Policy Management in Context-Aware Networks

NUPUR BHATIA

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2007

COS/CCS 2007-04

Policy Management in Context-Aware Networks

Nupur Bhatia

2007-02-14

Master of Science Thesis Project
Conducted at Ericsson Research

Industrial supervisor: Theo Kanter (Ericsson Research)
Academic supervisor: Gerald Q. Maguire Jr.
Examiner: Gerald Q. Maguire Jr.

School of Information and Communication Technology (ICT)
Royal Institute of Technology (KTH)

Abstract

The Ambient Network (AN) Project is part of the European Commission's 6th Framework Programme and aims to enable cooperation between heterogeneous networks, using current and future wireless technologies, minimising the effort of mobile users to gain access to the services that they are interested in - irrespective of their location or the network they are currently using. Because of the highly mobile nature of users and a demand for instant and dynamic access to services, these networks have to be composed 'on the fly' *without* any pre-configurations.

The use of context information in AN can remove the need for pre-configuration of networks, hence making them autonomic. However, a concern exists that the free and uncontrolled dissemination of context information could breech the privacy of the participants. It is extremely important to address these privacy issues in order to control who has access to what context information. This control can be achieved through the use of well defined policies. This creates a requirement for a framework in the ContextWare architecture for protecting context information.

This masters thesis project is part of an effort to create a policy based infrastructure for authorisation of access to network context information within the AN. The thesis investigates, models, and designs an architecture for a policy management system based on OASIS XACML, that creates an extension to the architecture for management of context information in the AN. In addition to a policy management architecture within an AN, a policy management architecture for composing ANs is also created. To facilitate the transfer of requests and policies, the thesis creates a Policy Management Protocol. The designed architecture was then implemented to create a proof of concept.

The designed architecture and protocol were evaluated by running tests on the prototype. The measurements from the tests are analysed and presented in this thesis. The analysis of the experimental data indicates that a policy management system is both feasible and practical. The results show that the delay overhead caused by introducing policy management in a distributed context provisioning system, ranges from 1.7% in a system without load to 6% in a worst case scenario. The throughput of the policy management system is 15 requests per second under load.

Sammanfattning

Ambient Network är ett EU-finansierat project inom det 6:e ramprogrammet. Projektets mål är att möjliggöra samarbete mellan heterogena nätverk, som använder både dagens men även framtidens trådlösa teknologier, för att minimera slutanvändarens insats för att nå den tjänst de är intresserade av – oberoende av plats eller vilket nätverk de använder. På grund av den stora delen av mobila användare som kräver omedelbar och dynamisk tillgång till tjänster måste dessa nätverk gå samman 'on the fly' utan tidigare konfigurering.

Användningen av context information i Ambient Networks kan elmininera behovet av förkonfigurering av nätverk, följaktligen blir de då autonoma. Dock, ett problem som uppkommer med detta är att den fria och okontrollerade spridningen av context information bryter integriteten för deltagarna. Det är väldigt viktigt att ta itu med detta problem för att kunna kontrollera vilka som har tillgång till vilken context information. Den här kontrollen kan uppnås genom väldefinierade policies. Detta skapar ett behov av ett ramverk inom ContextWare arkitekturen för att skydda den tillgängliga context informationen.

Den här uppsatsen är en del i ansträngningen att skapa en policy baserad infrastruktur för attestering av tillgång till context information inom Ambient Networks. Uppsatsen undersöker och designar en arkitektur för ett policy handhavande system som är baserat på OASIS XACML, den bygger vidare på arkitekturen för handhavande av context information i Ambient Networks. Utöver policy hantering inom ett ambient network skapas också policy hantering mellan ambient networks när de förenas. Den framtagna arkitekturen är därefter implementerad för att visa på konceptets hållbarhet. En sammanslagning av två policy handhavande system när två nätverk slås ihop är behandlat endast i teorin, det är inte implementerat.

Designen utvärderas genom att köra test på den implementerade versionen och därefter analysera och visa resultaten i rapporten. Dessa test innehåller mätningar av fördröjningen av en enda begäran samt flera, responstiden i ett system med policy-hantering jämfört med utan samt prestandan i ett policy-hanteringssystem med en liten mängd policies jämfört med en större mängd policies.

Acknowledgements

I would like to thank Professor Gerald Q. Maguire Jr. for being my academic supervisor and guiding me with his valuable comments and questions throughout my thesis; which lead me in the right direction.

I would like to express special thanks to my supervisor at Ericsson, Theo Kanter for giving me an opportunity to work on this thesis project. I have reverted to him a number of times when I felt myself in a deadlock situation and discussions with him have always helped me bring things into focus. His support, suggestions, and feedback have led me in the correct direction during my thesis work.

I would also like to extend thanks to my colleague Markus Swenson with whom I have bounced many ideas and had frequent discussions in the area of my work, helping me to crystallize my ideas.

Many thanks to Börje Ohlman from Ambient Network Project for giving me directions during our discussions in the initial phase of the project.

The figures, Fig.1 to Fig. 8 in this report have been used with the permission of the Ambient Network Project.

Last, but not the least, I would like to thank my family for their patience, understanding, and support during my studies. I dedicate my thesis to them.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACAS	Adaptive & Context-Aware Services
ACS	Ambient Control Space
AN	Ambient Networks
ANI	Ambient Network Interface
API	Application Programming Interface
ARI	Ambient Resource Interface
ASI	Ambient Service Interface
CCH	Cross Layer Context Handling
CCP	Context Control Plane
CIB	Context Information Base
CLA	Context Level Agreement
CM	Context Management
CME	Context Management enabled Entities
ConCoord	Context Coordinator
CUA	Context User Agent
CXP	Context Exchange Protocol
FA	Functional Area
FE	Functional Entities
FP6	Sixth Framework Programme
IST	Information Society Technology
LG	Location Generator
LI	Location Information
LR	Location Recipient
LS	Location Server
OASIS	Organization for the Advancement of Structured Information Standards
PAN	Personal Area Network
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PMT	Policy Management Tool
QoC	Quality of Context
RM	Rule Maker
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
UCI	Universal Context ID
UDP	User Datagram Protocol
URI	Universal Resource Identifier
URN	Uniform Resource Name
WP6	Work Package 6
WWI	Wireless World Initiative
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

1 Introduction

This chapter introduces the general area of the thesis and presents the problem that this thesis aims to solve.

1.1 The Ambient Network

Mobile and wireless networks today are evolving beyond GPRS and 3G to include WiFi and WIMAX and this heterogeneity brings with it both differences in access technologies and differences in mobility support. In addition to multiple access technologies, the networks themselves may be mobile, including both moving access networks as well as mobile Personal Area Networks (PAN). Today there is a need to provide a means to seamlessly traverse network to network, operator to operator, and device to device, so that the user is provided with an affordable, apparently homogeneous, and ubiquitous network.

The Ambient Network (AN) project [1] is co-sponsored by the European Commission under the Information Society Technology (IST) priority under the 6th Framework Programme and undertakes the task of bringing the appearance of homogeneity into the above mentioned networks.

> "Ambient Networks offers a fundamentally new vision based on the dynamic composition of networks to avoid adding to the growing patchwork of extensions to existing architectures. This will provide access to any network, including mobile personal networks, through instant establishment of inter-network agreements."[1]

To enable the dynamic behaviour needed in such an AN, the networks have to be aware of context information. Awareness means both acquiring and acting on this information. Context is any information that explains the situation of a person, place, or object [2]. For the AN, the relevant context could be related to storage in devices, data rates of the links in each (access) network, mobility handoff decisions, user location, etc. Moreover, in a mobile scenario this context will change and so the network should adapt to these changes to ensure that the user is always presented with optimised services and applications: ideally without any need for user intervention. Context Networks are networks which have the ability to collect, manage, and distribute context information from and to the various network entities, user services, and applications.

The Ambient Network project proposes to transparently integrate heterogeneous networks when needed, on the fly, without being preconfigured by the involved network operators. The task could become very complex when taking into account user mobility and the fact that this complexity should be hidden from the user. The key to reducing this complexity lies in using context information to make autonomic decisions [4] thus providing the user with the best connectivity possible based on the users' requirements at that time and the current capabilities of the available networks. **Work-Package 6: Context Aware Networks** (WP6) in the Ambient Networks project is dedicated to exploiting the advantages of context awareness to "make mobile communication networks simpler, more efficient, and more powerful" [1]. The

task of WP6 has been to create a system architecture for context management called ContextWare. The aim of this ContextWare architecture (refer section 2.4) is to make the network itself context aware by making use of user context and network context information, thus enabling intelligent decision making, hence providing better services to users. [3]

1.2 Authorization within Ambient Networks

The definition of context in terms of Ambient Networks includes all information related to any entity in the environment where these entities could be humans, applications, devices, and software agents. In addition context information represents some aspect or property that describes the entity and this property could be an absolute physical property such as the temperature of an entity, or the capacity of a network, or it could be a logical property such as the cost of a service, or the security level in a network (which could be different in different situations and is determined by the creator of the context). Managing this wide range of context data to aid the AN in self management is a complex task. Moreover the context information is sensitive information which could be used to alter the behaviour of the network or service and so access to this context information must only be provided to entities that have the appropriate access. Security is not just related to controlling who gets into the AN (authentication). What these entities are allowed to do (authorisation), once inside the network, is just as critical.

Control means not only preventing the leakage of data, but also permitting access to necessary information. Access control permits users to access only the information they have privileges for.

Policies are required to grant access to context information in a secure way, thus translating the technology independent high level system goals to low level actions. Here the policies are related to context information and the context information itself is considered to be the resource for which access is needed.

In addition to providing Access-Control, policy-based networking has been used to provide a degree of automation, by linking actions in the network to system-events. Policies permit adaptation of network behaviour without modification of the implementation and introduce network management flexibility as they seek to control network behaviour using sets of high-level rules.

ANs require a flexible and scalable policy-based management system that can fulfil management requirements of all other functional entities in the Ambient Control Space (see section 2.1). Responsibilities of this management system include network management, service management, and context management. To realize these responsibilities, the system must deal with different types of policies such as those for handling access control, mobility, security, context, composition, QoS, service provisioning, etc.

The management system must also provide the means to define security policies that can specify ownership and permissions, and are “well written” in order to avoid conflicts with other policies within the network.

Policies are expressed as a sequence of rules to control the behaviour of an entity (i.e. as a condition-action pair). Policies can be utilized to **realize** core management functionalities such as context management, network management, and service management. Policies can also be used for processing raw context information through aggregation, transformation, or filtering to produce new context information. Finally, access policies can be used to securely control and monitor context dissemination to services and applications which can either request context values or decisions of actions to be performed according to the current context.

This thesis implements the management of access to context information through the use of policies. The aim of this thesis is to provide the ContextWare architecture with a privacy sensitive authorisation framework utilizing policy management, in order to control the access and distribution of context information within the boundaries of the ambient network and for inter network distribution. The specifics of how this is to be done are described in the next section.

1.3 Longer problem statement

Previous work in the area has been related to designing an architecture and communication protocol for the collection, management, storage, and distribution of context information in the Ambient Network [6]. Although there are lot of benefits achieved by making networks context aware and enabling sharing of context information between the various network entities to make the networks self-organising and self-managing, it is very important to keep in perspective the threats of exposing this context information to undesired entities in an uncontrolled manner.

This motivates the development of a policy framework for authorization of access to context information. This framework must take into account the nature of the ContextWare architecture, hence it should provide a high degree of flexibility in order to support a wide range of services. Authorisation policies should express the context source's desires regarding the distribution of context access to other entities [7].

The task of this thesis is to investigate, model, structure, and evaluate a policy management extension to the ContextWare support in the scope of the Ambient Network project.

The tasks can be further elaborated as below.

- The main task is to create a **policy management architecture** to enable ContextWare (context aware) decisions about authorisation of access to both ContextWare entities and context. The authorisation scope should be within the Ambient Control Space(ACS), between different ACSs via the Ambient Network Interface(ANI), or from the service and application layer via the Ambient Service Interface(ASI). (For additional details about ACS and interfaces see sections 2.1 and 2.2)
- Develop a **protocol** to parameterize policies with context information from ContextWare. This protocol should provide a way to filter, aggregate, or correlate context information from various sources. The architecture of the policy

management extension will be based on OASIS XACML. (For additional details about OASIS XACML see section 3.5)

- Develop a **prototype** based on the designed architecture to manage access to context information within the Ambient Network.
- Carry out measurements during the testing phase to determine that the computations regarding authorization and dissemination of policy decisions with respect to context data can be made in a timely fashion. (These results are presented in Chapter 6).
- Propose further work based on the results regarding extensibility of the approach and standardisation. (Future work is summarised in Chapter 7).

1.4 Methodology and Report Structure

The applied methodology and structure of the thesis are as follows:

1. Background study in the area of Ambient Networks. (Chapter 2)
2. Study of existing technologies and background for access control through policy management. (Chapter 3)
3. Discussion of the design principles and design of the policy management system architecture with example use cases and design of the policy management protocol. (Chapter 4)
4. Prototype implementation details along with key features and limitations. (Chapter 5)
5. Evaluation and analysis of the designed architecture and protocol based on tests performed on the prototype. (Chapter 6)
6. Overall conclusion and suggestions for future studies. (Chapter 7)

2 Background related to Ambient Networks

This chapter provides an introduction to the Ambient Network Project and specifically to ContextWare concepts. It also briefly covers previous projects done within the area. Figures one through six in this chapter have been used with the permission of the Ambient Network project.

Ambient Networks is part of the Information Society Technologies (IST) priority in the Sixth Framework Programme (FP6) of the European Community for research, technological development, and demonstration activities. FP6 is part of the Wireless World Initiative (WWI) which aims to maintain European leadership in wireless technology by bringing together an industry led consortium of leading operators, vendors, and research organizations that work together with the determination, skill, and critical mass to create cross-industry consensus and to drive standardization. [8]

The Ambient Network project aims to connect the heterogeneous networks belonging to different operators and technology domains by creating a common network control layer so that these networks appear homogeneous to users. The design paradigm adopted in the Ambient Network is a horizontal layer of networks with a common control function, the Ambient Control Space (Fig. 1), that offers services to a wide range of applications. The central ideas are instant and dynamic composition of networks without a need for pre-configuration, in order to allow rapid adaptation of the network topology as needed for moving networks or users. This approach would enable network operators to easily manage and dynamically setup network configurations, while the users would benefit from transparent transitions from network to network and from access to services in different networks without needing manual intervention.

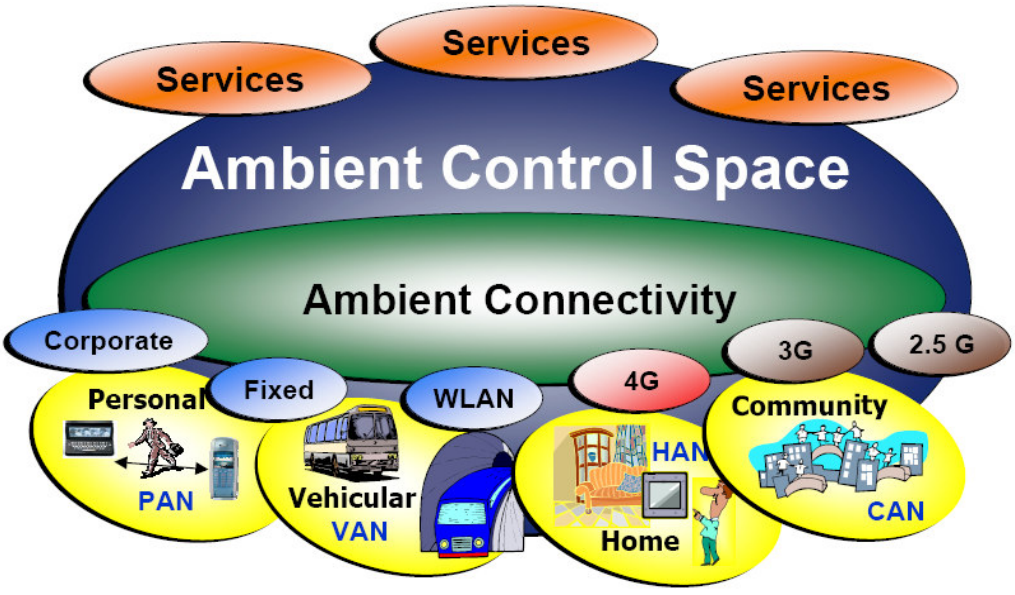

Fig. 1: High level view of the Ambient Networks with the common Ambient Control Space for the various network technologies [9]

The advantage of the Ambient Network paradigm is that it uses both existing and new networks as building blocks and connects all of these networks together into a larger system. As illustrated in the figure above, users could own the networks themselves, as in the case of a Personal Area Network (PAN), i.e. a network of devices directly associated with the user. This PAN can, using the Ambient Network architecture, connect seamlessly to other networks such as WLANs and the 3G networks.

The goals of such ambient networks are to provide a scalable and affordable mobile communication network that provides rich and easy to use communication services in a cost effective manner, while promoting competition and cooperation amongst operators and technologies; thus allowing for incremental market introduction of new technologies. [9]

2.1 Ambient Control Space (ACS)

The framework of AN functionality is a coordinated set of control functions, collectively referred to as the Ambient Control Space (ACS), which acts as a control layer to connect existing heterogeneous networks. The ACS manages the underlying data transfer capabilities and presents a set of well defined control interfaces towards other ambient networks, the supported services and applications. The ACS can be subdivided into two types of Functional Entities (FE): the actual Control FE and the Control Space Framework FE. The actual Control FE includes the Composition Management entity, Mobility Management entity, etc. These are embedded into the Control Space Framework (shown as boxes in Fig. 2). Whereas the Control Space Framework FE consists of functions, needed to assist the actual control functions in control and management tasks and coordination with other FEs in the control space. The different Control FEs achieve connectivity with each other by sending messages via the Control Space Framework FE.

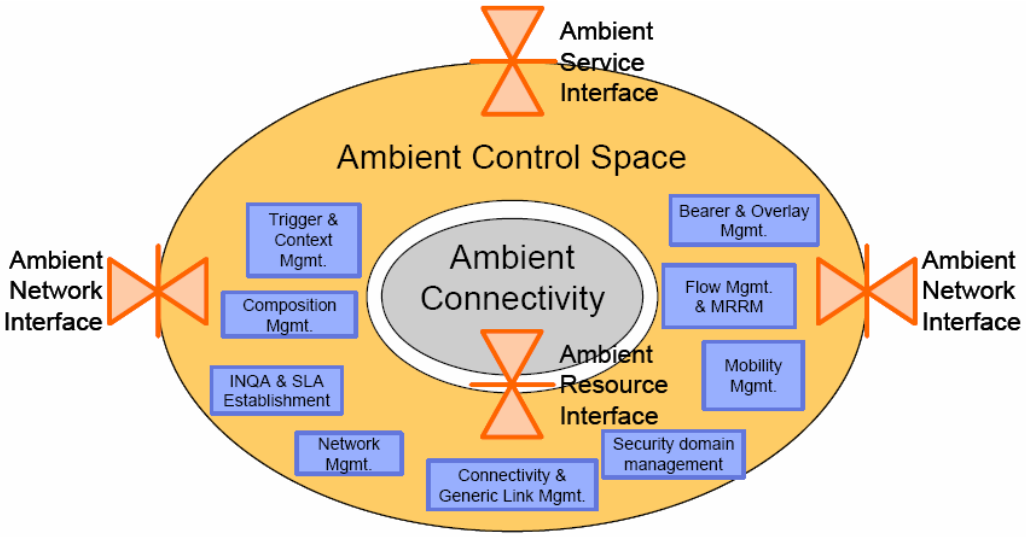

Fig. 2: Ambient Control Space and the Ambient Interfaces [11]

The key feature of the ACS architecture is the modularity achieved by decomposition into a set of FEs. Ambient Networks, internetworking is based on different technologies using existing internetworking approaches, such as translation or global layering.

ACS offers networks the flexibility to move either physically or logically at any time. As an example, a user can achieve restricted mobility within a network, but when she moves out of the range of this network, into another network, she might have to initiate connectivity from scratch again. Instead, using ACS, this connectivity is achieved transparently by providing a uniform abstraction of the underlying connectivity to the control space. By using the ACS functionality in the AN, both networks would have a common control layer, and this would thus help bridge the control gaps between connectivity islands, allowing a seamless transition for the user from one network to the other, giving the user the illusion of using a homogeneous network. [12]

The Ambient Control Space together with network connectivity via Internet Protocol (IP) is called an Ambient Network. The main characteristics of an Ambient Network are

- Well defined control interfaces to other ANs, applications, and services.
- Provides all or a subset of the ACS functions.
- Enable dynamic composition of ANs to form a new AN.

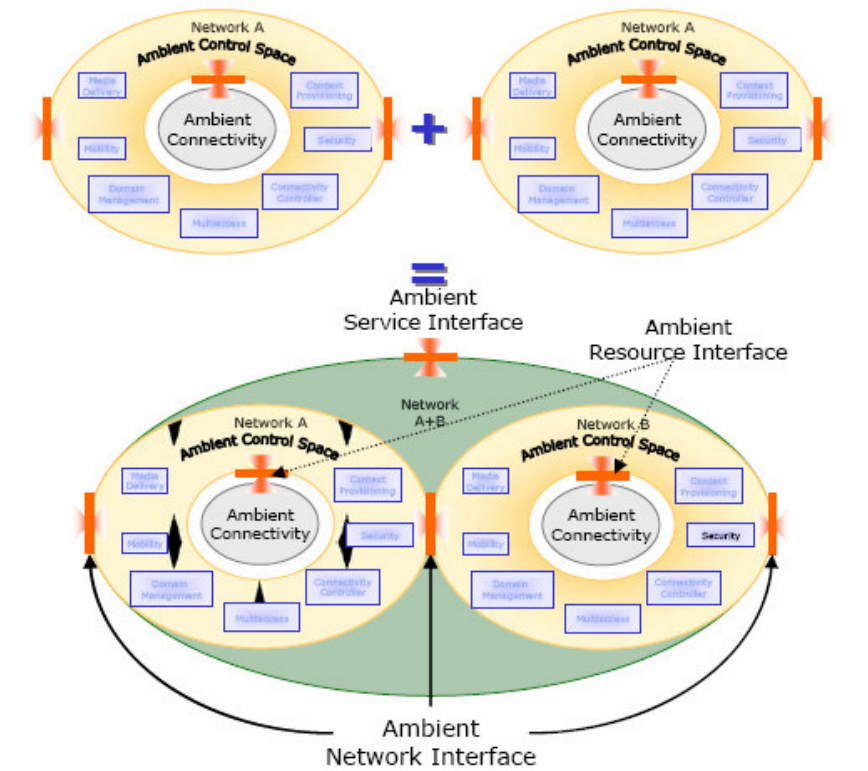

Fig. 3: Ambient Network composition creating a larger AN [12]

2.2 Ambient Interfaces

The Ambient Network provides access to the ACS through the use of three interfaces, the Ambient Network Interface (ANI), the Ambient Service Interface (ASI), and the Ambient Resource Interface (ARI). These interfaces are used by other ANs, applications, or connectivity resources (such as routers, media transcoders, etc.) to connect to the ACS.

2.2.1 Ambient Network Interface (ANI)

The Ambient Network Interface is an (horizontal) interface that connects the ACS functions of different ANs. The ANI creates a common ACS composed of different ANs by facilitating communication between the control spaces of participating ANs. [12]

2.2.2 Ambient Service Interface (ASI)

The Ambient Service Interface is an (vertical) interface that exposes the connectivity and control functions of the AN to the application layer. This interface lies between the ACS and the application and makes it possible to implement services without needing to worry about the heterogeneity of the underlying access networks. Using the ASI, network context information is made available to applications. [12]

2.2.3 Ambient Resource Interface (ARI)

The Ambient Resource Interface is another (vertical) interface located *within a node* between the ACS and the physical connectivity network. The ARI abstracts the specific control techniques of the underlying connectivity network, thus allowing the control functions of the ACS FE to operate seamlessly over a heterogeneous network infrastructure. The ARI provides control mechanisms for managing resources such as routers, radio terminal interfaces, access points, media transcoders, etc. [12]

2.3 Context Aware Networks

Context information is the situational or environmental information about a person, place, or thing. Making a system context aware enables the creation of an intelligent system. However, traditionally context awareness has only been used at the application level. Along with recent increase in the number of available access technologies and access networks, there has also been an increase in the number of devices available with a user, so that a user can interact with more than one access network. This creates the need to provide (heterogeneous) pervasive computing at the network level, thus creating the requirement for context information management *across networks and devices*. The aim of WP6 in the AN work package 6 (WP6), is to bring context awareness to the network level, thus making the network entities context aware.

> "ANs aim to support a common framework for context awareness across all functions in the ACS in order to adapt service availability and service delivery in heterogeneous networks and dynamic environments." [10]

Context information can be categorized into different categories as under [3]:

- Volatile or non-volatile context information -- depending on how fast the context information changes
- Real time or non-real time context information
- Private or public context information

- Network- centric (network connectivity and bandwidth, resources available, etc.) or user-centric (user profile, location, etc.) context information

Bringing context awareness to the network level requires network entities to collect relevant context information and to adapt based on this context information. So instead of the traditional concept of applications adapting to context information, in context aware networks, the network protocols adapt based upon the context information. This benefits the network services available through the functions of ACS, by providing them with raw context information and also the possibility of processing and using this processed context information. The user services can also benefit from network context information across different networks and domains.

2.4 ContextWare architecture

ContextWare is a term coined by researchers participating in WP6. This term is the name for a proposed architecture for management of context information. In broad terms, the task of ContextWare is collection of context information from context providers, management of context information within the ACS, and distribution of context information to context clients. These context sources and context clients could be either user applications or network services/entities. Through the use of the ContextWare architecture, the complete network can be made context aware. This enables not only applications, but also network management and protocols to adapt themselves based upon this context knowledge, thus directing the network entity's operation and decision making.

The various network services are referred to as the Functional Areas (FA) in an AN, and one of the goals of WP6 is to make these FAs context aware. ContextWare functionality mediates between the various FAs by managing the collection, processing, and distribution of context information. It simplifies the interactions between the FAs, making them efficient by reducing the number of interactions and the overhead of control traffic thus improving performance. The ContextWare architecture should to generic and not be dependent upon specific applications or devices.

The basic concept underlying the ContextWare architecture [3] [18] is context association which is a unidirectional relation from a context source to a context client. It is the direction of the flow of context information. Each context association has certain attributes linked to it such as Context Level Agreements (CLA), Quality of Context (QoC), modes of retrieval (such as server push or client pull mechanisms) etc.

CLA are required for the establishment and enforcement of the policies governing the kind of context information that is allowed to be exchanged between different parties. CLA can be negotiated both between functions within one AN, and between two different ANs to allow inter-AN context information utilization. Quality of Context (QoC) refers to the nature of the context information in terms of precision, probability of correctness, trust-worthiness, resolution, validity period, etc.

2.4.1 ContextWare Components

The ContextWare architecture has five main components, specifically:

- Context Coordinator
- Context Managers
- Context Information Base
- Context Sources
- Context Sinks

The ContextWare architecture is realized through two main Functional Entities(FE), the first implements the interface between the ContextWare architecture and other FAs in the AN and the second implements the internal operations in the ContextWare architecture. The two FEs are called the Context Coordination FE (ConCoord FE) and the Context Management FE (CM FE) and are briefly covered below. [4]

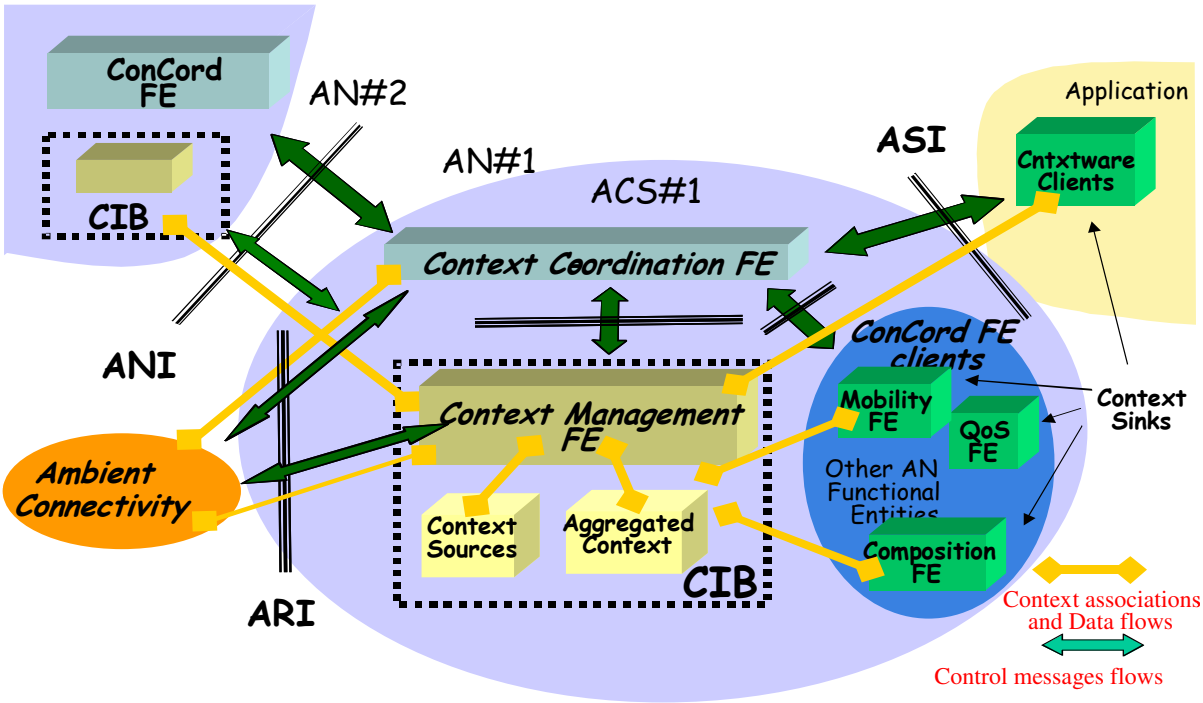

Fig. 4: ContextWare architecture [4]

In addition to these two main FEs, the ContextWare Architecture includes Context Sources, Context Sinks, and a Context Information Base (CIB). A Context Source is the source of context information and could be a context sensor, network service, or network resource. A Context Sink is the consumer of context information and places its request with the ConCoord to receive specific context information. The CIB is a repository of context information, collected from various Context Sources and it makes this information available to the Context Sinks.

2.4.2 Context Coordinator FE

The context coordinator (ConCoord) is a distributed registry and is the first point of contact for all clients trying to access context information. The context coordinator does not store the actual context information, but rather stores pointers to it. It returns the address of the context source (that has previously registered the requested UCI with the ConCoord) in response to a context client's query for a particular UCI. The context sources register with the distributed registry corresponding to the ConCoord and specify their location and UCI after proper authentication and authorization. The ConCoord also authenticates and checks the authorization rights of the context client before providing it with the location of the context source containing the requested context information. The primitives involved in the above context protocol are REGISTER, RESOLVE, GET, SUBSCRIBE, and NOTIFY and are depicted in the Fig. 5.

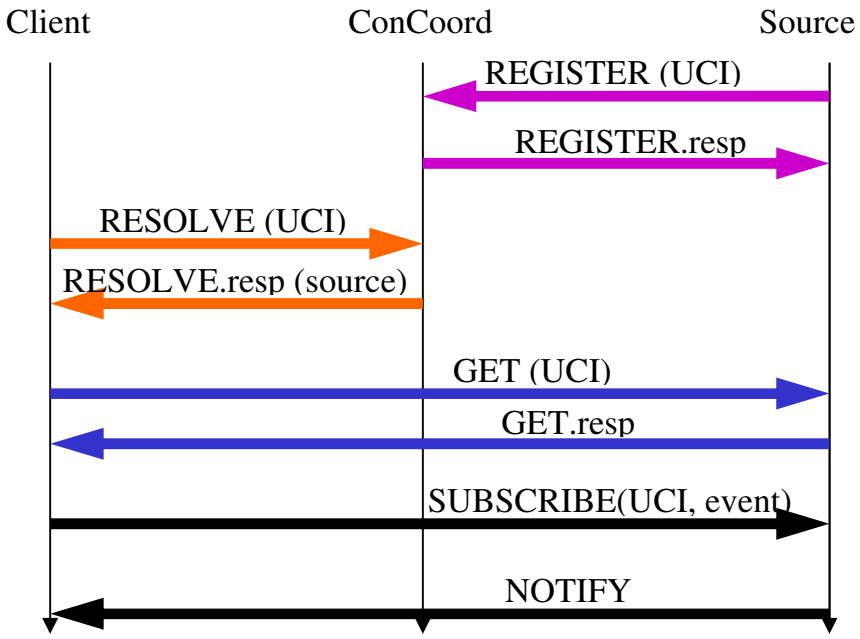

Fig 5: Flow sequence of context primitives [4]

The context sources REGISTER the UCI of their context information with the ConCoord. When the context clients contact the ConCoord using RESOLVE to get access to a specific item of context information, the ConCoord replies with the address of the source where the requested context information exists, if access is granted to the context client. The context client then directly fetches the context information from the context source using the GET primitive. The context client could alternatively SUBSCRIBE to a specific item of context information and whenever there is a change in this context information, the context client would receive a notification through a NOTIFY primitive.

2.4.3 Context Manager FE

The Context Manager (CM) entity is responsible for implementing the core internal operations within the context provisioning system which include collection,

management, and distribution of context information to interested entities and also management of context information sharing across domains.

The CM FE, if delegated, takes responsibility on behalf of the context sources to manage and distribute context information to the appropriate clients. The CM FE manages the contents of the CIB, distributing and storing the context information in the most appropriate location in the CIB to address the issue of scalability, access rates, and update rates. The CM FE also creates appropriate context associations for managing scheduled interactions between context sources and sinks as well as for aggregating context information to meet the client's requirements. [18]

Similar to the Context Sources, the Context managers, once created register their output type and capabilities with the ConCoord. The ConCoord's registry therefore maintains mappings of context information to the location of context sources and of context managers.

The context data received originally from context sources is in its raw form. This information can be used as such directly by context clients, but sometimes the context clients require information that has been collected from multiple sources and processed in some way. The processing of information, in the form of aggregating, correlating, or filtering of context information is done by the Context Manager (CM). The entities involved in context processing are the context clients, the context sources, and the CMs.

Lets assume a context client requires context information from one or more sources of type T_1, T_2, …, T_n. The context manager contains a processing function 'f' that transforms the input context information of type T_i to output context information of type T. This resulting context information is then made available to the context client through the context protocol. [4]

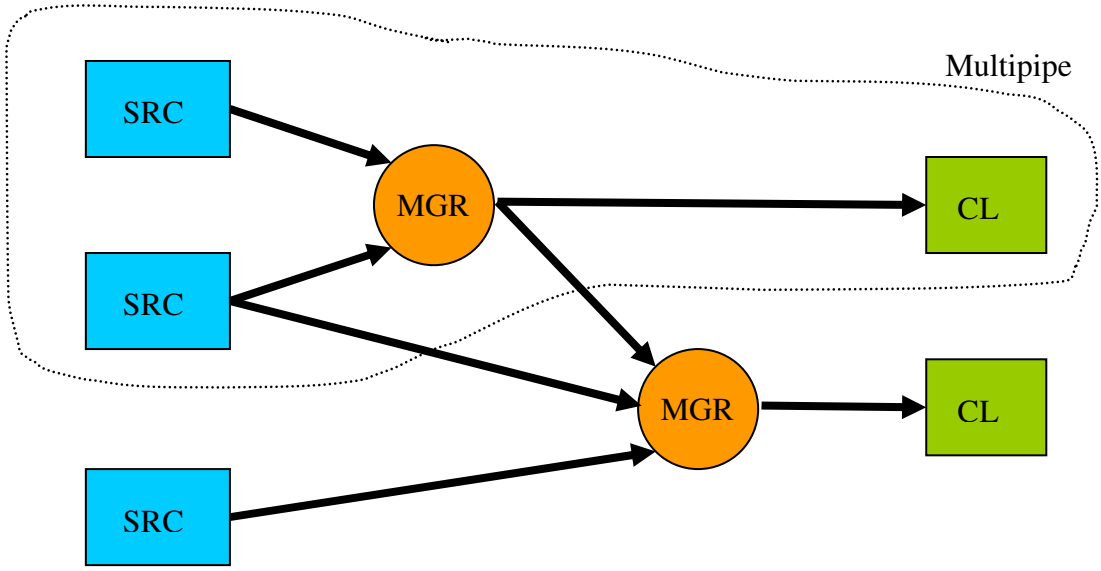

Fig 6 : Directed Acyclic Graph of Context Associations [4]

As seen in Fig. 6, the initial sources are nodes that have zero inputs, the final clients are nodes that have zero outputs and the CM are nodes that have non-zero inputs and outputs. In relation to a particular client, the graph that links all the sources and managers that lead to the client is called a multi-pipe graph for that client. In addition to getting inputs from initial sources, the CM could also get input from another CM - as long as the context information is type consistent. This enables recursive multi-pipe establishment in a distributed way. This method is advantageous in the sense that the partially processed context information can be used by different CMs in conjunction with different initial sources to produce varied context information, thus reducing (or avoiding) repeated processing. The ConCoord locates the final context manager, which locates the managers for its input, which in turn locate the managers for their input, and so on, until the inputs are all initial objects (i.e. basic context sources).

A CM can perform various processing functions, (such as filtering, converting, logging, aggregating, or correlating) on the 'raw' context information and depending on the type of processing, the CMs are of different types. CMs with only a single input are either filters, loggers, or converters; while CMs with multiple inputs are aggregators or correlators. Filter CMs extract specific input from the sources, logger CMs create logs and records, and converter CMs transform the format of the input context information.

2.4.4 Universal context ID

An item of context information can be considered as a data object, thus to reference context information, some form of unique identification is needed. This identification in the ContextWare architecture is called a Universal Context ID (UCI). A UCI is a type of Uniform Resource Name (URN) which is a Universal Resource Identifier (URI) that identifies a context by name in a particular namespace. The UCI simply uniquely denotes a context, without indicating its location or how to dereference it. The UCI can be represented in the following format [4]:

ctx://domain/path?options

where, "ctx" denotes the URI scheme for context identifier, "domain" is the domain name of the Ambient Network where the context object is defined, the "path" specifies the hierarchy of the existing context information using a set of case sensitive words separated by slashes and the "option" specifies further modifiers to the context information in the form of "parameter = value".

The context information can be referenced locally within a domain by omitting the network name as [4]:

ctx:/path?options

The difference between the two UCIs is that the fully-qualified UCIs contains a double slash after the colon and a slash after the domain name, while local UCIs contains only one slash and omits the domain name. The main advantage of local UCIs is that context clients can always retrieve their desired context information using the same name, even when the network changes. A given context object can have multiple UCIs in the form of aliases. Two different UCIs are equivalent if and only if

they refer to the same context information. There is work ongoing to standardize the UCI namespace in order to keep the UCIs unique.

2.5 Work done by Sergio Quintanilla Vidal

Sergio Quintanilla Vidal [6] has used a centralized architecture and communication protocol called Context Exchange Protocol (CXP) for collection, management, and distribution of context information in the Ambient Network. This design moves the complexity from the users' device to the network, thus minimizing the use of the resources in these devices. CXP is built over UDP, which is a stateless best effort datagram protocol. Thus CXP implements some reliability measures, specifically the addition of a sequence number, acknowledgements, buffering at the sender and receiver, and retransmissions. CXP messages use the Extensible Markup Language (XML) for communication. The centralized entity through which all other entities communicate is the CM. Each device may have some context information to share, as well as it could be interested in context information provided by other entities, thus each device or entity could both be a context source and context client and is given a generalized name of Context User Agent (CUA). Each CUA would not necessarily have access to all the context information provided by other CUAs and the required access rights are managed through access policies of each CUA. These policies specify which entities have rights to access the context information belonging to a CUA, but this is not sufficient for a real-world implementation as it doesn't specifically deal with control of a subset of the context information that could be accessed. Since the design uses a centralized architecture, the core entity is a very critical node and constitutes a single point of failure.

2.6 Adaptive & Context-Aware Services (ACAS) project

The ACAS project [5] deals with providing adaptive, user-centric Internet services to users moving within a heterogeneous infrastructure. Unlike the Ambient Network Project, where context exchange is at network level, the ACAS project exchanges context information between application layer entities that own several devices. At the core of the ACAS project is mobile middleware that enables application layer entities to automatically configure services amongst themselves, without any prior knowledge, by using context information. The context information is collected from sensors, then processed, and distributed to context managers, which make this information available to applications and services. The infrastructure that enables the services to be context aware is called the Context Information Network and the context managers in this system are called Context Management enabled Entities (CMEs). The Context Information Network is based on the IETF Session Initiation Protocol (SIP) for Instant Messaging and Presence Leveraging Extensions (SIMPLE).

A CME is the central entity in the Context Information Network and represents the application layer entity such as user, organization, or location. This application level general CME manages the context information held in devices linked to the user or application by communicating with the local CMEs present at the devices. This device CME communicates with the general user CME which is interconnected to other user CMEs too in the form of a network and this enables devices, entities, and applications to share context information. When the access to these CMEs is controlled through

rule guided policies, it is termed as a context service, which is a distinct entity that can be addressed.

3 Background related to Policy management

This chapter introduces the previous work done in the area of policy management and access control. It covers in some depth the policy and access control language, OASIS XACML which would be the base for the design of the policy management architecture in the next chapter. Figure seven and eight appear in this chapter with the permission of the Ambient Network project.

The use of context information in Ambient Networks can remove the need for pre-configuration of networks, hence making them autonomic. However, a concern that exists is the free and uncontrolled dissemination of context information breeching the privacy of the participants. It is extremely important to address these privacy issues in order to control who has access to what context information. This control can be achieved through the use of well defined policies. This creates a requirement for a framework in the ContextWare architecture for protecting context information. Ambient Networks require a flexible and scalable policy management system that can fulfill the requirements of all other functional entities in the ACS. Responsibilities of the management system include network management, service management, and context management. To realize these responsibilities, the system must deal with different types of policies such as those for handling mobility, security, context, composition, QoS, service provisioning, etc.

The management system must also provide a means to define security policies that can specify ownership and permissions and are "well written" in order to avoid conflicts with others within the network. To enable creation of policies that are "well written", we have to resort to the concept of Well Defined Services. Well Defined Services provides a common understanding of the service semantics for all the entities in the participating networks to identify the common services [23]. The policies defined on a high level outlining the system goals must be technology independent. These policies can then be interpreted into low-level actions thus improving scalability with fewer policies to manage and less policy maintenance time.

Policy-based networking has been used to enable an increased degree of automation, by linking actions in the network to system-events. Network management flexibility can be achieved without the need of modification in the implementation, using policies that dictate network behaviour based upon the available context information. Policies seek to control network behaviour using sets of high-level rules. It is possible to enhance the performance of such a system, by using systems that are context aware.

3.1 GEOPRIV (Geographical Location/Privacy) architecture

The GEOPRIV architecture deals with the authorization, integrity, and privacy requirements of information handling in a location based system. A lot of work has been done in this area and because of the similarities, the ContextWare architecture can use parts of the GEOPRIV privacy framework in order to control access to information stored in or referenced by the Context Information Base.

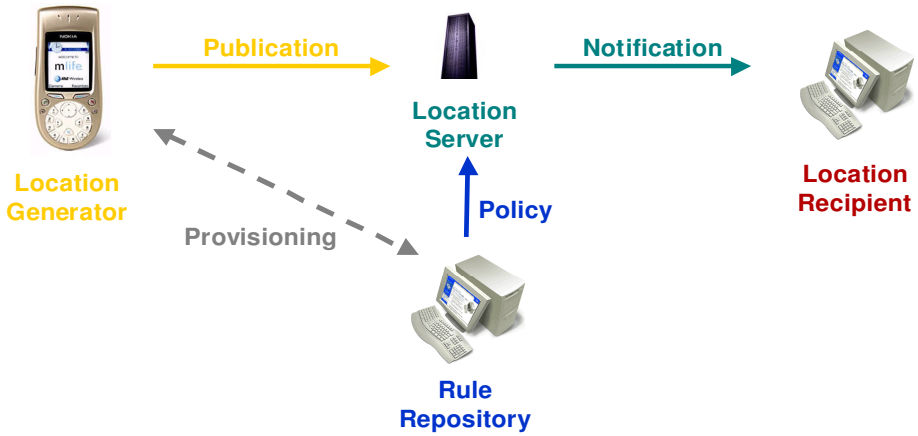

Fig 7: GEOPRIV architecture [4]

RFC 3693 [14] describes a protocol-independent model for access to geographical location. As shown in Fig 7, the Location Generator (LG) gathers the location of the Target and creates Location Objects (or in other words it produces Location Information (LI)) and publishes this information to the Location Server (LS). The LS receives this LI and applies the authorization policy rules received from the Rule Maker (RM) to this Location Object. The LS optionally receives subscriptions from a Location Recipient (LR). The authorization policies are rules that specify the specific conditions under which the LS is authorized to forward LI to the LR and with what precision. Depending on the rules, the identity of the LR, and the specific request from the LR to access location information of a particular target, the access to LI is granted to the LR with the reduced or enhanced data precision.

Comparing the components in the GEOPRIV architecture to ContextWare, and based on similarities of task, we see that the LR maps to the context client, the LG maps to the context source, the LI is the context information, and the LS maps to the ConCoord and CM. The LS in the GEOPRIV architecture or the CM and ConCoord in the ContextWare architecture play the role of Policy Decision Point. Further, the ConCoord handles the coarse grained policies, while the CM and context source handle the fine grained policies that are linked to particular context information.

3.2 IETF PBM Framework

The Policy Based Management (PBM) framework developed by IETF [16] is a framework that can represent, manage, share, and reuse policies and policy information. It is a model for policy management utilizing Policy Decision Points (PDPs or alternately policy servers), Policy Enforcement Points (PEPs), a policy repository, and a Policy Management Tool (PMT). A PDP handles requests by applying the relevant policies retrieved by querying the policy repository, making decisions, and distributing the result of the request to the PEPs. PEPs are entities (e.g. routers) where the actions are actually enforced. The PMT is used to create, edit, and administer the policies in the policy repository. Some protocols are necessary for the interaction between the entities, for example, COPS (section 3.4) is used for interworking between PDP and PEP, while LDAP is used by the PDP to access policies in the policy repository.

The IETF framework mainly deals with policy based networking to facilitate network management, device configuration, and traffic policing relative to Quality of Service (QoS) constraints. The view taken by IETF framework is compatible with a centralised model, but does not support the peer-to-peer technology used within AN [19]. Because of the dynamic nature of an AN, and the need for a dynamic policy-based infrastructure, the IETF policy framework designed for static networks is inadequate.

3.3 Policy Based Management Framework for Ambient Network (PBMAN)

PBMAN [17] is a policy based management framework within the Ambient Network based on a distributed architecture. It is a policy environment comprised of information and data models, a simple policy language called LPBMAN, and a policy interpreter. A prototype in the form of a proof-of-concept has been implemented for PBMAN using the DHT-based X-Peer middleware[1] [24] as the main enabling platform. The Policy Decision Network (PDN) is the central concept used in PBMAN. It is responsible for storing and retrieving policies and making decisions about requests. The PDN is comprised of PDN-Nodes and Repositories. PDN-Nodes can interact with other PDN-Nodes and PEPs via a peer-to-peer infrastructure, thus providing load balancing, fault tolerance, and scalability. The PBMAN architecture recognises PDN ACS and Agent ACS (comprised of User and PEP ACS) as types of implementations of ACS. The PBMAN architecture adds three new FEs to the ACS, namely, a policy FE, a P2P FE, and a Data Management FE along with two information repositories: the policy repository and the Management Information Repository (MIR).

An information model has been built upon three main types of management entities: policies, targets, and associations (as shown in Fig 8). Targets are entities such as services or users that policies may be associated with.

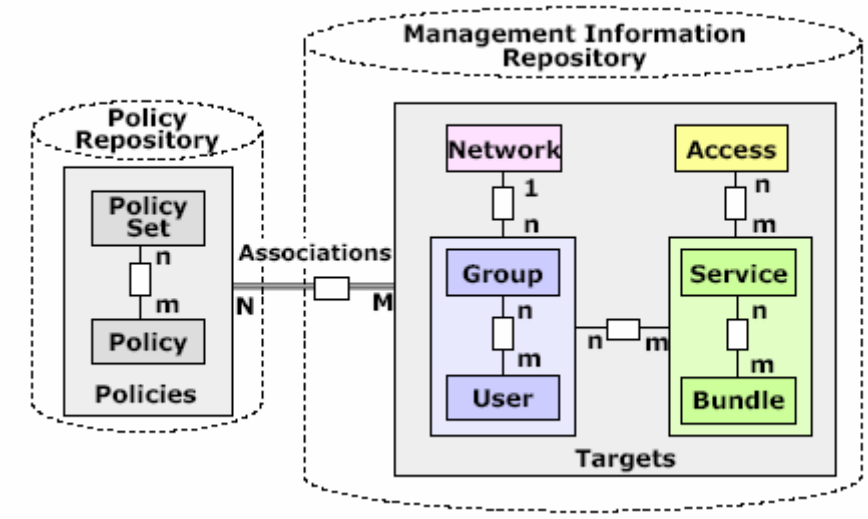

Fig 8: PBMAN Information Model

[1] Distributed Hash Table algorithm for performing lookups in peer-to-peer systems based on keys.

PBMAN also addresses the issue of composition in the Ambient Network and classifies compositions according to two criteria: type of ACS involved and mobility pattern. As mentioned earlier ACS recognises two types of ACS: Agent ACS and PDN ACS. Thus depending on the type of ACS, the composition could be: Agent/Agent, Agent/PDN, or PDN/PDN. Depending on mobility, the composition could be of type fixed/fixed, fixed/mobile, or mobile/mobile. Considering all the combinations of combining the two criteria, there could be nine types of composition in PBMAN.

To analyse and test the effectiveness of the PBMAN approach, a video on demand service was implemented, modelled, deployed, and tested using the X-PBMAN prototype which is implemented using the X-Peer peer-to-peer middleware [24].

The PBMAN approach used its own policy language which limits its use as a standardised PBM system. PBMAN does not address context awareness within the Ambient Network and deals only with Policy management without considering context information.

3.4 Common Open Policy Service (COPS) Protocol

As defined in RFC 2748 [15], the COPS protocol is a simple query and response protocol that employs a client/server model where the PEP, acting as a client, sends requests, updates, and deletes to the PDP; while the PDP, acting as a server, returns decisions back to the PEP.

The connection between the PEP and remote PDP uses Transmission Control Protocol (TCP) as its transport protocol making the exchange of messages reliable. The TCP connection is initiated by the PEP. Subsequently, the communication between the PEP and the remote PDP is mainly a stateful request / decision exchange. The COPS protocol is stateful in two main respects:

1. Request/Decision state is shared between client (PEP) and server (PDP). This means that requests from the client PEP are instantiated and remembered by the remote PDP until they are explicitly deleted by the PEP. Thus, decisions to the remote PDP can be generated asynchronously given a current instantiated request state.
2. State from various events (Request/Decision pairs) may be interdependent. This means that the server may respond to new queries differently because of previously instantiated Request/Decision state(s) that are related.

The protocol is extensible and was created for the general administration, configuration, and enforcement of policies.

The COPS protocol provides message level security for authentication, replay protection, and message integrity. COPS can also reuse existing protocols for security such as IPSEC or Transport Layer Security (TLS) [20] to authenticate and secure the channel between the PEP and the PDP.

The PDP can also send unsolicited decisions to the PEP, e.g. the PDP can force the PEP to change its behavior. Similarly, the PDP can send information for accounting or monitoring purposes to the PEP.

The PEP out sources the decision making to the PDP. Although the PEP can make local decisions under the direction of the local PDP, the final decision is made by the remote PDP.

If the remote PDP is not available, e.g. due to a network error, the PEP must try to connect to a backup remote PDP or revert to a local PDP. However, when the connection to the remote PDP is restored, the PEP should update this (remote) PDP based upon the decisions which occurred locally during the disconnection.

3.5 OASIS XACML [13]

The OASIS eXtensible Access Control Markup Language (XACML) is a standardised access control policy language utilising XML syntax for managing access to resources. It can be used for creating generic (i.e. usable in different environments), extensible, distributed (across different networks and application), and expressive policies.

XACML is the common language that creates a link between systems that: create access policy, collect whatever information is necessary to determine compliance with that policy, evaluate compliance, and enforce the policy.

The OASIS XACML standard provides an access control policy language that is for defining restrictions for accessing each resource together with a request and response language. It also provides a framework which uses the policy language. The request/response language supports queries and depending upon the request, either permits access or denies it. In addition, a Policy Decision Point (PDP) based on XACML, supports functions for finding a policy applicable to a request and also for evaluating the queries to decide whether access is granted or not.

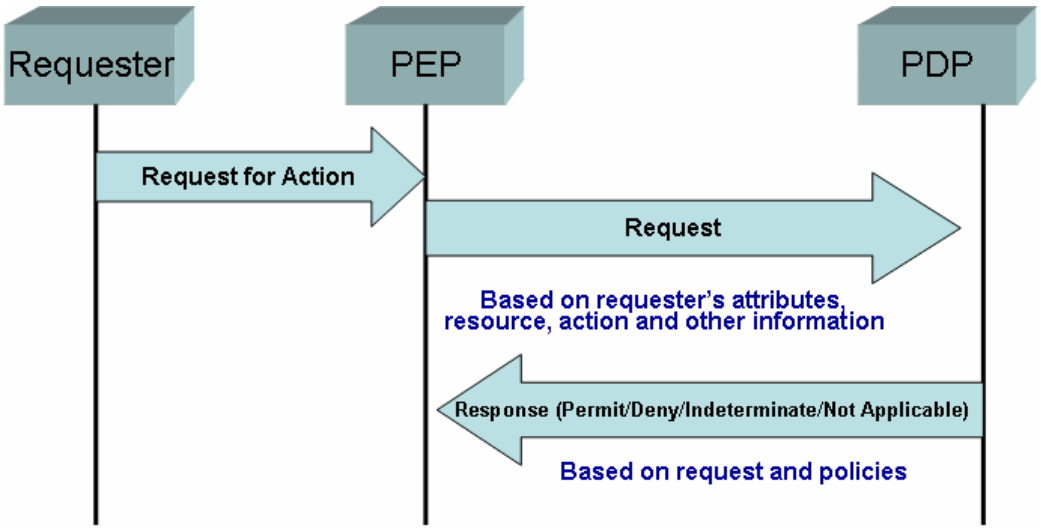

Fig 9: Flow Diagram of access control request/response language of XACML

In a typical scenario, a requester wants to take an action on a resource and forwards a request to the Policy Enforcement Point (PEP) that is associated with the resource and that protects the resource. As shown in Fig 9, the PEP formulates a decision request based on the request sent by the requester, the requester's attributes, the resource, and other relevant information, and forwards this request to the Policy Decision Point (PDP). The PDP processes the request based on policies relevant to the request and responds whether the access can be granted or not.

Policies are needed to handle requests from entities that attempt to access certain resources. The entity has to be an authenticated user to avoid leakage of information. The request should specify who is making the request, what resources are needed, and what action needs to be taken by the resource. In very simple terms, a request can be stated as:

```
<Request>
      <Subject>Alice</Subject>
      <Resource>Print</Resource>
      <Action>Print</Action>
</Request>
```

3.5.1 Rules and Policies

The policy relevant to a particular request, i.e., a decision request, could be composed of one or more rules or policies. The basic top-level policy elements in XACML are *Rule*, *Policy,* and *Policy Set*. A Rule element consists of basic Boolean expressions. Note that a rule element cannot be used independently in a PDP, but has to be encapsulated in a Policy. The main components of a rule are: Target, Condition, and Effect. The Target component is explained in section 3.5.3. The Condition is an optional component and refines the applicability of the rule beyond the applicability defined by the Target. The Effect component specifies the result of the Rule when it evaluates to true; it has one of two values: Permit or Deny.

A policy element consists of one or more Rules, a Target, a method or algorithm for combining the rules to make an authorization decision, and an optional obligation that has to be fulfilled by the PEP before the result of the request can be used by the requester. A Policy Set element consists of one or more Policies or Policy Sets or references to policies in other locations and a method or algorithm for combining these Policies and Policy Sets to make an authorization decision. In addition it may contain a Target and obligations.

3.5.2 Combining Algorithms

The algorithms for combining the results of individual Rules and Policies are called combining algorithms. There are standard combining algorithms defined in XACML and these can be identified by *RuleCombiningAlgID* or *PolicyCombiningAlgID* depending on whether it is used in a Policy or Policy Set element respectively. XACML also provides a syntax for creating user defined combining algorithms. The

standard algorithms within XACML are *Ordered or Unordered Deny-overrides*, *Ordered or Unordered Permit-overrides*, *First-applicable*, and *Only-one-applicable*.

3.5.3 Target

In addition to processing the request from the PEP, the PDP also has to find the Policy or Policy Set that is linked to the particular request. This is achieved through another feature defined in XACML called the Target. A Target is a set of simplified conditions for the Subject (an actor whose characteristics can be referenced in a condition statement), Resource (data, service or system component), Action (an operation on a Resource), and Environment (a set of attributes independent of the subject, resource, or action that is linked to an authorization decision), that must be met for a Rule, Policy, or Policy Set to apply to a given request. Thus the Target can find the relevant policies that apply to a given decision request. When a request comes to a PDP, the PDP verifies that the attributes defined in the target of the rule match and satisfy the attributes for the subject, resource, action, and environment defined in the request context. To enable quick lookup of policies, the Target also provides a way to index these policies.

Once the relevant Policy that applies to a given request is found, the Rules in the Policy are evaluated. The Rules are Boolean conditions that evaluate to a true or false result and for a true result the Effect of the Rule is either a *Permit* or a *Deny*. If the condition evaluated to an error, then the result of the Rule is *Indeterminate*, while if the condition doesn't apply to the request, then the result is *Not Applicable*.

Based on the combining algorithm type used for the rules or policies, the combined results would vary. As an example, in the case of deny-overrides, if a single rule or policy evaluates to deny, then regardless of results from other rules or policies, the combined result is deny. In contrast to this, for permit-overrides, a single permit result means a combined permit result regardless of results from other rules or policies.

The XACML schema for a target is:

```
<xs:element name="Target"type="xacml:TargetType"/>
<xs:complexType name="TargetType">
  <xs:sequence>
    <xs:element ref="xacml:Subjects" minOccurs="0"/>
    <xs:element ref="xacml:Resources" minOccurs="0"/>
    <xs:element ref="xacml:Actions" minOccurs="0"/>
    <xs:element ref="xacml:Environments" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Policies should specify to which requests they apply. This is done through the Target attribute. The policy should also specify if the policy is to be applied to a particular request, what effect will it have on the request. A single Policy can apply to multiple

subjects and the policy will apply to each of the subjects (disjunction) one at a time. A simple example of a policy applied to multiple subjects is:

```
<Policy PolicyId = "Simple Policy">
   <Target>
      <Subjects>
         <Subject>Alice</Subject>
         <Subject>Bob</Subject>
         <Resource>Printer</Resource>
         <Action>Print</Action>
      </Subjects>
   </Target>
   <Rule RuleId = "PermitRule" Effect = "Permit"/>
</Policy>
```

3.5.4 Attributes

The basic unit in XACML is an attribute. Attributes are characteristics of a Subject, Resource, Action, or Environment that is referenced in a Target. A request sent by a PEP to a PDP is composed entirely of attributes which are then compared to the attributes in policies to reach the access decision. A Named Attribute is an instance of an attribute that is recognized by an attribute name, type, the identity of the Subject, Resource, Action, or Environment that holds the attribute, and optionally the issuer of the attribute. A target element can be specified by multiple attributes defined by different identifiers and could be of different data types. All the attributes should match (conjunctive) for the policy to be effective. An example is shown below:

```
<Attribute AttributeId ="subject-id" DataType="string">
    Alice
</Attribute>
<Attribute AttributeId ="securityLevel" DataType="int">
    3
</Attribute>
```

The PDP processes the decision request, resolving the attribute values, using two mechanisms: *AttributeDesignator* and *AttributeSelector*. The *AttributeDesignator* is defined in the policy and specifies an attribute for the Subject, Resource, Action, or Environment with the given name and type. The PDP tries to match the values specified in the Policy with that of the value in the request. The *AttributeSelector* is defined in the Policy and finds the attribute values through the data type and an XPath expression which identifies the location of an attribute value.

3.5.5 Attributes with multiple values

The result of both AttributeDesignator and AttributeSelector can be multiple values and is always returned as a special attribute type called a Bag; even if none or only one value matches. A Bag is an unordered collection of values which may contain duplicate values.

To reach an access decision, the resulting attribute values in the Bag are passed through a set of Functions that can operate on any combination of attribute values to return the attribute value supported by the system. These functions make it possible for the PDP to unambiguously handle the case of multiple attribute values. These functions can be used in various ways including in nested functions arranged in a complex hierarchy. An example policy for finding the supported attribute is:

```
<Policy PolicyId = "Simple Policy">
  <Target><Subjects><Subject>
    <SubjectMatch MatchId="string-equal">
      <AttributeValue DataType="string">
        Alice
      </AttributeValue>
    <SubjectAttributeDesignator
      AttributeId ="subject-id"/>
    </SubjectMatch>
  </Subject>…</Subjects></Target>
  …
</Policy>
```

3.5.6 Policies in a Distributed System

Instead of storing all policies at a single point, the policies can be spread out in a distributed system. This means that the policies could be written by different people and enforced at various different points. This also enables the Policies to be modified and updated independently as needed. XACML facilitates the collection, combination, and processing of independent policies at a PDP by comparing attributes with the relevant Target element relevant to the decision request.

To make extracting and finding the correct Policy for a decision request manageable, it is necessary to index and identify the policies. This is done based upon the Target element in XACML. Policies can be stored in a database and the relevant policies can then be retrieved by the PDP through a database query. Or the policies could be locally stored at the PDP and the Target element could be evaluated for a particular decision request to select the relevant policy.

3.5.7 XACML context

The resources controlled by the different PEPs could be of various forms, e.g. a web server or a remote-access gateway, and they could each use a different format to issue decision requests. In order for the PDP to communicate and process requests from multiple resources, either the policies in the various PEP have to be written in all the

different formats understood by all PEPs in the system or the PDPs could utilize a standard format, such that any PEPs that use a format different from this would use a translator to convert the request into the format understood by this PDP.

The latter approach has been adopted in XACML and the standard format for requests and responses by the PDPs is called the **XACML context** and its syntax is defined as an XML schema. The PEPs that issue requests in a different format have to pass these requests through an intermediate step to convert them into XACML context in order to be understood by the PDPs. This approach enables the Policies to be written and analyzed independently of the environment in which they have to be enforced. The XACML context defines a recognized representation for the input and output of the PDP attributes. The actual conversion from the application environment to the XACML context is a task for the individual implementations and is outside the scope of the XACML specification.

3.5.8 Data Flow Model

Fig. 10 below shows a detailed data flow diagram encapsulating the access control request/response feature of an XACML supported system.

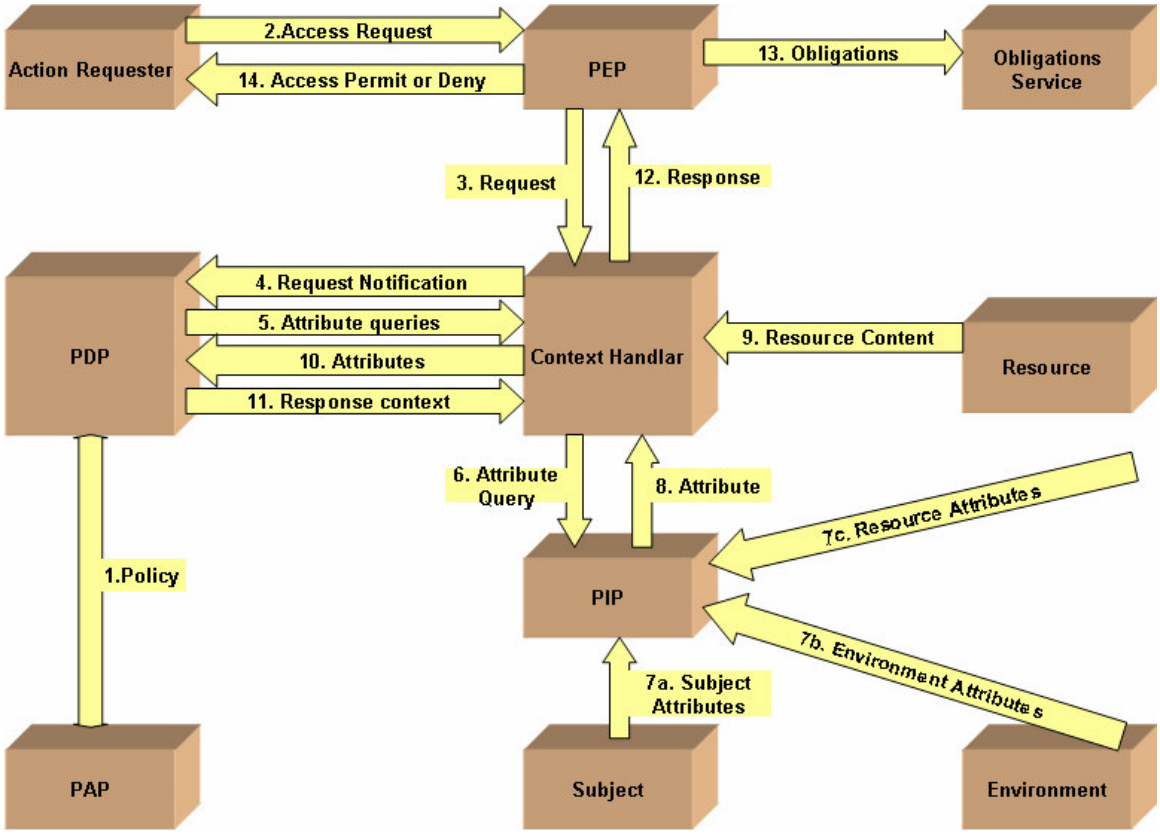

Fig 10: XACML Framework

The policies and policy sets are created by a Policy Administration Point (PAP) and made available to the PDP depending on the relevant Target. The requester that needs access sends an access request to the PEP, which then forwards this request, along with optional attributes for the Subject, Resource, Action, and Environment, in its native language, to the context handler which acts as a translator. The context handler

transforms the native request to XACML context and sends it to the PDP. The PDP queries the context handler for any available attributes. The context handler then forwards this query to the Policy Information Point (PIP). The PIP obtains the requested attributes from the relevant Subjects, Resource, and Environment and returns them to the context handler. The context handler then sends this information to the PDP along with the optional resource information. The PDP processes the request and evaluates the policy and sends the authorisation decision back to the context handler. The context handler transforms the result back into the PEP native language and returns it to the PEP. Even if the PDP has returned a Permit result, there might be some obligations defined for the PEP to fulfil before it can grant access to the requester. If the authorisation decision was permitted, the requester is granted access, otherwise the access is denied.

Let us assume an example control policy from a corporation named Medi Corp (med.example.com) that states: "Any user with an e-mail name in the "med.example.com" namespace is allowed to perform any action on any resource."

Following the flow in the model diagram,

1. The Policy Administration Point (PAP) that is responsible for creating Policies and Policy Sets, makes this policy available to the PDP.
2. When an entity, for example Bart Simpson, with e-mail name bs@simpsons.com, tries to get access to some context information that could be his medical record, he sends an Access Request to the PEP.
3. This request could be in the native request format of the requester and is forwarded to the Context Handler along with optional attributes for subject, resource, action, and environment, where it is translated into XACML context. In this specific example, the subject is the email address of Bart Simpson, the resource is his medical record and the action is read the medical record.
4. This request is forwarded to the PDP.
5. To process this request, the PDP requires attributes related to the request and sends a query to the Context Handler.

6, 7, & 8. The Context Handler forwards this query to the Policy Information Point (PIP) which collects the requested attributes from the relevant Subjects, Resources, Action, and Environment and passes them back to the Context Handler.

9 &10. The Context Handler returns the attributes to the PDP optionally also adding the resource, which is the medical record in this case, in the context. The PDP first locates the policy related to this request by comparing the subject, resource, and action in the request against the subject, resource, and action in the policy target. Having found the policy, the PDP processes the request in relation to the rules that are specified in the Policy. For our specific example, the PDP compares the Subject, Resource, and Action in the Request against the Target of the one Rule in this policy. The requested resource matches "AnyResource" and the requested action matches "AnyAction" in the Policy, but the requesting subject-id does not match "@med.example.com".

11. The PDP returns the authorization decision and the response context to the Context Handler which in this case would be Deny access.

- The Context Handler performs the translation of the response back to the native format and sends it to the PEP.

13. There might be some obligations that the PEP would have to fulfill to grant access to the Requester even with a Permit result.
14. Depending on the authorization decision, the Requester is granted or denied access.

3.5.9 The advantages of XACML

XACML is not in itself a complete authorization solution, but it provides a foundation upon which integrated solutions can emerge. It provides a host of advanced features that make it well suited for tying together large-scale authorization solutions. The many advantages of XACML compared with traditional proprietary languages are:

- **It's standard**. As with using any other standardized solution, once XACML is more widely deployed, the interoperability with other applications will become easier.

- **It's generic.** This means that rather than having to provide access control for each particular environment or each specific kind of resource separately, developers can use XACML in any environment. One policy can be written that can then be used by many different kinds of applications, and when one common language is used, policy management becomes much easier.

- **It's distributed.** This means that a policy can be written in a way that in turn refers to other policies kept in arbitrary locations. The result is that rather than having to manage a single monolithic policy, different people or groups can manage separate sub-policies as appropriate, and XACML knows how to correctly combine the results from these different policies into one decision.

- **It's powerful**. While there are many ways the base language can be extended, many environments will not need to do so. The standard language already supports a wide variety of data types, functions, and rules about combining the results of different policies. In addition to this, there are already standards groups working on extensions and profiles that will hook XACML into other standards such as SAML and XML Digital Signature, which will increase the number of ways that XACML can be used.

4 Design of a Policy Management System (PMS) Architecture

This chapter covers the design decisions and presents the designed architecture for Policy Management within an AN and for the composition scenarios. The design of the Policy Management Protocol for the transfer of messages is also presented here.

In Chapter 2, the background related to AN and ContextWare was presented. In Chapter 3, a few existing policy management architectures are covered along with an overview of the OASIS XACML standard. As mentioned in Chapter 1, the task at hand is to build a Policy Management architecture to manage authorisation of access to network context information and ContextWare entities within the ACS of an AN. This Policy Management architecture should also be capable of managing authorisation of access to context information from the application and service layer via the ASI and across an ACS of the constituent ANs via ANI. This architecture will be based on OASIS XACML.

Although, for the scope of this project, the Policy Management System (PMS) is only needed for handling access control policies, the actual scope for a PMS within the Ambient Network is more varied. A fully fledged PMS should be able to support policies for self management, triggering, composition, mobility, security, QoS, service provisioning, etc. in addition to access control policies. The policy management framework should be capable of Context Management, Network Management, and Service Management. The policies have to be authenticated in order to be secure and the framework should include policy negotiation capabilities to exclude conflicting policies. The policies need to be technology independent and should specify high level system goals that can be translated into low level actions. The design for a Policy Management System handling all the above capabilities is being developed within the Work Package D.3 within the AN project.

Taking into consideration the capabilities of a complete PMS, the PMS architecture for handling the access control feature has to be general and must be easy to extend.

4.1 Design decisions for Policy Framework and Policy Management in Ambient Networks

In this section we will examine the decisions made in the design of the Policy Management System.

4.1.1 Store all policies in a common repository

Within the Ambient Network, the access policies dictate rules for access to particular context information. These policies define the behaviour/access control of the network. The question is who should be responsible for defining the policies of a particular resource or UCI object? Is it the provider of the resource or is it some central administrator? Moreover there could me more than one entity that is interested in defining access policies for a single resource. In this case the decision has to be taken by all the policies defined by that resource. To provide a global view of the whole network it is necessary to consolidate the policies existing in a network.

The access policies specify the policies decided by the network administrator or the source of the context information regarding access to the context information it owns. These specific access policies might be in conflict with the general access policies (or some other policies) for the same context information existing in the Ambient Network.

Although policy negotiation and conflict resolution is out of scope for this thesis, in order to give a consolidated view of all policies within an AN, the design decision is to **store all policies within an AN in a common repository**. This repository is called a Policy Information Base (PIB).

Different entities may have conflicting configurations or policies, which may lead to inconsistent network state. With this approach, the conflicting policies would be resolved when storing the policies in the repository. This approach does not limit the distributed creation of policies. Thus policies relevant to different context information could be written by different policy writers and Policy Administration Points (PAP), but still stored at a common repository. The prototype implementation, implements a centralised file based repository. The decision of using a file based repository compared to a database repository is based on factors like performance and cost. Files tend to be faster to read than databases. When creating or modifying a repository instance the entire instance needs to be written, so file-based repositories are faster involving a single operating system operation compared to querying and taking appropriate action on a section of the database repository. It is easier to share data from database for redundant nodes compared to replicating file based repositories in the same state. But for the purpose of our implementation, we don't implement duplicate nodes and hence file-based repository is a good choice.

4.1.2 Centralised PDP and distributed PEP approach

The basic design decision that has been used in designing the architecture of the PMS for AN, is that **the management of access to context information is done in a centralised way**. This means that the entity that makes the decision about giving access to context information, based on policies, will work in a centralised manner.

But this centralised approach will only be restricted to the Policy Decision Point (PDP), thus the task of policy enforcement can be more flexible. The **Policy Enforcement Points (PEP) can be distributed** and exist in nodes belonging to different FEs. This decision is discussed further in section 4.2.

The Policy Management System is **based on the OASIS XACML** standard and uses the java XACML API's for the implementation.

ConCoord is the first point of contact for any entity joining the AN, and registering or requesting context information. Thus in our design, we designate the ConCoord as the first entity that controls access by initially accepting requests for context information, transforming this request into an XACML request, and subsequently enforcing the decision received from the PDP regarding access rights of the requester based on *general access policies* existing in the AN. If the decision is to permit access, then the location of the source of the context information is returned to the requester and the requester directly requests the context information from the source of the context

information. The rationale behind this is that a source registers an object only once with the ConCoord and similarly a client also resolves each UCI object at the ConCoord only once. Any subsequent interaction is then between the source and the client directly. In our design, access control based on detailed or *fine grained policies* for context information is enforced by the actual source that is responsible for the context data or the particular context manager, if the source has delegated responsibility for handling the context information to the CM.

Here the specific access policies for the requested UCI object belonging to the source or the CM come into play. The source may give access to specific context information to a client only for a period of time, or a time to live (TTL) value, after which the client would have to again request for access from the source. Thus although the client would know the location of the source of context information, the actual access to the context information can still controlled. The concept of having a TTL value for the evaluated access response has its advantage because it decreases the load on the PMS for repeated evaluations of the same UCI object.

Access to aggregated context in the form of filtered, transformed, or aggregated context information is controlled by the specific CM identified by the UCI. For aggregated context, the policies related to all the context information in the aggregated context have to be aggregated or combined into a common Policy Set and connected to the newly generated context information. The policy combination should be done in such a way so as to avoid conflicts. Conflict resolution can be done using some context resolution software (refer to section 7.2). When the common policy set has been created any additional policies needed, can be added.

4.1.3 Other Decisions

It is assumed that the nodes belonging to an AN are part of a security domain [25]. The domain manager of this domain issues public key certificates to member nodes whose cryptographically generated identifiers have been signed by the private key of the manager. Thus the **entities joining the Ambient Network are already authenticated** and a security infrastructure exists within the AN.

The policies are identified by a policy identifier <PolicyId> and this **policy identifier has to be unique within a PDP of the relevant domain or the ambient network** in which the policies apply. This motivates the need for creating a standardized mechanism for naming policies so that each policy has a unique identification within an AN and also in the new AN created after composition.

For the purpose of this thesis, the following policy naming convention is used.

> The Policy ID should have in addition to a policy name or number, an indicator specifying the source of its creation. Each source is an encrypted entity and has a unique public key ID - which is globally unique. The policy name or number has to be unique for each source.

For example, a policy with Policy ID prefix, P1 belonging to the Source S1 would have the complete Policy ID:

PolicyID = “P1:S1”

If there are more than one policy linked to a source, the policy id prefix for each policy has to be different, so that at any given time, the policy ids are unique.

For example another policy related to the same source S1 would have a Policy Id prefix different from P1 and could be:

Policy Id = “P2:S1”

In case of composition, when AN1 composes with AN2 to form AN3, the policies that would be used in the newly composed AN, would be a subset of policies from AN1 and AN2 and will be unique due to the fact that policy ids are based on the source of their creation. Thus the previously mentioned policy, P1:S1 in the composed scenario would still be unique as it is linked to a unique source.

4.1.4 Policy types

Depending on the type of management being enforced, the policies can be of different types. The policies related to access control can be categorised as:

1. **Authorization Policies**: Authorization policies define the tasks that a service or client is authorized to do on a target object. These policies are *target based* upon the relevance of a policy in terms of a particular request as decided based on the target. The policies could be either positive (i.e. Permit-biased) or negative (i.e. Deny-biased).
2. **Obligation Policies**: The obligation policies define the obligations or activities that a particular subject or Policy Enforcement Point (PEP) should fulfil before access can be granted to the subject. These policies are *subject based*, as it is the responsibility of the subject to interpret and enforce these policies.
3. **Delegation Policies**: The delegation policies define under what conditions the authorization responsibility can be delegated and to whom.

4.2 Analysis of the centralised approach for the PMS

The policies within an AN can be created and owned by different FEs. These policies are used to define access rights to the context information owned by these FEs.

One approach to manage the access to context information within an AN could be that each FE is responsible for managing access to the context information owned by it. This would mean that each FE would have its own PEP, PDP, and PIB for access control. There might be multiple access policies defined by different entities for a single UCI object. This distributed policy decision approach would not provide an overall view of the policies in the AN.

The approach we have taken in the design of PMS is to have a consolidated centralised Policy FE for the complete Ambient Network that manages access control to all the context information existing in the AN. To enable this, all the policies belonging to the different FEs are stored in a centralised repository. This approach can enable conflict resolution of policies at storage time. Management involves evaluating

the request for access to context information against the existing policies within an AN and arriving at a decision. Thus it is the policy decision entity i.e. the Policy Decision Point (PDP), that works in a centralised manner. Each FE using the services of the Policy FE would have its own PEP since policy enforcement is a task on the specific context information and would still be done at the location of the context information. The PEPs would be created as per the specific needs of the different FEs. Thus the PEP would be distributed in the AN.

The advantages of having a centralised approach for PDP is that the use of policy management within an AN is still a very new area and a centralised PMS makes it easier for the various FEs to define/handle/change/backup their policies. It is assumed that at this stage there are a limited number of policies in use. More over, having all the policies together makes it easier to detect and resolve conflicts. The policies in XACML are indexed based on their *targets* and the policies applicable to a particular decision request are identified based on the target element of the policies. This motivates use of a centralised PIB that can evaluate the request based on all relevant policies within the AN, rather than using a subset of the policies. The Fig. 11 depicts the centralised PDP and distributed PEP approach taken in the design.

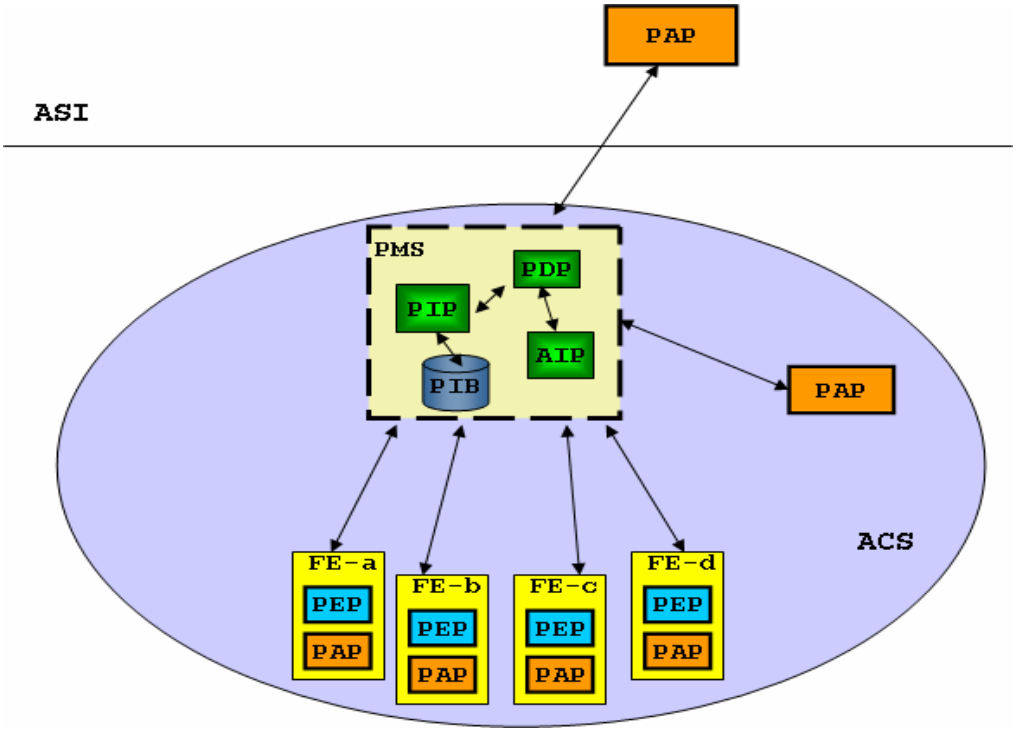

Fig 11: Centralized PMS and distributed PEP approach

Evaluating existing Policy Based Management Systems such as that one developed by IETF (see section 3.2), we see that policies are a centralised management instrument Thus using a centralised PMS belonging to an AN, is an initial step in exploring policy based management. Once the use of policy based management has reached substantial maturity, it can be adapted to a more decentralised model as a future step.

Although easier to implement and use, a centralised PMS has some major constraints. Having a single centralised PDP presents a **single point of failure**. The failure of this

node can affect the operation of the AN in a very drastic way. Thus it is essential to support redundancy of this node. This can be achieved by having a duplicate PDP and a duplicate repository. The duplicate repository must maintain the same state as the original repository. The duplicate PDP must also maintain the same state as in the original node since the access control decisions have a time to live factor and in care of the failure of the original node this data would be lost if not replicated which would mean re-evaluation of all the access requests. This re-evaluation would lead to high peak in initial load and would affect the performance of the PDP.

4.3 Policy Management Architecture

As discussed in Section 3.1, the basic elements of the GEOPRIV architecture can be mapped to the Policy Management architecture in AN. The general PMS architecture managing the ContextWare entities utilizes an architecture derived from the GEOPRIV architecture as shown in Fig 12.

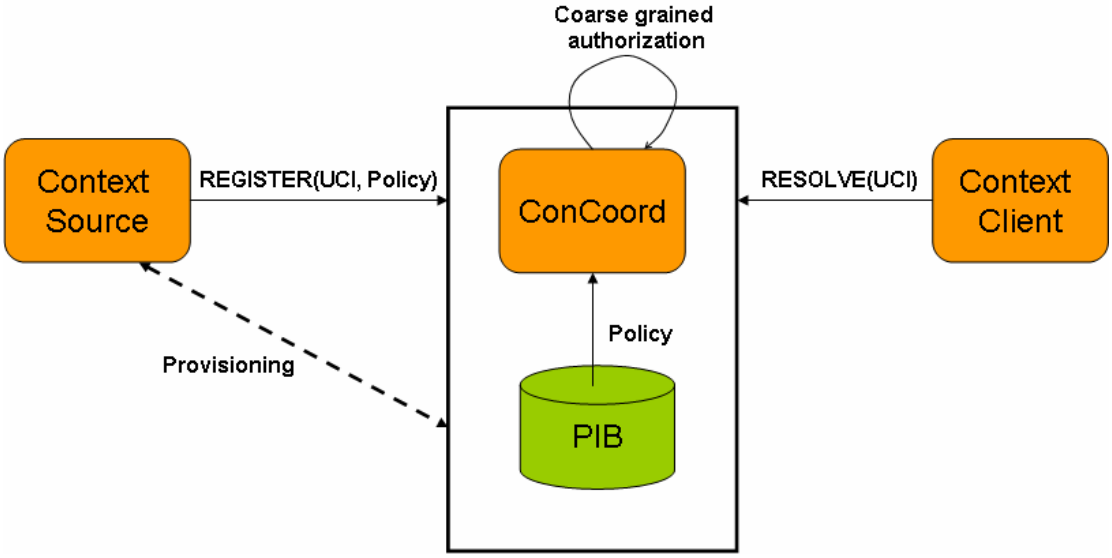

Fig. 12 Basic architecture of Policy Management System within AN

The Context Source registers the UCI of the context information with the ConCoord, while the policies related to the UCI are created by the source and are stored in the policy repository. When a Context Client tries to access certain context information, the access rights of the Context Client to the requested context information is checked first using the coarse grained policies of the ConCoord, then, if access is allowed, the location of the source of context information is revealed to the Context Client.

Applying the OASIS XACML framework and its elements on the high level architecture of Policy Management System within AN, the basic requirements for a policy framework in AN are:

- Policy Enforcement Points (PEPs) for communicating the access requests on behalf of Client entities and communicating the response along with the handling of obligations.
- Policy Decision Points (PDPs) for handling requests and evaluating them against the defined policies to reach a decision.

- Policy Information Base (PIB) which is the policy repository for storing policies.
- Policy Authoring Points (PAP) for creating and editing policies.
- Policy Information Point (PIP) or policy finder as an interface to the PIB and used for storing and fetching of policies.
- Attribute Information Point (AIP) or attribute finder which is an interface to the context repository and used for fetching the attributes related to a request.
- A Context Information Base (CIB) is the Management Information Repository which contains context information to which the policies are applied.
- A policy language such as XACML for defining policies for authorisation to context information.
- A policy mechanism to handle composition of ANs.

Based on the above requirements for the policy framework, the basic architecture of Fig. 12 can be further refined as shown in Fig. 13.

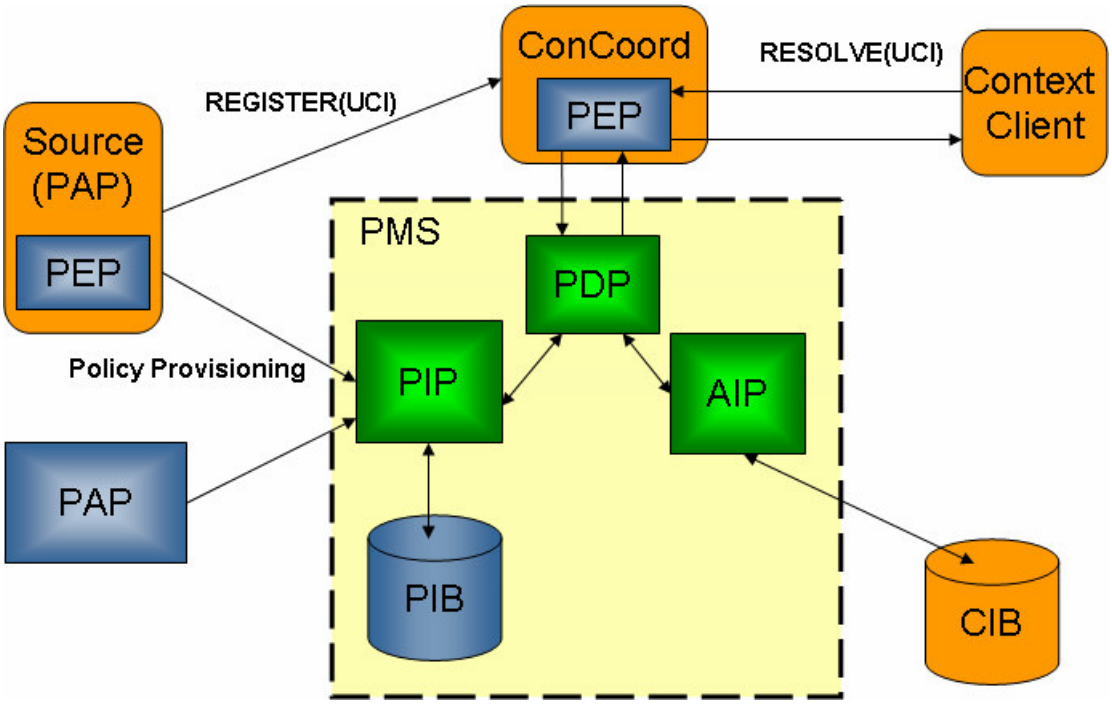

Fig. 13 Policy Management Architecture in AN

The Policy Management Architecture of Fig. 13 depicts a Policy Management System (PMS). The PMS can be used to facilitate the storing of the policies received from the source at the time of register and the policies received from the client at the time of resolve.

The ContextWare comprising of the ConCoord, the CIB, and the sources and clients of context information are assumed to be in a P2P network. When a context source joins the network and registers its UCI with the distributed ConCoord, the mapping of the UCI against its source location is stored with the ConCoord. The ConCoord provides the address of the location of the PMS to this source. The available access

policies belonging to this context source are registered by the source at the Policy Information Point (PIP) in the Policy Management System (PMS) which in turn stores the policies in the common repository, the Policy Information Base (PIB).

When a Context Client tries to resolve a UCI at the ConCoord, the ConCoord take the role of a Policy Enforcement Point (PEP) and transforms this UCI request into an XACML request and sends this request to the Policy Decision Point (PDP) in the PMS for evaluation. The PDP sends a query to the PIB via the PIP to retrieve the policies that are applicable to the decision request. This is determined by fetching the attributes using the Attribute Information Point (AIP) and evaluating the Target element of the retrieved policies or policy sets. The PDP sends back the response to the ConCoord PEP. The ConCoord PEP parses and analyses this response and if the response permits access, the location of the source responsible for this UCI is sent back to the Client.

As mentioned in section 4.1.2, the policies can be general and coarse grained, or they could be specific and fine grained. The first category of policies are related to the whole domain and carry a wider perspective thus they are evaluated initially when an entity contacts the ConCoord and requests for context information. The ConCoord here takes the responsibility of the PEP and forwards the request from the Client to the PDP and also conveys back the response to the Client after handling any obligations. The fine grained category of policies are more specific to the actual context information and are evaluated by the PDP either when the Client requests information from the source, i.e. the owner of the raw context information or the Context Manager, i.e. responsible for the processed context information. The Policy Management Architecture of Fig 13 is expressed as a special case for each of the categories of access policies.

Case 1: General access policies

As explained in the section 2.4.2, ConCoord is the gateway node in the ContextWare that is the first point of contact for coordinating context information between context sources and clients. Thus the ConCoord is the most suitable candidate for a PEP and being responsible for initiating the evaluation of general access policies pertaining to the whole Ambient Network. All the general policies related to the complete AN could be created by a PAP and would be stored in a common repository. When a Client joins the network and subscribes to some context information at the ConCoord, the request for access would be handled by the PDP in the PMS of the AN. The PDP would fetch the relevant policies from the PIB through the PIP and the concerned attributes from the CIB through the AIP and based on the policies take a decision regarding providing the location of the source of context information to the context client.

Case 2: Fine grained policies at source

When a source REGISTERs with the ConCoord at the time of joining the AN, the source stores the access policies related to its context information at the PIB through the PIP. Once the client has been granted general access by the ConCoord and learns the location of the source where the context information exists, it sends a GET or SUBSCRIBE directly to the source of the context information. Here the source acts as

a PEP and after transforming the UCI request into an XACML request, forwards the request to the PMS. This request is handled by the PDP in the PMS of the AN. The PDP fetches the fine grained policies relevant to the request from the PIB through the PIP and the concerned attributes directly from the source (if the source has not stored the attributes in the CIB) through the AIP and based on the policies makes a decision regarding providing access to the actual context information.

The architecture for this specific case is as shown in Fig. 14

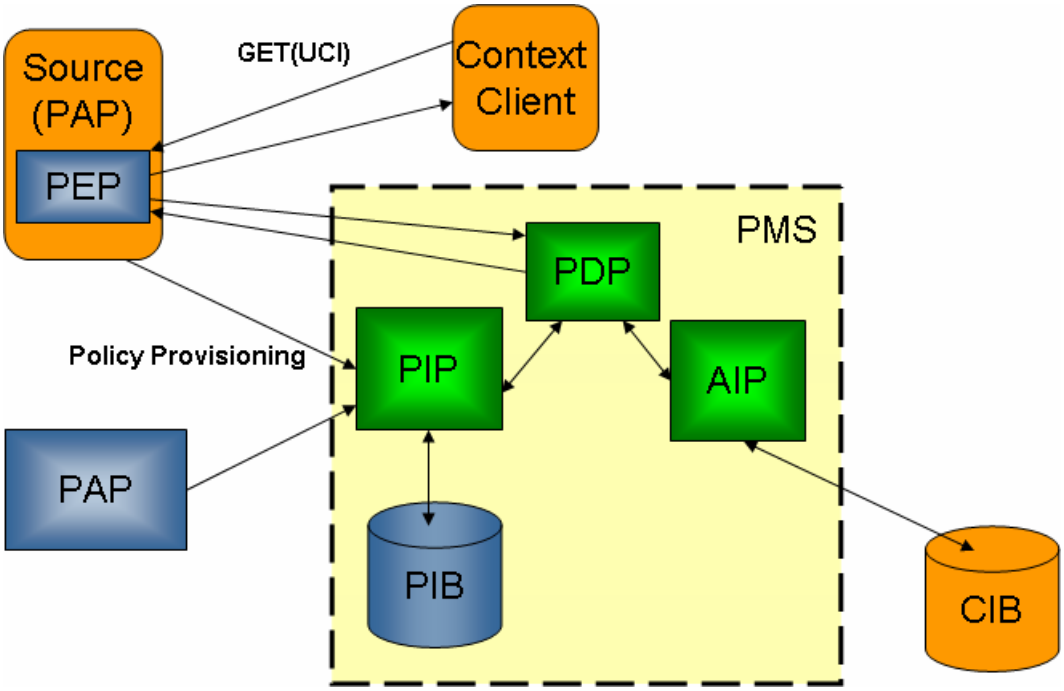

Fig 14 : Policy Management Architecture handling fine grained policies at Source

Case 3: Fine grained policies at CM for processed context information

When a source REGISTERS with the ConCoord at the time of joining the AN, the source may delegate the responsibility of handling its context information to a Context Manager for various reasons. A reason could be that the context information is updated very frequently and the source doesn't have sufficient resources or doesn't want to be bothered with notifying the clients of all of these changes. (Thus the source simply notifies the CM and the CM takes care of sending updates to the various clients.) It could also be that the context information should be changed to some other format through transformation or filtering, or combined with other context information from other sources to produce new context information. Based on the policies through which the context managers were created, access policies related to this new context information should also be dynamically generated through aggregation of policies of individual sources (or created by an administrator). These access policies are stored in the PIB through the PIP. Once the client has been granted

general access by the ConCoord and has the location of the CM associated with the context information (as covered in Case 1), it sends a GET or NOTIFY directly to the CM responsible for the context information. Here the CM acts as a PEP and forwards the request to the PMS. This request is handled by the PDP in the PMS of the AN. The PDP fetches the relevant policies from the PIB through the PIP and the concerned attributes from the CIB through the AIP, then based on the policies makes a decision regarding providing access to actual context information.

The architecture for this specific case is illustrated in Fig 15.

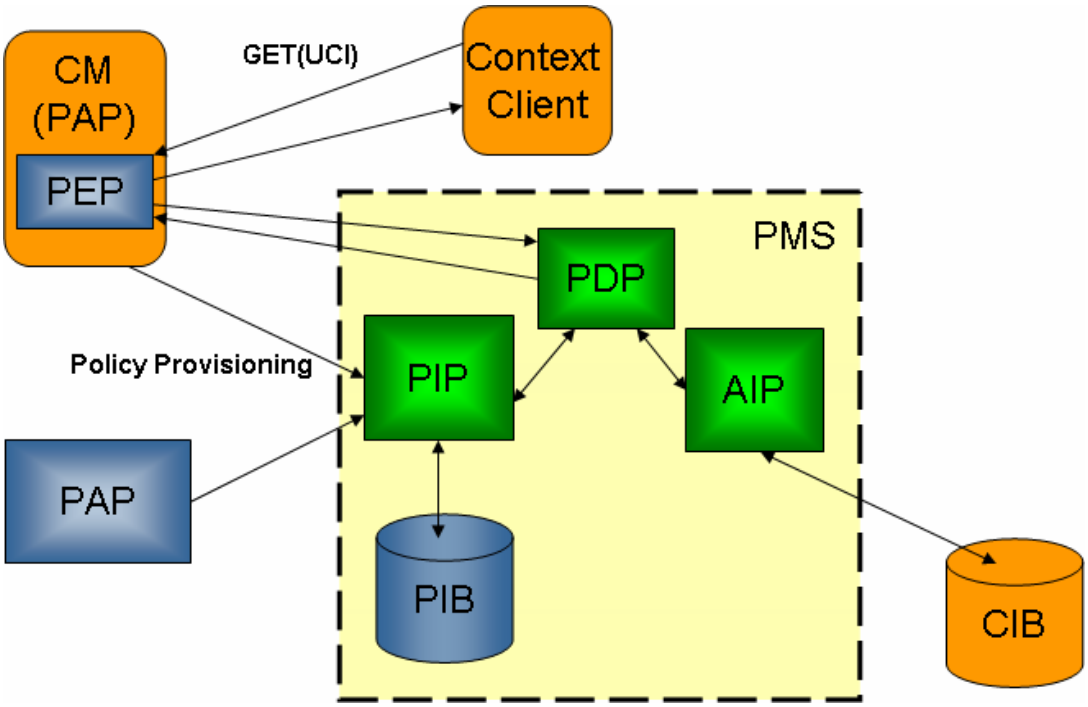

Fig 15 : Policy Management Architecture handling fine grained policies at Context Manager

4.4 The operation of the PMS architecture during Composition

Composition from the Ambient Networks' perspective is not limited simply to basic addressing and routing, but goes much further. Network composition within the AN is the composition of the functions for incorporating higher layer support including context distribution and management of context distribution based on policies in the composing networks and their combined policies .

Composition can be categorized into three types: interworking, network integration, and control sharing. These different categories of composition, impose different requirements on the composition of policies and the policy management system.

When interworking, the logical structure of the ACS is not affected, thus each ACS keeps its own Policy sets. In addition, new policies are added to both the networks to

take care of the composition agreements. Thus the PMSs in the composing networks would continue to maintain their identity and will have their own separate policy repositories.

For network integration, the ACSs of the compositing networks are merged into one, creating a new identity. For the PMS, it means the merging of the policy information base and all the policies from both ANs. Subsequently, the PMS of one ACS will take charge of handling all the requests and evaluating them against the combined policy database, while the other PMS is inactive. This means that all the requests for access to context information from the combined AN would be directed to the selected PMS. Although, beyond the scope of this thesis, it is worth mentioning that when the policy databases from the composing networks are combined, the policies have to be checked to resolve conflicts to ensure consistency of policies in the merged policy database.

In case of control sharing, some tasks are managed solely by the individual ACS and for some other tasks a higher level ACS can be created which controls the shared tasks and resources of both ANs. In terms of PMS it means that in this type of composition, the architecture behaves as in the case of network integration, as in the case of interworking, or both together. In other words, it is either full delegation, no delegation, or partial delegation of control.

Fig. 16 depicts the complete merge scenario, as in the case of network integration or full delegation under a control sharing scenario. Here the policy management of both networks would be done by one of the PMSs of the combining networks that takes the role of the master PMS of the combined ACS of the composed networks. This PMS uses the combined policies of the composed network to evaluate requests from the newly formed network. The architecture of the composed PMS would be the same as shown in Fig 13.

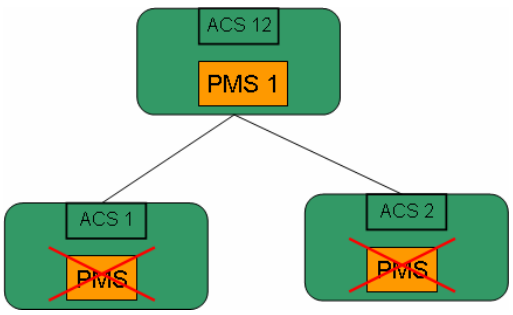

Fig. 16: Network integration composition or full delegation in Control sharing composition

Interworking or control sharing composition with no delegation is depicted in Fig 17. As shown in figure, the PMSs of the composing networks continue to control access to context information in their original domains.

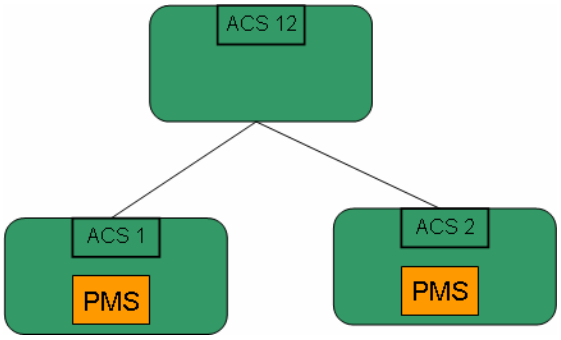

Fig 17: Network interworking composition or no delegation

in Control sharing composition

For the composition category of control sharing with partial delegation, access to certain context information would be limited to the scope of the original uncomposed networks; whereas for some other context information, the policy management would be delegated to the combined PMS of the composed network. This is shown in Fig 18. In this case a node in AN1 would have access to only some nodes and services in AN2 based on access policies.

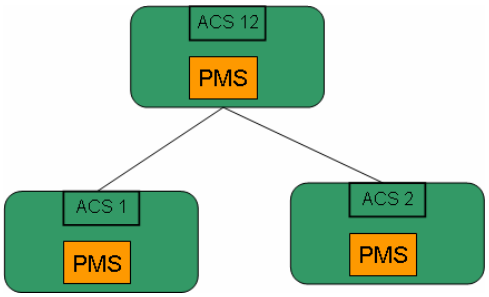

Fig 18: Partial delegation in Control sharing composition

When the composition is sharing control with partial delegation, the architecture would look like that shown in Fig. 19.

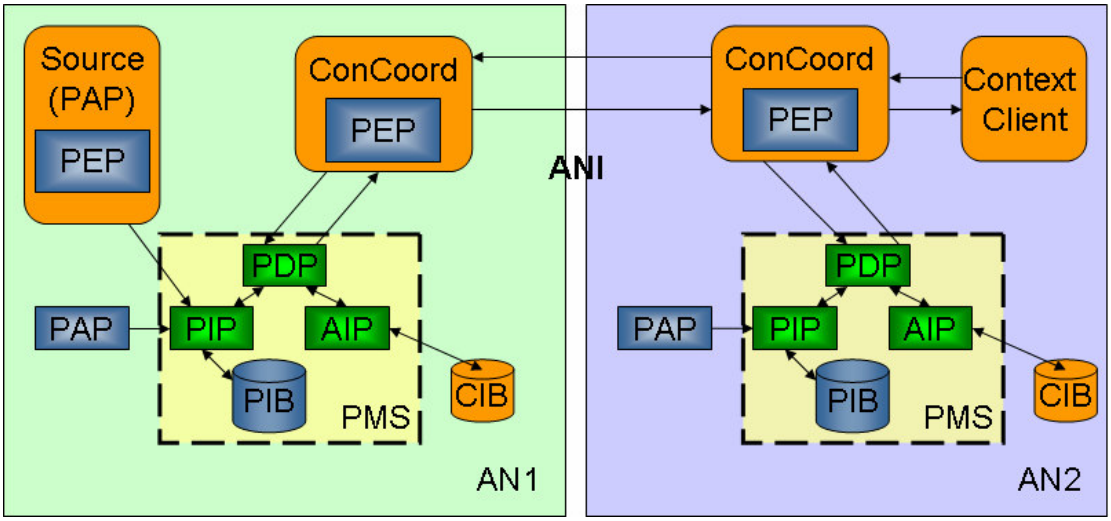

Fig. 19 : Operation of PMS architecture during Composition

When a Client in AN2 tries to get access to context information in AN1, the client contacts the ConCoord of the AN2 which in turn contacts the ConCoord of AN1 (based on partial delegation policies) over the ANI and requests access to the context information. The operation of the PMS would be identical to the operation described earlier for accessing context information within one AN. Once the Client gets access to the location of the source of the context information based on policies, the client of AN2 will directly send an access request to the source in AN1 and the operation of the PMS would be as described earlier.

4.5 Example Scenario

Here we consider an example scenario specifying both general access rules as well as access rules specific to particular context information. This and other scenarios would then be further developed into a use case and implemented in a prototype.

It is close to 8 am in London and Bob is at home getting ready to go to a business meeting in Liverpool. Bob has a PAN that consists of a laptop (with WLAN, WiMAX, and Bluetooth), a Smart Phone (with WLAN, UMTS, and Bluetooth), and a Bluetooth enabled headset. During his journey from London to Liverpool, the wireless interfaces of Bob's PAN proactively attempts to discover available wireless wide area networks (WWAN) (e.g., UMTS or WiMAX). The PAN receives advertisements from a number of WWAN operators, each advertising their services and rates. In addition these advertisements include a digital signature of the operator to prove the authenticity of the operator's information and the service offering (these are handled by the Security FE of the PAN).

For Bob's PAN to compose with any network, the policies in the PAN are used to find the most suitable network based on the current context information (here we consider the list of available access networks). The source of information about the available access networks are the wireless interfaces of Bob's PAN which are a WLAN interface, a WiMAX interface, a Bluetooth interface, and a UMTS interface. Each of these interfaces generated its own list of currently available access networks (along with other information). These sets of interface specific information serve as raw context information and are represented as context objects to which access is desired by the client (composition FE) to make its decision about the most suitable network based on criteria such as bandwidth, cost, etc.

Initially the Composition FE sends a request for access to this context information to the ConCoord that uses the coarse grained policies of the PAN to decide at a general level whether access can be granted or not. An example general policy could be, that access to any context information within the AN, can only be granted to a client that is within the same AN, and has been authenticated. If access can be granted, then the address of the source where the requested context information resides is sent back to the client. Subsequently, a context request is sent by the Composition FE acting as a client and received by the individual sources of context information, (i.e. the wireless interfaces or by a delegated context manager that has aggregated all the context information about the available access networks from various source, i.e. the various interfaces). These requests are evaluated by the PDP against existing fine grained access policies pertaining specifically to access networks and a decision is reached if access to this context information can be granted or not. An example of a fine grained

policy could be that access to the context information regarding available networks cannot be granted to the client if the firewall on the laptop is not running properly. The fine grained policy being enforced here may itself require access to further context information (e.g., the state of firewall on the laptop) before the policy can be enforced.

4.6 Use Case

The above example scenario is presented as use cases below. Here we present three use cases for each of the three cases presented in section 4.2

4.6.1 Use Case for General Access Policies

This use case presents the case where the coarse grained access policies are applied when a client joins a network and requests access to context information from the ConCoord for the first time. The policies applied are general access policies pertaining to the whole Ambient Network stored in the network by an authenticated PAP. These policies are applied first by the PDP when any client registers with the ConCoord. This client could be in the same network, or be a service or application over ASI, or be another node in another AN over the ANI.

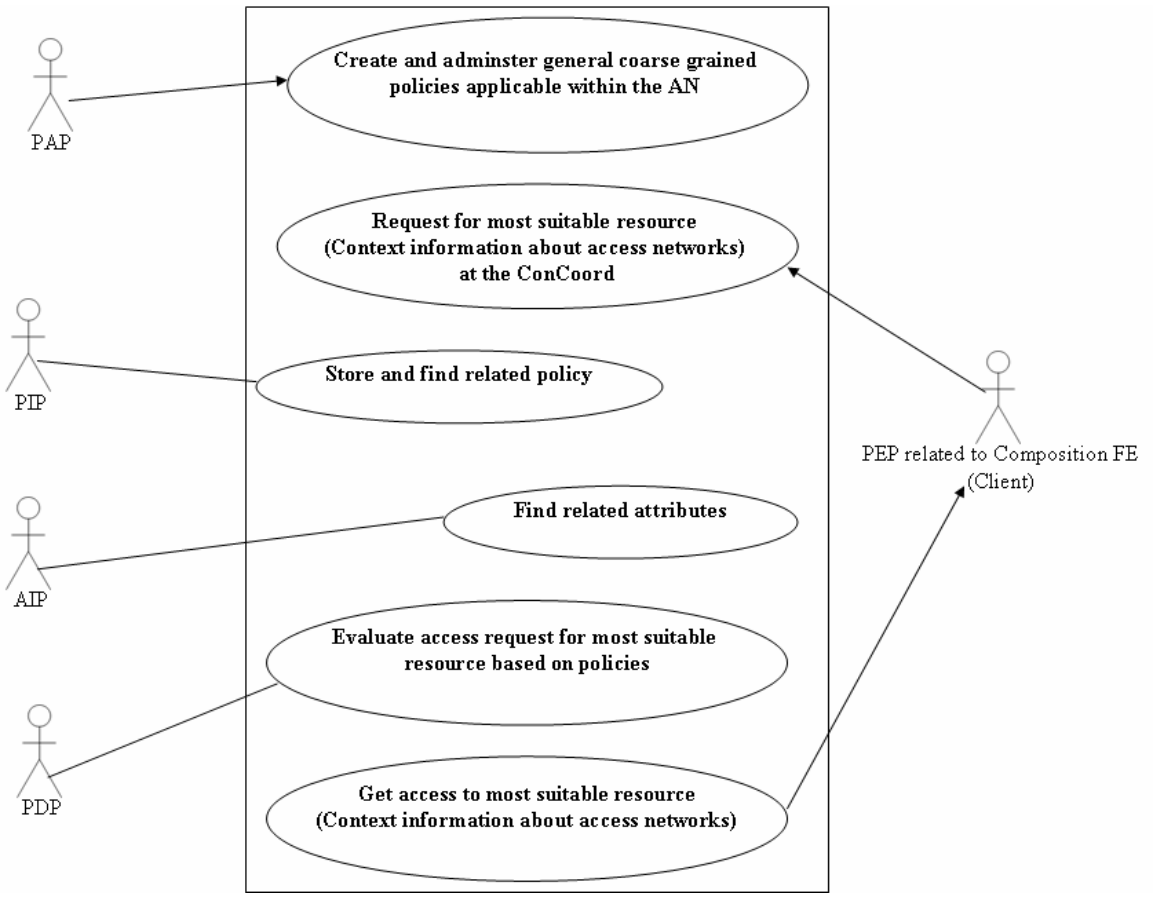

Fig. 20: Use Case for general access policies

4.6.2 Use Case for fine grained Access Policies at Source

This use case presents the case where the fine grained access policies existing at the source are applied, when a client, after joining a network and having received the location of the source of context information it needs, requests access to context information from the source. The policies applied are specific access policies pertaining to the specific context information. The evaluation of the request for context information is done at the PDP based on the fine grained policies.

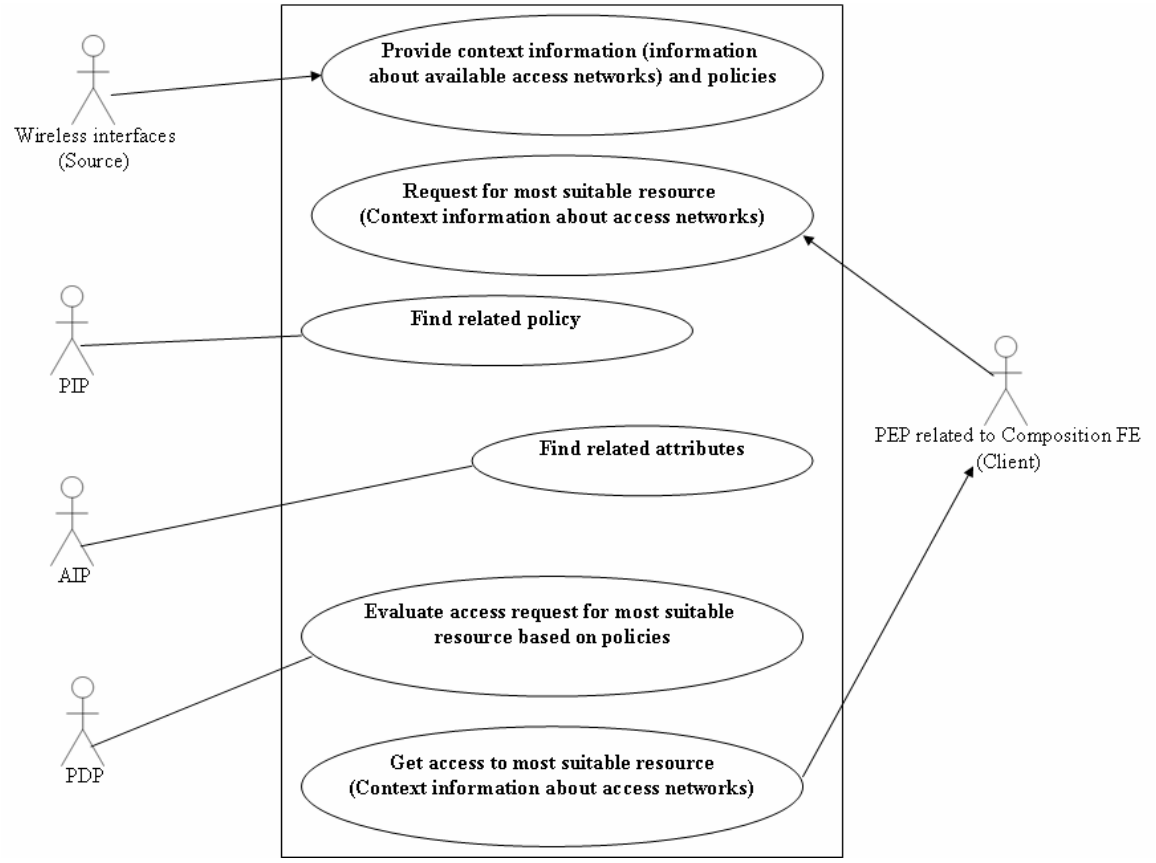

Fig 21: Use Case depicting the scenario

4.6.3 Use Case for fine grained Access Policies at Context Manager

This use case presents the case where the fine grained access policies existing at the Context Manager are applied, when a client, after joining a network and having received the location of the source of processed context information it needs, requests access to context information from the Context Manager. The policies applied are specific access policies pertaining to the specific processed context information. The evaluation of the request for context information is done at the PDP based on the fine grained policies.

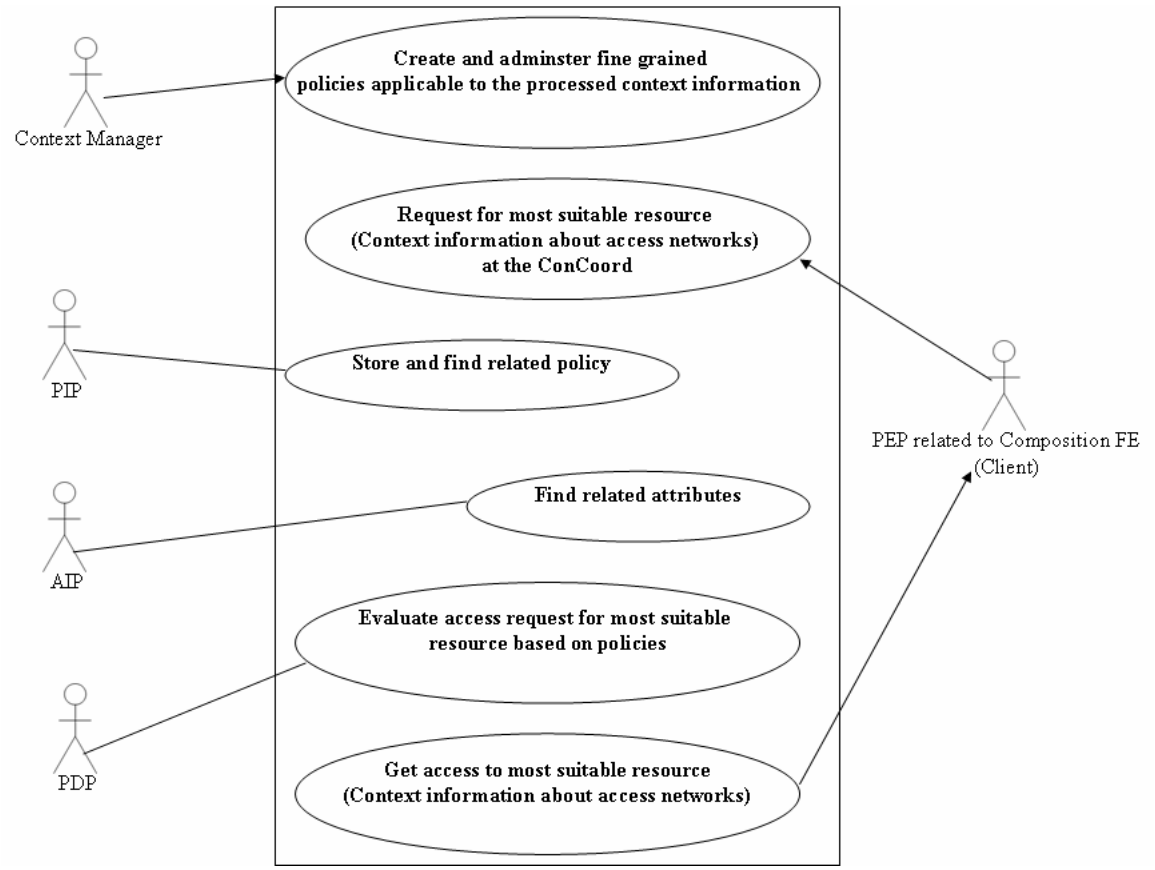

Fig 22 : Use Case for processed information

4.7 Policy Management Protocol (PMP)

The messages within the PMS are XACML messages and have a standard format. A Policy Management Protocol (PMP) has been designed for the messages to and from the ContextWare and also within the PMS to carry the XACML messages. PMP was inspired by the Context Exchange Protocol (CXP).[6].

Another alternative was to use COPS for these messages. In the following section, we will discuss the factors behind our choice, along with the features of PMP.

4.7.1 Transport Protocol

As discussed in section 3.4, COPS uses TCP as the transport layer for sending its messages. This requires the establishment of a TCP session. Using such a connection oriented protocol is suitable when multiple packets may arrive in bursts. However, ContextWare and PMS send single messages, such as REGISTER, RESOLVE, etc. - which do not need a TCP session. In contrast, the CXP has been designed to utilize the stateless, best-effort, transport layer, User Datagram Protocol (UDP). CXP incorporates reliability measures in the protocol to detect packet loss and resend lost packages.

Therefore, we based the design of the PMP on the Context Exchange Protocol. This means that like the CXP, the PMP utilizes UDP and also contains support for detecting and resending lost packets.

4.7.2 Message format

Similar to CXP protocol, PMP protocol uses XML for encoding messages. XML is widely used and is both machine and human usable. In addition because of its popularity, there exist a lot of XML parsers and this makes the design simpler to implement.

4.7.3 Reliability over UDP

As described in section 4.7.1, the protocol utilizes UDP. This creates the need to introduce some reliability so that lost packages can be detected and retransmitted. Similar to CXP, the following methods have been utilized:

- A sequence number is introduced in each packet to track the flow of messages between the sender and receiver. Thus the receiver would be able to determine if a packet has been missed or is out of order.
- An acknowledgement is sent from the receiver to the sender after successfully receiving a message. This acknowledgement can either be an explicit message or be included in the next message sent by the receiver to the sender. The sender waits for a defined amount of time for the acknowledgement message before resending the message.
- To check that messages are received in order at the receiver side and to facilitate retransmission by the sender in case of lost message, buffers containing the received and sent messages are maintained both at the sender and receiver side respectively. The buffers are kept up-to-date by deleting the appropriate messages from the buffer.

4.7.4 Policies within the AN

The policies within the Ambient Network are authored by authenticated Policy Authoring Points (PAP). These could be stand alone nodes acting as a PAP or various sources of context information that provide policies controlling access to their specific context information. These sources could belong to different FEs within the network, thus each FE would be responsible for providing policies to the PMS for managing access to its context information. It is assumed that all the FEs would have support for XACML, thus the policies provided would be in XACML format.

4.7.5 Policy Enforcement Points

The PEPs are not an integral part of the PMS, but belong to the various FEs. The PEP within ContextWare is located within the ConCoord. This PEP is responsible for performing multiple tasks. It is the first point of contact for a client trying to request access to context information. This PEP is also responsible for transforming the UCI request from the client into an XACML request. It then sends this request to the PDP,

which evaluates it based on the existing policies, and responds with a decision to the PEP. This decision may contain some obligations which the PEP has to fulfil before permitting access to the client. Based on the response received from the PDP, the PEP responds to the client. Thus communication between the client and the PMS is always through the PEP.

4.8 Message Flows

Here we will look at the messages within ContextWare and the PMS for registration of policies as well as for resolution of UCIs based on access control.

4.8.1 Policy Registration

When an authenticated source, client, or a Policy Authoring Point wants to register policies with the PMS, it first sends a message REGISTER to the ConCoord in ContextWare. This message is an empty message and doesn't contain the actual policy. In response to this message, the ConCoord returns the location of the PIP to the registering entity (source, client, or PAP).

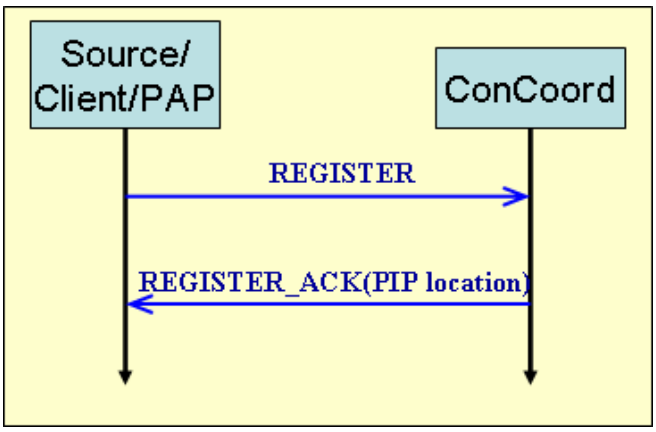

Fig 23: Policy Registration at ConCoord

Now the policy registering entity can register the policies with the PMS. It sends the policies directly to the PIP using the message REGISTER_POLICY which contains the policies in the body of the message. The PIP then stores them in the common policy repository, PIB. Once the policy has been successfully stored, the PIB returns an acknowledgement to the PIP which in turn sends an acknowledgment to the registering entity that the policy has been stored. The flow diagram for provisioning of the policies in the Policy Management System is shown in Fig. 24.

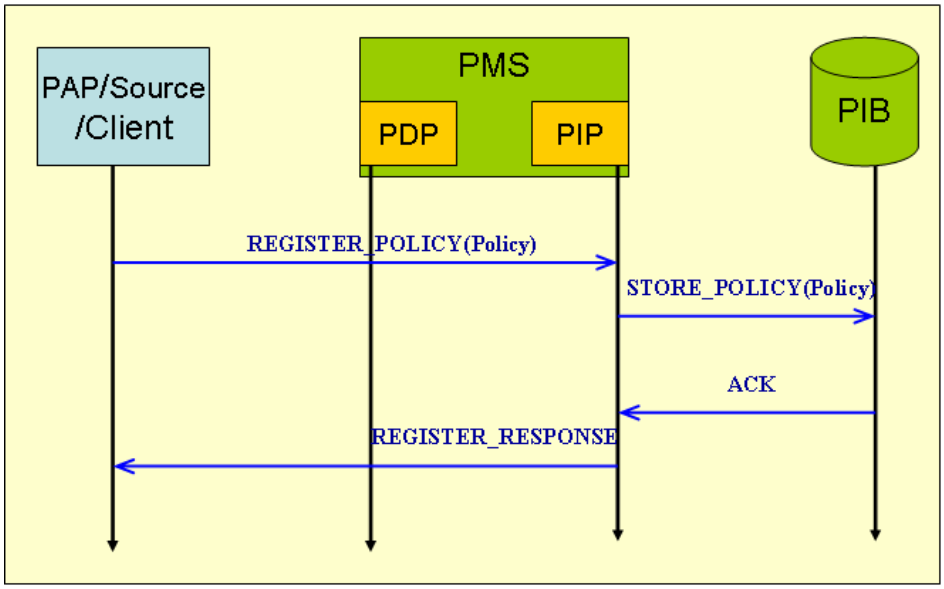

Fig 24 : Policy registration at PMS

4.8.2 Access Control for UCI resolution by client at the ConCoord PEP

As described in section 2.4.2, the ConCoord is the first point of contact used by a client trying to resolve a UCI. The discovery of the ConCoord by a Client is ContextWare specific and is not dealt with in the design of PMP. When a client first joins an AN, and requests context information from the ConCoord, the ConCoord acts as a PEP, transforms this request to an XACML request and forwards the resulting ACCESS_REQUEST to the PDP in the PMS. The ConCoord PEP also sends an acknowledgment to the client that it has received the request. The PDP similarly sends an acknowledgement to the PEP that it has received the ACCESS_REQUEST.

Based on the target element of the request, the PIP fetches the relevant policies and the AIP fetches the relevant attributes from the repositories, then the PDP evaluates the request. The messages used for this evaluation are not XML messages, but are internal XACML messages. Thus these are all collocated on the same node.

The PDP sends the ACCESS_RESPONSE containing the decision in XACML format within the body of the message to the ConCoord PEP. This response is acknowledged. Based on the result of this evaluation, if access is granted, the ConCoord PEP looks up the UCI-source mapping and sends the location of the source of the context information to the client in its RESPONSE. If access is denied, then the PEP informs the client that access cannot be granted. While granting access, if the decision contains some obligations to be fulfilled by the client, it is up to the PEP to ensure that the obligations are enforced before permitting access. Once the client receives the result, it acknowledges it irrespective of the nature of the result.

The flow diagram is as shown in Fig. 25.

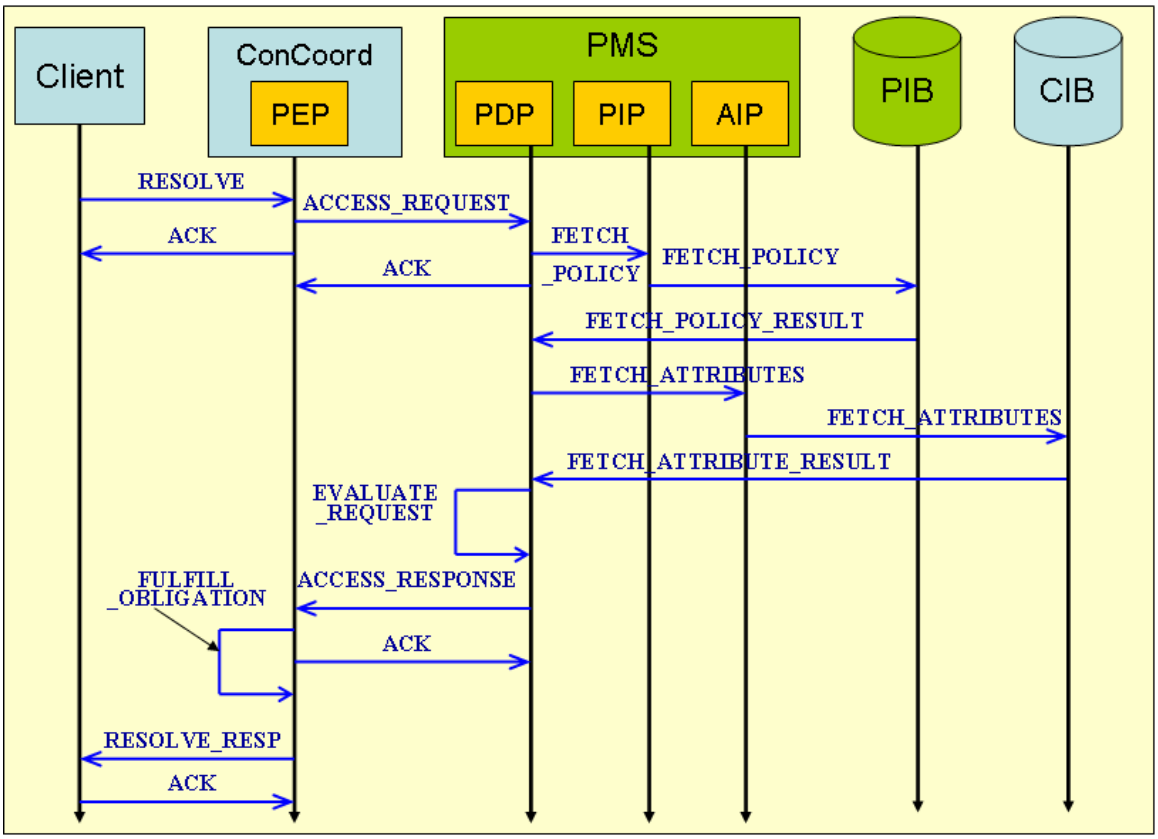

Fig 25 : UCI resolution at ConCoord PEP

4.8.3 Access Control for UCI resolution at the Source or CM PEP

If the client was permitted access by the ConCoord PEP and has received the location of the source or the Context Manager where the required context information exists, then the Client directly sends a GET_REQUEST to the source or the CM. Alternatively, it could send a SUBSCRIBE request to the source or CM to receive updates of the requested context information. The PEP acknowledges the receipt of this request to the client.

When the source or CM, acting as PEP, receives this request, it forwards the request to the PDP in the PMS and receives an acknowledgement. The request is evaluated by the PDP after the PIP and the AIP have fetched the relevant policies and attributes, then the result is sent back to the PEP (source/CM) as an ACCESS_RESPONSE. This message is again acknowledged by the PEP.

The result might carry some obligations that must be met by the client before access can be granted. It is the responsibility of the source (CM) PEP to ensure that the obligations are met before the access result is conveyed to the client. If access is permitted and all obligations met, the context information is made available to the client. What ever is the result from the PEP, the client acknowledges the receipt of this result.

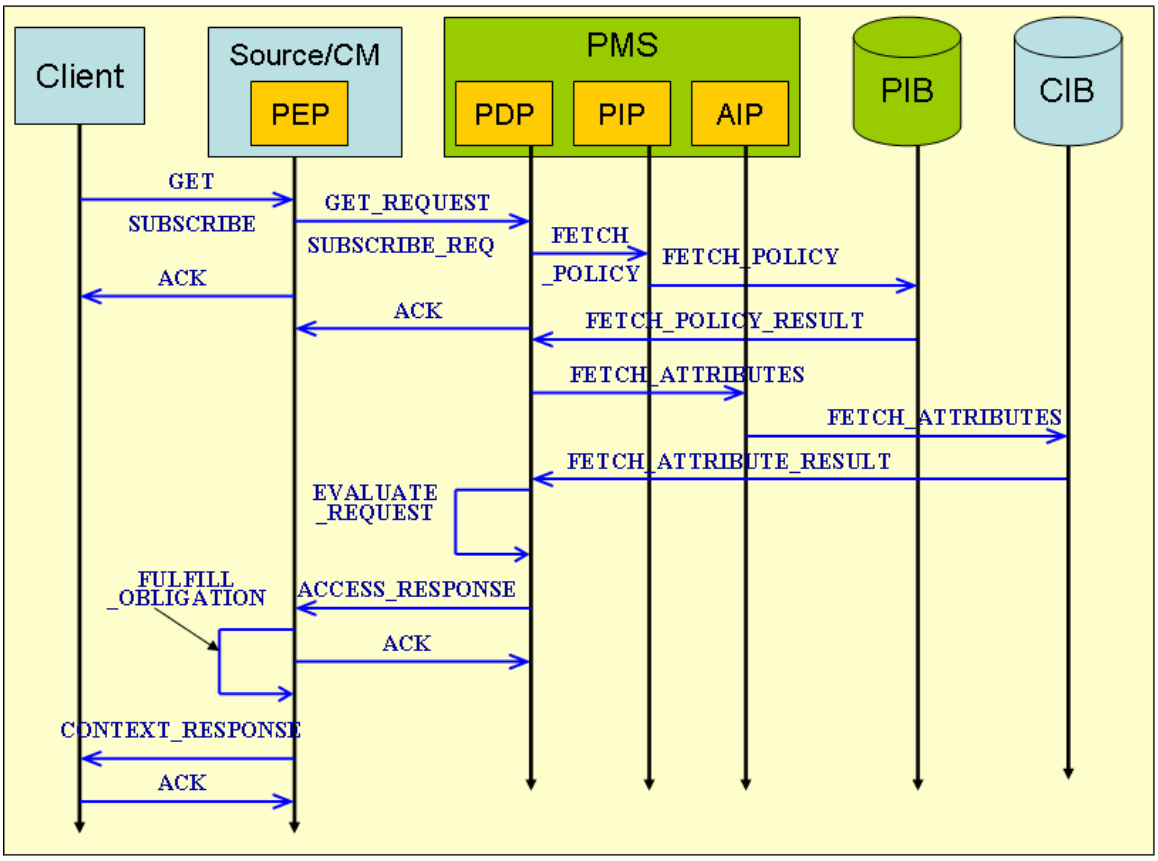

Fig 26 : Flow Diagram when source or CM has PEP role

4.9 Message Protocol

The message syntax of PMP is very similar to the message syntax of CXP designed by Sergio [6]. The PMP messages are XML messages and contain some header elements that would be carried in all the messages and the contents of the body element of each message would be different for different messages. The format of the PMP message is:

- **Element <method>:** This element specifies the name of the message that is sent from the sender to the receiver. Examples of this element are ACCESS_REQUEST, ACCESS_RESPONSE, etc.

- **Element <sequenceNum>:** This element contains the sequence number of the message sent. The sequence number helps the sender to match acknowledgements and helps the receiver to check the order of messages.

- **Element <ack>:** This is an optional element and contains the sequence number of the message being acknowledged.

- **Element <fromEntityId>:** This is the cryptographic identifier of the node sending the message.

- **Element <fromAddress>:** This entity contains the IP-address of the sender.
- **Element <toAddress>:** This entity contains the IP-address of the receiver.
- **Element <body>:** This element would contain the actual information that is conveyed. Depending on the method of the message, the body element would contain different information. This is elaborated below for each of the specific messages.

4.9.1 REGISTER Message

The body element of this message is empty. This message is sent from the entity trying to register a policy to the ConCoord to learn the address of the PIP within the PMS.

4.9.2 REGISTER_ACK Message

This message is sent in response to the REGISTER message and the body of this message contains the IP-address and port used by the PIP. Thus the body of this message has the elements **<pipIPaddress>** and **<pipPort>.**

4.9.3 REGISTER_POLICY Message

This message is used by the policy owner to register its policy with the PMS. The body of this message contains the policy in XACML format. Each message can contain only one policy and if an entity wants to register multiple policies, multiple REGISTER_POLICY messages will have to be sent.

4.9.4 ACK Message

This message is used when the receiver just wants to send an acknowledgement to the sender. This message contains the **<ack>** element with the sequence number of the message it is acknowledging; the body of the message is empty.

4.9.5 REGISTER_RESPONSE Message

This is a response message sent by the PIP to the source. The body of the message contains a **<result>** field to indicate if the policy was registered successfully of not. If the <result> is successful, it is the only field included in the body of the message. In case of an unsuccessful result, the **<error>** field is included that details the reason for the unsuccessful registration. The <error> field can be used by PIP in the future to convey to the policy owner any conflicts in the policy with the existing policies.

4.9.6 RESOLVE Message

This message is sent by the client to the ConCoord PEP requesting access to context information. The body of this Message contains the element **<UCI>** which contains the URI of the context being requested. Alternately, it may contain the **<request>** field which contains the request for context information in XACML format.

4.9.7 ACCESS_REQUEST Message

This message is sent from the PEP to the PDP. The body of this message contains the XACML request that was created by the PEP from the requested UCI.

4.9.8 ACCESS_RESPONSE Message

This message is sent by the PDP to the PEP. The body of this message contains the XACML result of the evaluation of the XACML request against the relevant policies. The result is a **<decision>** element which contains the actual access response and optionally may contain an obligation element.

4.9.9 RESOLVE_RESP Message

This message is sent by ConCoord PEP to the client. This message contains a **<result>** field in the body of its message. Based on the access decision from the PDP, the body of this message contains the either the location of the source of context information, or it contains a deny response.

4.9.10 GET/SUBSCRIBE Message

This message is sent by client directly to the owner of the context information. The owner could be the source of raw context information or the context manager, which is the source of aggregated, filtered, or transformed context information. The body of the message contains the **<UCI>** element containing the UCI of the requested context information.

4.9.11 GET_REQUEST/SUBSCRIBE_REQUEST Message

This message is sent by the PEP to the PDP for evaluation of access. The body of this message contains the XACML request that was created by the PEP from the requested UCI.

4.9.12 CONTEXT_RESPONSE Message

This message is sent by the PEP to the client. This message contains a **<result>** field in the body of its message. Depending on the decision response from the PDP, the body of the message would either contain deny or would have the actual requested context information.

5 Implementation

This chapter describes the implementation phase, covering the prototype's details along with environmental details, including the software and hardware used for implementing the prototype. The problems encountered during this phase are also covered here.

The main goals of creating a prototype for the PMS were to create a proof of concept and to measure the additional time delay and resource requirements introduced by adding access control to context information in an Ambient Network.

5.1 Hardware Equipment

The prototype consisting of Policy Management System (PMS), Policy Enforcement Point (PEP), and a client has been implemented using two laptops. The PMS, PEP, and client nodes have been distributed over the two machines in such a manner that there is no direct communication between nodes on the same machine. Each of the three nodes uses a different UDP port to allow them to co-exist on the same computer.

The client and the PMS have been implemented on a HP Compaq nc6000 laptop with Intel® Pentium® M processor with processing speed of 1400 MHz and the Microsoft Windows XP Professional operating system.

The PEP has been implemented on an IBM X31 laptop with Intel® Pentium® M processor with processing speed of 1600 MHz and operating system Microsoft Windows XP Professional.

Fig. 27 shows the placement of the nodes and the messages between these nodes.

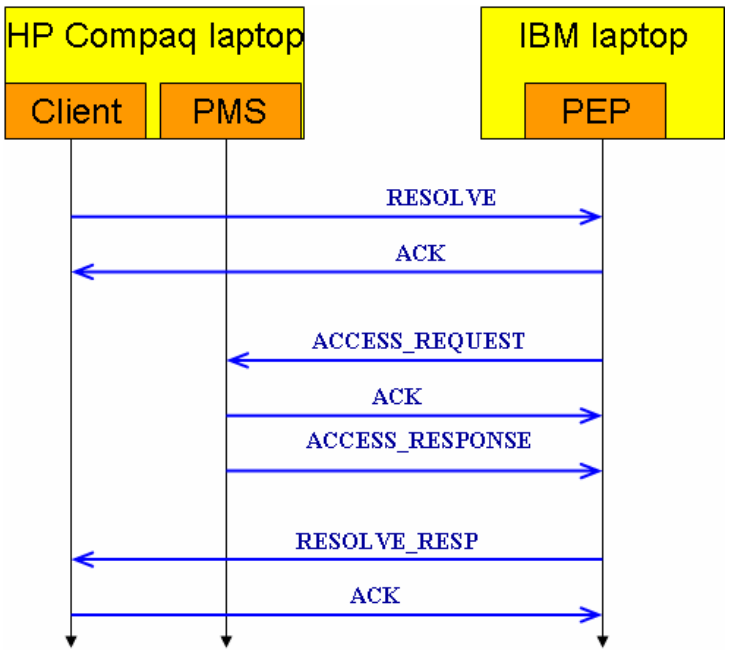

Fig 27: Prototype node distribution and message flow

5.2 Software Environment

The prototype has been implemented in java using J2SE version 5.0. The prototype also uses the XACML implementation for policy management initially developed by Sun Microsystems, which is known as SunXACML. SunXACML is an open-source implementation, implemented completely in Java and is currently being developed as a SourceForge project [27]. The prototype makes use of the SunXACML release supporting XACML version 1.2.

The implementation has been done using the Eclipse SDK 3.2.0 [28].

5.3 Network Environment

The prototype has been developed and tested within the Ericsson Research test bed using an IEEE 802.11b WLAN network. Both the machines were located on the same subnet and thus could reach each other using the local IP addresses. Both machines were associated with the same access point, so that all communications between the machines were sent over the WLAN twice, once to the access point and once back to the other machine.

5.4 Implementation Description

Here we will look at some of the key features of the prototype.

- **XML Parser**

  As described in subsection 4.9, the messages between the different nodes are XML messages. Thus the prototype has to ensure that the messages are constructed and received in a correct format. To interpret XML messages, the prototype makes use of existing standard parsers, specifically the kXML 1.2 parser [29].

- **File repository**

  The policies used for the prototype are stored as XML files in a Microsoft Windows file directory located on the relevant machine (thus there was no additional network traffic required to access these files).

- **Creation of an XACML request by the PEP**

  The prototype is designed in such a manner that the PEP takes the requested UCI that it receives from the client in the RESOLVE message. The PEP converts this request into an XACML request that is understood by the PMS before sending it to the PMS.

  Alternately, if the request for the UCI from the client is already in the form of an XACML request, the PEP simply forwards this request to the PMS in its ACCESS_REQUEST message.

- **TIME_OUT**

  The prototype includes a time-out for receiving ACK messages, thus if an ACK is not received before the end of this time-out, the message is resent.

- **Buffered Requests**

  Both the PEP and the PMS nodes use buffers to store the requests, responses and the ACK messages. A thread listens to all incoming messages for the node and stores them in this buffer as a queue. Another thread checks the buffer for messages waiting and processes them one by one. Thus each node in the implementation uses two threads.

5.5 Prototype Limitations

Here we highlight the limitations of the prototype as compared to the design explained in the previous chapter. The reasons for each of these limitations are given in conjunction with the explanation of the limitation.

- **Discovery and bootstrapping**

  The prototype assumes that the client already knows the address of the ConCoord PEP to which it initially sends the UCI request. Thus the prototype does not need to support any bootstrapping mechanisms. In addition the assumption is made that the ConCoord PEP also knows the location of the PMS.

- **PIP and AIP are not implemented as separate entities**

  The main role within the PMS is that of the PDP that evaluates the access request against the existing policies. The basic task of the Policy Information Point (PIP) and the Attribute Information Point (AIP) are to retrieve the relevant policies and attributes for evaluation of a request. For the purpose of the prototype, the PIP and the AIP are not implemented as separate modules, but are implemented as integral part of the PDP. Thus the PDP in the prototype first fetches the relevant policies and attributes and then evaluates the request against these policies and attributes.

- **Policy creation by PAP**

  The prototype does not implement a Policy Authoring Point (PAP) for authoring policies within the Ambient Network. The assumption is that this task would be performed by the functional Entities (FEs) that are the owner of the context information and thus define the policies for the access to the context information belonging to them. This is because we only have focused on measuring the overhead created by access control (as without an understanding of this overhead, the ability to easily author policies is of limited use). For this reason we omit the creation of a PAP that creates and stores policies within the PMS. Instead we use already created policies that pre-exist (i.e., are manually placed) in the PIB database and evaluate the requests generated by the clients against these policies.

6 Evaluation

This chapter evaluates the design of the architecture and the protocol. The design is evaluated using both measurements and analysis. In other words, this chapter considers both qualitatively and quantitatively the quality of the design.

The PMS is designed as a centralised entity. This means that requests from all the nodes accessing context information would be directed through this centralised FE. Measurements are performed to calculate the latency and throughput that would result from such a design. The delay is first evaluated for a single request and then for a burst of requests. In addition to evaluating the performance of the PMS, these measurements are used to estimate the throughput of the PMS.

We also measure the delay when the requests are sent in a sequence with a delay of 2 seconds, 1 second, and 500 milliseconds between two consecutive requests, to estimate the performance of the PMS under different load conditions.

Another test compares the time difference between request evaluation time for context information in an ACS with a Policy Management System (PMS) and the same request for context information in an ACS without a PMS.

These tests and their results are described in the following section, but first we introduce the test environment.

6.1 Test Setup

To perform the measurements, we used Ethereal [21]. Ethereal is a Network Protocol Analyzer used to monitor network traffic (this software is now called Wireshark [26]). It is available on both Windows and Linux platforms.

In addition to using Ethereal, we also trace logs of the time stamps of specific events into log prints and use these for measuring the response times.

Ethereal is used on the machine running the PMS to capture the message packets. Referring to Fig. 27 in the previous chapter, we see that the PMS is co-located with the client on the same machine.

Using the analyser and the trace logs, we can see the time when a request is received by the PMS and also the time when the response is sent from the PMS, thus we can calculate the *PMS response time* which is the time required by the PMS to evaluate a request and generate a response. Referring to Fig. 28, this is the time difference between sending the ACCESS_RESPONSE at the PMS node and the receiving of the ACCESS_REQUEST at the same node and is indicated as the time x_4-x_2. This time also includes the time the request is waiting in the buffer at the PMS to be evaluated. We also made measurements for the actual *PMS request evaluation time* which excludes the time the request is waiting in the buffer and is the actual time taken by the PMS to evaluate a request after picking it from the buffer, evaluating the request, and sending a response.

In addition to measuring the response time of the PMS, we also measure the time taken by the client to receive a response for its request. This time is measured from when the RESOLVE is sent from the client to when the RESOLVE_RESP is received by the client; denoted in Fig. 28 as the time x_6-x_0.

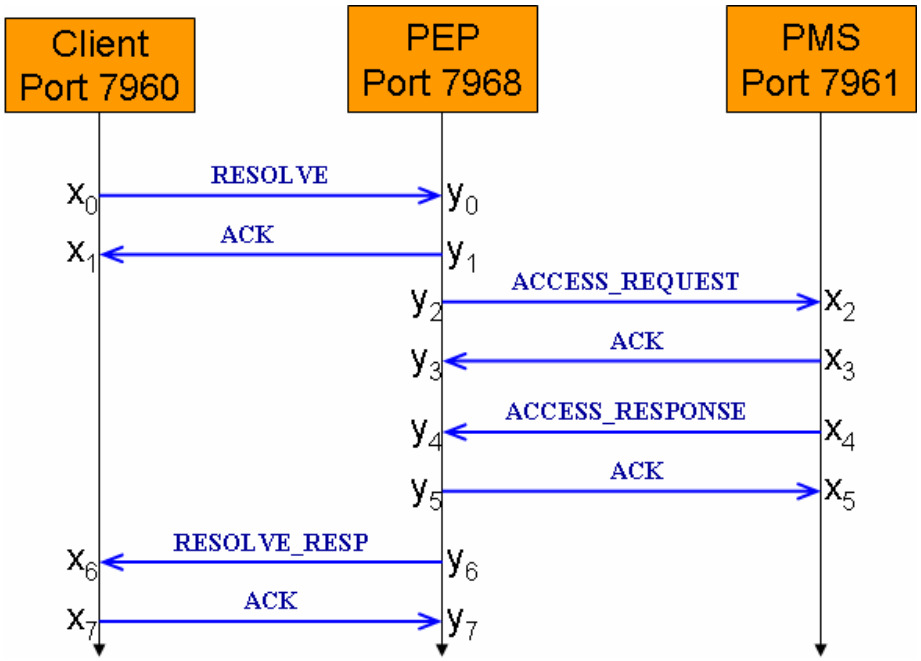

Fig 28 : Messages with time reference

6.2 Network Latency

The evaluation of the total response time also includes network latencies. Latency can be defined as a time delay between the moment something is initiated, and the moment its first effect begins. Specifically network latency is the latency to transfer the packet from one machine to the other. The total latency of processing the request is composed of the sum of the network latencies and the sum of the processing latencies.

Referring to Fig 28, the total latency from the client's perspective includes the request processing latency and the network latency. The total latency from the PMS perspective is the access request processing latency and the network latency.

To measure the network latency, we run the network analyser, Ethereal at both the machines to capture the UDP packets exchanged between the two machines. To distinguish between the client and the PMS on the same machine we utilize the port number of the sender or receiver of the messages.

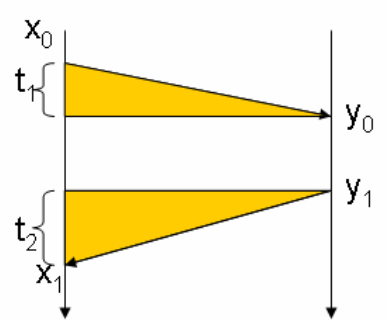

Fig 29 : Network latency measurement

Network latency can be calculated as the average of the total time delay of the packets received by the receiver from when they were sent by the sender. Here we have assumed that the network is symmetrical. From Fig 29, the (one way) network latency is

$$\text{Latency} = \frac{t_1 + t_2}{2} = \frac{(Y_0 - x_0) + (x_1 - y_1)}{2} = \frac{(x_1 - x_0) - (y_1 - y_0)}{2}$$

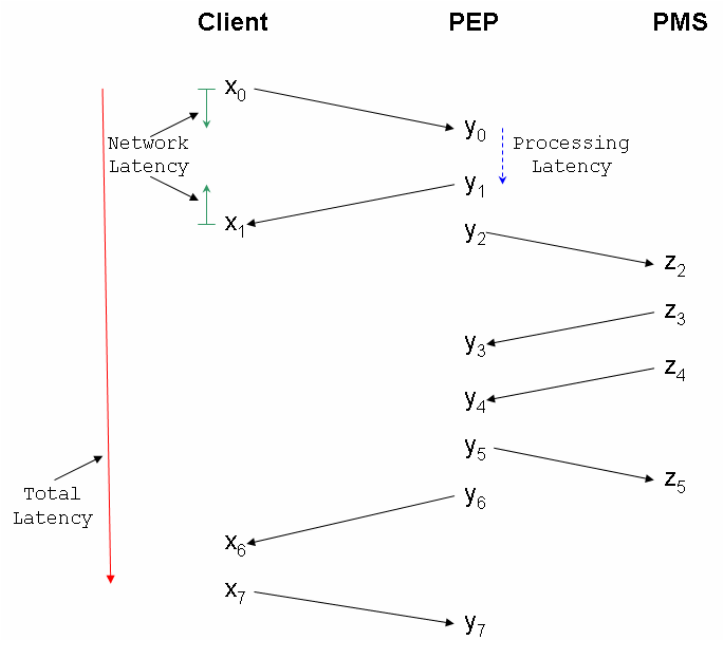

Fig 30 : Latency measurement, to extract the network latency components

Extending the above latency measurement to the complete message sequence from the time the client sends a request to the time the client receives a response, we can derive the sum of the network latencies.

The one-way network latency T_A between the client and the PEP is as calculated below:

$$T_A = \frac{1}{2}\left(\frac{(x_1\text{-}x_0) - (y_1\text{-}y_0)}{2} + \frac{(y_7\text{-}y_6) - (x_7\text{-}x_6)}{2}\right)$$

The one-way network latency T_B between the PEP and the PMS is:

$$T_B = \frac{T_1 + T_2 + T_3 + T_4}{4}$$

Where,

$$T_1 = \frac{(y_3\text{-}y_2) - (z_3\text{-}z_2)}{2}, \quad T_2 = \frac{(y_4\text{-}y_2) - (z_4\text{-}z_2)}{2},$$

$$T_3 = \frac{(z_5\text{-}z_3) - (y_5\text{-}y_3)}{2} \text{, and} \quad T_4 = \frac{(z_5\text{-}z_4) - (y_5\text{-}y_4)}{2}$$

Thus, the total one-way network latency T is the sum of the latencies T_A and T_B. To measure the network latency, we sent a single request from the client and measured the network latencies between the client and the PEP and the PMS and the PEP. This test was repeated 20 times. All the samples were collected in different captures or in other words, the test was not performed at the same time. This explains the variance in the results.

The captured data was then transferred to the Microsoft® Office Excel 2003 spreadsheet application and the result is displayed in Fig 31. The blue bars indicate the latency T_A and the maroon bars indicate the latency T_B, calculated using the formulas above. The total one-way latency for the complete message flow is shown by the yellow bars.

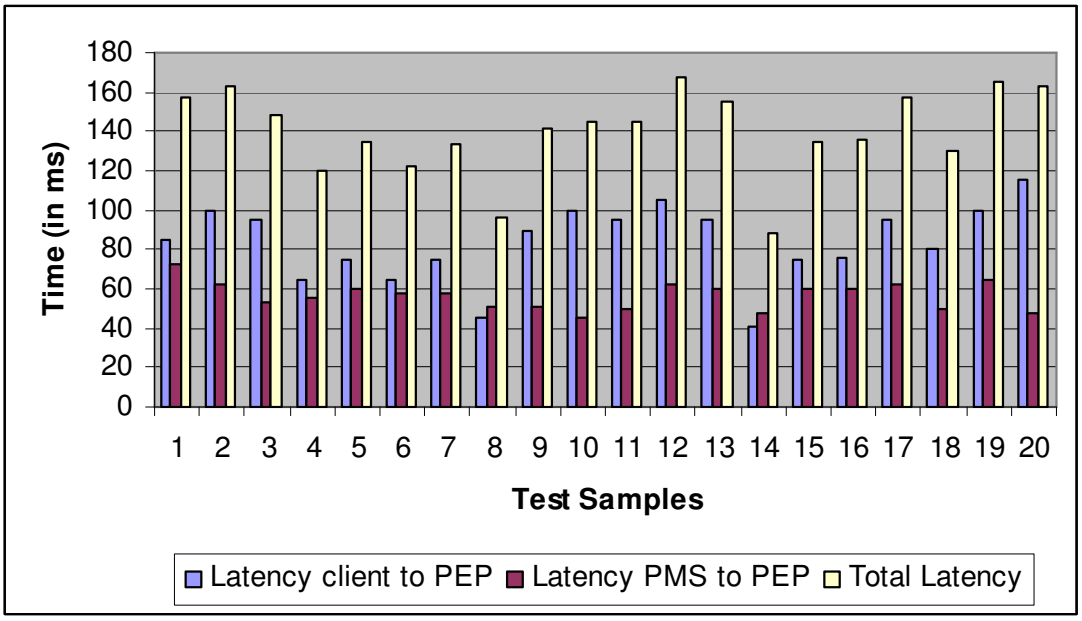

Fig 31: Total Latency in the Network

The average one-way latency or delay between the client and the PEP was found to be 79 ms and the average one-way latency between the PMS and the PEP was 57ms. Table 1 summarises these results.

One-way network latencies			
	Shortest	**Longest**	**Average**
One-way latency between client and PEP ($T_{A,}$ in ms)	41	115	**84**
One-way latency between PEP and PMS ($T_{B,}$ in ms)	45	73	**57**
Total Latency (in ms)	88	168	**140**

Table 1: Network Latency

6.3 Response time measurements for a single request

A client sends a RESOLVE message to the PEP with the UCI of the context information it wants to access. We sent a single request from the client and repeated the same test 20 times to get multiple samples. The results for the time taken by the PMS to evaluate the access request and return a response versus the total time taken for the client to receive back the response for its RESOLVE request are compared below.

To measure the time, we used timestamps inserted in the code to find the exact time an event occurs. We used the timestamps from the node on which the PMS and the client were running. Referring to Fig 28, the PMS response time is x_4-x_2 and the total response time is x_6-x_0. As discussed in subsection 5.4, a thread in the PMS stores the access requests in a buffer and another thread processes these stored requests.

The PMS response time is measured in two ways. The first way is as mentioned above and is the time difference between sending the access response and the time the access request reaches the node. We can call this the *PMS response time*. The second way is the difference between sending the access response and time the access request is pulled out of the buffer for processing. This second measurement is the actual time taken by the PMS to evaluate a request and doesn't include the time when the request is waiting in the buffer. We can call the *PMS request evaluation time*. As seen from the results in

Table 2, these two measurements are not very different in the case of a single request. The average PMS response time is 686 milliseconds and the average PMS request evaluation time is 644 milliseconds. The reason for this small difference is that since the test includes only a single request, the request doesn't have to wait in the queue for a long time.

The captured data from the 20 test samples for a single RESOLVE request are transferred to the Microsoft® Office Excel 2003 spreadsheet application and the results are calculated. Fig 32 shows the 20 test sample results for the first case where the PMS response time includes the time the request is queued in the buffer and the actual evaluation time.

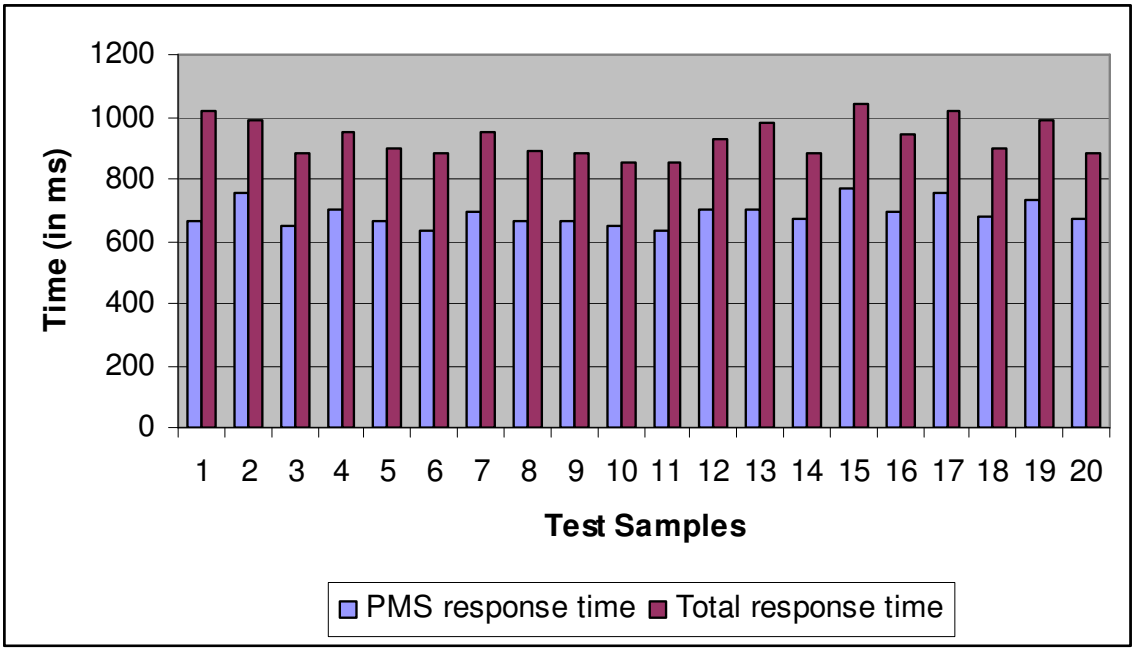

Fig 32: Response times for a single access request from a client

Fig 33 shows the test results for the 20 samples showing the PMS request evaluation time which excludes the time the request is queued in the buffer.

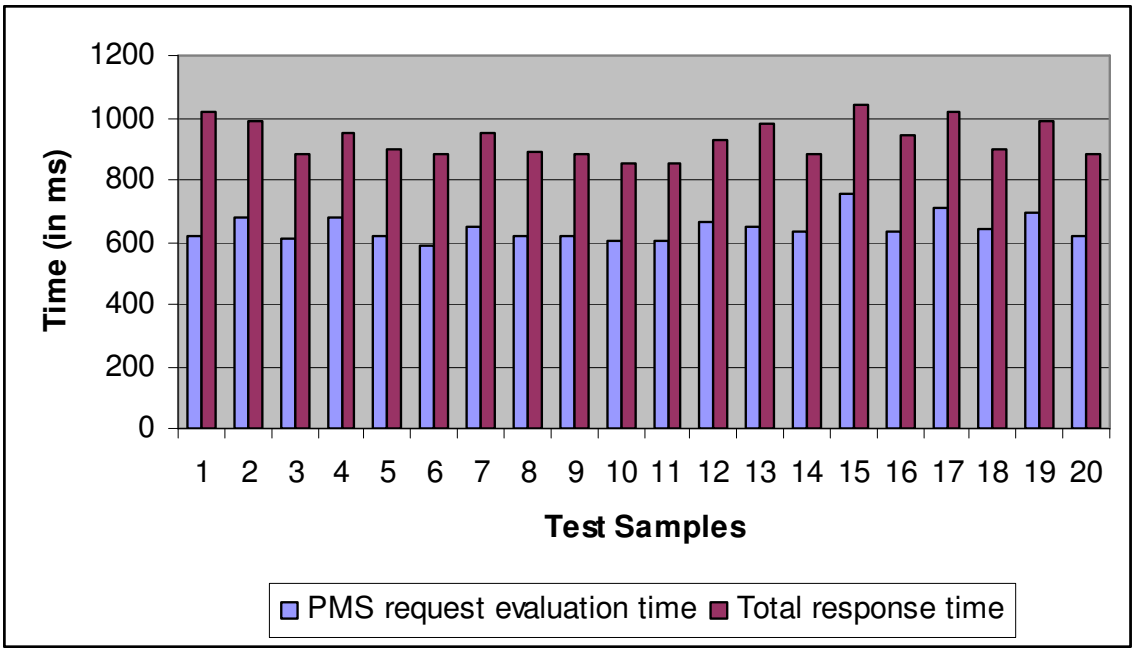

Fig 33: Request evaluation time for a single access request from a client

The values for the PMS response time, the PMS request evaluation time and the total response time are as shown in

Table 2.

Response time for Single Access Request			
	Shortest	**Longest**	**Average**
PMS response time (in ms)	631	771	**686**
PMS request evaluation time (in ms)	601	751	**644**
Total response time (in ms)	851	1041	**931**

Table 2: Response time for Single Access Request

6.4 Response time measurements for a burst of requests

In this test we sent continuous RESOLVE requests from the client to the PMS via the PEP. The test was repeated for a burst of 5, 10, 15, 20, 25, and 30 requests. The measurements were done using timestamp logs of events from the code. This data was

imported into the spreadsheet application and used for calculating the response times. The response times from each of these tests are illustrated in graphs. The results revealed some very interesting observations.

Fig 34 shows the bar chart for the test sample with a burst of 5 requests.

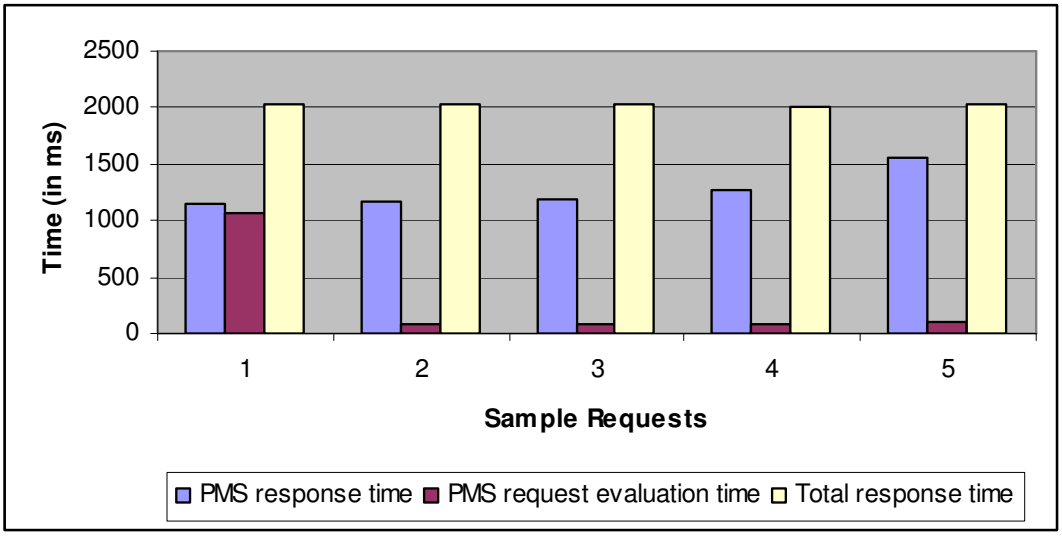

Fig 34: Response time for a burst of 5 requests

Let us first look at the PMS request evaluation time which is indicated by the middle bar in Fig 34. The first request takes 1061 milliseconds for access evaluation and the subsequent requests take an average of 89 ms for access evaluation. The reason for this huge difference in time between the evaluation of the first and the consecutive requests is that, in the implementation, the policies are loaded into the memory at the time of evaluation of the first request. For the subsequent requests, the evaluation takes much lesser time as the request is evaluated against the policies that are already loaded in the memory.

This observation leads to the conclusion that the PMS request evaluation time would reduce ten fold if the policies are read into the memory of the PMS as soon as they are defined or received from the Policy Authoring Point (PAP).

Again looking at Fig 34, we see that the PMS response time does not reduce in tandem with the PMS request evaluation time as the former also includes the buffer queuing time.

Fig 35 shows the chart for a burst of 10 requests.

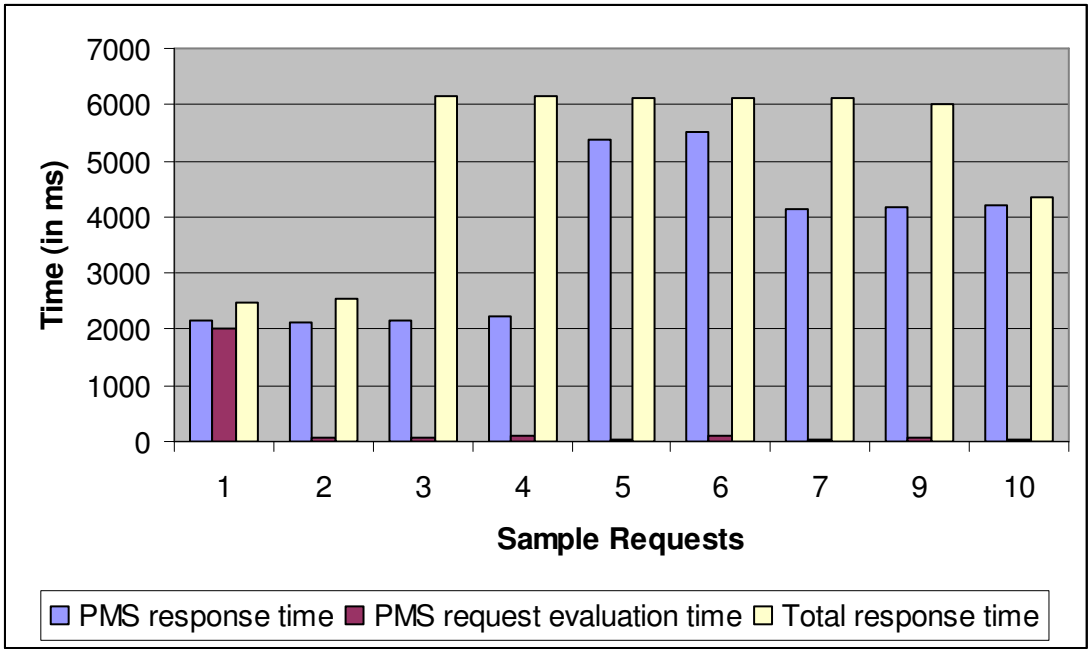

Fig 35: Response time for a burst of 10 requests

Here we again see that the result for the PMS request evaluation time reduces to an average of 61 milliseconds after the first request in which it also loads the policies. But we see that the PMS response time and the total response time deteriorates as the burst of request increases. The reason for this is that there is a single thread in the implementation that processes the buffer and clears the queue. The buffer includes the ACK messages in addition to the request messages.

By introducing a pool of threads, instead of a single thread for processing the buffer, we would be able to reduce the effective PMS response time and bring it closer to the PMS request evaluation time.

Taking account of the fact that the PMS response time includes the request evaluation time and the buffer queue time which would increase with the increase in the number of requests, we will only look at the PMS request evaluation time results for the tests for the burst of 15, 20, 25, and 30 requests.

Fig 36 shows the results for the test when a burst of 15 requests are sent from the client.

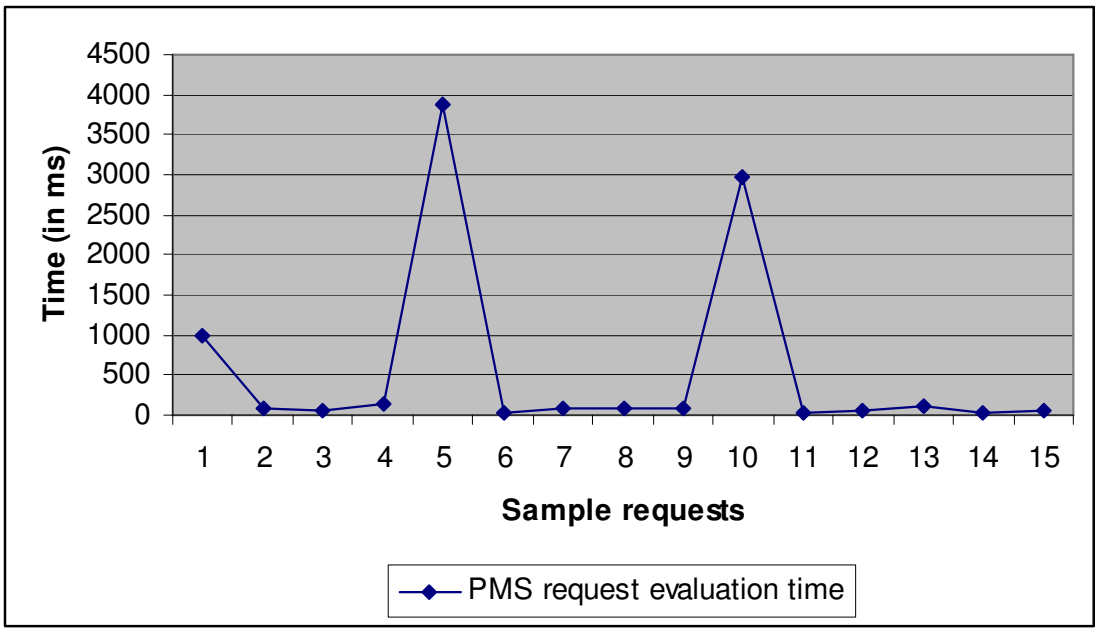

Fig 36: Request Evaluation time for a burst of 15 requests

The results above show the PMS request evaluation time which is the actual evaluation time excluding the buffer queue time. Looking at the line chart, we see that after the first request which loads the policies into memory, the average PMS request evaluation time for most of the requests (excluding the 5th and the 10th request) is 73 ms. There is a very large delay in evaluating the 5th and the 10th request and looking at the logs, we see that the thread from the PMS process evaluating these requests is suspended and the thread from the separate process of the client residing on the same node starts processing the sending of RESOLVE messages and receiving of ACK and RESOLVE_RESP messages.

By separating the applications or processes of the client and the PMS on to separate machines, the PMS evaluation of the request after it has been picked up from the buffer queue can continue uninterrupted and the delay seen in the Fig 36 would disappear.

Fig 37 shows the PMS request evaluation time for a burst of 20, 25 and 30 requests. Looking at the line chart, we see that the behavior seen in Fig 36 is also here. The reason again is that the client process takes control of the CPU processor and the PMS process is suspended.

Thus we can again deduce that placing the PMS and the client on separate nodes, the delay in evaluating the ACCESS_RESPONSE by the PMS can be reduced.

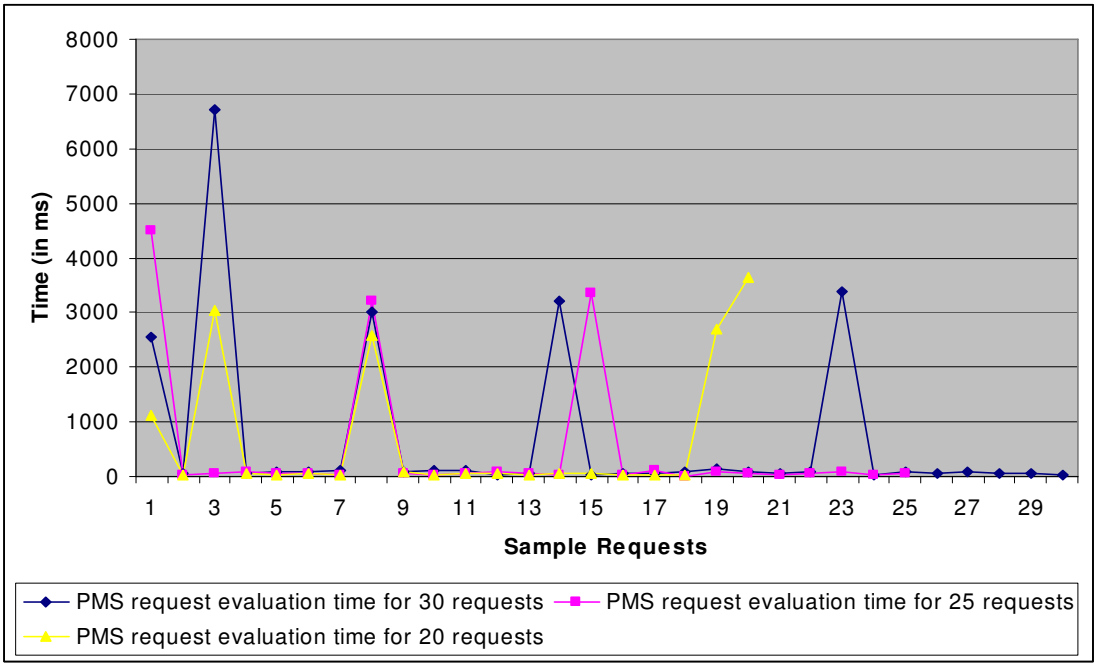

Fig 37: Request Evaluation time for a burst of 20, 25, and 30 requests

Excluding the first request evaluation which includes the policy loading time and also excluding the delay peaks introduced by the placement of the PMS and the client (for the prototype) on the same machine, the average PMS request evaluation time for 20, 25, and 30 requests are 51, 56, and 74 milliseconds respectively.

PMS request evaluation time							
Burst of requests	**5**	**10**	**15**	**20**	**25**	**30**	**Average**
Average Time per request (in ms)	89	61	73	51	56	74	**67**

Table 3: Average PMS request evaluation time per request

From the Table 3, we see that the average access validation time through the PMS is 67 milliseconds which gives us a **throughput of approximately 15 requests per second**.

6.5 Response time measurements for a sequence of requests

In this test we sent a sequence of RESOLVE requests from the client to the PMS via the PEP. The test was repeated three times for cases when each request was separated from the next request by a time factor of 2 seconds, 1 second and 500 milliseconds. The three tests were conducted to measure the response times in a situation when the PMS was not working under load (the spacing of 2 seconds between requests gave

enough time to the processes to complete their task), to situations when the load was gradually increased.

The test was done by sending sequences of 30 requests from the client with delays of 2 seconds, 1 second, and 500 milliseconds between two consecutive requests. The measurements were done using timestamp logs of events from the code. This data was imported into the spreadsheet application and used for calculating the response times. The response times from this test with the spacing of 2 seconds between two consecutive requests are illustrated in Fig 38.

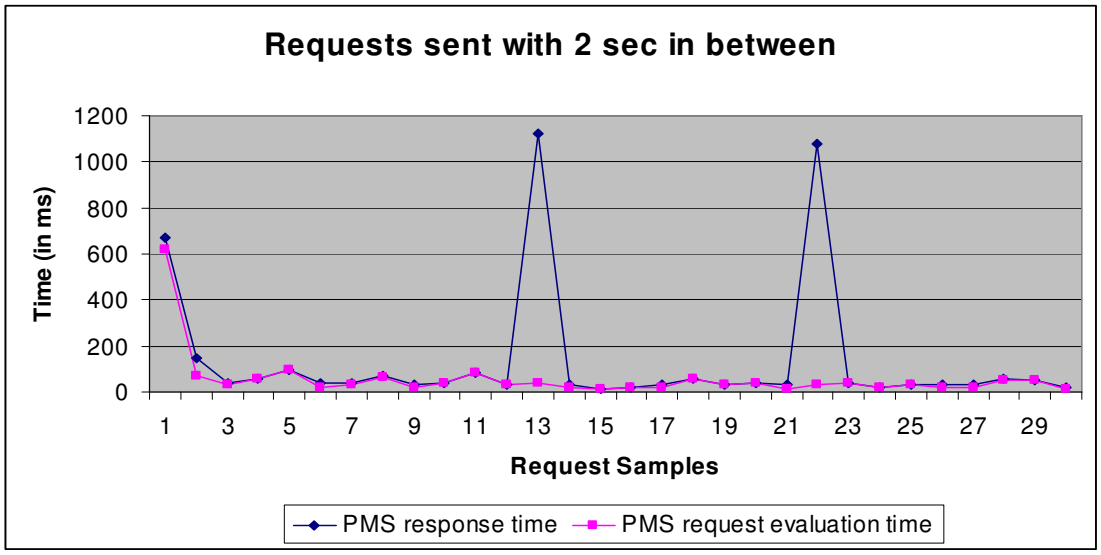

Fig 38: Response time for a sequence of 30 requests spaced apart by 2 seconds

Analysing the test results we see that the first request, which includes the time for loading the policies into the repository, takes 621 milliseconds for evaluation after extracting the request from the queue. The actual response time that includes the time the request is waiting in the buffer is not much more and is 671 milliseconds. After the first request and with the policies loaded in the memory, the subsequent average request evaluation time is 36 milliseconds and none of the request evaluations are delayed.

Looking at the PMS response time results, we see that except for the13th and the 22nd request (where the buffer delay is in the order of one second), all the other requests are picked from the buffer and processed as soon as they are received. As explained in section 6.5, if we introduce a pool of threads for processing the buffer queue, we would be able to overcome this delay.

The average access validation time through the PMS when it is not overloaded is 36 milliseconds.

Again repeating the above test with a spacing of 1 second between two consecutive requests, the response times obtained are shown in Fig 39.

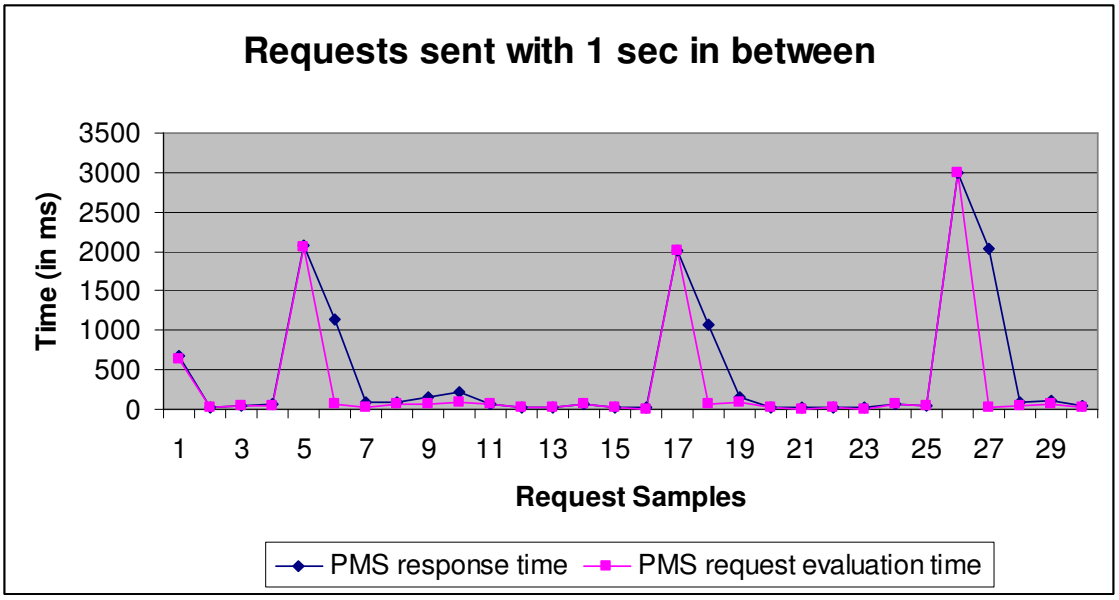

Fig 39: Response time for a sequence of 30 requests spaced apart by 1 second

The graph above shows that the first request that loads the policies, takes 641 ms for request evaluation and 681 ms as the PMS response time. Looking at the logs we saw that the thread evaluating the request after retrieving it from the buffer gets suspended for the 5th, 17th, and the 26th requests and the client thread becomes active.

Referring to the description for the same behavior seen in section 3.4, by separating the client and the PMS on to separate computers, the delay in PMS request evaluation time can be reduced. Average PMS request evaluation time excluding the delay peaks is 43 milliseconds.

The same test was repeated for a burst of requests with 500 milliseconds between two consecutive requests. The results are shown in Fig 40.

The PMS request evaluation time and the PMS response time for the first request which includes the time for loading the policies are 721 ms and 761 ms respectively.

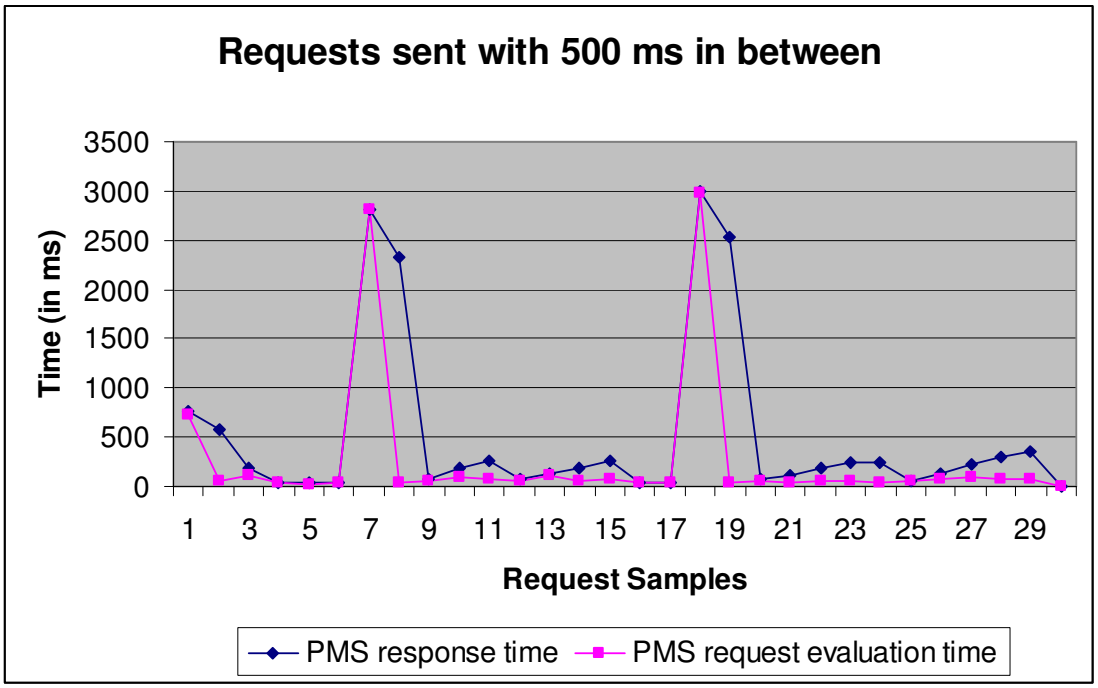

Fig 40: Response time for a sequence of 30 requests spaced apart by 500 milliseconds

We again see the peaks at the 7th and the 18th request caused by the suspension of the PMS thread evaluating the response and the transfer of control to the client thread.

Excluding the 1st, 7th, and the 18th requests from the average PMS response time calculation, the average value for the PMS response time is 58 milliseconds.

6.6 Latency in a system with PMS compared to one without a PMS

In this test setup, we combined the prototype for PMS with the prototype for a distributed context-aware network developed by Swenson [22] and evaluate the result. The test involved a client in the distributed network which directed its request to the PEP and the PMS for requesting access to context information and the result was conveyed back to the client existing in the distributed overlay.

From the measurements we saw that the following results.

Average PMS response time = 619 ms
Average total response time = 1950 ms

The PMS response time also includes the time for loading the policies into the memory. With policies preloaded into memory, and looking at the previous results, we can deduce that this time will be reduced ten-folds. Considering a worst case scenario, we assume that the average PMS request evaluation time excluding the policy loading time is 100 milliseconds.

Looking at the results in Swenson's report [22], we see that the best response time without using the PMS is 1680 ms and the worst response time is 2100 ms. If the PMS is used in the distributed context provisioning network, and using the assumed

worst case values of 100 milliseconds for access evaluation, the time delay overhead caused by access request evaluation would be from 4.7% to 6%.

Using the value of 67 millisecond for average PMS request evaluation time in a PMS running with load (from subsection 6.4), the time delay overhead caused by access request evaluation, in a distributed context provisioning network, would be 3.2% to 4%.

In addition to the access request evaluation, the policy management would also include time delay overheads for request generation by the PEP and request parsing. These actions are not part of the PMS node and would be handled by the individual Functional Entities (FEs) but would still account for an overhead in the actual policy management process.

6.7 Analysis

Evaluating the prototype, we came up with some very interesting observations. These are highlighted as below:

- The loading of policies onto the PMS memory is an expensive task in terms of time. The PMS request evaluation time would reduce ten fold if the access request evaluation task is kept separate from the policy loading task and the policies are read into the memory of the PMS as soon as they are defined or received from the Policy Authoring Point (PAP).
- By introducing a pool of threads, instead of a single thread for processing the buffer in the PMS, multiple requests waiting in the buffer queue could be evaluated simultaneously, thus reducing the effective PMS response time and bringing it closer to the PMS request evaluation time.
- By separating the applications or processes of the client and the PMS on to separate machines, the PMS evaluation of the request after it has been picked up from the buffer queue can continue uninterrupted and the delay caused by the client process taking up the processor time, would disappear.

The reason for the suggestion of separating the client from the PMS onto different machines is because in the prototype, all the requests were sent from the same client and these were then received via the PEP by the PMS. For example, if the client sends a burst of 100 requests, with the client and the PMS situated on the same machine, the CPU would have to distribute the processing time between the client sending 100 requests and the PMS evaluating 100 requests.

Performance measurements in java include the compilation time of translating Java bytecodes into native code at runtime [30]. Thus creating the prototype in a native language instead of Java would increase performance. Another suggestion is to run the prototype on different operating systems such as Unix or Linux instead of Microsoft ® Windows ® to compare the performance of the PMS.

For a practical implementation within an Ambient Network, the clients would be distributed on different nodes and the request would be received from the distributed PEPs. This would solve the problem seen in the prototype introduced by processor scheduling to some extent.

The average time taken per request for access evaluation is 67 ms leading to throughput result for the PMS working under load to be 15 requests per second.

Combining the prototype for policy management with the prototype for distributed context-aware network, and based on measurements and analysis, we saw that the time delay overhead caused by access request evaluation ranged from 4.7% to 6% for a worst case scenario.

7 Conclusion and suggestions for future work

This chapter concludes the thesis bringing into focus the accomplishments of the thesis in terms of the initial goal, the knowledge gained, and the contributions made. This chapter also contains suggestions for future work, discussing the topics and directions in which the thesis could be extended.

7.1 Conclusion

The goal of the project was to design and implement a policy management architecture within the Ambient Network project for managing access to context information. The project involved doing a background study of Ambient Networks, as well as a study of the existing policy management systems.

Policy management within an AN is a very wide area as policies in such an AN can be used for handling mobility, security, context, composition, QoS, service provisioning, etc. This thesis limited the scope of the project to management of access control policies.

Context information within the Ambient Network is distributed amongst the various FEs of the AN responsible for the different context information. For our architectural design of the policy management system we had to decide if the PMS should be distributed or centralized. The decision was to have the policy decision point and the policy repository as centralized entities and the policy enforcement points as distributed entities. This design principal enabled a consolidated and global view of the complete AN, and was a pragmatic decision at this point, as *there are currently a limited number of policies in use within the AN*. These policies are being developed by the different work packages responsible for the various Functional Entities(FE), to define the access rules for the context information associated with the FEs. The design also defined how the policy management system should behave under different types of composition.

The protocol design utilized a XML based messaging protocol over UDP which took care of policy provisioning from the policy authoring points to the policy repository as well as request/response messaging related to access control between the client and the PMS via the PEP.

A prototype based on the proposed architecture and using the defined protocol was implemented and tested. The implementation was made using laptops, as currently the XACML implementation is not supported on handheld devices. The prototype included an implementation of a client, a PEP, and a PMS using a XML file based repository.

The design was evaluated by conducting some tests on the prototype. These tests measured the performance of the PMS with regard to the response time for evaluating an access request and throughput. The response time was measured for a single request, a burst of requests, and a sequence of requests with 2 seconds, 1 second, and 500 milliseconds of delay between two consecutive requests to arrive at some conclusions regarding the performance and scalability of the PMS. In addition, the

prototype developed for the PMS was combined with the prototype developed by Markus Swenson [22] implementing a distributed context provisioning network and the response time in the combined prototype was measured. This measurement was used to arrive at the delay overhead values introduced by the policy evaluation process.

The results for all the measurements mentioned above are summarized in section 6.7.

Evaluating the results from the measurements, we can see that policy evaluation for access control **doesn't** cause a considerable delay (4.7% to 6%) compared to the provisioning of context information done without access control.

7.2 Suggestions for future work

- **Distributed PMS**

  The current solution is based on a centralised PDP and distributed PEP and PAP approach. Another approach that could be explored is to have a combination of distributed and centralised PDP. The idea would be to have distributed local PDPs at the different FEs collocated with the PEPs. The scope of this local PDP would be restricted to the policies within the associated FE. Thus the PDP would make decisions local to the FE. Since there might be other policies created by entities outside the FE which define access rights to the UCI object within the FE, it is essential to take a decision based on all relevant policies. To enable this, the policy decision from all the local PDPs could be propagated to a centralized PDP which has an overall view of the AN and makes a decision based on the combined partial results from the different FEs.

  With this approach, the work and thus the load of access decision would be distributed throughout the Ambient Network. Moreover the local decisions could have a time to live (TTL) and would not be re-propagated to the central PDP until their timeout, thus avoiding excessive congestion and saving resources in the network. This study would be able to compare the practicality of a distributed approach compared to a centralised approach in terms of latency and load overheads.

  The distributed approach would also have to explore the validity of the propagated partial access decisions in case of composition, depending on the type of composition.

- **Conflict detection in policies and conflict negotiation /resolution**

  Conflict detection and resolution of policies is a broad area with many unknowns and needs further study.

  In a highly dynamic environment like Ambient Network, context information can have many policies linked to it and these policies could be created by different entities and could by dynamic in the sense that they could change

with time. The policies defined for one resource could be in conflict with each other and would lead to ambiguous results. Thus it is essential to resolve the conflicts in the policies before they are used for evaluation.

Policy conflicts can occur at different levels and times and so their resolution cannot be restricted to only one method. These conflicts could appear at application layer as conflicts between different services or they could appear in the hardware level. The policies could start conflicting when two policy repositories are being merged at the time of composition, or they could start conflicting at runtime depending on the current context information, or because of conflicting policy actions being executed simultaneously.

As the policy conflicts can appear at different times, the resolution could also be carried out at different times. Static policy conflicts could be detected at the time of creation and while being stored in the repository. This method would enable the system to maintain a static policy conflict-free database.

Resolution of runtime policies can be done automatically by conflict resolution software that refers to some set of meta-policies using predefined precedence rules. Human intervention should be an optional feature initiated only as a last resort since the overall goal is to achieve unattended functioning of the AN. What effect the runtime conflict resolution would have on the performance of the policy management system would have to be evaluated.

Thus we see the complexity of the policy conflicts that could appear in the Ambient Networks and need to be studied further to arrive at suitable solutions.

References

[1] IST project 507134 Ambient Networks, http://www.ambient-networks.org/ (2006-01-11).

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

[3] Ambient Networks Project WP6, “Ambient Networks ContextWare - First Paper on Context-aware Networks”, Technical report, IST–2002-507134-AN/WP6/D61, 2005-01-07.

[4] Ambient Networks Project WP6, “Ambient Networks ContextWare - Second Paper on Context-aware Networks”, Technical report, IST–2002-507134-AN/WP6/D6-3, 2005-12-16.

[5] Adaptive & Context-Aware Services project. http://psi.verkstad.net/acas, project website, last accessed 2006-08-16.

[6] Sergio Quintanilla Vidal, “Context-Aware Networks: Design, Implementation and Evaluation of an Architecture and a Protocol for the Ambient Networks Project”, Final Thesis, LITH-IDA-EX--06/012—SE, Linköpings University, 2006-03-13.

[7] R. Giaffreda, H. Tschofenig, T. Kanter, C. Reichert, ”An Authorisation and Privacy Framework for Context-aware Networks”, Technical Publication, Work package 6, Ambient Networks, 2005-05-03.

[8] Wireless World Initiative, http://www.wireless-world-initiative.org/, (2006-07-13).

[9] T. Rinta-aho, “Introduction to Ambient Networks. Slides for NETS 1st seminar”, April 2005.

[10] A. Karmouch, R. Giaffreda, A. Jonsson, A. Galis, M. Smirnov, R. Glitho and A. Karlsson, “Context Management Architecture for Ambient Networks“,Wireless World Research Forum 11, Oslo, Norway, Jun 10-11, 2004.

[11] Ambient Networks Project WP1, “AN Framework Architecture”, Technical Report, IST-2002-507134-AN/WP1-D05, 2005-12-30

[12] Ambient Networks Project WP1, “Ambient Networking: Concepts and Architecture”, Technical Report, IST–2002-507134-AN/WP1/D08, 2005-02-17

[13] OASIS XACML, “eXtensible Access Control Markup Language (XACML)”, Version 2.0, OASIS Standard, 2005-02-01

[14] Geographic Location/Privacy (geopriv) Working Group Charter, available at http://www.ietf.org/html.charters/geopriv-charter.html, (April 2005).

[15] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry, "The COPS (Common Open Policy Service) Protocol", RFC 2748, January 2000.

[16] R. Yavatkar, D. Pendarakis, and R. Guerin, “A Framework for Policy-based Admission Control”, RFC 2753, Jan 2000.

[17] C. Kamienski, J. Fidalgo, D. Sadok, J. Lima, L. Pereira, and B. Ohlman, “PBMAN: A Policy-based Management Framework for Ambient Networks”, Jan 2006.

[18] Ambient Networks Project WP6, “Integrated Design for Context, Network and Policy Management”, Technical report, FP6-CALL4-027662-AN P2/ D10-D1, final draft, Dec 2006.

[19] Ambient Networks Project WP6, “Analysis of Policy-based Management Requirements and Tools for Ambient Networks”, FP6-CALL4-027662-AN P2/ UCL, Feb 2006.

[20] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1”, RFC 4346, April 2006

[21] Ethereal, Network Protocol Analyser, http://www.ethereal.com, accessed on January 2nd 2007.

[22] M. Swenson, “A distributed approach to context-aware networks”, Upcoming Master Thesis report, Royal Institute of Technology, Stockholm, January 2007.

[23] B. Ohlman, K. Jean, A. Galis, I. Herwono, and J. Nielsen, “Requirements for a policy framework for Ambient Network”, Wireless World Research forum.

[24] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, “Looking Up Data in P2P Systems”, Communications of the ACM, February 2003.

[25] A. Méhes and G. Selander, “Identifiers and keys for management and composition of Ambient Networks”, Annex 3 of Deliverable Report D7.2 “Ambient Network Security Architecture”, Ambient Networks Project, IST-2002-507134-AN/WP7/D02, December 2005.

[26] Wireshark, http://www.wireshark.org/, last accessed January 2007

[27] Sun's XACML Implementation, http://sunxacml.sourceforge.net, last accessed January 2007.

[28] Eclipse SDK, 3.2.0, http://www.eclipse.org/, last accessed January 2007.

[29] kXML-RPC project, http://kxmlrpc.objectweb.org/, last accessed January 2007.

[30] M. Cierniak, G. Lueh, J. M. Stichnoth, “Practicing JUDO: Java under dynamic optimizations”, May 2000.

COS/CCS 2007-04

www.kth.se