publishing requests, once extracted, are forwarded to the publisher which is responsible for conveying the advertisements to the underlying discovery mechanism. Each of these activities has been represented by a functional block within the CACI architecture. The CACI producer advertisement internal functional diagram is shown in Figure 4.2.

FIGURE 4.2. **CACI Producer Functional Diagram**




### 4.3.1 CBDL Description

The CACI component includes a CBDL description which specifies the context requirements of the component. In this way the CACI component is different from a standard OSGi component. This CBDL description is used by both context consumers and context developers to indicate to CACI their respective requirements.

A *CBDL Document* represents the context requirements of a component. This document can be used to specify any number of *binding* and/or *publishing* requests. The binding requests indicate that the component is a context consumer whereas the publishing requests indicate that the component is a context producer. These requests can both be simultaneously present in a CBDL description. In such a case the component is both a context consumer and a context producer.

The particulars of the binding/publishing requests are specified with the help of various CBDL tags. A typical publishing request is comprised of an *id,* a *producer entity*, a *context element* and a *context format*. The producer entity specifies who is producing the context information in the id field. The context element field is used to indicate a singular type of context information that is being offered. Finally, the format field comprises the kinds of formats that the advertised context information can be presented in. This field is flexible as the publisher may choose to use this field to specify formats. A sample publishing request is shown in

Figure 4.3.

**FIGURE 4.3. CBDL Sample Description**

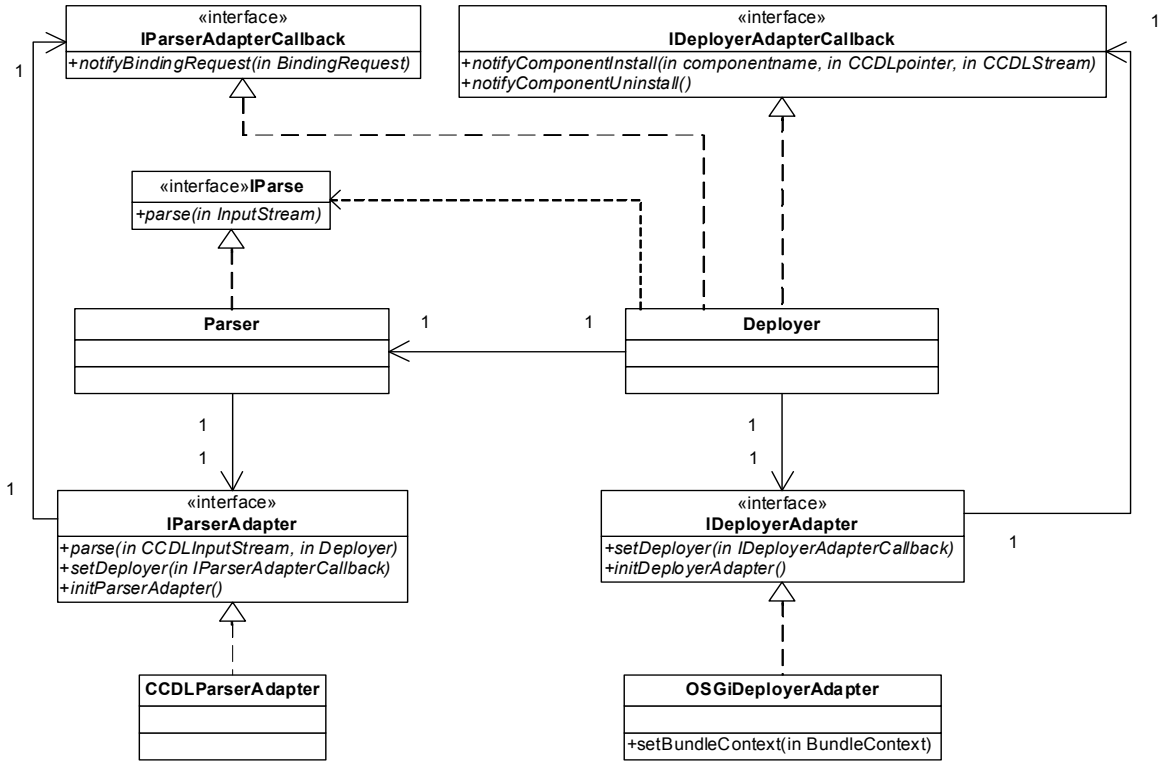
```
<?xml version="1.0" encoding="UTF-8"?>
<CBDLDocument
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="CBDL-schema.xsd"
UserID="Healthcarecentre"
ApplicationID="ESS_Healthcarecentre">

ContextAdvertisment PublishingID="Testbundle-Location">
     <Element>Speed</Element>
     <Entity>Tom</Entity>
     <Format>km/hr</Format>
</ContextAdvertisment>

</CBDLDocument>
```

### 4.3.2 Deployer and Parser

When a component is deployed in the CACI container, it will be intercepted by the deployer component. This is made possible by using the bundle event mechanisms implemented by the OSGi framework. More specifically a bundle listener event is deployed which monitors bundle events such as bundle installations, if a bundle is started, stopped etc. When a bundle is deployed a CACI binding event is instantiated.

The deployer then checks to see if a CBDL description has been included in this specific bundle. If a CBDL description is found; then it is forwarded to the parser. The parser is responsible for checking the validity of the document. It is also responsible for extracting any publishing requests that may be present in the CBDL document. The deployer also checks the ID of this publishing request to ensure that it is new in the system. The publishing requests are handed over to the publisher.

**FIGURE 4.4. The Deployer and Parser Adapter**

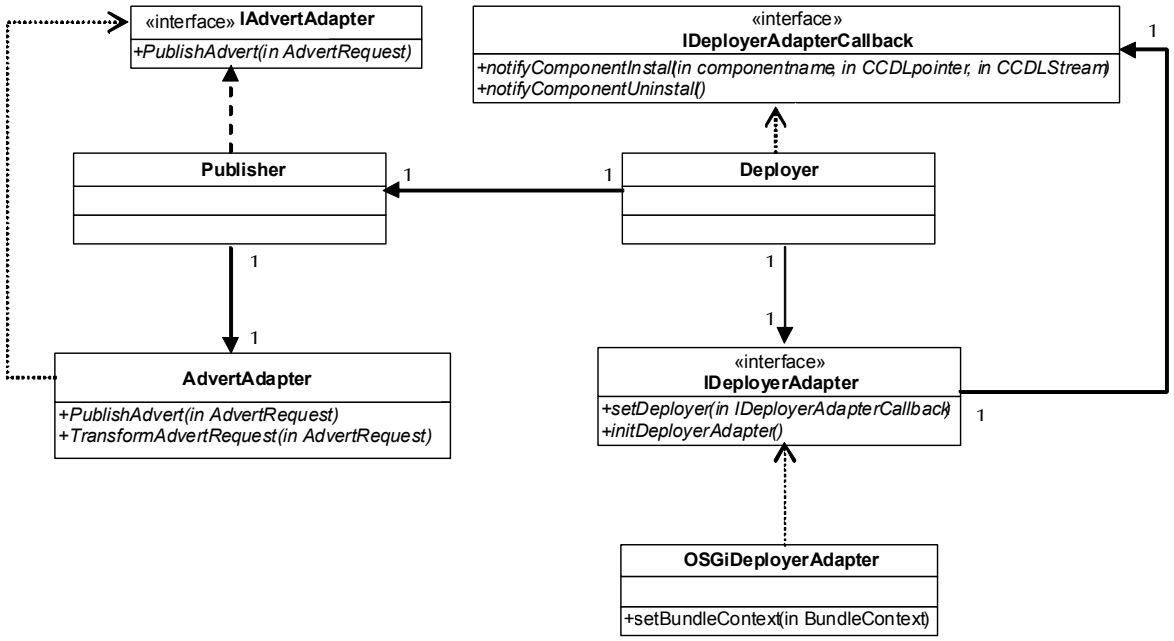



The deployer and parser functions have been implemented with adapters called the *IDeployerAdapter* and the *IParserAdapter* respectively (see Figure 4.4). A *OSGiDeployerAdapter* and a *CBDLParserAdapter* are also shown in Figure 4.4. The *OSGiDeployerAdapter* is responsible for notifying the deployer of any OSGi bundle events by using the notify method which is specified in *IDeployerAdapterCallback*. As the name suggests, the *CCDLParserAdapter* is responsible for using the notify function from *IParserAdapterCallback* to notify the deployer of the CBDL descriptions that it has parsed.

### 4.3.3 Publisher

The publisher lies at the heart of the support functions for context consumers. The publisher receives the publishing requests from the deployer and advertises these requests to the underlying discovery mechanism (See Figure 4.5). The new publishing request is also stored in a database. Adapter classes were also used to implement the publishing functions in CACI. The *AdvertAdapter* and its interface, the *IAdvertAdapter* were created to assist the transference of the publishing request to the discovery mechanism. The presence of the adapter allows the flexibility of changing discovery mechanisms. Please refer to the appendix for a detailed look at the classes associated with the publisher and relevant adapters.

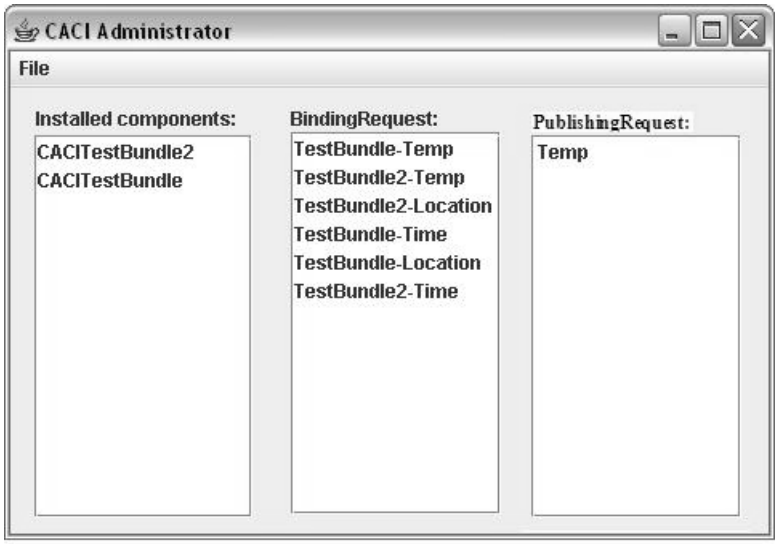
FIGURE 4.5. **Deployer and Publisher Adapters**




### 4.3.4 CACI Database and GUI

The CACI database is a repository of information that is used for administrative purposes. Information relating to the deployed components, various binding and publishing request are stored in it. Currently, the database is implemented in memory but future extensions envision a move towards more sophisticated implementations such as relational databases. Additionally, the current CACI GUI also relies on this database as it is a visual representation of the information stored in it. Figure 4.6 shows a view of the CACI GUI. Future work on CACI may include a more complex GUI.

FIGURE 4.6. CACI GUI




The support functions for context producers are therefore made possible with the above mentioned architecture. The actual coding and realization has remained true to the programming style that was used to program the context consumer functions. As mentioned previously, the discovery mechanism used by CACI is SimuContext. Future extensions of CACI should include testing CACI with different discovery mechanisms. Additionally, CACI needs to build in support for the more complex case of where an entity, and therefore its requirements, can switch roles i.e. it can take on the roles of both a consumer and a producer.

CHAPTER 5

# Securing CACI

The problem of security in the context of CACI can be summarized as the need to *ensure the secure exchange of contextual information between distributed context producers and context consumers*. Such a problem falls within the domain of network security. In this chapter we will focus on the different security concepts within this domain as well as how these concepts are implemented using cryptographic techniques.

## *5.1 Wired vs. Wireless Networks*

Before we look at the problem of security in more detail, we would like to address one important question: Will CACI be deployed on a wired or wireless network? The reason this question is important is because securing a wired network is typically simpler than securing a wireless network. Wired networks offer security based on physical access which is enough in most cases. For example, a home where CACI enabled sensors and corresponding consumer applications are deployed in a wired setting. The inherent access control based on physical protection of the network means that since only authorized users have access to the network (someone with a key to enter the house) network security does not need to be specifically addressed. Even if this network is connected to an insecure network like the Internet, security can still be maintained using suitably configured firewalls at network entry points.

Unfortunately the more general setting envisioned for CACI is based on wireless connectivity. This is because it is often impractical to require devices like sensors to be connected to the network using wires. Not only does this require a lot of access points but will also impact the mobility of the sensors. However the broadcast nature of wireless networks means that transmissions can be heard by anyone within range. Similarly illicit transmissions can be received by legitimate devices and may even be used for denial of service attacks.

## *5.2 Attacks: Passive and Active*

The main aim of security is to prevent attacks. Attacks are deliberate attempts to compromise the security of a system, usually exploiting weaknesses in the design or operation. Attacks against wireless communication are usually classified into two types: passive and active. Passive attacks attempt to learn information that does not affect the operation of the system e.g. eavesdropping on message content and traffic analysis. They are thus a threat against the confidentiality and anonymity of the communication. In section 5.3.3 we will introduce the concept of encrypting traffic to ensure confidentiality and in section 5.3.8 we will talk about anonymity.

On the other hand active attacks attempt to alter system resources or affect their operation by modifying and injecting packets into the network. Examples of active attacks are masquerading, replay, modification and denial of service. Section 5.3 will discuss the various mechanisms which are used to defend a framework against passive attacks. Passive attacks are typically more difficult to detect than active ones.

To summarize, it is much easier for unauthorized users to gain access to a wireless network in order to launch both active and passive attacks because:

- ◆Eavesdropping is easier.
- ◆Injecting fake messages into the network is easier.
- ◆Replaying previously transmitted messages is easier.
- ◆Launching Denial of Service (DoS) attacks is easier.

Next we look at some of the building blocks typically used to secure wireless communication. We will also give example of the commonly used security mechanisms to create these building blocks.

## *5.3 Security Building Blocks*

The next step in our design process is identifying the types of security that a typical system requires. Under Information Assurance (IA) security functions can be classified as: confidentiality, integrity, authentication, availability, and non-repudiation [42]. However it is important to note that a secure system does not necessarily need all of these features in place. The eventual implementation of some or all of these security functions is dependant on the needs of the system.

**5.3.1 Availability:** Availability means that the intended network services are available to legitimate parties when required. The threat to availability is called Denial of Service (DoS). It

should not be possible for an adversary to disable a node or a service given by a node in the network. Denial of service attacks on wireless ad-hoc networks can be of many different types. On the physical and media access control layers an attacker can use jamming techniques to interfere with communication [27]. On network layer, a malicious node participating in the network can easily pollute the routing infrastructure by injecting fake messages [28]. On higher layers, the attacker could interfere with service discovery etc.

In general, it is hard to completely protect distributed devices against denial of service attacks. Physical layer attacks based on jamming can be resisted using mechanisms such as spread spectrum [27]. Other DoS attacks target finite resources e.g. fake authentication attempts in order to cause battery exhaustion. Such attacks can be resisted by lowering the cost of authentication. Furthermore introducing priority rules on the tasks performed can give some level of protection against battery exhaustion.

**5.3.2 Authentication:** Authentication allows the communicating parties to be assured of the each others identity. In computer security, authentication is performed by proving knowledge of a cryotpgraphic secret. Without authenticity it is difficult to provide availability, integrity and confidentiality because an attacker would be able to impersonate a legitimate node. The challenge in providing authentication in a distributed environment such as that envisioned for CACI is in *key management*. We would like previously unassociated CACI producers and consumers to authenticate each other seamlessly. Creating such credentials in a transparent and automatic manner is a challenging task.

Authentication protocols based on challenge-response are the simplest and most widely used authentication protocols today. In such a protocol, Alice issues a challenge to Bob whose response proves that he knows a valid cyptographic secret. When authentication is mutual (i.e. mutual-authentication), Bob will also challenge Alice who must in turn prove her knowledge of a valid cyptographic secret. Other well-known authentication mechanisms are based on biometrics.

**5.3.3 Confidentiality:** Confidentiality ensures that the information transmitted between two parties can only be accessed by the intended receiver. Wireless transmission is easy to eavesdrop so messages traveling over the wireless medium must be protected.

In wireless communication confidentiality is typically enforced using encryption ciphers such as The Data Encryption Standard (DES), the Triple DES (3DES) and the Advanced Encryption Standard (AES) [27]. Note however that the confidentiality of *stored* information is a different matter altogether and is related to the problem of *tamper resistance*. The use of tamper resistant modules to store sensitive information is well known, but their widespread use is discouraged due to increased cost of devices.

**5.3.4 Integrity:** Integrity ensures that information traveling between two devices has not been altered or falsified without detection. The integrity of communication is provided using keyed-

Hash Message Authentication Codes (HMAC) such as MD5 and SHA-1 (section ).

**5.3.5 Authorization:** Given the insecure nature of wireless communication it is important that access to network resources should only be provided to legitimate entities. Authorization typically follows authentication and is this closely linked with it.

Authorization is generally provided using access control based on policies. Although the Policy Enforcement Point (PEP) [29] is almost always the service equipment, functionality related to the Policy Decision Point (PDP) [29] can be out sourced to a trusted third party. This is usually done in large centralized environments such as corporations where all policies reside and are evaluated at a central entity. Having a single point of control reduces the overhead of management and means that any changes made at the central entity are immediately reflected in the network.

**5.3.6 Replay detection:** Replay detection ensures the freshness of the messages received over the wireless link. It is usually carried out by adding sequence numbers or time stamps to transmitted messages. Duplicate messages are detected based on the repeated sequence number or expired time stamps and are then discarded.

**5.3.7 Non-repudiation:** Non-repudiation is the concept of ensuring that the sender or receiver of a message can not deny having sent it or received it. It is the opposite of plausible deniability. Non-reputation can only be done using cryotpgraphic techniques that are asymmetrical. Consequently the most popular way to guarantee non-repudiation in computer communication is to use digital signatures (section 5.4.5 ).

**5.3.8 Anonymity:** Often a basic expectation from the user is that sensitive information related to him should not be revealed without his consent. Sensitive information does not only includes the identity of the user but also things *linked* to his identity such as his geographical position.

In the next section we provide an introduction to cryptographic algorithms and protocols used to provide the main security services described above.

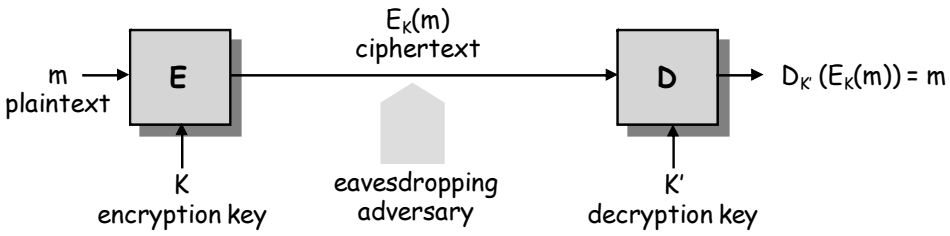
## *5.4 Cryptography*

The above security functions are usually implemented using cryptographic techniques. In computer communication cryptography is the process of modifying data using a secret known as a key such that it ensures some security properties. In this section we will introduce some relevant security mechanims based on cryptography. We will look at mechanisms for encryption, Hash function, Hash Message Authentication Codes and Digital signatures.

**5.4.1 Encryption:** When cryptography is used for encryption, the actual data is treated with a

mathematical algorithm (E) that uses the key (K) to transform the data (m) into an alternate form called the cipher text ($E_K(m)$). The ciphertext can be decrypted (D) back in the plain text using a correct key. An encryption algorithm is secure if it is extremely difficult, if not impossible, to revert back to the original information from just the cipher text i.e. without knowing the key. This is illustrated in Figure 5.1 .

**FIGURE 5.1. Encryption and Decryption**




A key is a piece of information, such as a password, that allows the holder to encode or decode information. As such a recipient of an encoded message must have the key or some related information to decode the message. Key based algorithms are usually categorized as asymmetric (section 5.4.3 ) or symmetric (section 5.4.2 ).

**5.4.2 Symmetric Encryption:** In symmetric encryption the same key is used for both the encryption and decryption of a message. Therefore in the figure above K = K'. Before an individual can send encoded information to someone, they must communicate the key to this person. This is because the recipient will be unable to decode the message unless he/she has access to this key. This form of encryption is very efficient and requiring minimal processing power. Key management is a major drawback of this type of cryptographic technique. If either the sender's or the recipient's key should get compromised, an intermediate person can decrypt and read the messages.

All encryption schemes can be classified into two basic categories. Those based on symmetric cryptography and those on asymmetric cryptography. When encrypting bulk data, symmetric encryption is preferred due to its lower overhead. The most popular asymmetric key encryption methods are several orders of magnitude slower than the best known symmetric key schemes.

The two main types of ciphers based on symmetric cryptography are those that process the plaintext in large blocks of characters and those that operate on individual characters of plaintext. The former are called block ciphers, and the latter stream ciphers. The exact cipher to use is beyond the scope of this document and depends on the type of information being protected. Reference [30] presents a detailed benchmark for a selection of block ciphers. The authors take into account the security properties, storage and energy efficiency of a set of candidates and arrive at a selection of block ciphers and operation modes.

**5.4.3 Asymmetric Cryptography:** Encryption in asymmetric cryptography takes place with

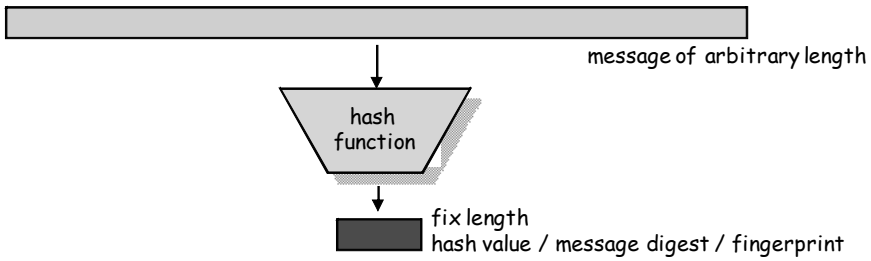
use of a pair of keys: the private key and the public key. Therefore in the figure above K!= K'. Furthermore it is computationaly infeasible to compute K' from K.

The public key is used to encrypt the message whereas the private key is used to decrypt it. In this scheme a sender can freely distribute his/her public key, however there must be a secure means of linking the public key with the identity of the person. One approach to preventing such attacks is by using a certificate authority. The certificate authority is a trusted third party which issues a digital certificate that has been signed stating that this key belongs to a particular person.

People wishing to send encrypted messages to the receiver can do so by using the public key of the receiver. The main advantage of asymmetric key cryptography is that as the key used for encryption is public, so there is no danger of the key being compromised. RSA, Diffie-Helman and El Gamal are examples of some popular public key algorithms.

**5.4.4 Hash Functions.** Hash Functions: Hash functions are a mathematical operation (usually ease to compute) that map a bit string of finite length in to a bit string of fixed length. This is illustrated in the figure below.

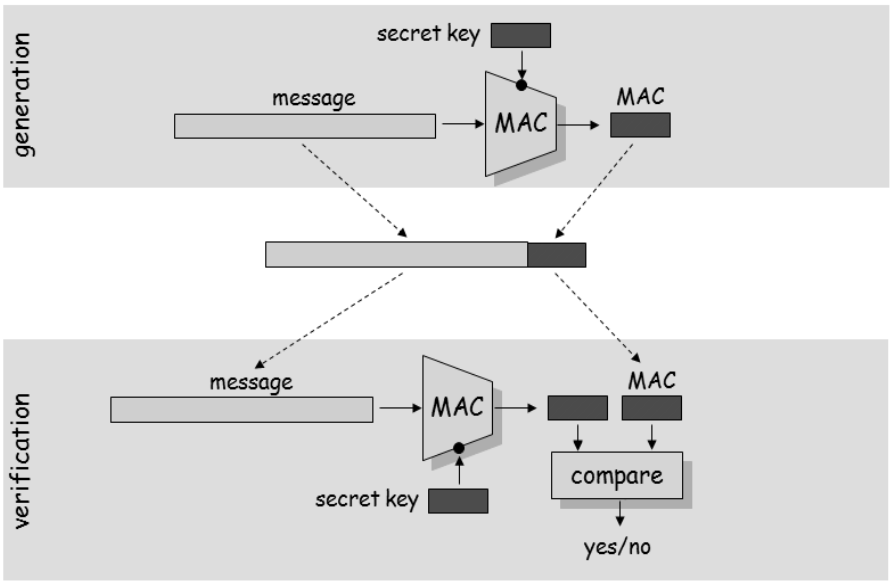
**FIGURE 5.2. Hash Function**




Although hash collisions due to the many-to-one mapping are unavoidable, finding such collisions of a given message is difficult. The hash of a message serves as a compact representative image of the message (like a fingerprint).

**5.4.4.1 Hash Message Autentication Codes (HMACs):** HMACs are hash functions that also accept key along with the message. Similar to hash functions, they also produce an output known as a Message Authentication Code (MAC) that is a bit string of fixed length. This MAC is appended to each message transmitted over the insecure channel. When the message is received by the destination it generates a new MAC using the shared key and compares it with the MAC included with the message. If the two MACs are identical it means that the message was sent by the correct sender and not modified in transit. This message is illustrated in Figure 5.3 .

FIGURE 5.3. HMACs




Given the mathematical properties of a hash function it is not computationally feasible to produce a correct MAC for a message without knowing the correct key. Therefore without knowing the secret key, attackers cannot modify the MAC of transmitted packets without detection, or generate new packets with a correct MAC. Consequently HMAC functions are used to implement data integrity and authentication.

**5.4.5 Digital signatures:** Digital signatures are based on public key cryptography and are created by signing the hash of a transmitted message with the senders private key. Anyone in possession of the senders public key can verify the authenticity of the message. Unlike HMACs, digital signatures have the additional property of non-repudiation. However the public key encryption scheme used for digital signatures is more computationally intensive than the symmetric key encryption scheme used for MAC generation, so digital signatures are not used to secure bulk data transfers. Digital signature schemes are either based on RSA, DSA (Digital Signature Algorithm) or ECDSA (Elliptic Curve DSA).

CHAPTER 6

# *A Security Framework for CACI*

CACI is a middleware that offers services to context consumers and producers in a distributed environment. Loosely coupled and physically distributed systems such as CACI rely on the transmission of messages and events that need to be secured before such systems can be widely deployed. The identity of entities involved, the authorization to access available services and the privacy and integrity of transmitted messages must all be established. Consider the typical example of a health monitoring application based on CACI. This context-aware application takes as inputs the medical data about a user and outputs an alarm to a health service if the heart rate goes above a certain threshold. We want to ensure that the information being provided by the application is only visible to the necessary individuals i.e. the doctors. Similarly, the doctors should have the assurance that the information has indeed come from the right person. Furthermore, we would also like to ensure that the information that is being transferred has not been tampered with in any way.

In this chapter we follow a systematic approach to designing a *security framework* for CACI. We begin by describing a typical use case for CACI that highlights its security requirements. Once we have matched these security requirements to the security functions of section 5.3, we describe the system constrains that must be kept into account when designing security for CACI. Next we analyze three generic security frameworks for distributed environments and select the one that best suits the requirements and constrains we identified earlier. Finally, we recommend a standardized security framework that corresponds to the generic model that we selected.

## *6.1 Use Case: Epilepsy Safety System*

**6.1.1 Overview:** Epilepsy is a chronic neurological disorder that affects millions of people around the world. People who suffer from epilepsy are typically plagued by unprovoked seizures. It is a disease which cannot be cured but can only be controlled by medication and in

some cases by surgery. Sufferers of epilepsy find that their quality of life suffers greatly due to the unpredictability of their seizures.

Research has shown that these seizures can be predicted by monitoring the various vital signs such as blood pressure, heart rate, brain activity etc. of a patient in question. The Epilepsy Safety System (ESS) leverages this knowledge to provide epilepsy patients with a system that can help improve their quality of life. The ESS utilizes context information such as the patient's current location and the location of the patient's care-giver to provide customized care. If a seizure is detected, the ESS takes appropriate actions such as directing the primary care givers to the patient's current position. The ESS also utilizes context information to conclude what to do should the primary caregiver be unreachable, whether to trigger first-aid assistance if the vital signs indicate only a mild seizure etc.

**6.1.2 Scenario:** Emily, who is an epilepsy patient, lives in Enschede. Upon discovering that the weather will be especially warm this weekend, she has decided to visit the beach in The Hague. Emily always travels with her PDA which is the local component of the ESS. That is, Emily's vital signs are monitored via a special bracelet that she wears. This bracelet communicated her vitals to the PDA which is responsible transmitting this information to her health monitoring service.

She takes the bus to the beach where she spends a few nice hours enjoying the sun. However, after a few hours the ESS realizes that her vitals are symptomatic of the onset an epileptic seizure. When the ESS detects the onset of a seizure it notifies the health monitoring service of Emily’s condition and her current location. This service in turn notifies a doctor in Emily's vicinity, Emily's primary physician and her family.

The doctor is provided with Emily's exact location so that he may provide her with the essential medical attention that she requires. Emily's primary physician in Enschede is also apprised of the situation (but not her location) because when Emily returns to Enschede he may want to alter her medicine or dosage. Finally, Emily's family may be informed (if so decided by Emily) so that they may be able to assist her if they wish.

**6.1.3 Analysis and Requirements:** This story illustrates how the ESS can be used to help a person such as Emily avoid a potentially life-threatening situation. Because her vital signs were constantly being monitored, Emily was not house bound and could go out and enjoy the weather. Furthermore, upon receiving indications that Emily might get a seizure, the health care providers were able to ensure that she had the required medical assistance that she needed on time. In essence the ESS being sensitive to Emily’s context is able to provide her with a better quality of life.

When considering security for such a scenario it is clear that issues such as privacy, trust, availability and anonymity need to be resolved. The following discussion attempts to highlight the need for each of these features.

***Privacy:*** It is important to realize that Emily's privacy needs to be protected. She needs to be sure that her vital signs are not sent to the wrong parties and unauthorized people should not be able to eavesdrop on such communication. Furthermore, even though Emily's doctor in Enschede is allowed to query her vital signs at any time he may not be allowed to query her location. For example in the ESS use case, the PDA communicated Emily's location to the health monitoring service but there was no need for other parties to be informed of Emily's location. Also it may be necessary to ensure that eavesdroppers are unable to see *whose* vitals are being monitored let alone *what* those vitals are. Thus the identity of the party may need to stay anonymous.

With respect to "Security Building Blocks" (see section 5.3), we can identify authentication, confidentiality, authorization and anonymity as the underlying security functions required to support privacy. In this work our focus will be on designing a security framework for CACI that supports our requirements for authentication and authorization. Once authentication is performed, other security functions such as confidentiality, integrity, replay detection etc. can be provided easily by leveraging on the resultant cryptographic keys and using any of the well researched encryption algorithms, for instance those mentioned in sections 5.3.3 and 5.3.4. In the above use case, anonymity can be provided by using virtual identities that can only be mapped to real world identities when presented with proper authorization. Since the exact mechanism to provide anonymity will vary from situation to situation, a in-depth framework for anonymity will not be a part of our generic security framework.

***Trust:*** Trust is another key feature of any context-aware system. When producers such as Emily generate context they need to be sure that the party consuming this context is trustworthy. Furthermore the consumer, such as the Health Monitoring Service (HMS), should be assured that the context information that it is receiving is indeed coming from Emily and not someone else.

With respect to the "Security Building Blocks" in section 5.3, we can identify non-repudiation, replay detection and integrity as the underlying security mechanisms required to support trust. The typical protocols used to support these security mechanisms have already been identified and are more of an implementation issue. Consequently they are not part of our generic security framework.

***Denial of Service:*** A critical application such as the ESS needs to be available at all times. A situation where the HMS or the doctor can not gain access to the ESS due to a denial of service attack must be avoided if at all possible. Therefore the security architecture must be designed such that the threat from denial of service attacks is minimized. Issues related to denial of service have already been addressed (see "Security Building Blocks" in section 5.3). Our design tries to reduce the cost of security operations performed at resource constrained CACI devices so that they are less vulnerable to denial of service attacks.

## 6.2 Security Requirements for CACI

As explained in the earlier chapters, the model proposed by CACI transitions through a number of steps before establishing the binding between a context consumer and a context producer. There are two distinct entities that handle context information in the context-aware domain, namely the context consumer and the context producer. In the scope of CACI, a context consumer arrives and states his/her context requirements in a context descriptor file. CACI is responsible for finding the right context producer based on this description. Furthermore, once CACI has discovered a suitable context producer, it will proceed to establish a binding between these two entities. The following is a discussion of the security requirements identified in section 6.1.3 in the light of the binding mechanism proposed by CACI. The next step will be to propose a high level security architecture which meets these requirements.

***Privacy:*** Privacy is defined as an individual's ability to allow, only those individuals whom they chose, access to their private information. A context producer can ensure his/her privacy by limiting the access it wishes to grant a context consumer. This can be accomplished in a number of ways such as allowing unknown context consumers' limited access to data, by limiting the amount of time access maybe allowed to a consumer, or by allowing only authorized users access. Privacy can be provided via the help of policies which specify who are the rightful recipients of sensitive context-information.

A context producer may not wish for his/her identity to be known to the users. In such a scenario the binding between the context producers and consumers need to be established so that the identities of the party wishing to remain anonymous is protected. Anonymity for the producer means that the producer will only reveal his real identity to authorized consumers. If the producer needs to provide his read identity it must be disclosed to consumers after they have successfully authenticated with the producer. In most cases the producer may wish to use virtual identities. Anonymity for the consumer mean that the producer does not need to know the real identity of the consumer-- only that the consumer is authorized to access this service. This can be implemented using for example authenticable virtual identities.

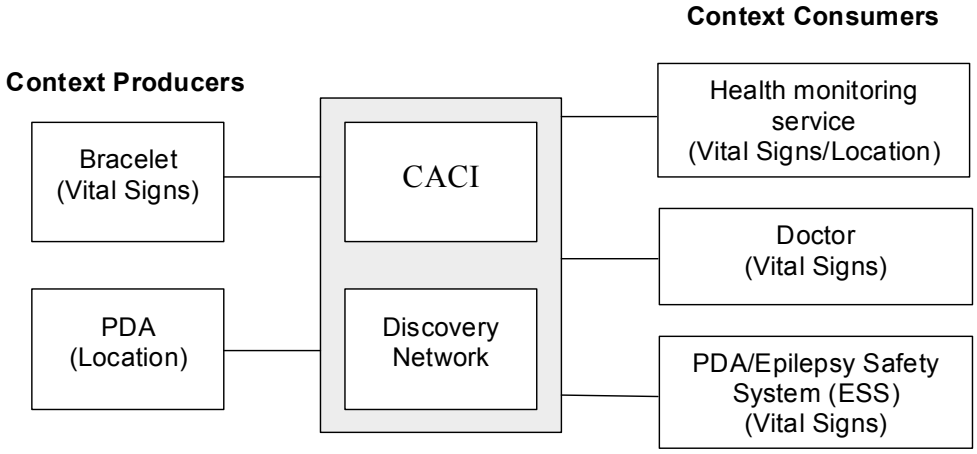
***Trust:*** Another security concern for context aware applications is that of trust. Trust implies that the context consumer can rely on the information provided to it from the context producer. Reciprocally, the context producer also trusts that the information that it is providing to the context consumer is not being disclosed or misused. Trust in a context producer will influence the binding between consumers and producers. For example, if a context consumer only trusts the data that is receiving from a particular source it can request CACI to bind it to only one specific context producer.

When referring to trust, with respect to context-aware applications, several subtopics are incorporated such as trust establishment and trust management. In order for two entities to trust each other, a relationship of trust or an understanding of trust needs to be established. The

entities continue to trust each other so long as this relationship is intact. It is therefore necessary to engage in trust management to handle the trust relationships between various parties.

The establishment of trust between a pair of unassociated producers and consumers typically involves the secure exchange of a secret. In section 6.4 we will look at the different mechanisms which can be used to create trust. Figure 6.1 identifies the context consumers and producers in the ESS use case.

**FIGURE 6.1. Use Case -- Context Consumers and Context Producers**




## *6.3 Non-Security Requirements for CACI*

The requirements for our security model also needs to incorporate certain non-security related constraints placed on our design.

**6.3.1 Usability.** The use case presented in section 6.1 makes it clear that CACI is meant to support applications that are being used by the average person. This means that it is important to protect CACI while maintaining good usability. A more usable system reduces the user involvement and simplifies it when it is necessary. This implies that processes should be self-organizing, where possible, intuitive and easy to use otherwise. Furthermore, the user should not be presented with complex security options the consequences of which he will not understand and therefore will not use them correctly.

**6.3.2 Constrained Devices.** CACI has been designed as a light weight solution for binding between distributed context consumers and producers. It is intended for use on devices like sensors which have limited resources. Devices such as biomedical sensors, digital bracelets, belt computers etc. may be constrained in their:

**Available Energy:** Most mobile devices are powered by small batteries. As users begin to carry around more and more electronic devices, there is a growing need for devices to last

longer without requiring a recharge. For example, resource constrained sensors must last months on a pair of AA batteries.

**Computing Speed:** In line with the limited available energy, some mobile devices are powered by minuscule processors with limited computing capabilities. Such processors, often supporting a minimal instruction set, are not suitable for performing complex mathematical operations like those required by asymmetric cryptography (i.e public/private keys) [43, 44].

**Limited Storage:** Although recent advances in memory technology mean that storage costs and dimensions have reduced substantially, mobile devices still have limited storage capabilities when compared to e.g. personal computers.
Cost: As the success of a device is proportional to its cost, any cost overhead of security, for instance through the use of specialized cryptographic hardware, must be relative to its benefit (and totally avoided if possible).

## *6.4 Generic Security Frameworks*

In this work, we will propose an Identity and Access Management (IAM) framework for CACI. The verification of a person's or application's identity is carried out through authentication whereas access control or the granting of access rights is realized through authorization. Once authentication is performed, confidentiality, integrity, and replay detection are provided by using the resultant session keys and any of the cryptographic algorithms mentioned in sections 5.3.3 and 5.3.4.

We have already stated that authentication between CACI producers and consumers needs to be performed using cryptographic secrets called keys. One of the challenges when designing security for a distributed environment, like CACI, is in the safe distribution of such keys which must be carried out prior to service access. In computer security this is known as the problem of *key management.*

Authorization is typically enforced using what are known as access control policies which detail the usage rights of various users or applications. Unless authorized through one or more access control policies, users have no access to any functions of the system. Access control is defined as the enforcement of specified authorization rules based on positive identification of users and the systems or data they are permitted to access. The exact types of policies used for authorization, for example, Access Control Lists (ACLs) or Role-Based Access Control (RBAC) [31] depends on the usage scenario and is beyond the scope of this work. What is important for us is that our proposed authorization framework must meet our usability requirements (see section 6.3.1).

Legacy applications have traditionally handled their own authentication and authorization. However the new trend in network communication such as SOA (Service Oriented

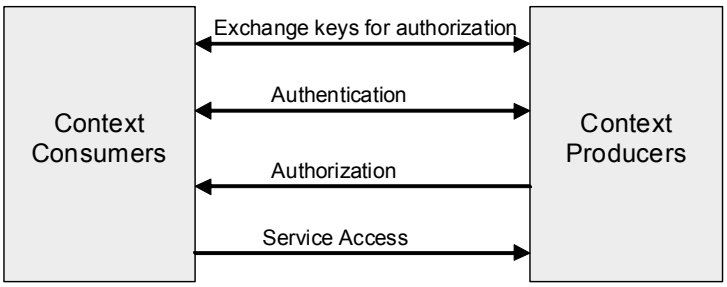
Architecture), Web 2.0 and Identity Federations are focused on centralizing both of these functionalities in central location. Not only does this ease administrative overhead but having one point of control also increases security because any changes such as user account deletion only need to be entered once for them to have a system-wide effect.

Broadly speaking, we can choose one of three possible security frameworks. Our security framework can require both authentication and authorization to require user assistance (we call this the fully user assisted approach), for the user to only configure the latter (we call this the partially user assisted approach) and for none of the two to require user assistance (we call this the minimally user assisted approach). Each of these three approaches has its benefits and drawbacks which we discuss in the next section.

### 6.4.1 Fully User Assisted Approach

The fully user assisted approach requires user interaction for both the authentication and the authorization steps that must precede the actual service access. Figure 6.2 shows the steps involved in a successful interaction between the context consumer and the context producer using a time sequence diagram.

**FIGURE 6.2. Fully User Assisted Approach**




Before authentication can take place the context consumer must have a valid key to identify itself to the context producer. For mutual authentication the context producer must also have a valid key. In the fully user assisted model the users must ensure the secure exchange of such keys. We will see that requiring the user to exchange the necessary keys manually, prior to each service access, is not very scalable or desirable.

The secure exchange of keys between two co-located devices using user assistance has been well researched. Typically it involves setting the devices in a certain state where they are able to receive the key, or the information to derive the key. There is of course the more trivial solution where, prior to the actual communication, the two users enter the secret information by hand into the two authenticating devices. However given the size of keys, especially public keys it is easier and less error prone to not require the user to type alot of characters. The keys to be exchanged can be either symmetric (a shared key) or asymmetric (two public keys). The

IST MAGNET [32] and IST SHAMAN [33] projects have both proposed establishing shared keys by mechanisms based on an authenticated Diffie-Hellman protocol. Others such as [34] use a location limited side channel to ensure the secrecy of a key transferred in plain text. Location-limited side channels are separate from the main wireless link, and have the security properties of demonstrative identification and authenticity. This is by virtue of the medium over which the data travels, for example direct physical contact or directional channels such as infrared. Once the secret has been exchanged, mutual authentication between the two devices can be based on any suitable challenge response authentication mechanism.

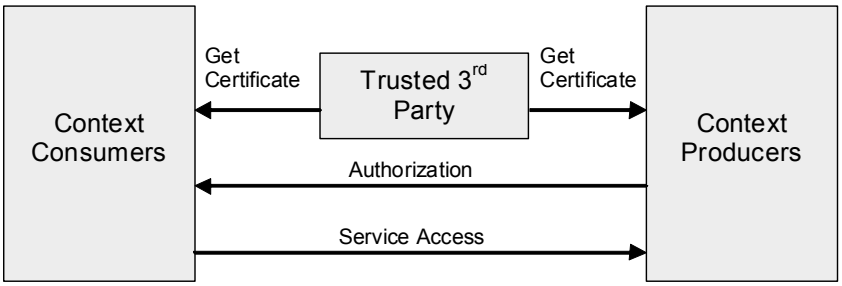
In the fully user assisted approach, authorization for service access can also be a tedious affair. This is because the user is now responsible for creating and maintaining the access control policies at each context producer. Policy rules can be expressed in many languages (e.g. XML based X-RBAC [31]) and are a powerful and flexible means of representing access requirements. However we believe that policy rules as used today are very difficult to express and understand, even if they are generated by simpler front end tools. Thus the user will have to acquire the technical know-how to write and manage these policies. This is best avoided.

In the use case described earlier, Emily has a doctor who can access her vitals. If the design of the security architecture is fully user assisted, then Emily is responsible for configuring both the authentication and authorization. This implies that she will have to exchange keys with each doctor and configure CACI to use these keys for authentication. She will also have to create and maintain the policies that will be used by CACI to enforce access control relating to each doctor. Furthermore, if her current doctor changes she will not only have to exchange keys with the new doctor but will also have to update the policies to reflect this change. These updates will need to be mirrored in CACI. Such actions defeat the purpose of context-aware applications, such as the ESS, that aim to provide the user with a hassle free and ubiquitious service. Consequently we reject the fully user assisted approach.

### 6.4.2 The Partially User Assisted Approach

The second security approach simplifies the responsibilities of the user by introducing the concept of a trusted third party. This trusted third party is responsible for enabling authentication (the user is still responsible for enabling authorization). Figure 6.3 is a diagrammatic representation of a typical message interchange between a context consumer and a context producer using a Public Key Infrastructure (PKI).

FIGURE 6.3. **Partially User Assisted Approach**




In this scenario, each participating CACI device is issued with a certificate from the trusted third party. Although this initial configuration step requires user interaction (to authenticate the CACI device to the trusted third party before it can issue the credential) this only needs to be performed once per participating device. The CACI context consumers and context producers can then leverage these credentials to authenticate each other automatically and without requiring any user interaction during authentication.

Security models in which each CACI device can be uniquely identified by a credential derived for it by a third party can generally be classified into two categories. The first category is based on online verifiable mechanisms and requires the entity issuing the credential to be online at the time of authentication. The second category is based on offline verifiable mechanisms and does not require the entity issuing the credential to be online when the two devices want to communicate.

Using certificates issued by a mutually trusted Certificate Authority (CA) is an example of *offline* verification. Each device knows the public key of the certificate authority and can use it to verify the certificates presented to it by other devices. On the other hand, approaches that require *online* access to a central authority for authentication are the Needham Schroeder protocol [35] and its most well known derivative, Kerberos [36].

The disadvantage of using digital certificates is that because it is based on expensive asymmetric cryptography, it may not suitable for all potential CACI devices (see section 6.3). Additionally, this approach also has the problem of certificate revocation. Although certificate revocation is often taken for granted, it is by no means a trivial challenge. Online authentication using lightweight symmetric cryptography (e.g. Kerberos) will not work in cases when online access to the central authority is not available. Since we assume that Emily is always online (so that distress messages can be sent out when there is an emergency and also so her doctors can access her vitals remotely) this is not a problem. Given the constrained nature of mobile devices, an approach based on online authentication using lightweight symmetric cryptography in the form of Kerberos, is promising. This also means that we do not have to worry about the problem of credential revocation.

If Emily is in distress it makes sense that any doctor in her immediate vicinity should be able

to read her vitals from her bracelet even when there is connectivity with the ESS. For this reason it is useful for the system to support access to critical context producers using a suitable location limited side channel (section 6.4.1). For example, if Emily is fine she can control access to her vitals by limiting physical access to her bracelets side channel; but if she has had a seizure, any health professional can take a reading directly. This is similar to checking an accident victim's wallet for ID.

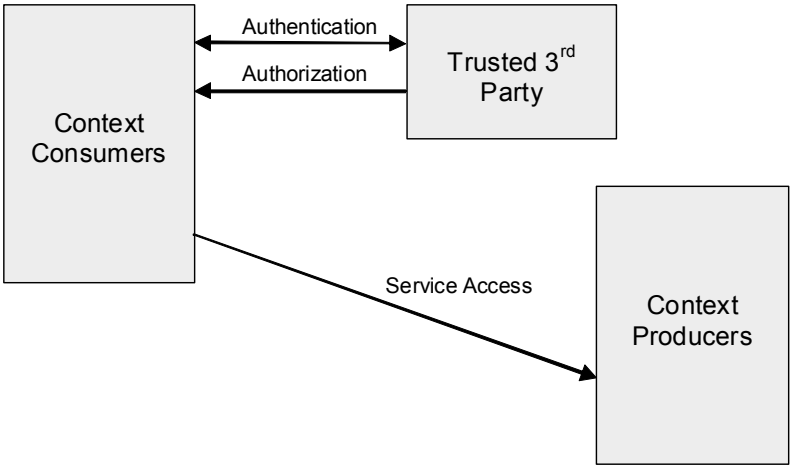
Lastly as the authorization is still being configured by the user he/she is still responsible for creating and maintaining the relevant policies for CACI. As mentioned earlier, this is not an ideal solution because policy rules are rather difficult to understand and express. Nevertheless this approach has a much lower user overhead than the fully user assisted approach described earlier and is a step in the right direction.

### 6.4.3 The Minimally User Assisted Approach

This final model further reduces user intervention by transferring both the authentication and authorization steps to the trusted third party. Like the previous case, each participating CACI device is issued with a credential which is only valid for subsequent authentication with the trusted third party. What is new is that there is also an access policy configured for each CACI producer at the third party. Since access policies are difficult to configure and maintain for the average user, this policy is created and maintained by trained professionals. This is done with either direct consultation with the administrator of the producer or indirectly, for example, if the Emily were to sign a form at the doctor's office authorizing him to query her vitals. Note that centralizing policy enforcement at such a central location also makes it possible to define more complex policy models than tradition ones like Role-Based Access Control (RBAC), Access-Control Lists (ACL), Discretionary Access Control (DAC) etc. using the XACML (eXtensible Access-Control Markup Language) policy language. Since this requires defining a new processing model and not merely configuring an existing one, it will require trained professionals.

In such a scenario the context consumer must contact the trusted third party before it can access a service from a particular context producer. The trusted third party first authenticates the context consumer (using the existing credential) and then verifies if the consumer is authorized to access the requested service. If the access request is deemed valid, the consumer is given a token. The context consumer will then present the token to the context producer. Based on the information present in the token, the context producer will be able to authenticate the consumer and grant him the approved access. These interactions have been shown in Figure 6.4.

FIGURE 6.4. **Minimally User Assisted Approach**




This approach meets our requirement of usability (as a large part of the overhead of security has been moved to the trusted third party) while being lightweight for constrained devices (as it is based on symmetric cryptography). The CACI producer is only responsible for *enforcing* the policy decision taken at the trusted third party which is contained within tokens.

In the next section we look at a standardized security model that matches our selected framework. It is designed for Web Services and allows us to embed Kerberos tokens and access policy in XML based SOAP messages.

## *6.5 A Standards Based Approach*

Having established our requirements and chosen a generic security framework, we now look at a suitable standardized approach that closely resembles our chosen generic security framework. This model is known as 3PAC [5] and was proposed as an extension to Web Services (WS) security, enabling credential based access policies for both service discovery and access.

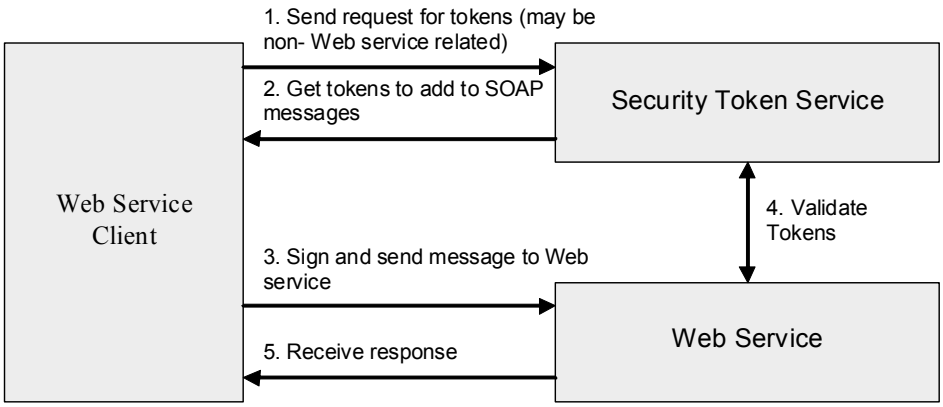
Although CACI is not a WS (since it does not support the core protocols specifications like SOAP etc. required be called such), the 3PAC extension to the WS security framework has much in common with our generic approach. WS security describes enhancements to SOAP messages that can be used to accommodate a wide variety of security models and cryptographic technologies. It is important to know that these mechanisms by themselves do not provide a complete security solution but are merely the foundation on which secure applications can be based.

Web Services are standardized Web APIs for service access over the Internet that are used by clients and servers to communicate using XML messages that follow the Simple Object Access Protocol (SOAP) standard. It is a general-purpose mechanism for associating security

tokens with SOAP messages. Figure 6.5 shows the typical message flow in a WS security framework. Note the similarities with our selected generic security framework (Figure 6.4) also based on a trusted third party.

The Security Token service can be Kerberos, PKI, or even a username/password validation service. It may not even be WS-based i.e. the Kerberos ticket might be retrieved through built-in operating system security functions. Once the client gets the tokens it wants to use in the SOAP message, the client will embed those tokens within the message.

**FIGURE 6.5. Typical Message Flow for a Web Service**




Unfortunately the WS-Security specification only provides a set of mechanisms to secure SOAP message exchanges through the use of message integrity, message confidentiality, and single message authentication. Besides these basic mechanisms the specification does not proposing a standards based approach to the *distribution and enforcement* of credentials-based access policies for Web Services. 3PAC builds upon the Web Services security (WS-security) specifications by proposing a model for these features.

In [5] the authors argue that 3PAC proposals scale well and can be implemented in existing Web Service deployments. The 3PAC architecture utilizes a trusted third party for the authentication of users and the resolution of policies to users. It breaks down the access of a webservice into two phases. The first phase takes place during service discovery and corresponds to the evaluation of access control policies. In the second phase the resultant policy is enforced at the web service.

Similar to 3PAC, our security architecture also differentiates two phases for context usage and access. The first phase is the context discovery phase in which CACI is responsible for the actual discovery of a context source. Once CACI has discovered a source it will proceed to establish a binding between the context consumer and the context producer. The second phase is the context access phase. In this phase the context consumer attempts to establish a binding or to gain access some context information. At this point CACI is responsible for ensuring that this user is granted access based on the policy that is relevant to this user.

### 6.5.1 Context Discovery Phase

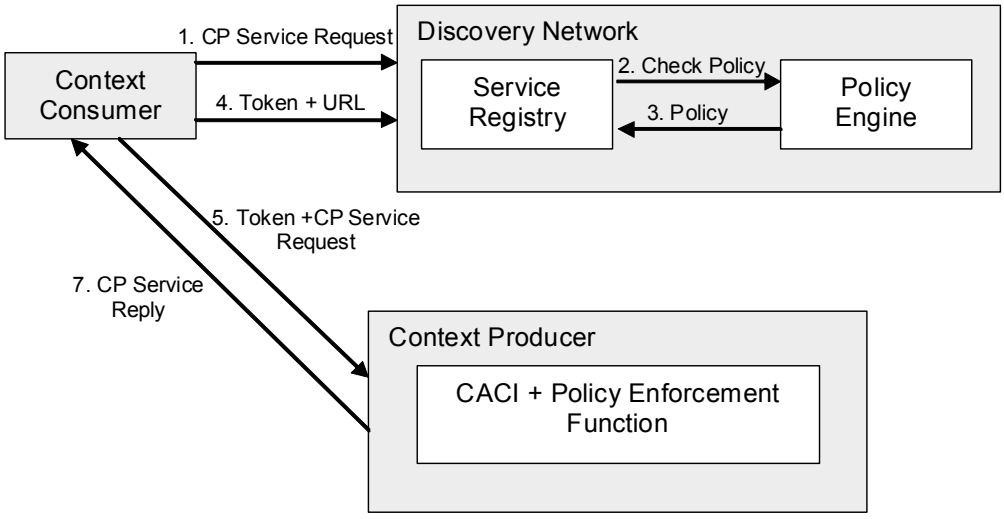
When a context consumer wishes to gain access to context information, it must first authenticate itself to the third party which is the Policy Decision Point (PDP). Upon verification, the request will be forwarded to the policy engine. The policy engine is responsible for resolving the request based on the stored policies. Once the request has been successfully resolved, the policy engine returns a token to the context consumer. The context consumer in turn will present this token for the establishment of binding between itself and the context producer. This token will also be used when attempting to access the context after the establishment of the binding. Note that context producers that wish to register their service at the service registry and add a policy for its use will also need to authenticate with the third party. This policy information will most likely be represented using XACML.

### 6.5.2 Context Access Phase

The context producer is the Policy Enforcement Point (PEP) in our architecture. The context producer receives the token to authenticate the context consumer. In order to authenticate the token provider, an access controller component needs to be added onto CACI. The access controller authenticates the context consumer (the token provider). Once the consumer has been verified, it is granted access based on the policy that was contained within the token.

Requests that do not contain a token can be handled by the access controller in one of two ways. The first option is to discard any requests the do not contain tokens as faulty. This design decision ensures that only authenticated and authorized users are allowed access to information. Another option is to have a default policy present at the context producer that allows any request without a token (i.e. anonymous users) certain restricted access. Figure 6.6 shows the step by step procedure described above.

**FIGURE 6.6. Security Solution for CACI**



Note that we also assume that the access controller and the context producer functionality belongs to the same device. In the case that the access controller is not resident on the same device, a trust relationship must also exists between the access controller and the context producer. This relationship can be enabled using shared keys or self signed certificates.

### 6.5.3 Revocation

Revocation of credentials issued by a third party is traditionally difficult to implement. A good example is the Public Key Infrastructure (PKI), which has struggled to provide a good solution to this problem. However when compare to an offline verifiable mechanism like PKI which is based on using digital certificates signed by the public key of a trusted third party, our proposed mechanism has more in common with an online-verifiable mechanism like Kerberos which does not require revocation check at the time of service access. This is because our tokens have short term validity and there is really no point in incurring the additional overhead of revocation which normally requires a “revocation check” at a specified URL against a list of revocations, for every request.

### 6.5.4 Inter-domain authentication

Our basic architecture assumes that the context discovery component, the context consumers and the context producers all belong to the same administrative domain i.e. they share the same trusted third party. Note that there exist inter-domain authentication models based on brokers or trust hierarchies to overcome this restriction, however they are outside the scope of this work. We will give some ideas on over coming this issue in future work.

CHAPTER 7

# Related Work

In the first part of this masters assignment we designed and implemented the CACI producer component in the CACI prototype. Since the consumer binding component had already been developed, our design decisions for the producer advertisement component had to be compatible with CACI's existing architecture.

In this second part of this masters assignment we have followed a systematic approach to recommending a security framework for CACI. We began by identifying CACI's security requirements and possible system constraints. Based on these we came up with a generic security framework and made a recommendation of a specific, standards-based approach. We proposed using the trusted third party model of Kerberos to authenticate CACI producers and consumers. We chose Kerberos because it is fast and lightweight as it is based on pure symmetric cyptographic primitives. This makes it more suitable for battery operated mobile devices typically found in CACI use cases. The authorization information related to this service access as well as the Kerberos tokens are encapsulated in SOAP messages that are protected using the WS security specification. We recommended the 3PAC security model because it built upon the WS security specifications by proposing a solution to the problem of distribution and enforcement of credentials-based access policies.

This chapter will examine technologies related to CACI and the security framework that was recommended for CACI. The discussion in this chapter highlights:

- The distinguishing features of CACI when compared with related technologies
- The use of our basic design features in other related security technologies.

## *7.1 CACI*

The importance of assisting with the creation and maintenance of context binding has been

debated for some time now. Context-sensitive Bindings [39], Service Oriented Network Sockets [40], and OSGi (Extended) Service Binder [41] are some example of solutions of this problem. Although the basic premise to CACI is similar to the aims of these solutions, CACI has been geared specifically towards advanced context-aware applications. An important contribution of CACI has been to formally identify and define components which are consumers and producers. Another important contribution of CACI has been to formalize and simplify the manner in which the bindings between the context consumers and producers are created and managed.We will briefly examine these above mentioned approaches in this section.

### 7.1.1 Context Sensitive Bindings

Context sensitive bindings aim to allow programmers the ability to bind their program to code that is discovered at run time. The programmer has the ability to define the requirements for the code or object in policies. Based on these policies, bindings are maintained transparently with the program.

The designers of Context Sensitive Bindings have proposed their scheme as an alternative to situations where static bindings are ill-suited. They foresee context sensitive bindings being used in adhoc environments which are typified by frequent disconnections and transient interactions. Furthermore, they feel that their approach will work well in any situation where it is beneficial to decouple the object interface from the implementation and where different interfaces can be selected.

The context sensitive binding approach differentiates itself based on:

Transparent Maintenance of Binding: The binding between the interface and the object specified by the programmer is maintained transparently. The interface is used to make any procedure calls to the object. The actual selection of the object is hidden from the programmer.

Policy based Choosing: The selection of an object for an interface is based on programmer defined policies.

Metric Based Evaluation: The best object out of the group of criteria matching objects is selected based on some predetermined metrics.

Continuous Binding for Ad Hoc Mobile Environments: This is the corner stone of the context sensitive binding approach. It provides continuous bindings in adhoc mobile environments in the face of disconnections. It also provides seamless shifts between objects based on quality of service criteria.

In order to demonstrate the feasibility of their concept, a middleware was implemented in JAVA which emulated these properties.

### 7.1.2 Service Oriented Network Sockets (SoNS)

Service oriented network sockets were designed to provide access to network services in a dynamic environment. A service oriented network socket can connect opportunistically to a service based on the high level requirements that an application states. The following design feature were provided to a system that attempts to provide opportunistic access to services in an adhoc environment:

Resource Discovery and Selection: The system is able discover services based on a high level specification. Furthermore the system has the capability to choose the closest matching service from amongst the list of discovered services.

Extensibility: The service requirements are specified in a manner that supports a diverse set of application requirements, network characteristics and standards.

Connection and Rebinding Semantics: The system has the ability to rebind when a better alternative becomes available. This rebinding can be facilitated by allowing the application to identify certain parameters that permit rebinding. These include the context within which it would allow rebinding for example, the room, the subnet etc. Additionally, the application should also be able to identify how long the system should wait before reacting to a change in order to avoid changes that are not of relevance to the application.

### 7.1.3 The OSGi Extended Service Binder

The OSGi Extended Service Binder is a model that promotes the discovery and binding of components running on distributed network nodes. This model is an extension of the Service Binder Model as offered by OSGi. The Service Binder Model allows developers to encapsulate code into components that offer and require services from each other. The services that are offered or required are simply declared by each component.

The Extended Service Binder model has three main components -- the export factory, the binding factory and the lookup service. The export factory is a service that makes a Java object remotely available. It also publishes a ‘binding description’ based on the services that it wishes to make available. The binding factory is used to bind to a given service based on the binding description that has been provided. The lookup service provides a service registry where service providers can register and deregister their services. Service clients can use this registry to lookup required services and be notified about specific services

This architecture allows for real peer to peer dynamic composition to occur at run time. However the drawback of this approach is that the Java interfaces must be agreed upon in advance between the service providers and the service requesters. The creators of this extension hope to tackle this short coming in their future work.

### 7.1.4 Distinguishing Features of CACI vs. Related Technologies

While each of the above solutions identifies with the need for the presence of a binding mechanism which can function robustly in a dynamic environment, they do not address context aware applications in specific. This is an important distinction as context consumers and producers have distinct requirements that need to be sufficiently addressed by the binding mechanism in order to provide full support to the application developer.

CACI leverages the capabilities of current context-aware middleware infrastructures and extends them to handle the dynamic availability and QoS of the various context providers. Another distinction of the approach taken by CACI is the use of a high level language called CBDL that is used to specify context requirements. This frees the application developer from having to program technology specific binding requirements. CACI is responsible for the interpretation and resultant bindings as specified in the high level descriptions.

## *7.2 Security*

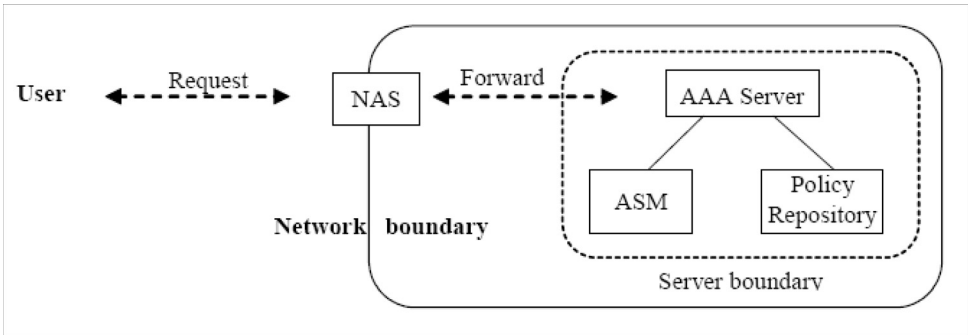
Many publications in the topic of security for context aware applications focus on just one aspect of the overall security architecture, specifically the best policies for access control [45, 29, 50]. On the other hand, our focus in this work was on the security framework for CACI. Consequently topics of interest for us were the generic AAA model [46, 47] for network security, the X.509 PKI standard [48], Kerberos [36], SAML [49], WS-* [51] and OpenID [53].

### 7.2.1 Authentication, Authorization and Accounting

Authentication, Authorization and Accounting (AAA) is a term for a generic architectural framework for controlling access to computer resources. This framework allows for the consistent and coherent configuration of the three central features. These features are essential for effective network management and security. Since our work focuses solely on security issues we will only investigate the first two functions, specifically authenticating users and handling their authorization requests.

Figure 7.1 illustrates how the AAA framework is used to control who is allowed access to the network and what services they are allowed to use once they have access.

FIGURE 7.1. **The AAA architecture**




AAA is the defacto standard for controlling network access in large corporations, universities etc. For example, when a user wants access to a network, it sends a request to the Network Access Server (NAS) which forwards the request to an AAA server. The AAA server checks to see if the user's credentials are valid. The NAS acts as a gateway to guard access to the network and its resources and depends on the answer from the AAA server to allow or deny access to the user. As a generic framework, AAA does not specify the exact authentication and authorization protocols used between the client and the NAS.

The generic AAA architecture is centralized and consists of three basic components: the AAA server, the policy repository and the Application Specific Modules (ASMs). The AAA server has rules to inspect authentication requests and come to a decision. The policy repository contains the list of available services about which authorization decisions can be made and the policy rules to make them.

Since the applications requiring AAA services may have their own unique needs in terms of configuration and initialization, the generic AAA server has an application interface to a set of ASMs. Authorization is performed by the AAA server based on the content of the policy repository and the decision by the ASMs (with their application specific knowledge). The ASMs may also use their application specific protocols to communicate with application speci.c service components.

In the AAA framework authorization can be performed using three available models (1) *Agent* (2) *Pull* and (3) *Push*. Each of these leads to a different sequence for the order in which the operations are performed. In the *Agent* model the AAA server forwards requests between the client entity and the service equipment. The client entity sends a service request to the AAA server which, after a successful authorization, forwards it (possibly with other configuration information) to the service equipment. The service equipment is the entity that provides the actual service. The service can be something that the user considers as a service or even something that provides some functionality within the network. The latter type is usually transparent to the user. After the service equipment is prepared (e.g. sets up state and so on) to provide the service, it acknowledges to the AAA server that it is ready. The AAA server replies to the client entity that the service is set up. In the *Pull* model the client entity sends the

request directly to the service equipment. The service equipment forwards the request to the AAA server which evaluates the request and returns an appropriate response. Service is then granted or denied based on the response. In the *Push* model, the client entity gets a token from the AAA server. Anytime that it requests a service from the service equipment, it presents the token from the AAA server as a way of verifying that it has been authorized to have access to the service.

The advantage of the centralized AAA approach is the ease of management for the network administrator who only needs to make changes (for both policies and user credentials) at the AAA server for them to have a system wide effect. Any changes at the AAA server e.g. to access rights are immediately reflected since every authorization request is evaluated at the AAA server. Furthermore, from a security perspective it is also more secure to have user credentials and access policies stored in a single, secure location. The main disadvantage of a centralized AAA approach is having a single point of failure, however this can be easily mitigated by having one or more backup servers.

### 7.2.2 X.509 Standard

The X.509 is an ITU-T standard for a public key infrastructure (PKI). The X.509 specification provides a format for public key certificates as well as a certification path validation algorithm. The path validation algorithms are based on trust verification using a hierarchical system of certificate authorities (CAs). A certificate authority is trusted third party which issues digital certificates for use by other parties. Although the X.509 specification also includes mechanisms for credential revocation using certificate revocation list (CRL), this is not a trivial task. There is a time lag for performing the CRL checks which also requires online access to the revocation server. Moreover using only digital certificates for securing CACI still does not solve the problem of authorization.

### 7.2.3 Kerberos and the KDC

The authentication process of Kerberos involves a trusted third party known as the Key Distribution Centers (KDC). The KDC shares a secret key with each client and service in the system. In order to access a service, the client has to submit a valid Kerberos ticket to the service. The ticket is a message encrypted using the secret key of the desired service and contains information about the client and a secret session key that is used for secure communication with the client. The client obtains the ticket as well as a copy of the session key from the KDC after the necessary authentication and authorization. Kerberos was identified early on as the authentication model well suited to our selected generic security framework due to its lightweight primitives and ease of revocation. Revocation of any client can be done easily by updating the information at the third party. However by itself Kerberos cannot be used in a context aware application scenario because it lacks mechanisms for discovery and access control. Consequently we looked at WS security and particularly 3PAC to unify service discovery, Kerberos authentication and policy based access control into a standards based approach. We have given an overview of both technologies in section 6.5.

### 7.2.4 Security Assertion Markup Language (SAML)

"*SAML, developed by the Security Services Technical Committee of Organization for the Advancement of Structured Information Standards (OASIS), is an XML-based framework for communicating user authentication, entitlement, and attribute information*" [49]. This information is exchanged between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). After the user authenticates to his identity provider he is able to access some personalized or customized resources at another service provider which, if confident of the origin of the assertion, can choose to log in the user as if he had authenticated directly.

As a standard means of exchanging security information, SAML is instrumental in solving the problems associated with Identity Federation. Typical use cases include, Multi-Domain Single Sign On (MD-SSO), Attribute based authorization and securing web-services. Related approaches, abet with different areas/use cases of focus, are WS-Trust [51] and OpenId [53]; we will look at them shortly.

In simple words, users authenticate themselves with an identity provider of a domain with whom they have a existing trust relationship. Once they are authenticated they are able to access resources at providers in other domains. Because SAML is both extensible and flexible, it has also been adopted by other standards such as WS-Security. The following are some of the benefits of employing SAML:

Platform Neutrality The security framework is abstracted from the security architecture and vendor implementations.

Loose coupling of directories User information does not have to be maintained or synchronized between directories.

Improved online experience for users This refers to the benefit from a multi-domain single sign on. The users experience is improved as they can authenticate themselves at an identity provider and then access other access providers without additional authentication. Furthermore, the linking of multiple identities with SAML also promotes privacy while increasing user experience.

Reduced Administrative costs for service providers The burden of maintaining account information is moved from the administrator to the identity provider.

Risk transference As SAML allows the maintenance of identities to be transferred from the service provider to the identity provider, the risk of maintenance of the identities is also transferred. This is usually in line with the business requirements of the service provider.

SAML identifies three components namely assertions, protocols and bindings. Assertions are

packages of information that contain one or more SAML statements. A sample assertion with a Subject, Conditions and Authentication statements is shown in Figure 7.2. Details are beyond the scope of this document but can be found in [50].

**FIGURE 7.2. A sample SAML assertion**

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  Version="2.0"
  IssueInstant="2005-01-31T12:00:00Z">
  <saml:Issuer Format=urn:oasis:names:SAML:2.0:nameid-format:entity>
    http://www.example.com
  </saml:Issuer>
  <saml:Subject>
    <saml:NameID
      Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
        j.doe@example.com
    </saml:NameID>
  </saml:Subject>
  <saml:Conditions
    NotBefore="2005-01-31T12:00:00Z"
    NotOnOrAfter="2005-01-31T12:10:00Z">
  </saml:Conditions>
  <saml:AuthnStatement
    AuthnInstant="2005-01-31T12:00:00Z" SessionIndex="67775277772">
    <saml:AuthnContext>
      <saml:AuthnContextClassRef>
        urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
      </saml:AuthnContextClassRef>
    </saml:AuthnContext>
  </saml:AuthnStatement>
</saml:Assertion>
```

There are three different kinds of assertions. Authentication assertions validate the users identity, while attribute assertions contain specific information about users and authorization assertion identifies what a user is authorized to do. Protocols define how SAML requests and receives assertions. Bindings define how SAML assertions are mapped to SOAP exchanges.

### 7.2.5 WS-Trust

WS-Trust is proposed by the WS-* initiative which works towards creating specifications for different aspects of web services including security. Although WS-Trust supports the use of SAML assertions inside its tokens, however when compared with SAML, WS-trust decouples protocols from tokens. Whereas the SAML protocol presumes the use of SAML tokens, WS-Trust makes no such assumptions about its tokens. WS-Trust also defines a Security Token Service (STS) which translates tokens of various formats like Kerberos into tokens required by the service providers. This means that when using STS a service provider can establish trust relationships across a wide ranges of protocols and token formats. WS-Trusts is always used in conjunction with WS-Security which has been introduced in the pervious chapter.

### 7.2.6 Open-Id

The Open-Id approach uses URL based identity specifications. Unlike WS-Trust and SAML, Open-Id has no affiliation with WS-Security or SOAP. User names include a resolvable DNS components. Website that want to enable Open-Id sign ons must add some version of Open-Id libraries to their site. Users on the other hand do not need to download any client software, nor

instal plug-ins into their browsers. To login to a site with Open-Id the user first asserts rights to a particular address on the Internet by specifying a URL (essentially a user name). The service provider communicates directly with the site and uses its SSL certificate to receive a shared key. The user also authenticates to this site, receives the same shared key and uses it to identify with the service provider. When compared with the other technologies mentioned above, Open-Id is (a) fully decentralized and (b) has a much less configuration overhead. However, SAML and WS-* remain standards for enterprise class identity federation because they have a much stronger security posture than Open-Id. Service providers depend on DNS to connect them with the given Open-Id provider who provides a certificate which the service provider can validate. However, the rest of the data is asserted and managed by the user.

CHAPTER 8

# Conclusions and Future Work

## 8.1 Conclusions

### 8.1.1 CACI

The first half of this masters project focused on extending CACI to include support functions for context producers. This section will focus on the benefits and drawbacks of CACI.

**Advantages of CACI**
The most important benefit of CACI is the decrease in the amount of work for the context-aware application developer. The developer no longer needs to create tedious code to ensure the proper selection, use and deselection of a context producer with a context consumer. CACI is used to oversee this binding process. Developers can now focus their energies on creating other aspects of the context-aware application. Furthermore, the user of context aware applications will ultimately benefit as application developers will be able to improve the quality of the product that they produce.

Another key advantage of CACI is that it is lightweight and therefore portable. It is not a "theoretical" model but a realistic approach to the development of context-aware applications. CACI can therefore be easily deployed in most environments in which context-aware applications function.

**Limitations of CACI**
While the requirements defined at the start of this project with regards to the CACI context producers were met, there are still some limitations present. These limitations should be overcome to make CACI a more complete middleware. We refer the reader to section 8.2 for a discussion on possible directions for future work on CACI.

### 8.1.2 Security Framework for CACI

**Advantages of the Security Model**
From a high level perspective, utilizing a third party as the policy decision point allows us to provide the user with a simplified user experience as it reduces his administrative/ configuration load significantly. This is inline with the aim of CACI. If the responsibility for authentication and policy decision making were present at the end points, the user would then be required to handle most of the complexity associated with the creation of trust and writing and management of policies. Such overhead just to enforce security defeats the basic premise of context-aware applications which aim to provide the user with an ubiquitious and pleasant experience.

From a technological perspective we have proposed the use of the 3PAC model, which is an extension of the web-services framework.

**Drawbacks of the Security Model**
Availability of infrastructure is central to any architecture that relies on a model which provides security with the help of an online trusted third party. Any consumer that wishes to access information from a context producer that is protected with our security framework needs to have access to a trusted third party. Even though such access cannot always be guaranteed, we do not consider this a serious drawback since a lot of communication between producers and consumers will also be taking place over the Internet. An alternative solution in the case of event that there is no connection available to the infrastructure and the context producers and consumers are geographically co-located, is to access the services using a suitable location limited side channel configured for temporary access.

Similarly, another short coming of a centralized security architecture is that there is only one point of failure. The security of the entire architecture is invalidated if the trusted third party is compromised or unavailable. As far as the latter point is concerned, the *reliability* of the system can be easily increased using one or more backup servers.

## *8.2 Future work*

### 8.2.1 CACI

CACI was designed as a middleware that specializes in providing binding transparency to the context-aware application developer. This project focused specifically on the construction and integration of support functions for context producers. As mentioned, earlier while these support function have been designed and incorporated in to the CACI prototype, more work is required to make CACI a robust middleware. These include the following [54]:

◆ The producer mechanism of CACI should also be extended to publish to external discovery

mechanisms. Currently, the context producer publishes its services to a local repository.

◆ During the course of this project, performance testing was carried out on the CACI prototype. It was discovered that CACI performance can become unstable in the event of the frequent arrivals and departures of context producers and of frequent changes in the context requirements. Research needs to be carried out to minimize the effect of such events so that the stability of CACI can be increased.

◆ More focus needs to be placed on how the system recovers from the sudden absence of a context producer. Steps need to be put in place that can handle the deregisteration of the producer component from the discovery mechanism.

◆ Currently CBDL research has only been carried out to define the syntactical requirements for a CBDL document. In order for CACI to accurately match the context requirements to context offerings, the semantic of documents using CBDL also need to be defined.

◆ Furthermore, CACI can not identify conflicting QoS requirements within a document. These maybe intentional or unintentional, in either case CACI should have a mechanism that can identify such a discrepancy and inform the application developer of the oversight.

### 8.2.2 Security Framework for CACI

The implementation of support functions for CACI and the recommendation of the security model make CACI better rounded. However more work must be done to assess the exact empirical viability of the security model by making the necessary modifications to CACI and running it in a manner that actualizes the model suggested. Such performance evaluation will help paint a better picture for future development.

**A model for more fine-grain authorisation**

As stated in Section 6.5.1, when a consumer wishes to access a service, it must ask the third party for a token which is valid at the producer. This token includes both authentication and authorisation information, which will be used by the service provider. As far as authentication is concerned, this model is does not leave anything to be required. It is unlikely that the identity of the consumer changes anytime during service access. However authorisation as implemented through the permissions included in the token only enables rather coarse grained access control. Access policies enforced during service access can enable applications to have a more fine grain access control. However this will require applications to send authorisation requests to the third party during service access, and this increases the overhead in the system. Furthermore such services will have to be either developed with the required SDK which will enable dynamic authorisation or use some standardized interface like the web services interface (if it is supported by the third party).

**Inter-domain authentication**

Earlier we stated that our basic architecture assumes that the context discovery component, the context consumers and the context producers all belong to the same administrative domain i.e. they share the same trusted third party. Such a model where the service provider is part of the same authentication domain as the consumer and the identity provider is called the Identity 1.0 paradigm. The new trend, known as Identity 2.0 breaks from this paradigm and has the identity provider in a different domain from the service provider. This is also known as identity federation, and in the last few years there has been a lot of standardization work in the context of inter-domain authentication and authorisation based on SAML. SAML assertions provide a standards based and platform independent means of exchanging security related information between an identity provider and a service provider that belong to different domains.

# *Terminology*

| | |
|---|---|
| AAA | Authentication, Authorization, and Accounting |
| ACL | Access Control List |
| AES | Advanced Encryption Cipher |
| API | Application Programming Interface |
| ASM | Application Specific Modules |
| CA | Certificate Authority |
| CACI | Context Aware Component Infrastructure |
| CBDL | CACI Binding Description Language |
| CORBA | Common Object Request Broker |
| CRL | Certificate Revocation Lists |
| DAC | Discretionary Access Control |
| DES | Data Encryption Standard |
| DNS | Domain Name System |
| DoS | Denial of Service |
| DSA | Digital Signature Algorithm |
| ECDSA | Elliptic Curve DSA |
| ESS | Epilepsy Safety System |
| GUI | Graphic User Interface |
| HMAC | Hashed Message Authentication Code |
| HMS | Health Monitoring Service |
| IA | Information Assurance |
| IAM | Integrity and Access Management |
| JAR | Java Archive |
| JCAF | Java Context Awareness Framework |

| | |
|---|---|
| KDB | Key Distribution Center |
| LAN | Local Area Network |
| MAC | Message Authentication Code |
| MDSSO | Multi Domain Single Sign On |
| NAS | Network Authentication Server |
| OSGi | Open Service Gateway initiative |
| PDA | Personal Digital Assistant |
| PDP | Policy Decision Point |
| PEP | Policy Enforcement Point |
| PKI | Public Key Infrastructure |
| RBAC | Role Based Access Control |
| SAML | Security Assertion Markup Language |
| SDK | Software Development Kit |
| SOA | Service Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| SOCAM | Service Oriented Context Aware Middleware |
| SSO | Single Sign On |
| STS | Secure Token Service |
| URL | Universal Resource Locator |
| WAN | Wide Area Network |
| WS | Web Services |
| XACML | Extensible Access Control Markup Language |
| XML | Extensible Markup Language |

# *References*


1. A.K. Dey, *Providing Architectural Support for Building Context-Aware Applications*, PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.
2. T. P. Moran, P. Dourish, *Introduction to the special Issue of Human-Computer Interaction*, Volume 16, 2001. (http://hci-journal.com/editorial/si-context-aware-intro.pdf)
3. M. Korkea-aho, *Context-Aware Application Survey*, Department of Computer Science, Helsinki University of Technology, 2000. (http://users.tkk.fi/~mkorkeaa/doc/context-aware.html)
4. A.K. Dey, G.D. Abowd, *Toward a Better Understanding of Context and Context-Awareness*, College of Computing, Georgia Institute of Technology, 1999. (http://www.cs.cmu.edu/~anind/context.html)
5. J. v. Bemmel, M. Wegdam, and K. Lagerberg, *3PAC: Enforcing Access Policies for Web Services,* IEEE International Conference on Web Services (ICWS), 2005.
6. The Active Badge System. (http://www.cl.cam.ac.uk/Research/DTG/attarchive/ab.html)
7. The Xerox PARCTAB. (http://www.ubiq.com/parctab/)
8. The CyberGuide Project Page. (http://www-static.cc.gatech.edu/fce/cyberguide/)
9. J Barton, T. Kindberg, *The CoolTown User Experience*, HP Laboratories, 2001.
10. K. Mitchell, *A Survey of Context Awareness*, Department of Computer Science, Lancaster University, 2002.
11. K. Henricksen, J. Indulska, T. McFadden, S. Balasubramaniam, *Middleware for Distributed Context-Aware Systems*, School of Information Technology and Electrical Engineering, The University of Queensland, 2005.
12. M. Roman, C. Hess, R. Cerqueira, K. Nahrsted, R. Campbell, *Gaia: A Middleware Infrastructure to Enable Active Spaces*, Digital Computer Labortatory, The University of Illinois at Ubrana-Champaign, 2002.
13. T. Gu, H. K. Pung, D. Q. Zhang, *A Service Oriented Middleware for Buidling Context-Aware Services*, Department of Computer Science, the National University of Singapore, 2004.

14. G. Blair, J.B. Stefani, *Open Distributed Processing and Multimedia*, Addison Wesley, ISBN 0201177943, 1997.

15. IEEE Distributed Systems Online. (http://www.computer.org/portal/site/dsonline)

16. S.Vinoski, D. C. Schmidt, *The Corba Component Model: Part 1, Evolving Towards Component Middleware*, 2004. (http://www.ddj.com/dept/cpp/184403884)

17. Common Object Request Broker Architecture, Carnegie Melon Software Engineering Institure. (http://www.sei.cmu.edu/str/descriptions/corba_body.html)

18. B. N. Schilit, N. Adams, R. Watt, *Context-Aware Computing Applications*, 1994. (http://sandbox.xerox.com/want/papers/parctab-wmc-dec94.pdf)

19. N. Ryan, J. Pascoe, D. Morse, *Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant*. (http://www.cs.kent.ac.uk/people/staff/nsr/mobicomp/Fieldwork/Papers/CAA97/ERFldwk.html)

20. A. K. Dey, *Context-Aware Computing: The CyberDesk Project*, 1998. (http://www-static.cc.gatech.edu/fce/cyberdesk/pubs/AAAI98/AAAI98.html)

21. I. Sommerville, *Software Engineering*, 7th Edition, Addison Wesley, ISBN 0321210263, 2004.

22. A. v. Halteren, D. Quartel, M. v. Sinderen , T. Broens, *Dynamic Context Bindings in Pervasive Middleware*, Proceedings of the 4th IEEE International Workshop on Middleware Support for Pervasive Computing (PerWare), 2007.

23. OSGi Alliance. (http://www.osgi.org)

24. A. Perrig, R. Szewczyk, V. Wen, D. Culler and J.D. Tygar. *SPINS: Security protocols for sensor networks*, 7th International Conference on Mobile Computing and Networking (MobiCom), 2001.

25. T. Broens, A. v. Halteren, and M. v. Sinderen, *Infrastructural support for dynamic context bindings*, 1st European conference on Smart Sensing and Context (EuroSSC), 2006.

26. T. Broens and A. v. Halteren, *SimuContext: Simply Simulate Context*, International Conference on Autonomic and Autonomous Systems (ICAS), 2006.

27. T.S. Rappaport, *Wireless Communication: Principles and Practice*, Prentice Hall, 2nd Edition, ISBN 0130422320, 1996.

28. Y. Hu, A. Perrig and D. Johnson, *Ariadne: A Secure On-demand Routing Protocol for Ad Hoc Networks*, 8th ACM Conference on Mobile Computing and Networking (MobiCom), 2002.

29. J. Vollbrecht, et al, *AAA Authorization Framework*, RFC 2904, Aug 2000.

30. Y. W. Law, J. Doumen and P. Hartel, *Survey and Benchmark of Block Ciphers for Wireless Sensor Networks*, 1st IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), 2004.

31. R. Bhatti, E. Bertino and A. Ghafoor, *A Trust based Context-Aware Access Control Model for Web-Services*, IEEE International Conference on Web Services (ICWS), 2004.

32. The IST MAGNET project. (http://www.ist-magnet.org/)

33. The IST SHAMAN project. (http://www.isrc.rhul.ac.uk/shaman/docs/)

34. D. Balfanz, D. K. Smetters, P. Stewart and H. C. Wong, *Talking to strangers: Authentication in ad-hoc wireless networks*, Network and Distributed Systems Security Symposium (NDSS), 2002.

35. R. Needham and M. Schroeder, *Using encryption for authentication in large networks of computers*, Communications of the ACM, 21(12):993–999, Dec 1978.
36. C. Neuman, T. Yu, S. Hartman, and K. Raeburn, *The Kerberos Network Authentication Service (V5)*, RFC 4120 (Proposed Standard), July 2005.
37. T. Broens, D. Quartel, and M. v. Sinderen, *Towards a Context Binding Transparency*, EUNICE, 2007
38. T. Broens, D. Quartel, and M. v. Sinderen, *Capturing Context Requirements*, EuroSSC, 2007
39. Sen, R. and G. Roman, *Context-Sensitive Binding, Flexible Programming Using Transparent Context Maintenance*, in Technical Report WUCSE-2003-72. 2003, Technical Report WUCSE-2003-72, Washington University.
40. Saif, U. and M. Palusak. *Service-oriented Network Sockets*, International conference on mobile systems, applications and services (MobiSys'03). 2003. San Francisco, USA.
41. Bottaro, A. and A. Gerodolle, *Extended Service Binder: Dynamic Service Availability Management in Ambient Intelligence*, International Workshop on Future Research Challenges for Software and Services (FRCSS'06). 2006: Vienna, Austria.
42. Information Assurance, http://en.wikipedia.org/wiki/Information_assurance
43. N. Okabe, S. Sakane, et. al, "*Security Architecture for Control Networks using IPsec and KINK*", International Symposium on Applications and the Internet (SAINT), Trento, Italy, Jan 2005
44. H. Yan and Z. J. Shi, " *Studying Software Implementations of Elliptic Curve Cryptography*", 3rd International Conference on Information Technology: New Genereations (ITNG), Las Vegas, Nevada, USA, April 2006
45. J. Hu and A. C. Weaver, "*A Dynamic Context-Aware Security Infrastructure for Distributed Healthcare Applications*", Pervasive Security, Privacy and Trust (PSPT2004), Boston, MA, August 2004.
46. J. Vollbrecht, et al, "*AAA Authorization Framework*", RFC 2904, August 2000.
47. J. Vollbrecht, et al, "*AAA Authorization Requirements*", RFC 2906, August 2000.
48. C. Adams, S. Farrel, T. Kause, and T. Mononen, "*Internet X.509 Public Key Infrastructure Certificate Management Protocol"*, RFC 4210, September 2005.
49. SAML, http://xml.coverpages.org/saml.html
50. SAML Technical Overview, http://xml.coverpages.org/SAML-TechOverview20v03-11511.pdf
51. WS-Trust, http://www.ws-i.org
52. WS-Security: SAML Token Profile, http://docs.oasis-open.org /wss/oasis-wss-saml-token-profile-1.0.pdf
53. OpenID, www.openid.net
54. Borens, T. , "*Dynamic Context Bindings: Infrastructural Support for Context-Aware Applications*", Phd Thesis, November 2008