# DISTRIBUTED CLIENTS' PROFILE MANAGEMENT SYSTEM FOR PERVASIVE SYSTEMS

BY
**HAILELEUL TEGENE**

**A THESIS SUBMITTED TO**
**THE SCHOOL OF GRADUATE STUDIES OF ADDIS ABABA UNIVERSITY**
**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS**
**FOR THE DEGREE OF MASTER OF SCIENCE**
**IN COMPUTER SCIENCE**

**NOVEMBER, 2005**
ADDIS ABABA

# Acknowledgments

I owe large debt to my advisor Dr. Dawit Bekele for his consistent support in commenting and showing ways through the thesis work. I'd also like to thank Dr. Jean-Mark Pierson for his comments and invaluable suggestions during the initial state of the work.

I gratefully thank all of my friends for their encouragement that has become momentum during my study.

Finally, I would like to extend my appreciation to my parents, Tegene Mekuria and Ayelech Abebe, who have bestowed a love of learning in me.

# Table of Contents


1 Introduction ..... 1

1.1 Background ..... 3

1.2 Problem Statement ..... 6

1.3 Objective of the Thesis ..... 6

1.4 Scope of the Thesis ..... 7

1.5 Methodology ..... 7

1.6 Organization of the Paper ..... 10

2 Profile Information, its Representation, and Communication methods ..... 11

2.1 Profile ..... 11

2.2 Profile Representation Tools ..... 16

2.2.1 XML (eXtensible Mark-up Language) ..... 17

2.2.2 RDF (Resource Description Framework) ..... 19

2.2.3 CC/PP (Composite Capabilities and Preference Profiles) ..... 22

2.3 Communication Method used for profile exchange ..... 30

*2.3.1 Communication Using Mobile Agents* ..... 31

*2.3.2 Communication Using SOAP (Simple Object Access Protocol)* ..... 35

3 Related Work ..... 40

4 The Adaptation/Delivery Infrastructure ..... 43

5 Design of Profile Management System ..... 46

5.1 Design Issues ..... 46

5.1.1 Characteristics of the client's Profile ..... 46

5.1.2 Constraints in Pervasive systems ..... 47

5.2 Evaluation of Profile Storage Sites ..... 48

5.2.1 Storing on Client Devices ..... 49

5.2.2 Storing on Local Proxy Servers ..... 50
5.2.3 Storing on a Separate Profile Server ..... 50
5.3 Proposed Architecture for Client's Profile Management System ..... 51
5.3.1 Proposed Profile Placement ..... 51
5.3.2 Management of the Fragments ..... 52
5.4 Distribution of Client's Profile on Local Proxy Servers ..... 62
5.4.1 Distribution of dynamic profile ..... 62
5.4.2 Distribution of static profile ..... 65
6 Prototype ..... 71
6.1 Overview ..... 71
6.2 The Development Environment ..... 72
6.3 Implemented Components ..... 73
7 Conclusion and Future Works ..... 78
7.1 Conclusion ..... 78
7.2 Future Works ..... 79
Reference ..... 81

## List of Figures

Figure 2.1: Hierarchical Data Structure of Profiles ..... 15

Figure 2.2: An RDF statement ..... 20

Figure 2.3: An instance of several RDF statements combined into a single diagram. .... 20

Figure 3.1: Architecture Overview----------41

Figure 4.1: Service-based distributed content adaptation infrastructure----------44

Figure 5.1: the Proposed Client's Profile Management Architecture----------54

Figure 5.2: The Client Side Profile Manager (CSPM)----------56

Figure 5.3: Server Side Profile Manager (SSPM) and its interaction with external components----------58

Figure 5.4: Showing interaction of client device with LP during signup----------63

Figure 5.5: Architecture Overview and data flow upon a user request----------69

Figure 6.1: User Login Form----------74

## Listings

*Listing 2.1: XML serialization of the RDF model of figure 2.3----------20*

*Listing 2.2: CC/PP Profile Components----------22*

*Listing 2.3: CC/PP Profile Components in XML----------24*

*Listing 2.4: A Complete Example of CC/PP Profile----------26*

*Listing 2.5: A Complete Example of CC/PP Profile in XML----------28*

*Listing 2.5: The SOAP envelope----------35*

*Listing 2.6: Remote method invocation with SOAP----------37*

# Abstract

The heterogeneity of device capabilities, user's context and communication network that exists in pervasive systems has accentuated the need for a content adaptation. The purpose of the content adaptation is to convert the original content before delivery so that the new content would fit to the capability of the client's device being used, the available bandwidth of the communication network, and the specific need of the user. In order to perform such an adaptation, the adaptation system is required to maintain adequate information about the clients' profile, the network profile, the content profile and others. This thesis work deals with the management of the clients' profile that includes device and user profile. Specifically, it tries to set up an appropriate way of managing clients' profile in pervasive environment in relation to content adaptation. To this end, a distributed clients' profile management scheme is proposed in the current work. In the design of the management scheme, two issues have been considered: (a) client's profile characteristics such as existence of confidential information in the client's profile, temporal nature of some of the profiles, etc. (b) constraints that exist in the pervasive systems such as heterogeneity of the client devices in terms of storage, processing, and display capabilities, mobility of the clients, etc. Based on the profile characteristics and the constraints of pervasive systems, we proposed to fragment the client's profiles into two: those that can be kept on the client devices and those that can be kept on the server side. The main consideration of this fragmentation is the security issue of confidential profiles and the storage size constraint of the client devices. In the proposed architecture, the server side profiles are, in turn, distributed on the local servers to meet the requirements such as availability of profile information in partial failure of the system, performance, scalability, etc.

The proposed system consists of different components that interact with each other through web interfaces for the management of distributed client's profile in a transparent way. It also provides interfaces to external components such as automatic profile extractors so that the dynamic profiles will be built in the course of service provision.

# CHAPTER One

## 1 Introduction

Over the last couple of years advances in digital electronics have made computers smaller, cheaper, and faster. This trend, along with other industrial advances, has promoted the development and rapid market growth of small computers that can be carried from place to place. It has also paved a way for computers to be embedded in everything from household appliances to automobiles. Consequently, computation is made possible almost anywhere; for example, in vehicles while driving, in planes while traveling, in the parks while touring, etc.

Advancement in computer network has brought high speed data exchange with high bandwidth communication that enables to transport large size of data within a short period of time. Area coverage of the network also varies considerably, from the local area network (LAN) that spans only over a few meters to the wide area network (WAN) , for example Internet that covers the entire world. Development in network technology has also introduced wireless communication that has enabled connectivity among numerous and scattered small devices.

Emergence of these small wireless communication devices along with high-speed wireless network has brought a new need in the field of computing. This need is to have ubiquitous access to data and computation; i.e., having access to data and computation at anytime, from anywhere, and using any available computational device. The idea behind ubiquitous computation is to enable invisible computation so that to enhance life in the real world. A system that provides such services is known as pervasive system [Celeste Campo, 2002].

Pervasive computing, which is also called ubiquitous computing, is a technological evolution heading to allow computation from anywhere and anytime on invisible computing devices that surrounds a person. In such an environment, a person is invaded

by numerous computational devices so that to enable the person to get access to the system using any of the devices. As a result, the technology is also sometimes referred to as the third wave of computing, that is many computers per person. The idea of this evolution is first conceived at Xerox PARC laboratory by Mark Weiser in 1988 [8].

To provide its functionalities, pervasive computing deals with a number of challenges. One of such challenges is to be able to avail information to a user regardless of the device he is using and the state of the network he is currently located in. To address this challenge, the system is required to make some sort of modification on the content to be delivered to the user. In pervasive computing terms, this process is referred to as *content adaptation*. Content adaptation is performed based on the context[1] of the computing environment including capability of the client[2] device being used, connectivity of the network, adequate information about the content to be delivered, and preference of the user. In general, this type of information is known as *profile*. To succeed the adaptation, the system needs to have a means to gather and maintain profile information.

This thesis work aims at devising a means for managing client's profile that includes information of the users and their devices. The client's profiles to be maintained are targeted for content adaptation in pervasive systems. To provide solution for the problem, various requirements regarding client's profile and constraints in pervasive systems are identified. Using the identified parameters, a distributed profile management scheme is proposed. The proposed system incorporates various modules that manage client's profile in a transparent way. To provide the required functionalities, the modules are distributed over the adaptation/delivery infrastructure. The modules maintain the well defined interfaces to interact with each other as well as with external components. This paper details the proposed solution for client's profile management system.

[1] Refers to the circumstances or situation in which a computing task takes place.
[2] Anything that consumes service/data from the system

## 1.1 Background

One of the aims of pervasive systems is to be human centered. That is, these systems try to deliver the services based on the current context of the user. User's context includes useful information such as what the user is doing, where he is located, what the surrounding resources are, what his intent in using the resources is, etc. Such information helps the pervasive systems to offer relevant services to the user that meets his current need. For instance, during meetings, the system should help the user giving him/her information about the people with whom the discussion is held, and during his travel for tour, it should help him/her by providing information on the places which he is intended to visit.

The pervasive systems also encompass a wide spectrum of client devices such as cellular phones, desktop PC, PDA, etc. These devices greatly vary in size. For example, the system includes from very small devices such as sensors, which can be embedded virtually on any device, to desktop PC, which is not portable. The size difference of the client devices has brought variation in displaying capabilities, processing capabilities, storage spaces, power consumption, and network connectivity of the devices. As a result, contents cannot be equally accessed by all of these devices unless they are customized appropriately to fit capability of the devices. For instance, consider an image of size 420x450 pixels. This image can effectively be displayed on a desktop computer of 15 inch monitor, whereas it is not possible to fully display on a cell phone of screen dimension 120x150 pixels.

The above facts drove the pervasive systems towards having one important service that is not common in traditional systems. The aim of this service is to customize the content so that it would fit to the capability of the devices. Moreover, the service also aims at providing contents that meet preference of the user. This type of service is known as *content adaptation*. Basically, it is a process of modifying the original content as per the requirements of the system. Content adaptation may include format transcoding (e.g. XML to WML, JPEG to WBMP), scaling of images as well as video and audio streams, media conversion (e.g. text-to-speech), omission or substitution of document parts (e.g.

substitution of an image by a textual representation), document fragmentation, language translation, etc. [2]

Based on the location for adaptation operation, content adaptation frameworks can be categorized into four [2]: *(i) Server-based approach*: the adaptation operation is performed at the server by extending the functionalities of the traditional servers *(ii) Client-based approach:* decision for adaptation operation is left to the client. After receiving the content from the server, the client does adaptation in its accord *(iii) Proxy-based approach:* a proxy intercepts every communication in between the server and the client to do adaptation of the content received from the server before delivered to the client *(iv) service-based approach:* this approach sees the content adaptation from the service point of view where adaptations are done by third-party service providers.

Among the above four approaches, service-based approach is relatively recent and it is introduced with surfacing of Web Service technologies [2]. This approach is based on specialized, third party services to perform the adaptation. There are various adaptation services each of which is dedicated to a particular adaptation service. This thus enhances the adaptation services. In this case, the adaptation services are accessed through subscription, either by the users or the content providers. Complete description of the service-based content adaptation intended for pervasive systems, which was developed by Girma Berhe, is given in [2].

Whichever approach is used for the content adaptation, the adaptation operation is done based on information that describes the content, user preference, device capabilities, network availability, etc. Each of this information can be obtained from different sources. For example, information that describes the content, also called metadata of the content, can be extracted from the content itself, information about the user preferences can be obtained directly from the user or indirectly during user interaction with the system, information on client devices can be accessed from the vendor's website through the net, etc. All such information, which is required for adaptation, is known as profiles.

Generally, profiles can be categorized into two groups [2]: *(i) Content or media profile:* this profile represents object description such as location where the media is stored and

media features such as type, format, size, etc of the content *(ii) Context profile*: this includes client's profile such as user's profile and device profile, network characteristics such as bandwidth and latency, and service profile that includes information on content adaptation such as service type, service location, price, etc.

Thus, content adaptation system must take both content and context profile into account for the adaptation decision. The quality of the content adaptation service is determined by the extent and relevance of the profile information that is available to the adaptation system. Details on profiles are given in section 2.1.

Due to its high importance, collection and management of profile information is at the heart of pervasive systems. The work of this thesis is thus aimed at devising appropriate way of managing client's profile that would suit for pervasive environments.

## 1.2 Problem Statement

The core problem of this work is the management of clients' profile in adaptation based pervasive systems. In this thesis, we consider the various constrains that exist in pervasive systems in addition to the characteristics of the clients' profile for the design of clients' profile management system that would suit to pervasive environments. Management of the clients' profile includes: acquisition of the profiles from various sources such as profile extractors and profile entry forms, proper storage of the acquired profiles, retrieval of the profiles, updating, etc.

## 1.3 Objective of the Thesis

The general objective of this work is to propose an architecture for client's profile management system, specifically targeting pervasive systems. The main concern of the architecture is to be able to store as much profile information as supplied by external components such as profile extractors. Furthermore, the architecture facilitates accesses to the clients' profile by external components that require profile information for content adaptation.

Another objective of the thesis work is to develop a prototype for the client's profile management system based on the proposed architecture. It is aimed to verify validity of the proposed solution.

## 1.4 Scope of the Thesis

This thesis confines itself to the management of client's profile for pervasive environments. The work focuses on the management tasks of the profile data, specifically client's profile. Here, client's profile refers to profile of the client devices and profile of the user. All the remaining profiles are not in the scope of this work. The work is not concerned with the following tasks either:

- *Profile representation tool*: It does not go into creating a new tool for profile representation; instead explores existing standard tools for profile representation and will use one.
- *Profile extraction*: It assumes that there exist profile extractors that continuously supply various forms of client's profiles so that the profile management system to be developed would focus only on the management tasks.

## 1.5 Methodology

In this thesis work, our aim is to design non-centralized client's profile management system so that issues such as scalability, efficiency, and reliability are well addressed. To complete the work, series of tasks, which are categorized into three phases, are performed.

***Phase-I:***

In this phase, intensive study and explorations were made on the areas related to the thesis work so as to have adequate understanding on the problem and its settings at large. Study is also made on the research works, which are already completed as well as underway, in the areas related to the current work through literature review. Among the activities performed in this phase are: studying the pervasive systems to clearly see the challenges, reading about content adaptation and its requirements, exploring characteristics of client's profile and profile representation tools, and exploring the various inter-process communication methods that exist in distributed systems.

The outcome of this phase has become important material for the next phase.

***Phase-II:***

In this phase, based on the study and investigation made in phase one, solution for the thesis problem is proposed. To this end, client's profile management scheme was proposed. In addition to the outcomes of the first phase, a thorough discussion with advisors was made to come up with the proposed management scheme.

The followings are summary of the main activities performed in this phase:

- Identification of design issues such as characteristics of client's profile and major constraints in the pervasive systems was made.
- Identification and evaluation of profile allocation alternatives were made.
- Profile allocation strategy and distributed profile management architecture were proposed.
- Study on issues related with distribution such as consistency problem is made. Based on the study, solutions for each of the identified problems were proposed.

Result of this phase is used as input to the next phase that deals with prototype development.

***Phase-III:***

Here, based on the scheme proposed in phase two, a prototype for the client's profile management system was developed. The main aim of this phase is to demonstrate the validity of the proposed solution to the original problem of the thesis work. The work of this thesis is related to the research, known as Distributed Content Adaptation Framework (DCAF), which is being conducted by Girma Berhe at INSA, Lyon France (see section 3.2 for further detail). Thus, the prototype of the current work is also related to the prototype of DCAF, which was developed at INSA. As a result of this, among other

activities, a study on the prototype of DCAF was made. The followings are list of the activities that have been carried out to complete the prototype development:

- Studying the DCAF's prototype.
- Identifying the requirements for the prototype.
- Studying the development environment such as development tools and protocols.
- Based on the identified requirements, developing a design for the prototype.
- Implementation of the design and then testing its functionalities.

## *1.6 Organization of the Paper*

The remaining part of this report is organized as follows. Section 2 discusses profile information including the representation tools and communication methods. Section 3 presents overview of literatures related to the current work. Section 4 details the proposed approach for the client's profile management. Section 5 briefs the prototype developed for the client's profile management system, based on the proposed architecture. Finally, section 6 concludes the report and indicates avenues for future works.

# CHAPTER 2

# 2 Profile Information, its Representation, and Communication methods

## *2.1 Profile*

Profile can be defined as any information about a user, device, network, content, and location that is useful for offering a better response to a given request [1]. Such type of information is significant in improving the response to a user request. In pervasive systems, the diversity in devices and user preferences have made the 'content personalization' an important requirement in order to achieve results that satisfy the user's need and also that fit to the capability of the client's device. The extent of content personalization of a system depends mainly on the amount of profile information available for adaptation. If, for example, the system knows more facts about the user's interest, then it is possible to provide a better service that best suits the interest of the user by applying appropriate adaptation on the content.

Profile information is broadly classified into two: context profile and media profile [2]. Context profile refers to any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical and computational object [3]. A media object profile, on the other hand, is a meta-data of the content that describes the content itself [2].

Media profile includes:

- Media features such as type (image or text), format (JPEG or MP3), size, color depth, etc.
- Available modalities or content version. For example, an X-ray image might have textual description.
- Location where the media is stored.

Context profile includes:

- User profile – information that encompasses personal data of the user such as security number and credit card of the user, current location of the user, as well as user preferences such as language preference, media type (image or text), response time, etc.

- Device profile - information about a device that includes description of the hardware and software:

  - Hardware capabilities: name of the vendor; model of the device; type of the device (PDA, cell phone etc); screen size (98x60 pixels, 160x160 pixels); colors; CPU type and speed; memory size; available input devices; secondary storage; loudspeaker; etc.

  - Software capabilities: level of HTML support; application brand and version; supported XML vocabularies; supported RDF vocabularies; supported scripting languages; etc.

  Information of this type is more or less stable. It is filled and completed at the beginning and does not require regular recharging.

- Network connectivity such as the network availability, bandwidth of the network, etc.

Client's profile is a context profile that includes only user profile and device profile [2]. To structure the concept of context, context profile is classified into two general categories: human factors and physical environments [10], where the human factors include user information, social environment and tasks and the physical environment includes location, infrastructure, and physical conditions.

To collect context information, various methods can be employed. For example, a user can explicitly enter his/her preference or the system itself may study the trend of user interaction with the system and come up with possible inference about the profile information. Similarly, the user's current location can be acquired with the help of low-

level sensors. Device profile can be gathered from the user or retrieved from profile repository of Vendor Company using device identification.

Applications that use context information to provide task-relevant service or information to a user are known as context-aware applications. For a specific application, only a subset of available context information is collected and used. That is, a given context-aware application does not require the entire context information; instead, it uses the one that relates to the type of service to be given by the application.

As pervasive systems are highly dynamic, the applications running in such environments should be flexible to provide services to the user without his/her attention/involvement. For instance, during a meeting, the system should help you by giving information about the people with whom the discussion is held, and during your travel for tour, it should help you by providing information on the places, which you are intending to visit, etc.

For applications to have this nature, they should be more and more context-aware.

Characteristics of context information [10]:

a) *Static and dynamic*: In pervasive environments, context information can be characterized as static or dynamic. As pervasive systems are dynamic, most of the context information is changing and hence dynamic. But there are still invariant context profiles to be considered. The persistence characteristic of context information then determines the means by which the profile information must be extracted. Accordingly, most of static context profiles such as capability of the client's device and user's personal data are obtained directly from the user. On the other hand, the dynamic context information such as user preference on media type and device location are indirectly obtained using automatic profile extractor software and sensor devices, respectively.

b) *Imperfect*: Context information may not reflect the actual state of the world that it models; i.e. it may have contradictory information, or may not adequately describe the world. For example, consider a user preference that gives description about the user's interest. Profile of this type may be extracted using automated

tools while the user is interacting with the system or getting service from the system. If the user, for instance, browses soccer site for a couple of times for some reason, the extractor learns that the user is interested in soccer games and hence considers this as the user's preference. Thus, this profile is used in future service provision to the user, while the user may not be interested in soccer.

c) *Multiple Representations*: Due to the dynamic nature of pervasive systems, some of the context information such as device location is collected dynamically with the help of sensors. But, the output of the sensors is not suited for the high level target applications that use the location information. This implies that, there have to be various forms of processing on context information so that it is represented in a form that can be used by the high level applications.

d) *Context Information is Interrelated*: Context information is highly interrelated; for example, there is ownership relation between people and devices. And also some of the context information is derived from the others; for example, the person's current activity may partially be derived from the person's current location and his/her past activities.

The profile data structure can be described using a hierarchical representation, see figure 2.1 below. The hierarchical data structure facilitates parsing of the profile data [2].

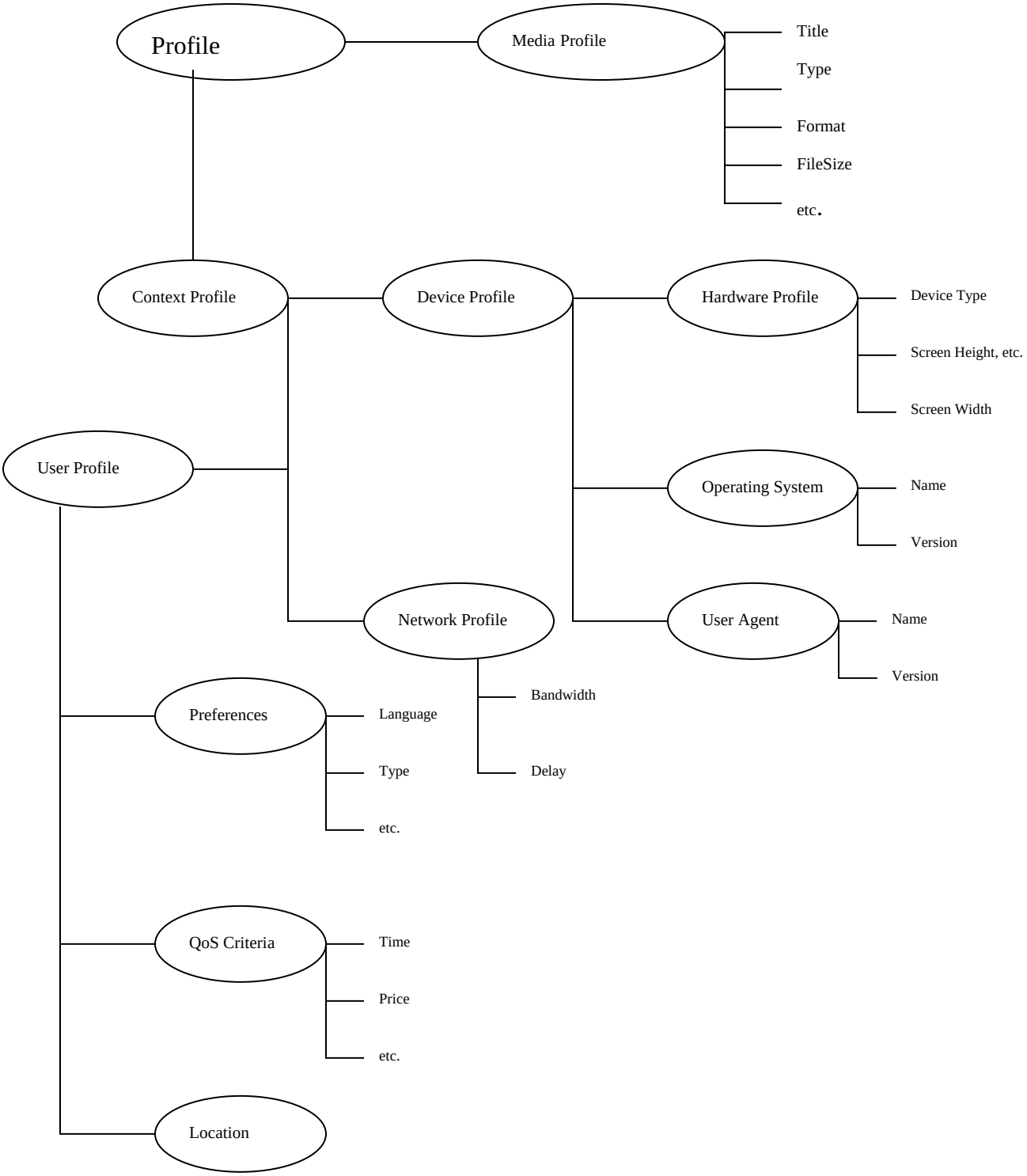



**Figure 2.1: Hierarchical Data Structure of Profiles**

## 2.2 Profile Representation Tools

In dynamic and heterogeneous computing environments such as pervasive systems, context-aware service provision is a key concept to meet varying requirements of the clients. To allow such functionality, context information must be collected and delivered to the application performing adaptation. This, therefore, requires having a common profile representation language that will be applicable throughout the entire process of context management. The need to have a common representation language has brought a couple of requirements concerning the representation format. The context profile representation should be [5]:

- *Structured:* Context profiles may represent a large volume of different context information. Structured representation of such information provides a means to retrieve/filter relevant information easily. It also eases unambiguous attribute naming, as attribute names can be interpreted context sensitively.

- *Interchangeable:* Context profiles should be interchangeable among various components of the system such as the client devices and the servers. To have interchangeable profile, the representation language should be flexible enough so as to retrieve any size of the profile upon the need. This requires a serializable representation of the context profile. In addition to transferring the whole profile, it may be required to transfer a sub-tree of the profile or a single attribute. By doing this, a profile need not be completely retransferred after the change on a single attribute.

- *Decomposable/Composable:* this feature of profile representation language enables the profile to be stored and maintained in a distributed way. For instance, we can find a default device profile from the device vendor's web site whereas the deviation of a particular device from the defaults can be stored at the device itself. This, thus, removes the

need of transporting the entire device profile from the client device to the servers.

- *Uniform*: A uniform representation of all kinds of context profiles such as device profiles, network profiles, and user profiles simplifies the interpretation during the process of service mediation and content adaptation.

- *Extensible:* As it is not possible to identify exhaustively the list of context profiles and also the future applications may need additional profile to be included, the profile representation should be extensible. Therefore, a profile representation format should make future extensions possible.

- *Standardized:* Context profiles are to be exchanged among different entities of the system that do not typically belong to the same administrative domain. For example, profiles on the device, which can be accessed form vendor's web site, should be in the same format as the user preference on the server side in order to use them together for content adaptation. Therefore, there is a strong need for a standardized representation of the context information.

Thus, profile representation tools have to meet the requirements mentioned above. We have chosen to use a profile representation language that is based on the XML language since XML satisfies most of the above constraints as discussed in the following section.

### 2.2.1 XML (eXtensible Mark-up Language)

The problem of lack of compatibility between applications is long existed. For instance, data created by or formatted for one application is hardly understood by another application. This problem was a major obstacle for interoperability among applications and hence hindered effective communication on the Internet. This, thus, has called upon having a common data format so that all applications will be able to understand and use

the data easily. In an effort to find a common and standard data format, XML was developed.

XML is a meta-language[3] officially recommended by World Wide Web Consortium, W3C. XML was meant to allow transmission of information of all flavors in a common and standard format over the Internet.

The concept of mark-up language was originated by IBM in the late 1960s to manage technical documentation [11, 12]. At that time, people in IBM developed a language for this purpose and called it a General Mark-up Language (GML). Later on, this language was adapted by International Organization for Standardization (, ISO) and was given the name Standard Generalized Mark-up Language (SGML).

Since SGML is a language for creating other more specialized languages, it is called meta-language. One of the specialized languages based on SGML specification is HTML, which is created to be used as a web language. Even if SGML is good enough for creating new languages, it suffers from its inherent complexity. For this reason people started to think of creating some other meta-language that would be easy for use. Hence, created a new but simplified version of SGML, namely XML. XML is, therefore, a descendent of SGML; and it was made to be easier for use, making it a preferable language for the Internet and for any web based services for that matter [11].

As XML is a meta-language, it has become a tool for spawning many new and specialized languages. One of such languages is eXtensible HyperText Mark-up Language (XHTML) which has been developed to replace HTML. XHTML is specialized for creating web pages in an easy way. There are also many other languages created out of XML to represent data in different domains. For example, GML for spatial data, MathML for mathematical data, CML for chemical data, etc. There are also profile description languages such as CC/PP and RDF derived from the XML meta-language.

[3] A language used to create another language

An XML document is a simple, plain text file that can be read using a regular text editor. This enhances interoperability among applications. Moreover, as XML documents are text files, they are comparatively small in size and hence can travel over the web quickly.

XML documents are built from text content marked up with text tags. Examples of such tags are, <name>, <author>, <title>, etc. These look like HTML tags, but in HTML there are limited number of tags that describe web page formatting, whereas in XML one can create as many tags as required and the tags do not describe formatting, instead, describe the type of the content they mark up.

### 2.2.2 RDF (Resource Description Framework)

The main problem in resource description is the lack of common conventions followed to effectively use meta-data. This is because of the possibility to describe the meta-data in different semantics, syntax, and structure. In response to address this problem, RDF was specified.

RDF is an XML-based meta-data description framework designed by the W3C. It provides a model for describing resources [18]. A resource is defined as any object that is uniquely identifiable by a Uniform Resource Identifier (URI*)* and that has a property (attributes or characteristics). A resource has property-types and the corresponding values. Property-types express the relationships between values and the associated resources. The collection of these properties that refer to the same resource is called a description [17, 19].

The RDF graphical description model is given by the following structure [19]:

[*Subject*] → p*redicate* → [*Object*]

where, predicate is a property-type that labels the graph edge from subject to object. The subject must actually be a resource. Object can be another RDF resource or value of the property.

Consider the sentence: *This music is played by Abraham*.

This is called a statement in RDF with three structural parts: a subject (This music), a predicate (is played by), and an object (Abraham).

Using the above RDF structure the statement can graphically be represented as,

[This music] →Played-by→ [Abraham]

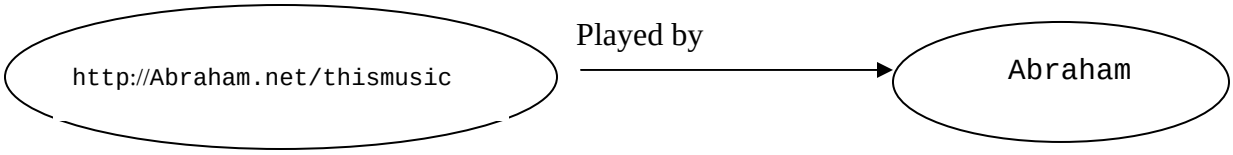



**Figure 2.2: An RDF statement**

In the above representation, the object is a string: "Abraham". But an object can also be a resource. This is shown in figure 2.3.

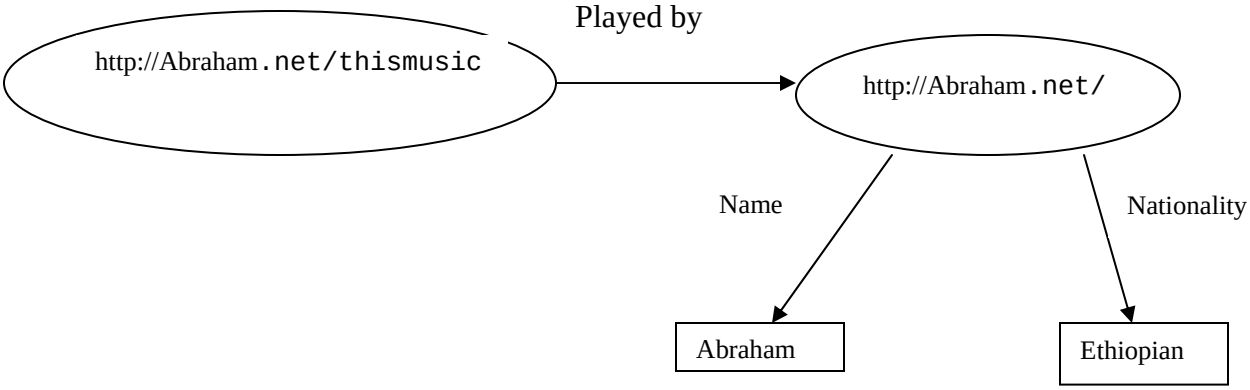



**Figure 2.3: An instance of several RDF statements combined into a single diagram.**

All of the RDF statements are expansions of the above structure [17].

The RDF representation depicted in the above figures can be serialized using the XML as shown in listing 2.1:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://schemas.Abraham.net/rdfexample/">
  <rdf:Description about="http://Abraham.net/thismusic">
    <played-by>
      <rdf:Description ID="Abraham.net">
        <name>Abraham</name>
        <nationality>Ethiopia</nationality>
      </rdf:Description>
    </played-by>
  </rdf:Description>
</rdf:RDF>
```

*Listing 2.1: XML serialization of the RDF model of figure 2.3*

The above serialization is just one of the many forms that are provided for XML expression. This diversity of serialization is somehow a challenge for implementing RDF. Despite the diversity, there is one constant in all RDF serializations - it is the element `rdf:RDF` that is used to wrap up the RDF statements.

The `rdf:RDF` element has namespace attributes that are meant to avoid ambiguities in using similar names in different contexts. RDF relies heavily on XML namespaces to resolve naming conflict. There are several element and attribute names that must be in the

namespace defined by RDF. Thus, all the RDF predicates must use a namespace to clarify their meaning, as shown in the above example [18].

In the above listing, inside the wrapper element rdf:RDF, there is a description element that indicates subject of the enclosed statements. In this example, we used the *about* attribute that points to an external resource (`http://Abraham.net/thismusic`) as subject. This subject has only one statement indicated by the predicate <played-by>.

To wrap up the statement, there should be an object of the statement. The object is another resource but it does not have an external URI (Uniform Resource Identifier) in the above example. It is represented by the embedded *`Description`* element with an *`ID`* attribute. The URI of this resource is found by combining the URI of the RDF file as a whole with the value of the ID attribute. This resource with ID "Abraham.net" in turn is found to be the subject of the two statements. The predicates of the two statements are represented by the child elements *name* and *nationality*. The objects of these statements are literals: "Abraham" and "Ethiopia".

### 2.2.3 CC/PP (Composite Capabilities and Preference Profiles)

CC/PP is a framework for describing device capabilities and user preferences developed by the W3C. It defines the basic structure of context profiles and introduces a mechanism for profile decomposability. The intended use of such profiles is for content customization. CC/PP is based upon the Resource Description Framework (RDF).

RDF is a way of representing statements, each of which contains a subject, predicate and object as it has been explained in the previous section. RDF can be serialized in many different ways, but CC/PP uses the XML serialization. The basic data model for a CC/PP is a collection of tables. The simplest form of each table in CC/PP is a collection of RDF statements with simple values [18, 19].

***CC/PP Profile Structure:*** A CC/PP profile is constructed as a 2-level profile tree with the following parts [19]:

- profile *components*, and
- component *attributes*.

***Profile components***: the top of the tree structure describes major components of the client such as hardware platform, software platform, and the application running on the client's device.

The following is a graphical representation based on the following three components: TerminalHardware, TerminalSoftware and TerminalBrowser [19].

```
[example:MyProfile]
 |
 +--ccpp:component-->[example:TerminalHardware]
 +--ccpp:component-->[example:TerminalSoftware]
 +--ccpp:component-->[example:TerminalBrowser]
```

*Listing 2.2: CC/PP Profile Components*

The corresponding XML representation of the abstact in listing 2.2 is given in the following listing:

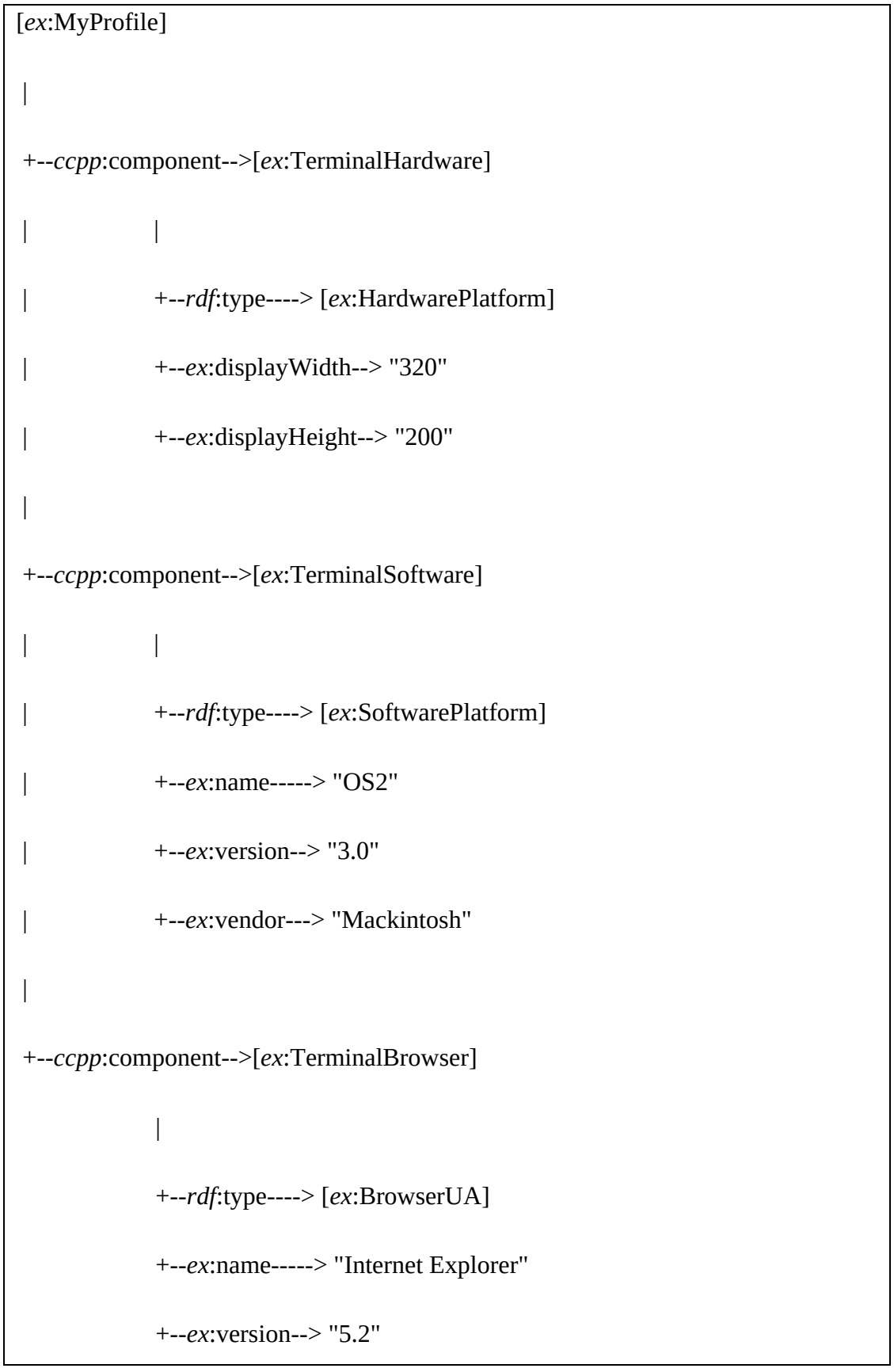
```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

        xmlns:ccpp="http://www.w3.org/2002/11/08-ccpp-schema#"

        xmlns:example="http://www.example.com/schema#">

  <rdf:Description rdf:about="http://www.example.com/profile#MyProfile">



    <ccpp:component>

      <rdf:Description

          rdf:about="http://www.example.com/profile#TerminalHardware">

        <!--  TerminalHardware properties here  -->

      </rdf:Description>

    </ccpp:component>

    <ccpp:component>

      <rdf:Description

          rdf:about="http://www.example.com/profile#TerminalSoftware">

        <!--  TerminalSoftware properties here  -->

      </rdf:Description>

    </ccpp:component>
```

```
    <ccpp:component>

      <rdf:Description

          rdf:about="http://www.example.com/profile#TerminalBrowser">

        <!--  TerminalBrowser properties here  -->

      </rdf:Description>

    </ccpp:component>



     </rdf:Description>

</rdf:RDF>
```

*Listing 2.3: CC/PP Profile Components in XML*

***Component attributes***: Each of the CC/PP component has a number of attributes using which client capabilities and preferences are decribed. The description of each component is taken as a sub-tree whose branches are the capabilities or preferences associated with that component [19]. CC/PP has specified attributes, each having a simple and atomic value, that are used to describe the capability of client's device. In the cases where there are complex values, they can be constructed as RDF subgraphs. One typical case for complex attribute values is to represent alternative values; e.g. When a browser supports multiple versions of HTML.

The following listing illustrates a hypothetical profile with inclusion of attribute values on the basic structure of CC/PP.


[ex:MyProfile]
|
+--ccpp:component-->[ex:TerminalHardware]
| |
| +--rdf:type----> [ex:HardwarePlatform]
| +--ex:displayWidth--> "320"
| +--ex:displayHeight--> "200"
|
+--ccpp:component-->[ex:TerminalSoftware]
| |
| +--rdf:type----> [ex:SoftwarePlatform]
| +--ex:name-----> "OS2"
| +--ex:version--> "3.0"
| +--ex:vendor---> "Mackintosh"
|
+--ccpp:component-->[ex:TerminalBrowser]
|
+--rdf:type----> [ex:BrowserUA]
+--ex:name-----> "Internet Explorer"
+--ex:version--> "5.2"

```
                +--ex:vendor---> "Microsoft"

                +--ex:htmlVersionsSupported--> [ ]

                                |

                  ---------------------------

                 |

                  +--rdf:type---> [rdf:Bag]

                  +--rdf:_1-----> "3.2"

                  +--rdf:_2-----> "4.0"
```

*Listing 2.4: A Complete Example of CC/PP Profile*

The corresponding XML representation of the abstact in the above listing is given by the following:

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

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

        xmlns:ccpp="http://www.w3.org/2002/11/08-ccpp-schema#"

        xmlns:ex="http://www.example.com/schema#">

  <rdf:Description

     rdf:about="http://www.example.com/profile#MyProfile">

    <ccpp:component>

     <rdf:Description

        rdf:about="http://www.example.com/profile#TerminalHardware">
```

```
    <rdf:type
        rdf:resource="http://www.example.com/schema#HardwarePlatform" />
    <ex:displayWidth>320</ex:displayWidth>
    <ex:displayHeight>200</ex:displayHeight>
  </rdf:Description>
</ccpp:component>
<ccpp:component>
  <rdf:Description
      rdf:about="http://www.example.com/profile#TerminalSoftware">
    <rdf:type
        rdf:resource="http://www.example.com/schema#SoftwarePlatform" />
    <ex:name> OS2</ex:name>
    <ex:version>2.0</ex:version>
    <ex:vendor> Mackintosh </ex:vendor>
  </rdf:Description>
</ccpp:component>
<ccpp:component>
  <rdf:Description
      rdf:about="http://www.example.com/profile#TerminalBrowser">
    <rdf:type
```

```
        rdf:resource="http://www.example.com/schema#BrowserUA" />
      <ex:name>Internet Explorer</ex:name>
      <ex:version>5.2</ex:version>
      <ex:vendor>Microsoft</ex:vendor>
      <ex:htmlVersionsSupported>
        <rdf:Bag>
          <rdf:li>3.2</rdf:li>
          <rdf:li>4.0</rdf:li>
        </rdf:Bag>
      </ex:htmlVersionsSupported>
    </rdf:Description>
  </ccpp:component>
 </rdf:Description>
</rdf:RDF>
```

*Listing 2.5: A Complete Example of CC/PP Profile in XML*

Evenif CC/PP does not meet some of the requirements of profile representation tools that have been discussed in section 2.2, we choose to use it for representation of clients' profile in the current work. This is due to the fact that currently there is no profile representation tool that meets the entire requirements. As CC/PP has only two levels for profile representation, it fails to represent the context profiles with complex struturing.

## 2.3 Communication Method used for profile exchange

Due to device and user mobility in pervasive systems, a client gets services from various points in the system. In most cases, the client's profiles (especially those that change through time) are discovered and collected during service provision. This leads to the fact that the client's profiles in pervasive systems are found distributed in the system. Thus, to manage the distributed profiles in the pervasive systems, the profile management system should make communication among its components found distributed in the system.

One of the basic requirements of distributed system is inter-process communication. The processes to communicate in profile management case are the various modules that are in charge of performing profile management tasks at various machines in the system.

Communication in distributed systems is based on a low-level message passing as offered by the underlying network. Expressing communication through the low-level message passing technique is very hard to develop distributed applications and hence, high-level communication methods have been invented [8]. Here we briefly mention the major types of high-level communication methods with their pros and cons.

One of the high-level message passing methods in distributed systems is *RPC* (*Remote Procedure Call*). RPC allows programs to make calls to procedures located on remote machines. It is aimed at hiding most of the intricacies of message passing. When a process on one machine calls a procedure on remote machine, the calling process is suspended, and execution of the called procedure takes place on remote machine. Information flow from the caller to the callee takes place in the parameters and comes back in the procedure result. From programmer point of view, no message passing is visible at all. In general, it looks like a conventional procedure call and therefore, location hiding of the procedure is achieved [8].

*Remote object invocation* is another method of message passing in distributed systems and it is an improvement on the RPC model. As it involves the concept of object, data of an object can only be accessed by invoking methods that are made available through an object's interface. In this model, interface of the object, also called proxy, is placed on

one machine (the calling machine) while the object itself resides on another machine. In order to call method of an object, the client first binds to the object. Such an invocation is known as remote method invocation or simply RMI [8].

RPC and RMI are good enough at hiding communication in distributed systems. But they set certain constraints: expecting that the server side is running at time of calling; and supporting only synchronous communication, by which a client is blocked until the request has been processed [8]. Message-oriented communication or messaging comes to resolve these problems. Basically, messaging assumes configuration of a host on sending side, communication server, and another host on the receiving side. Applications are executed on hosts, where each host gives an interface to the communication system and hosts are connected through network communication servers, which are responsible for passing and routing messages among hosts. An example of application that requires such a configuration is an electronic mail system.

Messaging, thus, does not require the receiving side to be active at the time of sending as the message temporarily stored on the servers found in between the sending and receiving sides. If the receiver is not active, the message can be kept in the communication servers. Moreover, once the message is delivered to the communication server, the sending host can continue with other activities.

In addition to the above communication methods, there are more advanced forms of communications in the distributed systems such as mobile agents and SOAP (Simple Object Access Protocol). In the following section we present description of these communication methods in relation to pervasive systems.

### *2.3.1 Communication Using Mobile Agents*

Agents are self-contained software processes or threads that can stand by themselves. Each agent has its own thread of execution which means that it can perform tasks on its own initiative. Based on their ability to move, they can be classified into two [13]:

a) Static - if they reside and remain on a single machine, and

b) Mobile - if they move from one machine to another on their own accord.

Code mobility can be seen as the capability to dynamically change the bindings between code fragments and the location in which they are executed. The concept of code mobility was begun long before. As of its inception various forms of code mobility have been identified [13]:

i) *Code on demand*: This is essentially made by downloading the executable code from the server machine to the client environment upon the request of the client. One typical example of this is the download of JavaApplet code in a web browser.

ii) *Remote Evaluation*: This allows the clients to execute a code close to the resources at the server side, thereby reducing network interaction. For example, SQL can be used to perform queries to a remote database.

iii) *Mobile Agent*: mobile agents can explicitly relocate themselves across the network. Examples of such a system are Telescript and D'Agents.

Currently, mobile agents are the most commonly used form of code mobility [13]. They provide very appealing, intuitive, and apparently simple abstraction.

The traditional client-server and the message-based architectures provide location transparency to treat local and remote resources in the same way. That is, from the user's point of view, all information is stored locally. In contrast to the traditional technologies such as *remote evaluation*, the mobile agent migrates to remote hosts where information is stored so that the user's request can be executed on remote host. The advantage of the later approach is that it reduces network traffic by keeping the data to be executed where it is. It also enables agents to make decisions or fulfill previously designed operations in an autonomous way on the remote site, even if the home host becomes temporarily unavailable. This feature, called mobile logic, gives flexibility to the design of distributed systems [14].

Local interaction and mobile logic also lead to the achievement of load balancing in the whole distributed system. The agents move from host to host to execute the user's requests and return the result to the host where they were created. They act autonomously according to the mobile logic they incorporate [14].

#### 2.3.1.1 Mobile Agents and Pervasive Systems

Mobile agents can play an important part in pervasive systems [15]. Pervasive computing is meant to be incorporated in various devices such as computers, cars, entertainment units, appliances, etc., to support many aspects of work and everyday activities. In order to provide appropriate service using variety of devices on the basis of 'anytime and anywhere', the system should: be aware of its environment and available resources, detect the changes in the context, and also be able to adjust/adapt its functionality to the changes based on the profile information. One of the applications of mobile agents in pervasive systems can be profile management. In profile management, the agent's role will be to maintain the user profile: constantly adding, deleting and modifying profile information based on new information.

Anything that can be done with mobile agents can also be done with conventional software techniques such as remote procedure call (RPC) and messaging [16]. Mobile agents, however, have several appropriate uses in pervasive systems where handful constraints exist.

The first use of mobile agent is its ability to make dynamic installation of code to extend functionality of existing system. This would avoid the user interaction with the system to install code. The second use is its ability to make disconnected operations. One typical advantage of disconnected operation is to save battery life. Suppose you are using personal digital assistant (PDA) for which major consideration is battery capacity and hence connection time. The agent can run on your PDA, where you can interact with and instruct it. But, when you turn off your PDA, you may still want the agent continue functioning and the agent actually does this by migrating to the other machine in the system. Finally, the most frequently proposed use of mobile agent is its ability to migrate

to servers to do customized searching, particularly when the servers lack the necessary procedures to perform the desired processing themselves.

Though mobile agents are the most powerful form of code mobility, they are not warmly accepted in Internet environment when the other forms of code mobility are in wide-spread use. Some of the down sides of the mobile agents are identified and listed below [13]:

- *Mobile agents do not perform well.* According to [13], some evaluations have shown that mobile agents, in the general case, provide worse performance than other mechanisms, such as remote evaluation and remote procedure invocation due to the overheads introduced as a result of moving both the code and the state of an agent.

- *Mobile agents are difficult to design.* Most applications in distributed environment can be designed using the well known software architecture concepts such as modularization and modeling of inter-component interactions. In mobile agent approach, it is difficult to identify the components that interact with each other and how to model the interaction.

- *Mobile agents are difficult to develop.* This is because the code needs to be implemented so that it runs in unpredictable environments, possibly interfacing with other mobile agents.

- *Mobile agents are difficult to authenticate and control*. As mobile agents are supposed to travel through and access information from various environments, access control mechanisms are required upon entering an environment. The problem is that there are many identities associated with a mobile agent. Hence, it is not clear which identities should be authenticated and how the access control mechanisms should take into account this information.

- *Mobile agents can be brainwashed*. This is due to their movement across multiple hosts to complete their tasks. During their movement they are vulnerable to a number of attacks.

- *Mobile agents cannot keep secrets*. As agents are autonomous and hence can interact with their environment and other agents, secret information can not be effectively concealed. This is, therefore, a threat for confidential transactions (e.g. signing a contract), to transmit using agents.

Due to the above drawbacks of mobile agents, we do not recommend to use mobile agents for communication in the current work. Especially, they deteriorate performance of the system due to the added overheads in moving the code and state of the agent as described above.

### *2.3.2 Communication Using SOAP (Simple Object Access Protocol)*

SOAP is an extensible XML messaging protocol and it forms a foundation for Web Services[4]. SOAP is a high-level protocol that defines only the message structure and a few rules independent of the underlying transport protocol. Currently, Hyper Text Transfer Protocol (HTTP) is the most frequently used protocol for SOAP messaging.

There are three basic parts included in SOAP specifications: the SOAP envelope, a set of encoding rules, and a means of interaction between request and response.

**i. The Envelope**

The SOAP envelope is an XML document. It provides information about the message that is being encoded in a SOAP payload, including data relating to the recipient and sender, as well as details about the message itself. The envelope contains a header (optional) and a body (mandatory). The body part is always intended for the final recipient of the message, while the header entries may target the nodes that perform intermediate processing [20]. For example, the header may be used to provide digital signatures for a request contained in the body. In this case the authentication and authorization server is supposed to process the header entry that can be done independent of the body. After validation, the rest of the envelope would be passed onto the SOAP server where body of the message would be processed.

[4] Functionality that can be programmatically accessible via the Web

A SOAP envelope can also include the encoding style, which assists the recipient in interpreting the message. The following listing, Listing 2.5, gives sample SOAP envelope.

```
<soap:Envelope

   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"


soap:encodingStyle="http://myHost.com/encodings/secureEncodi
ng">

  <soap:Body>

    <article xmlns="http://www.ibm.com/developer">

      <name>Soapbox</name>

      <url>

        http://www-106.ibm.com/developerworks/library/x-
soapbx1.html

      </url>

    </article>

  </soap:Body>

</soap:Envelope>
```

*Listing 2.5: The SOAP envelope*

### ii. Encoding

The encoding part of the SOAP specification brings simple means of encoding user-defined data types. With SOAP, XML schemas can be used to easily specify new data types. And those new types can be easily represented in XML as part of a SOAP payload.

Due to the SOAP ability to integrate with XML schema, it is possible to encode any data type in a SOAP message.

### iii. Invocation

SOAP enables to build applications by remotely invoking methods on objects. It removes the requirement that two systems must run on the same platform or be written in the same programming language. A SOAP package uses XML, a text-based syntax for making method calls.

Therefore, SOAP is a means for web service applications to make remote method invocation without a need for having an agreement between the requesting application and the remote object.

***RPC through SOAP***

SOAP decrees a set of rules that enables clients and servers to do remote procedure invocation using SOAP as a communication framework. It can work well as an RPC-type protocol and object serialization is a mechanism that gives SOAP-RPC its vigor.

A SOAP message is fundamentally a one-way transmission between SOAP nodes, i.e. from a SOAP sender to a SOAP receiver. Even if SOAP messages are one way transmissions, they can be combined to implement request/response mechanisms. This is how RPC is implemented using SOAP. To do this a few conventions must be followed:

- Request and response messages must be encoded as structures. For each input and output parameter there must be an element with the same name as the parameter. This is illustrated in the following sample SOAP message which has only body part of the envelope.

***Request:***

```
<SOAP-ENV:Body>

<m:getProfile xmlns:m="some-URI">

<name>DEF</name>

</m:getProfile>

</SOAP-ENV:Body>
```

***Response:***

```
<SOAP-ENV:Body>

<m:getprofileResponse xmlns:m="some-URI">

<name>PDA</name>

</m: getprofileResponse>

</SOAP-ENV:Body>
```

(where getProfile is a method name on the remote machine.)

*Listing 2.6: Remote method invocation with SOAP.*

The receiver responds to the request by appending ***Response*** to the method name that has been issued. In the above example, getProfile is the method name issued by the sender; and after invoking getProfile method, the receiver sends back the result to the sender by packing in the element getProfileResponse.

In order to do RPC, a lower-level protocol like HTTP is needed.

To invoke a SOAP RPC, the following information is needed [21]:

1. The address of the target SOAP node.

2. The procedure or method name.

3. The identities and values of any arguments to be passed to the procedure or method together with any output parameters and return value.

4. A clear separation of the arguments used to identify the Web resource which is the actual target for the RPC, as contrasted with those that convey data or control information used for processing.

5. The message exchange pattern which will be employed to convey the RPC, together with an identification of the so-called "Web Method" to be used.

6. Optionally, data which may be carried as a part of SOAP header blocks.

In the current work, our aim is to develop clients' profile management system so that to supply profiles for content adaptation in pervasive systems. Specifically, we intend to integrate our work into the content adaptation/delivery framework known as DCAF, details are found in section 4. As DCAF is service-based content adaptation/delivery framework that pre-supposes existence of web-service, our current work also requires web-based communication in order to interface with various components that exist in the DCAF. Thus, we have found that SOAP technology is an appropriate choice for communication among the various profile managing modules that are found distributed in the local servers and client devices to manage the distributed profiles.

# CHAPTER 3

# 3 Related Work

**Overview of [Agostini et al, 2003] Work "Towards Highly Adaptive Services for Mobile Computing"**

This section describes a work in the area of profile management and compares its approaches to the problems of this thesis.

The work of [Agostini et al, 2003] proposed an architecture enabling integrated representation and handling of profile information as well as a mechanism to resolve conflicts in description. It focuses on the representation and management of distributed profile data.

For the management of profile data, three main entities involved in the task of building an integrated profile are identified: the user, the network operator, and the service provider. As profile information is subject to privacy restriction, the management tasks are distributed to three managerial modules, each associated with one of the above three entities involved in the task. Thus, the distribution is mainly based on privacy concern. For example, the user may not want untrusted parties to gain access to his personal data. Similarly, the service provider may want to keep proprietary data that are not intended to be shared with other entities due to privacy or business reasons. According to this work, all client profiles that encompass both the device profiles and the user profiles are made to be stored on the user side in one repository. The general architecture of the work is shown below:

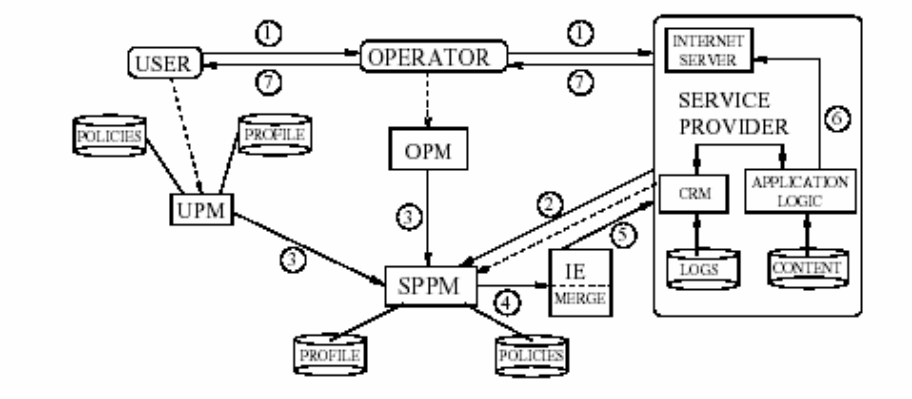



**Figure 3.1: Architecture Overview**

The profile managing modules in the system are User Profile Manager (UPM), network Operator Profile Manager (OPM), and Service Provider Profile Manager (SPPM).

i. The UPM stores and manages information related to the user and his devices. These are client's profile and include, among other things, personal information, user's interests, context information, and device capabilities.

ii. The OPM stores the attributes describing the current network context such as location of the client, connection profile, and network status.

iii. The SPPM is responsible for managing service provider proprietary data including information about users derived from previous service experiences.

This system may be suitable to the Internet environment where the devices that are involved in the system are capable of storing large amount of data and also where the network bandwidth is reasonably high.

Unlike the Internet case, in pervasive systems, the client devices are often too small to accommodate the entire profile of the client. The centralized approach also implies a single point of failure as well as performance bottleneck. In addition to this, as the network bandwidth in pervasive systems changes frequently and is limited, the profiles should be placed in such a way that communication is minimized. Therefore, centralized client's profile management proposed by this work is not appropriate for pervasive

systems. This, in effect, calls upon devising non-centralized means of managing client's profiles.

This thesis work focuses on the non-centralized way of client's profile management; i.e. distributed approach, in order to alleviate the problems mentioned above.

For the representation of profile data, [Alassandra Agostini et al, 2003] explored profile representation language. A common profile representation format is required as the profile is managed by different entities that are found under different administrational domains. For example, the profile representation language at user side should have the same format as that of network operators. They thus, chose to adapt composite capabilities/preference profile (CC/PP), which was designed to represent profile data in a standard format. The structure and vocabularies in CC/PP are aimed to represent the capabilities of the client's device and preferences of the user. CC/PP profiles are a set of components each one containing attributes with associated values. These components and attributes should refer to the CC/PP vocabularies where they are defined. The vocabulary is an RDF schema that defines CC/PP components and attributes, their semantics, and allowed values.

# CHAPTER 4

## 4 The Adaptation/Delivery Infrastructure

This section discusses the infrastructure for content adaptation/delivery system, which is developed by Girma Berhe at LIRIS (Laboratoire d'Informatique en Images et Systèmes d'Information) – INSA, Lyon. The overall service-based adaptation/delivery system is known as Distributed Content Adaptation Framework (DCAF) [24].

The DCAF is composed of a number of distributed components that perform various operations for the purpose of content adaptation and delivery. The followings are the major activities performed by the DCAF components:

- Handling communication with the client: accepts the clients' requests for content and provides the requested content
- Storing and providing access to multimedia content.
- Collecting and analyzing various profiles to determine the appropriate type of adaptation to be performed on a certain multimedia content.
- Carrying out the necessary adaptation based on the results of the analysis.

The adaptation operations that are performed by DCAF are called service-based content adaptations. In a service-based adaptation, the adaptation operation is performed on the fly by the third parity service providers. Service-based content adaptation distributes the adaptation processes so that the overall service provision will be enhanced. As per the type of adaptation to be made, an appropriate adaptation service provider is contacted. For example, an adaptation service provider that does only language translation can be used whenever there is a content to be translated. As this service is provided by the server that has specialized in only language translation, it is more efficient than any other server that does additional tasks.

Our aim in the current work is to manage and provide clients' profile for the purpose of content adaptation. Hence, we used DCAF as a reference model in proposing an architecture for clients' profile management system. The following is illustration of the framework and its various components.

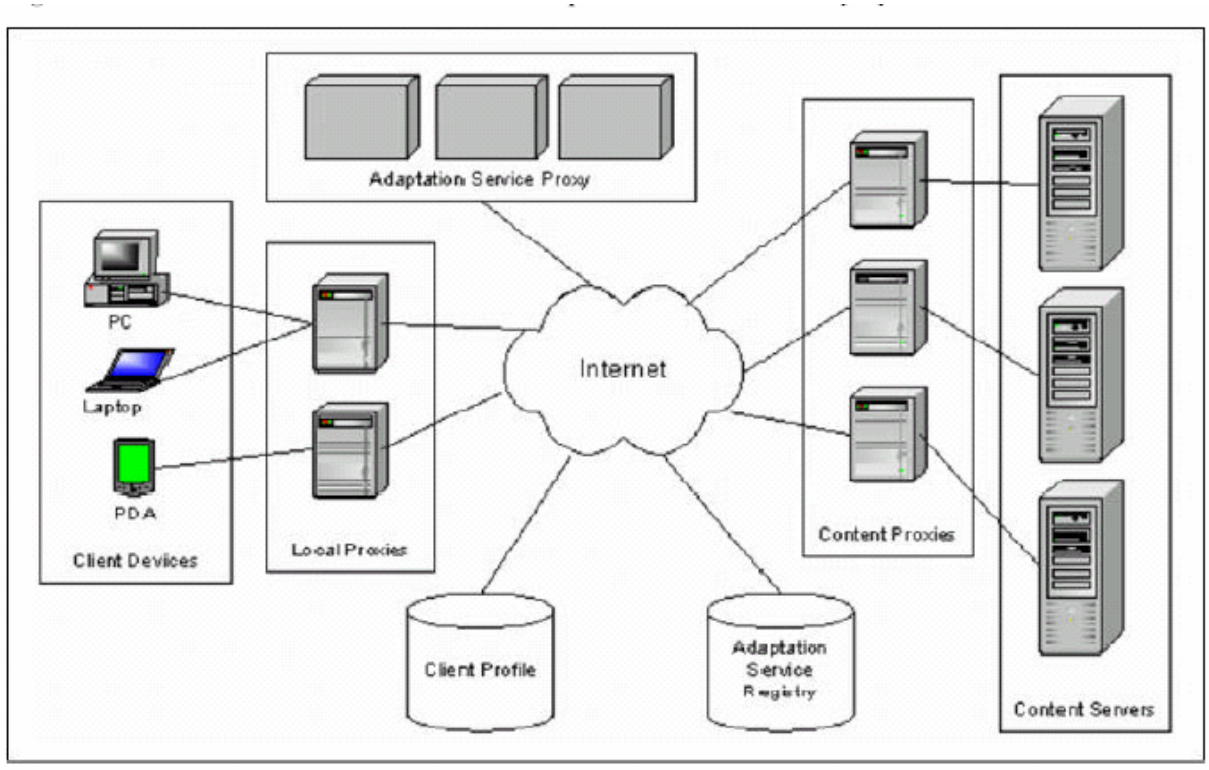



**Figure 4.1: Service-based distributed content adaptation infrastructure [2]**

### *Local proxy (LP)*

The tasks performed by this proxy are: retrieving client's requests and profiles, analyzing client and content/media profile, planning adaptation strategy, integrating and forwarding the adapted content to the user. It also caches adapted content for future use. To perform these tasks a module called CNAM (content negotiation and adaptation module) is developed by Girma Berhe, et al., [2]. It is responsible for creating an optimal adaptation plan according to the delivery context such as client's profile.

***Adaptation Service Registry (ASR)***

The adaptation service registry is a server that allows for the look up of adaptation service providers' interfaces. It stores service profile such as service type, location, supported media formats, network connection, price, status (on/off), and performance. When CNAM needs to make adaptation, it first contacts this server to discover available adaptation service providers. Based on the adaptation plan it has performed, it selects one of the available adaptation services.

***Adaptation Service Proxy (ASP)***

The adaptation service proxy provides adaptation service for content delivery. According to the service based content adaptation, content adaptation is performed by the third party service providers, which are accessed through ASPs. Thus, after getting service profile for the required adaptation from the adaptation service registry, the CNAM communicates the adaptation service proxy to get the content adapted.

***Content Proxy (CP)***

The content proxy provides access to the content server, formulates the user's request to source format, performs generic adaptation and caches generic documents. It also manages and provides content/media profile to local proxy and does a general adaptation such as anonymity (for security reason).

All the above components interact with each other to perform the content adaptation before delivery to the client. When a client sends a request, the local proxy gathers profiles from their respective repositories and, based on the profile, makes negotiation with the user to filter the request. The new request is forwarded to the content proxy that is supposed to retrieve and deliver the content back to the LP. After receiving the content, the LP requests the Adaptation Service Registry (ASR) for list of adaptation services and selects the one that meets the adaptation plan. After selecting one of the adaptation service based on its current need, the LP forwards the data to the Adaptation Service Proxy that makes adaptation on the given content and returns the result back to the LP. The LP, finally, delivers the adapted content to the client.

# CHAPTER 5

# 5 Design of Profile Management System

In this chapter, we will first present the design issues that should be taken as input to propose client's profile management system and then discuss, with respect to the design issues, various alternative sites for storage of profile data. In the subsequent two sections, we will detail the proposed solution for the management of the profile data. Finally, we will describe interaction of the various components that are identified for the management of profile data.

## *5.1 Design Issues*

To design the client's profile management system for pervasive systems, there are two main parameters to be considered: characteristics of client's profile and constraints of pervasive systems.

### 5.1.1 Characteristics of the client's Profile

Clients' profile is mainly composed of information of the users and the client devices (see section 2.1 for details). Each of these profile data has its own characteristics. The profile characteristics are part of the factors that determine where to allocate various profile data in the system and also how to make the overall profile management.

- *Confidentiality*: Some of the client's profile is confidential and hence, requires secured place for storage. For example, personal data of the user and certificates are among the confidential data.

- *Dynamism*: Client's profile such as user location and some of the preferences change dynamically. As pervasive systems support mobility, location information of the user changes frequently. Some of the user preferences are also dynamic and continuously built out of the interaction between the client and the system to have a better knowledge (understanding) about the user's

interest. The more the information discovered on the preferences of the user, the better the service that can be provided to the user. As a result, continuous profile extraction and updating is needed to gain a better knowledge about the user's interest.

- *Availability*: In pervasive systems, the contents should be adapted before delivery due to the constraints on device capability, network bandwidth, and user preferences. This implies that adaptation is one of the important activities in service provision of the pervasive systems. For adaptation to take place there has to be profile information available since adaptation is made based on the context of the system. Hence, the profile management should be designed to ensure availability of such information in case of partial failure of the system.

- *Incompleteness*: Some of the profile data such as user preferences are not required to be complete since profiles of this type are continuously built during user interaction with the system. Even if part of such information is missing, the service can be provided. For example, if the user preference regarding color is not found, the system can provide service using a default color.

- *Precision*: Some of the client's profile such as device location needs to be exact while some other profile information such as user preferences need not be precise. For example, type of movie that the user prefers need not be indicated precisely. Thus, it is not necessary to have the latest version of such information.

### 5.1.2 Constraints in Pervasive systems

Pervasive systems are characterized by enabling ubiquitous computations that allow computation from anywhere and anytime using any computing devices that surrounds a person [8]. In enabling ubiquitous computation, there are handful of constraints to be dealt with. The followings are only the main constrains that have major impact on the system that will be proposed.

- *Device Heterogeneity*: Pervasive systems encompass a wide range of devices with varying degree of capacity in terms of size, processing power, and storage space. As a result, some of the devices in the system are not capable of storing large amount of data and they are also weak in their processing capabilities. This, therefore, poses constraints on the amount of data to be stored as well as on the type of computation to be made on the client devices.

- *Low network bandwidth*: the network bandwidth between the client devices and the service providers is weak and unreliable as the connection, in most cases, is made through wireless links. The weak and intermittent connection thus demands to limit the data transfer between the client device and the server.

- *Mobility*: As pervasive systems are meant to provide services at anytime basis, mobility of the client devices is inherent to the systems. Mobility of the devices makes possible for the user to get service through different local servers in the system. Due to this, profile of the client may be collected at different servers. Thus, client's profile in pervasive system is inherently distributed.

The characteristics of profile information and the major constraints of pervasive systems discussed above have implication on the selection of profile data storage.

## *5.2 Evaluation of Profile Storage Sites*

In this work, the overall purpose of maintaining profile information is to be able to provide the required profile for adaptation plan and thereby for content adaptation. According to the service-based adaptation infrastructure, which is discussed in section 4.1, adaptation plan is made on the local proxy servers. Thus, the client's profile is needed on the local proxy servers. Since putting data near to its user increases performance, the system would benefit if the entire profile information is kept on the local proxy servers. But, there are issues such as profile characteristics and constraints of

pervasive systems, which are discussed above, that are worth considering in determining the storage sites for client's profile. In this section, we identify and evaluate various alternatives for profile storage.

### 5.2.1 Storing on Client Devices

This choice is particularly good for some of the profile information that require security and also for those that change regularly on the client side. But, it has a number of limitations, such as:

- *Limitation in storage capacity*: The client's devices in pervasive systems can be of any size with varying capability in terms of storage space and processing power. As there can be devices incapable of storing large size of data, keeping the entire profile of the client on the client device is not feasible. For example, if we take cellular phone, it may not be able to store the whole profile of the client.

- *Limitation in network bandwidth*: When the entire profile data is kept on the client device, it requires movement of the profile data from the client device to the servers. This is because, as it has been said above, profile data is needed for adaptation plan that takes place on the local proxy server.

- *Dynamism of profile*: There are profile data that change on the server side. For example, some of the user preferences such as music or movie preferences of the user may change on the server side as they are discovered during user interaction with the system. As it has been discussed in section 2.1, dynamic profiles are studied and extracted from the user's activities such as, for example, from what he/she usually retrieves, visits, etc. When such information is kept on the client device, it will result in too many communications that arise due to update request since it is extracted on the server side.

### 5.2.2 Storing on Local Proxy Servers

Server side storage is good for some of the client's profiles that regularly change on the server side. When there are newly extracted facts, the update is made on the local proxy servers. Again, as the entire profile information is needed only on the local proxy servers, there will not be profile transmission between the client devices and the servers. As a result, the low bandwidth between the client devices and the servers is not used for the transportation of profile information. But, on the other hand, keeping the whole profile on the servers has the following drawbacks:

- *Discloses confidential information*: As client's profile involves confidential data such as user's personal data, certificates, etc., keeping the whole profile on the server side risks data security. This is due to the fact that servers might be accessed by both loyal and non-loyal parties.

- *Generates too many communications*: Profile information such as location of the client device is dynamic and in most cases originates on the client's side. If such profile is kept on the servers, whenever the device changes location, the update request should propagate to the servers to refresh the location profile. When such changes become frequent, too many communications are generated between the client device and the servers introducing communication overhead.

### 5.2.3 Storing on a Separate Profile Server

This approach requires a separate profile repository to store profiles of all existing clients. The profile server, thus, can be accessed through the network from anywhere. Since profile location in this case is fixed, it addresses the problem of user and terminal mobility. However, it has the following limitations:

- *Single point of failure*: As the entire profile of each of the clients is kept on a single repository, failure of the profile server results in inaccessibility of the profile information. Thus, this approach does not meet availability

requirement of profile data that has been discussed in section 5.1.1. However, this problem can be slacken by creating backup servers; hence it may not be major problem of the approach.

- *Performance degradation*: As there is only a single server holding the entire profile, whenever profile is required for service provision to various clients, access is made from that server. Thus, the profile server becomes a bottleneck that degrades performance of the entire system. This problem may be reduced by caching the profiles at the local proxies. But, this is not recommended as there are dynamic profiles that change regularly. If profiles are cached at local proxies, recharging the cache so that to maintain consistency will become considerable overhead to the entire system.

- *Lack of scalability*: When more clients are needed to be supported, the centralized profile server becomes a bottleneck inhibiting expansion of the system.

## *5.3 Proposed Architecture for Client's Profile Management System*

### 5.3.1 Proposed Profile Placement

As seen above, the three alternatives for profile storage are not viable for pervasive systems. And we have not come across any work that proposes to distribute the profiles at both sides. However, we found that it is interesting to fragment and distribute the profiles on the client devices as well as on the servers (local proxy servers) depending on the characteristics of the profiles. By doing this, we attempt to meet the conflicting requirements such as:

a. Keeping track of dynamic profile information without requiring too many communications. The dynamically changing profiles are kept locally where they are generated/collected. For example, the location information, which is extracted on the client side, is stored on the client device, whereas user preferences can be

stored on the servers as they are extracted there. This, thus, limits the number of message exchange between the client and the server.

b. Preserving confidentiality of the profile information without requiring too much space on the client side. Some of the sensitive client's profiles that require security are stored on the client side where only the user will have full control on the data. As only small portion of the profile information is to be stored on the client device, space requirement is affordable.

To formalize the fragments, let us denote the client's profile by CP. CP is, thus, a set of profile on the server side, CPs and those that reside on the client side, CPc.

CP = { CPs, CPc }

*Profile information to be stored on the client side (CPc) includes:*

- Part of the personal information that may be subject to privacy.
- Frequently changing information (on the client side) such as location of the user and the terminals.

*Profile information to be stored on the server side (CPs):*

- Device profile such as hardware description of the device, description of the operating system, and description of the agents running on the client device.
- Part of the personal information that does not require security.
- Part of the user preferences that do not require confidentiality.

### 5.3.2 Management of the Fragments

As per the above profile fragmentation, two profile managers, with different functionalities, are identified to perform the entire profile management tasks. The first manager, called client side profile manager (CSPM), resides on the client machine and is responsible for the management of profile data on the client side, CPc. The second

manager, called server side profile manager (SSPM), resides on the server machines (local proxy servers) to manage the profile data stored on the local servers, CPs.

Though the above fragmentation has addressed some of the problems with profile management in pervasive systems, there are more issues to be considered. These issues are discussed in section 5.1 and include availability requirement of the client's profile, scalability requirement of the system, and the constraints imposed by mobility of the client's device, etc. To deal with these issues, the profile information on the server side is in turn distributed, using both replication and fragmentation, on various local proxy servers. The distribution has the following advantages:

- *For fault tolerance*: Since the profile data is not deposited centrally, in the cases where some of the servers are down, the system will continue to function. That is, there will not be a single point of failure due to redundancy in profile distribution. This thus ensures the availability requirement of profile information.

- *For load balancing*: As the profile data are distributed and managed on various servers, the management tasks are also distributed and carried out by different servers. This increases the performance of the system as it avoids having a single server to carryout all the computations related to profile management.

- *For scalability*: As the client's profile is distributed on various local proxy servers, a new client can easily join the system without creating significant overhead on the performance.

- *For increased performance*: while distributing, the profile data can be placed near to its user to increase performance of the system.

Profiles on the server side exhibit two different characteristics with regard to frequency of changes. Hence, based on the degree of changes, they can be classified into two: those that exhibit dynamism and those that are relatively stable.

Thus, CPs is a set of dynamic profile, DP, and static profile, SP. These two classes of profiles are stored separately so as to use different distribution techniques for each. Detail on the classification is discussed in section 5.4.

CPs = {DP, SP}

The following figure, Figure 5.1, shows architecture of the client's profile management system that is integrated into service-based content adaptation infrastructure. In this architecture, separate profile repository is omitted, which was part of the original infrastructure described in section 4.1. This is because according to our proposal, profile information is going to be kept in local proxy servers and client devices. Hence, profile management modules are integrated as part in local proxy servers and the client devices, as shown below.

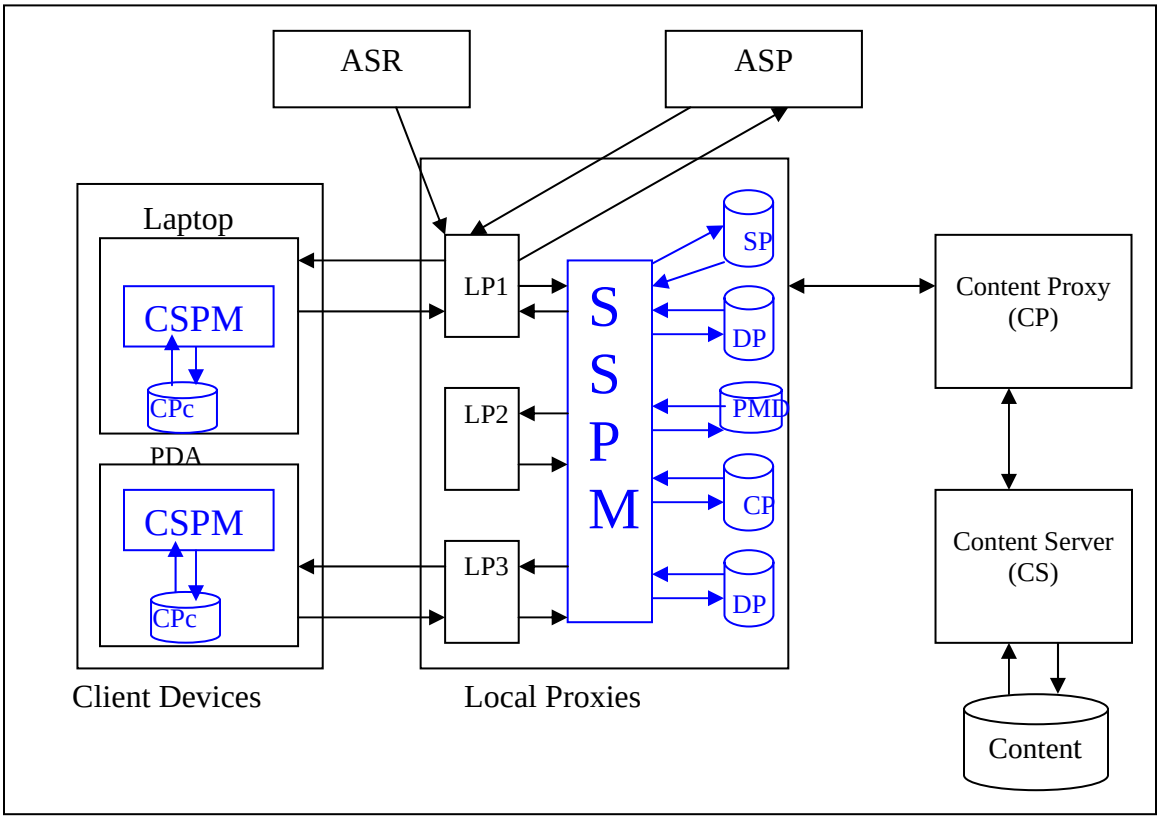



**Figure 5.1: the Proposed Client's Profile Management Architecture**

In the architecture, there are two profile managers: *CSPM* and *SSPM*. Here, we present discussion on the functionalities of the two profile managers.

#### 5.3.2.1 Client Side Profile Manager (CSPM)

CSPM is profile managing software consisting of two modules: *User Interface (UI)* module and *Profile Manager (PM)* module. *CSPM* is located on the client devices as shown in figure 5.2. *UI* module provides interface (set of forms) through which the user can sign up, sign in, enter initial profiles, update existing profiles, and delete unwanted profiles. *PM* is responsible for maintaining part of the client's profile kept on the client device. It has functionalities such as creating new profile, updating the profile, removing unwanted profile, and retrieving the required profile from the repository. For each of these functionalities, it provides an interface to the external components such as *UI*. For example, one of the functionalities is retrieving profiles from the profile repository. When a user signs in through the *UI*, the system has to get the client side profile from the repository. Thus, for profile retrieval, the *PM* provides an interface with the following general format:

***getProfile(client_Id) : Profile***

*Where*,

> ***client_Id*** is a sign-in id of the user that is used to identify the profile to be retrieved.
>
> ***Profile*** is a document containing profile of the user. If profile of the user does not exist, it returns NULL.

Once the *UI* gets a profile document of the user, it sends the document along with the login name of the user to the local proxy server through the underlying network. For communication between the various components of the system, we use *Simple Object Access Protocol (SOAP)* message passing. *SOAP* is an extensible *XML*-based messaging protocol; details are found in section 2.3.

The *CNAM*, which is found in the local proxy, receives the message that has been sent from the client device. As the profile received from the client side is just part of the entire profile, the *CNAM* requests the server side profile manager, *SSPM*, for the remaining client's profile. The complete functionality of *CNAM* and all other components engaged in content adaptation and delivery is discussed in section 4.1.

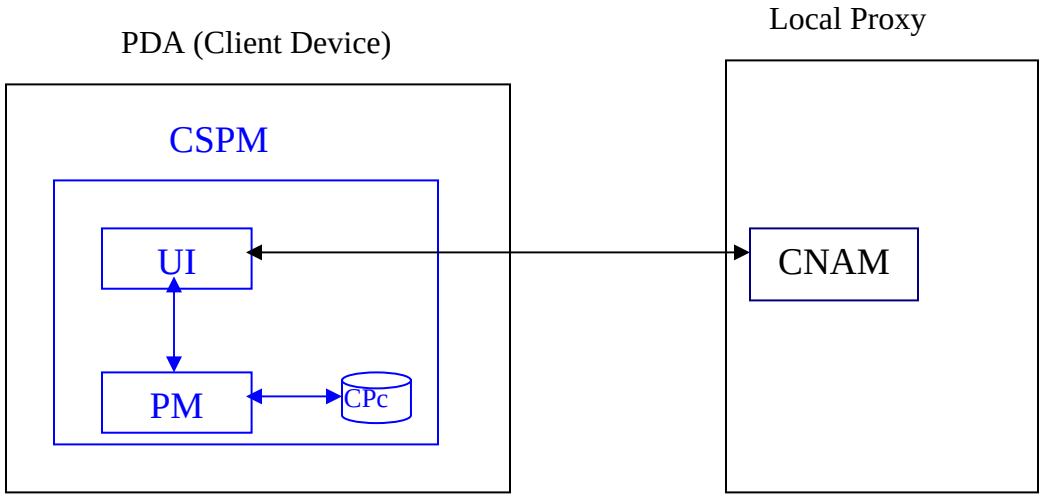



**Figure 5.2: The Client Side Profile Manager (CSPM).**

In addition to the above interface, PM also provides the following interfaces to perform management of server side profile.

- To create profile document for a new subscriber:

  **createProfile(client_Id, attrib_list,value_list):state**

  Where, **client_Id** is a unique identifier of the client's profile to be created. This identifier will also be used to find a particular profile document in the system. **attrib_list** is an array of attribute names of the client's profile for which values have been supplied. **value_list** is an array of values for the corresponding attributes in *attrib_list*. **state** returns either *true* or *false* depending on the status of the operation. It becomes *true* if the operation is successful; *false* otherwise.

- To update the profile when changes are issued:

  **updateProfile(client_Id, attrib_list,value_list):state**

Where, **client_Id** has the same meaning as in *createProfile( )* above. **attrib_list** is an array of attribute names for which changes have been made and **value_list** is an array of values for the corresponding attributes. **state** is a Boolean flag that returns true or false value depending on the result of the operation. It becomes true if the update is successful; and it becomes false otherwise. For example, if the client has not sign up yet, it returns false as it is not possible to update profile of non-existent client.

- To remove profile of a client:

  **deleteProfile(client_Id):state**

  Where, **client_Id** and **state** have the same meanings as in *createProfile()* above. **state** becomes false if the deletion is not succeeded, in case for example, when the specified client does not exist.

#### 5.3.2.2 Server Side Profile Manager (SSPM)

SSPM is server side profile managing component containing three modules: *Profile Manager (PM)*, *Metadata Manager (MDM)*, and *Name Resolver (NR)*. This component is deployed in local proxy servers to manage profile data, which have been distributed on the local servers, in a transparent way, as shown in figure 5.3.

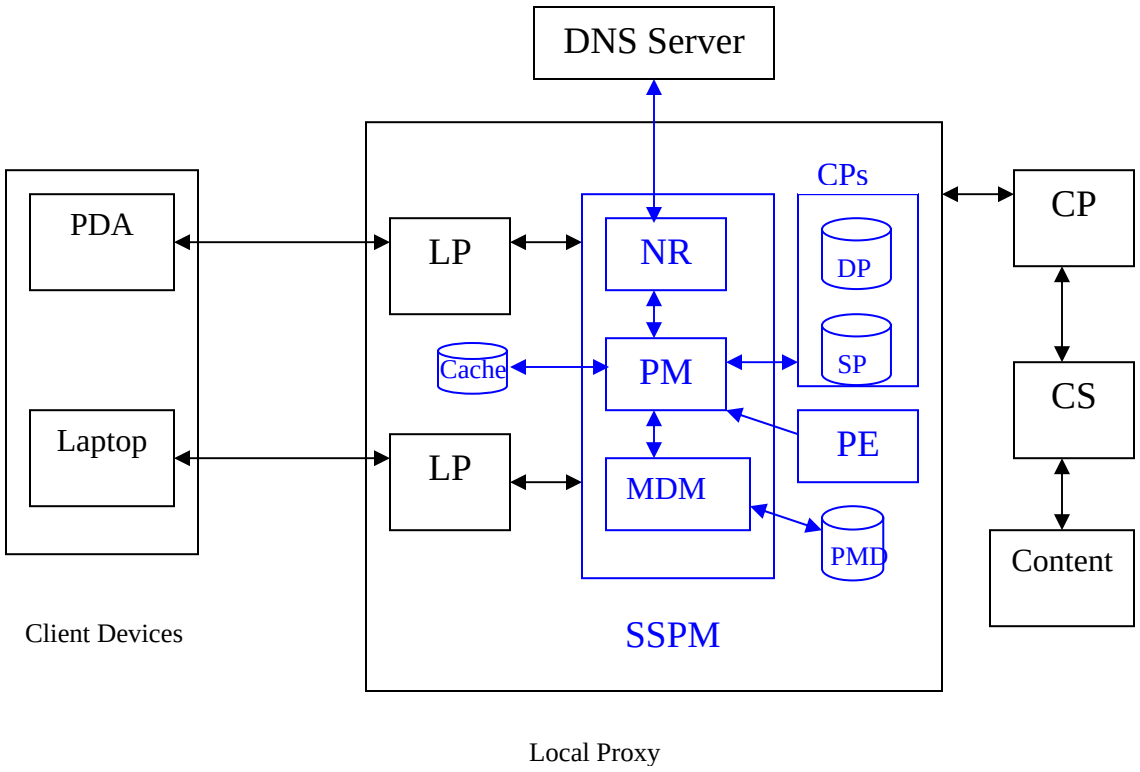



**Figure 5.3: Server Side Profile Manager (SSPM) and its interaction with external components**

Functionalities of the three modules included in the SSPM are described below.

**a. Profile Manager (PM)**

The *PM* is responsible for creating new profile, updating profile, removing profile, and retrieving profile. It provides an interface for the external components that are in charge of content adaptation or performing adaptation plan such as *CNAM*. The *PM* also provides an interface to the profile extractor, *PE*, so that it would be able to get newly discovered profiles by *PE*. In addition, the *PM* is responsible for caching profile metadata as well as client's profiles collected from various sites. The interfaces provided by *PM* to perform the mentioned functionalities are discussed below.

*i. To update profile:*

**setProfile(client_Id, attrib_list,value_list):state**

Where, **client_Id** is a unique identifier of the client that is used to identify the client's profile in the entire system. **attrib_list** is an array of attribute names for

which changes have been made and **value_list** is an array of values for the corresponding attributes. This interface is provided to the external component such as PE that supplies new profiles. After receiving the profile, the PM checks whether profile of the client already exists in the current local server. If the client's profile is not found locally, the *PM* creates a profile repository for the client by invoking *createProfile( )* method. If the repository already exists, it updates the existing content by invoking *updateProfile( )* method.

ii. *To create a new profile document:*

**createProfile(client_Id, attrib_list,value_list):state**

*Where,* **client_Id, attrib_list,value_list** and **state** have the same description as in *setProfile( )* method discussed above.

iii. *To update profile:*

**updateProfile(client_Id, attrib_list,value_list):state**

*Where,* **client_Id, attrib_list, value_list** and **State** have the same description as in *setProfile( )* method discussed above.

iv. *To retrieve profile:*

**getProfile(client_Id,device_Id):profile**

*Where,* **client_Id** has the same meaning as in the above methods and **device_Id** is used to identify a particular device being used by the client. This method is an interface to external components such as CNAM that wants to access the client's profile. It returns an XML profile document of the specified client through **profile**.

*Profile Metadata (PMD)*

*PMD* is a data repository that holds information about the client's profile. Primarily, *PMD* holds location information of each of the fragments of the client's profile and

names of the local proxy servers that hold replicas of static profiles; see section 5.4.1.1 for details.

**b. Metadata Manager (MDM):**

MDM is responsible for managing metadata[5] of the client's profile. The system maintains metadata to keep track of locations of the profile fragments and the replicas. Detail of this is given in section 5.4.1.1.

The functionalities of the MDM are: creating a new metadata, updating the metadata, retrieving the metadata, and removing the metadata. For these functionalities, it provides the following interfaces:

*i. To create a new metadata storage:*

**createMetadata(client_Id):state**

Where, **client_Id** and **state** have the same meanings as above.

*ii. To update the metadata:*

**updateMetadata(client_Id,attrib_list,value_list):state**

Where, **client_Id** and **state** have the same meanings as above. **attrib_list** is an array of field names, such as *server_name* and *fragment_Id*, to be updated. **value_list** is an array of values of the corresponding attributes.

*iii. To retrieve the metadata:*

**getMetadata(client_Id):metadata**

Where, **client_Id** has the same meaning as above and **metadata** is a metadata document retrieved from the repository.

[5] data that describes another data

iv. To remove the metadata:

**deleteMetadata(client_Id):state**

Where, **client_Id** and **state** have the same meanings as above. This interface is used to remove the client's metadata when, for example, the client is no more part of the system. This happens when the client is removed from the system.

**c. Name Resolver (NR):**

NR is responsible for resolving a given name of the client's home server – a local server that holds metadata of the client's profile – to the corresponding IP address. Given a name (domain and host name), it prepares a DNS request that it forwards to the DNS server, which is supposed to resolve the name to its IP address. For this, NR provides an interface to the external components such as PM. The interface provided by NR has the following general format:

**getAddress(server_Name):address**

Where, **server_Name** is the domain name of the client's home server. For example, it can be specified as *lp4.aau.edu*. **address** is an IP address for the corresponding server name.

## *5.4 Distribution of Client's Profile on Local Proxy Servers*

The profile information on the server side, CPs, is distributed on local proxy servers, as it has been shown in figure 5.1 above. The two types of CPs such as dynamic profiles (DP) and static profiles (SP) are stored separately as they exhibit different characteristics. Storing DP and SP separately enables us to use different distribution techniques that would suit to their characteristics. In this section, we present distribution strategies used for the management of CPs.

Dynamic part of CPs exhibit continuous change whereas static profiles such as storage size of client's device are somehow stable and do not require continuous updating. Since the static profiles, SP, do not change frequently, the system benefits if they are replicated on some of the local proxy servers. On the other hand, the dynamic profiles, DP, are subject to changes and hence replication of such information will require too many update requests, resulting in communication overhead. Thus, instead of being replicated, profiles of this type are distributed by storing the fragments on various local servers.

### 5.4.1 Distribution of dynamic profile

As described above, dynamic profiles are distributed by keeping their fragments in various local proxy servers. The fragmentation is made in such a way that when new facts (profiles) are discovered by the profile extractor, PE, on a particular location, it is kept on the local proxy server found in the vicinity to the client. This profile becomes just one of the fragments of the entire dynamic profiles of the client. Since fragments of dynamic profiles of a given client are kept in various servers, an issue of locating each of the fragments arises. Thus, the following section discusses this issue and proposes a solution to it.

#### 5.4.1.1 Keeping track of distributed fragments

To identify which fragment is stored at which server, the system maintains metadata of the profiles for each client. The metadata contains information such as client's identification (client_Id), location of the profile fragments, names of the servers that hold

replica of the static profiles, etc. The overall purpose of maintaining metadata per client is to be able to locate the replicas as well as the fragments of the client's profile.

Metadata of the client's profile is created initially when a user signs up in the system. During user's registration, the local proxy server, which is currently connected to the client device, creates the metadata storage. Then after, this server becomes a home server/location of the client as it holds information of the client's profiles that are distributed in the system.

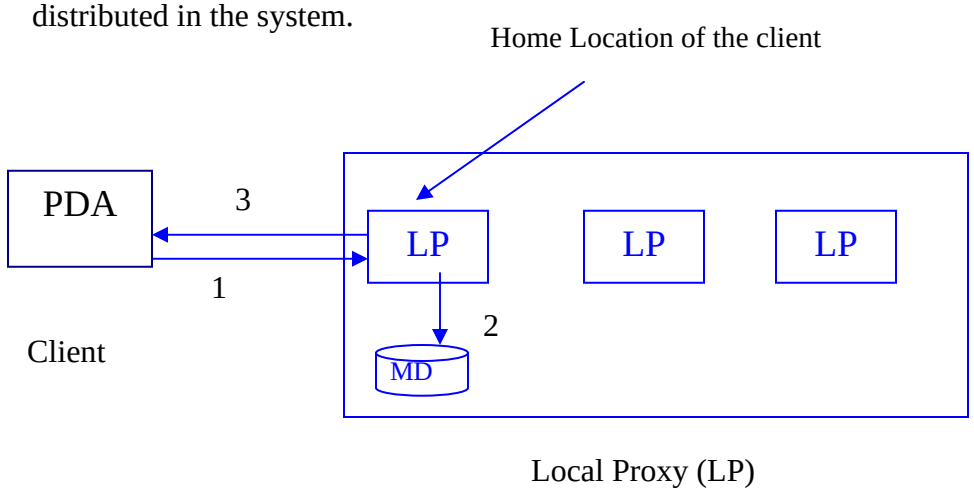



**Figure 5.4: Showing interaction of client device with LP during signup**

1. Sign up request to the nearby LP (by PDA)

2. The LP creates metadata of the client's profile

3. Confirms the registration by returning sign-in name to the client

In this work, we use home-based approach to locate the profile fragments of each client. According to this approach, it is the home server of the client that keeps information of the profile such as name of the servers holding profile fragments and replicas. Thus, from wherever the client logs to the system, the home server of the client should be identified and communicated to get metadata of the client's profile. The approach meets scalability requirement of the system, regarding managing distribution of the profiles as it uses different local proxies for storage and management of metadata of different clients. One of the major drawbacks of this approach, however, is the use of fixed home location. For one thing, it requires the existence of home location all the time [8]. If not, finding the

metadata and subsequently the fragments will become impossible. This problem can be alleviated by keeping copy of the content on some other servers, i.e. by creating backup server. The requester of the metadata is then offered to choose among the servers containing metadata of the client. If one of the given alternatives fails to function, then the other one will be requested.

To locate the home servers of each user, a table that maps users to their respective home servers can be maintained. As there might be too many clients in the system, the size of the table can easily grow up to hinder scalability of the system. A better approach that can scale up to the growing size of pervasive systems is to use distributed naming to locate home servers of the clients.

During user's registration (sign up to the system), home server of the client provides a name that will be used by the user to login (sign in) to the system. This name is domain name of the home server and it will thus become a means to locate home server of the user. It has a form of: ***domain_name\client_Id***.

> Where, **domain_name** is domain name of the home server and **client_Id** is a unique name given to the metadata storage. **client_Id** will also be used to uniquely identify the client in the system.

For example, consider a name *lp4.aau.edu\jhon.* Here, the first part (lp4.aau.edu) is a domain name that identifies the home server of the client - who is identified as *jhon* in the system. This name, hereafter, is referred to as login name of the user.

When a user logs in from anywhere to the system, he has to supply his/her login name so that the system would identify home server of the client. Identification of the home server enables the system to locate profiles of the client that are distributed in the system. Since user login is not one time activity, after registration the login name can be kept on the client's device. These days, it is very common that people move from place to place with their computational devices such as PDA and laptop. Thus, keeping the login name on the client's device removes the burden of re-typing it, provided that the user is using his/her device. But it is sometimes possible that the user may login using a device that does not

have the client's profile. In this case, the user is supposed to provide his complete login name to the system.

In any case, whether the login name is found directly from the user or from the client side profile repository, during login to the system, the name is given to a local server, which contains systems such as *CNAM* and *SSPM* whose descriptions are given in section 4.1 and section 5.3, respectively. The *CNAM*, after receiving client side profile including login name of the client, requests the *SSPM* for server side client's profile by providing the login name. Given the login name to the name resolver (*NR*) module of *SSPM, NR* forwards a request to the *DNS* server to identify address of the client's home server. After the name is resolved, the *SSPM* communicates with the home server to get metadata of the client. Once the metadata data is accessed, the *SSPM* can collect various fragments of the profile so that to deliver it to the profile requester, *CNAM*.

### 5.4.2 Distribution of static profile

As explained in section 5.4, static profiles are distributed by putting their replicas on various local proxy servers. Essentially, the replication will enhance performance and reliability of the system. However, replication creates problem on the consistency of the data store, which is physically distributed across multiple machines.

Replication of the static profiles does not require stringent consistency. For one thing, there is only one process to perform all sorts of changes on the profile replicas of a given client. This process is *profile manager, PM*, module that is found on the home server of the client. Whenever there are changes on the static profile of the client, the current local server informs the changes to the home server of the client. The *PM* on the home server then propagates the changes to the replicas found on various servers. Consequently, there will not be *write-write conflict*[6] on the data store (i.e. replicas of the client's profile). In addition to this, as the replication is made on the static profiles, changes are so rare.

Here, the only requirement is that the changes need to be propagated to all of the copies/replicas. Therefore, it is possible to use a weak consistency model that only

[6] Conflicts resulting from two operations that both want to perform an update on the same data store.

guarantees propagation of the updates to all of the replicas. A model that meets this requirement is *eventual consistency model [8]*. Limitation of eventual consistency model is that when the client is accessing different replicas due to mobility, which is a case in the current system, it can see different versions of the replicas. Consequently, the model does not guarantee consistent state of the data store. This problem is alleviated by using client-centric consistency. Client-centric consistency guarantees a given client concerning the consistency of accesses to a data store by that same client wherever the access is made.

When the client changes its location, all the changes made on the static profiles, before the client is moved to another location, should be propagated to all of the remaining replicas so that the replica on the current location would have a new version of the static profile. Among the client-centric models the one that fulfills this requirement is *read-your-writes model*. A data store is said to provide read-your-writes consistency, if the following conditions hold [8]:

> *The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process.*

It means that a write operation is always completed before successive read operations by the same process, no matter where that read operation takes place.

#### 5.4.2.1 Implementation Issues

There are two issues to be considered for the implementation of replication: distribution protocols and specific consistency protocols.

However, here we discuss only the first issue, distribution protocols. This is because that eventual consistency model, which is selected to be used in this work, does not deal with update conflicts. Since there is only one process performing update on a given data store, there will not be update conflicts in this work, as described in section 5.2..2 above.

#### *5.4.2.1.1 Distribution Protocols*

*a. Replica Placement*

One of the major design issues for distributed data store is deciding where, when, and by whom copies of the data store are to be placed. Regarding these issues, static profiles are managed in the following way:

- Initially the static profile, SP, is entered through client's devices during user subscription to the system. These profiles are then kept in the home server of the client.

- When a remote server requests for static profiles of the client (at times when the client is moved to a remote area), the home server does one of the followings:

    I. Sends just the required profile to the server for the current use. In this case, the home server increments a counter on the frequency of the profile request from that particular server.

    II. Creates replica of the static profile on the remote server. It does this when value of the counter exceeds some limit, which is set in the system. In this case, the home server keeps name of the remote server in the metadata of the client so that it will identify location of the replica.

As it is the home server that decides whether to create a replica on the remote machine, this type of placement is known as *server-initiated* replica placement. The home server creates a replica on the remote server only when the data access frequency of a particular machine exceeds a threshold frequency set in the system. For instance, if the threshold frequency of access is four, then the first four requests from a particular server will not result in having replica. It is only the fifth request that will make the home server to create a replica on the remote server. When the home server creates replica on the remote server it keeps name of the server in the metadata of the client's profile so that it will be able to make update on the replica when changes are made on the SP.

By using a counter on the frequency of profile access, we avoid creating too many replicas. First it checks, through counting the access frequency from a particular location, whether the client stays for long time in the current location. It only creates the replica when the frequency reaches some reasonable value. This is due to the fact that in pervasive systems it is common for clients to change their locations but they may not stay long.

*b. Update Propagation*

For the fact that static profile is stable, the frequency of changes on the profile is low and consequently, the read-to-update ratio at each replica is relatively high. Hence, a push-based approach is used for the propagation of the updates. That is, as soon as the changes are notified to the home server, the home server does update on the existing replicas. In the push-based approach, also called server-based protocols, updates are propagated to other replicas without those replicas even asking for the updates. It, thus, guarantees to maintain a relatively high degree of consistency, i.e. it enables to keep the replicas identical.

The requirement of push-based approach is that the server initiating the update has to know locations of the replicas. As the home server keeps names of local servers that are hosting the replicas (see section 5.4.2.1.1(a) above), push-based update propagation can easily be employed by the home server.

As to the decision of whether to use unicasting or multicasting, it is the underlying propagation protocol that governs. Since multicasting can be efficiently combined with a push-based approach to propagating the updates, multicasting is preferred to be used. In this case, the server that is meant to push the updates to a number of other servers simply uses a single multicast group to send its updates.

The following figure shows the interaction between different profile managing components and dataflow upon a user request.

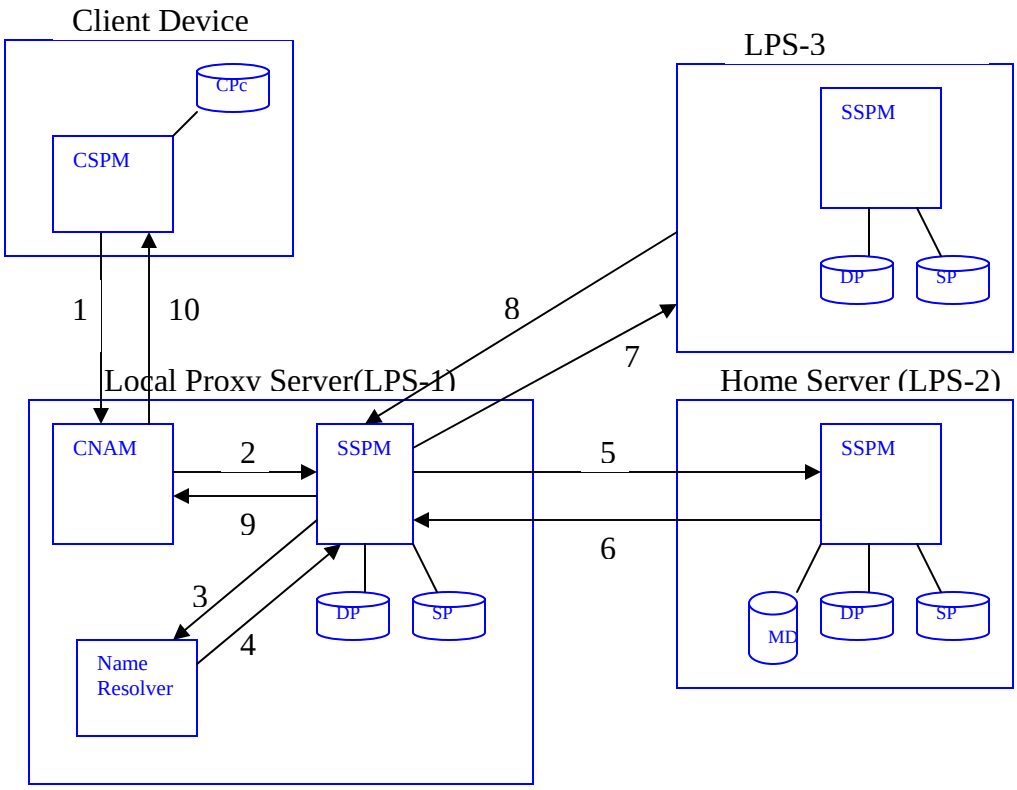



**Figure 5.5: Architecture Overview and data flow upon a user request.**

Figure 5.5 depicts overview of the proposed architecture of the client's profile management system. Here, we describe the basic steps involved in a user request to illustrate the system behavior. (1) A user issues a service request to the system through his device. The request contains login name of the user that will later be resolved to the client's home server address, which stores metadata of the client's profile. Then the request is intercepted by the local proxy server (LPS) that is found in the vicinity of the user's current location. According to the service based content adaptation architecture, content adaptation plan is performed by a module called CNAM (Content Negotiation and Adaptation Module), which is found in the local proxy server. Thus, the service request is directly given to this module. The module then requests the content server for the required content/service. (2) The CNAM requests the SSPM (Server Side Profile Manager) module, which is found on the same machine as CNAM, for the client's

profile. (3) The SSPM forwards the login name to the name resolver so that it would know the address of the client's home server. (4) The address of the client's home server is returned to the SSPM. (5) The SSPM forwards a request to the home server of the client for the client's metadata. (6) The client's home server returns the metadata to the requester. The SSPM then caches the metadata for future use. From the metadata, the SSPM identifies location of the various fragments of dynamic profile of the client and also location of the replicas of static profiles. (7) Based on the location information from the metadata, SSPM dispatches profile request to various local proxy servers. (8) The requested LPSs return the required profile fragment and/or replica. (9) The SSPM returns the profile to the CNAM that has made the request for the client's profile. Based on the given profile, the CNAM performs a plan for the adaptation and then chooses one of the adaptation services that are available. Finally, the content to be adapted is given to the selected adaptation service. (10) The adapted content/service is then delivered to the client's device.

# CHAPTER 6

# 6 Prototype

In this thesis work, we have proposed an architecture for the clients' profile management system with the intension to provide as much profile information as required for the content adaptation in pervasive environments. Complete description of the architecture is given in chapter 5. In this chapter, we provide discussion of the prototype developed based on the proposed architecture. Section 6.1 gives an overview of the prototype. Section 6.2 briefs the prototype development environment. Section 6.3, describes major components of the prototype. Finally, section 6.4 discusses the results of the prototype.

## *6.1 Overview*

This prototype is aimed at validating the proposed architecture for the clients' profile management system so that to avail the required profiles to the adaptation and delivery system. In the architecture, two modules are proposed to manage the profile fragments that are kept on two sides: the client devices and the local servers. Accordingly, in the prototype, we fragmented the entire client's profile into two so that to keep each on the two sides. For the management of the fragmented profiles, two modules are defined in the prototype. The modules are CSPM (Client Side Profile Manager) and SSPM (Server Side Profile Manager); they are deployed on the client devices and local servers, respectively. In each of the modules, there are well-defined web interfaces that enable communication between themselves and with external components such as profile extractors.

In addition to the management activities, the prototype system provides various forms for user interaction with the system. There are forms for login, registration, and displaying results of activities such as signup and login. The login form allows entering user id, password, and login name, see figure 6.1. On the other hand, the signup form provides data entry fields to enter server side profiles, client side profiles, id, and password.

During signup (user registration), a user is required to specify his/her id, password, profiles that will be kept on the client devices and also those that will be kept on the server side. The server side profile, along with the user id, is sent to the local server found

in the vicinity of the client device. After receiving the registration request, the local server creates metadata of the client and it becomes home server/location for the client. Consequently, it generates a login name for the user and returns to the client. The login name is used to identify home location of the user while he/she logs into the system from anywhere.

During login, the user provides his/her id, password, and optionally login name. The user is supposed to provide the login name when he/she is using a new device that does not have the profile of the client. Using the id and password supplied by the user, the system performs authentication. For valid login, the client side profile of the user is retrieved from profile storage and sent to the content adaptation module on the local server, along with login name and id. The adaptation module then requests the server side profile manager for more profiles of the client. The profile manager uses login name to identify home server of the client and consequently, based on the metadata information, collects the client's profiles that are found distributed on various servers. Finally, the collected profiles are delivered to the adaptation module that has posed profile request.

## *6.2 The Development Environment*

The prototype is developed on Microsoft Windows platform (Windows 2000 and Windows XP). For storage of various forms of information that the system maintains such as clients' profile and profile metadata, we used MySQL database management system (version 4.1.1). XML specification is used to represent the distributed profile data in their repositories. Message based communication among the different components of the system is used for exchanging information such as client's profile. For the purpose of exchanging message between the components, SOAP protocol over HTTP is used through the APIs provided by the Java programming language for SOAP and XML processing. For coding, we used Java Programming language (version 1.4.2). The interfaces provided by the two modules and local proxy server are implemented as Java Servlets using the Java Servlet technology. For implementation of Java Servlet, Apache Tomcat (version 5.0) is used as a web-server.

## 6.3 Implemented Components

For the management of profile data that are maintained on the server and client sides, we defined two modules - Client Side Profile Manager (*CSPM*) and Server Side Profile Manager (*SSPM*). Each of the modules provides interfaces for communication between themselves and also with external components. The profile information is stored in the database tables in the form of *XML* document.

**Module CSPM:**

This module consists of various components for the management of part of the client's profile that are kept on the client devices. It also contains a component for provision of user interfaces such as login form, profile entry form, and result display form. Each of the components is implemented as java classes. The followings are major components of the module that are implemented in the prototype.

**Class UI:** this class implements User Interface component of the *CSPM*. The *UI* class provides various forms that are used as interface between the user and the profile managing system. The forms are web-based interfaces that are used by users for login, signup, and profile entry. Based on the user actions received from the forms, *UI* invokes an appropriate method defined in *PM* for the management of profile data. Figure 6.1 shows an instance of user login form, which is provided by this class.

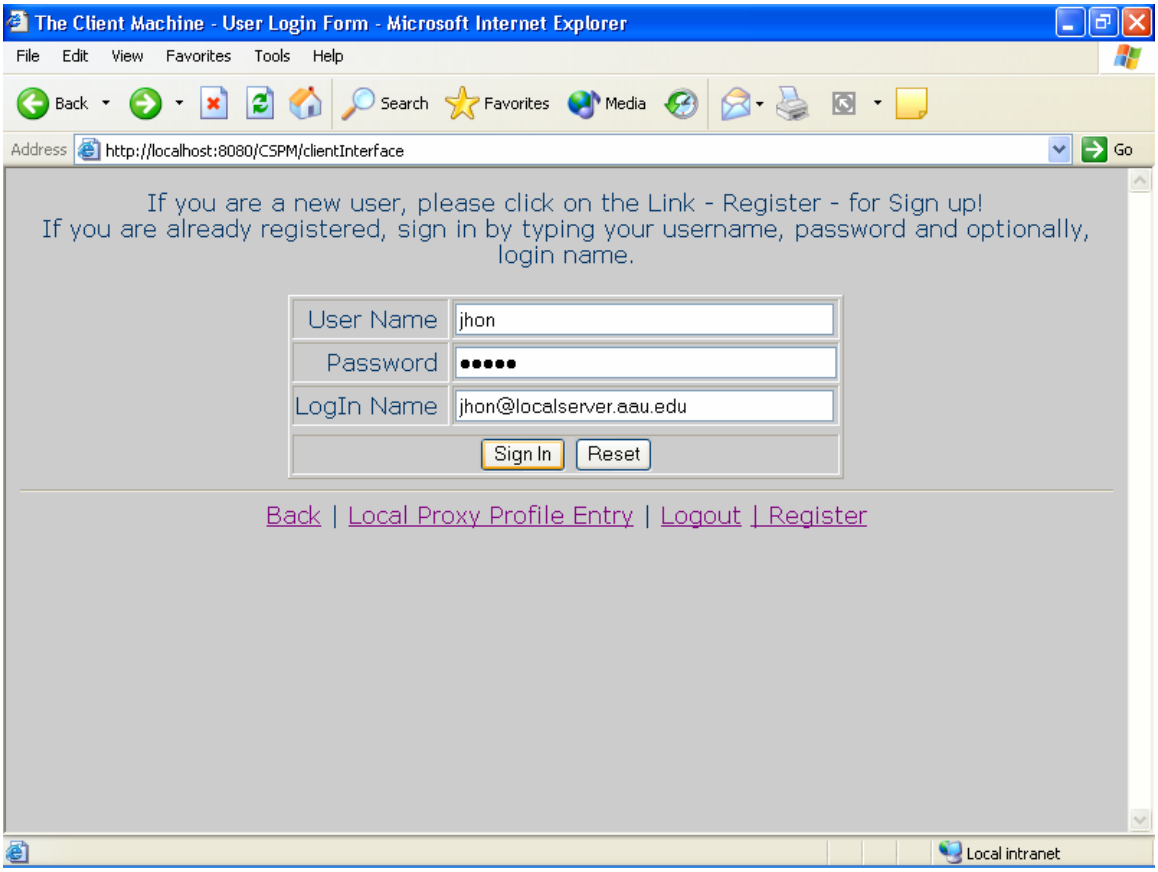



Figure 6.1 User Login Form

**Class PM:** this class implements the Profile Manager component of the *CSPM*. It consists of various methods that are intended for the profile management functionalities such as inserting new profiles of the clients, retrieving and updating the existing profiles of the clients. Some of the methods in this class also make communication with the remote applications, those that are hosted on the local proxy servers. For example, the signup method, after receiving the initial profile of the client, inserts client profile on the client device and sends the server side profile to the server machine. Upon completing the registration, the server creates and returns the login name back to the client.

In addition to the above major classes, the *CSPM* module also contains more utility classes that are intended to provide supportive functionalities such as generating *XML* document for a given profile, querying the database for insertion, deletion, retrieval, etc., parsing the *XML* document, creating *SOAP* message, etc.

**Module SSPM:**

This module contains components that are engaged in the management of the server side profiles. It, in addition, contains a component that provides user interface for the entry of dynamic profiles. The components are implemented as java classes and the major ones are described below.

**Class LPI:** This class implements the Local Proxy Interface component of the *SSPM*. It is a *Java Servlet* class that binds to a certain port and listens to the incoming requests from the client devices and from the *PM* class on the remote proxy servers. When there are incoming requests, the *LPI* accepts the requests through web interfaces. During user signup, for example, the *PM* class of *CSPM* module sends registration request to the *LPI*. Upon accepting the request, the *LPI* performs registration of the user and returns login name of the user back to the requesting module. The *PM* classes on the remote servers send request to *LPI* when they need some data such as profile fragment, profile replica, and profile metadata.

In addition, this class also provides interfaces for requests coming from the user form. There is one user form on the local servers that is intended to collect dynamic profiles. The form simulates profile extractor module. When the *LPI* class is delivered dynamic profiles of the client, it invokes the *creatFragment ( )* method of PM module so that the local server would keep the profile fragment of the client extracted on the current server.

**Class PM:** This class implements the Profile Manager component of the *SSPM*. It contains definition of various methods that perform profile managing activities. Among the functionalities of this class are: insertion of new profiles to the profile repositories, retrieval of profiles for delivery, retrieval of profile metadata, updating the metadata, and creating replica on the remote server. When, for example, it is requested for the server side client's profile, it transparently collects the profiles that are found distributed in the system. For collection of distributed profiles, it sends *SOAP* messages to the *LPI* classes that are found on the remote servers.

**Class MDM:** This class implements the Metadata Manager component of the *SSPM*. It performs various activities related to the management of profile metadata. During user registration, for example, the *MDM* class creates profile metadata record in a database table. In the metadata table, each record contains profile metadata of a client.

The metadata table has three fields: *id*, *replicas*, and *fragments*. The *replicas* field contains list of server names that host replica of the client's profile for each client. Information in this field is used by the home server to update the replicas when, for example, changes are notified for such profiles. The *fragment* field contains list of servers that contain fragments of the client's profile for each client. When the server side profile of a client is requested, first metadata of the client is retrieved from the client's home server. Then, based on the location information of in *fragments* field, collection of the profile fragments from remote servers is made.

The MDM class also provides functionalities such as updating the metadata content, and removing the metadata record.

**Class NR:** This class implements the Name Resolver component of the *SSPM*. Given a *login* name, it returns the corresponding server address. Normally, this class would interface with the DNS server to get the login name resolved, but as the prototype is implemented in the local area network, this is not the case in the current prototype. Here, we used a simple table that maps names to addresses is used to simulate the actual name resolving system. In the large systems that incorporate a large number of clients, it may be necessary to interface with DNS server to make use of scalable naming system, see section 5.4.1.1 for details.

**Class LocalProxy**: This class is not part of the profile management component; it provides web interfaces to accept the incoming requests from the profile managing components such as the *PM* of *CSPM* module. It simulates the *CNAM* module of the service based content adaptation architecture, which has been discussed in section 4.1. During user login, the *PM* class of *CSPM* module sends client side profile of the client along with the login name to the *LocalProxy* class. After receiving the client side profile, this class invokes the *getProfile ( )* method of *LPI* class for more profile information that is maintained on the local servers.

In addition to the above classes, the *SSPM* module also contains many more classes that are intended to provide functionalities such as creating and manipulating *XML* documents, composing and sending *SOAP* messages, inquiring the database, etc.

**Result**

For demonstration purpose, two local proxy servers are configured and the SSPM module is deployed in each of the servers. To simulate the client device, a desktop PC is used. The constraint imposed on the desktop PC is that it stores only part of the client's profile to show space limitation that exists on the client devices in pervasive systems. The CSPM is deployed on the client device and in addition to profile management, it provides user forms for login and signup, see section 6.3 above. During registration of a new user, the CSPM keeps client side profile on the client device and sends server side profile to one of the servers at random. The local server to which the profile is sent becomes home-server of the client. Each time the client device tends to connect to a local server, it selects one of the local servers at random. The reason for random section is that the prototype is implemented on local area network where each of the servers is equally available to the client.

During signup of a client, the two modules, CSPM and SSPM, deliver profiles of the client to the content adaptor. Since content adaptation is not in the scope of the current work, the prototype does not implement its functionalities. After getting the profiles of a client, the prototype system only displays on a form since there is no content adaptation component to make use of the client's profile. When this system is integrated to the complete content adaptation/delivery system, the client profiles can be delivered to the component that does content adaptation.

# CHAPTER 7

# 7 Conclusion and Future Works

## *7.1 Conclusion*

In the current work, we have attempted to address the various issues related to client's profile management system in a pervasive environment. Specifically, characteristics of profile information and constraints in pervasive systems are taken into consideration in proposing solutions to the problem. Based on the characteristics of the profiles, we propose to fragment the client's profile into two: the one that can be maintained on the client device and the one that can be kept on the servers (local servers). Thus, the proposed profile management architecture includes two major components to carry out the management tasks on the two sets of profile information: Client Side Profile Manager (CSPM) and Server Side Profile Manager (SSPM) that are deployed on the client devices and the local servers, respectively.

The CSPM consists of two modules: User Interface (UI) module and Profile Manager (PM) module. The UI module is responsible for creating forms using which the users perform login to the system, sign up, enter initial profiles, etc. The PM module is responsible for creating a new profile, updating the profile when there are changes, removing unwanted profile, retrieving profile from its repository, etc.

The SSPM consists of three modules: *Profile Manager (PM)*, *Metadata Manager (MDM)*, and *Name Resolver (NR)*. These modules cooperate to provide a transparent system that manages distributed profiles. They communicate with each other by message passing. The PM is responsible for creating a new profile, updating existing profile, retrieving the required profiles, etc. The MDM manages metadata of the client's profile. Creating and retrieving the metadata are among the tasks of the MDM. The NR handles name resolving task. Given, for example, domain name of a server, it returns address of that server to the PM.

For validation of the proposed architecture, a prototype system is developed. The prototype system is composed of two modules that are intended to manage profiles kept on the two sides: the client devices and the local servers. The modules are composed of Java classes and perform management of the distributed profiles in a transparent way. That is, if profile of a client is needed (for example, for content adaptation), a request is made to the profile manager found locally. The profile manager then does collection of the client's profiles that are found distributed in the system. Communication among the modules is performed using SOAP messaging through web-interfaces implemented using Java Servlet. For profile entry and result display, the prototype provides various user forms.

## *7.2 Future Works*

In this work, we have proposed an architecture for the client's profile management system that suits for pervasive environments. Though most of the issues regarding profile management are addressed in the architecture, there are more issues to be considered in the future works:

1. *Developing Profile Extractors*: Dynamic profiles are discovered through time while the users interact with the system. Thus, there should be profile extractor modules that learn how the user interacts with the system and consequently infer attribute values of the dynamic profile of the client.

2. *Developing Profile Integrator*: Due to mobility, a client may get service through different local servers during which dynamic profiles are extracted. In the current work, such profiles are kept on the servers, where they are extracted, as profile fragments. When profile of a client is requested by some external component (by content adaptor, for example), the fragments that are kept in various servers are collected for delivery. But before delivery, they have to be integrated. Thus, the profile management system needs to have profile integrator component that would combine the collected profiles by resolving conflicts, if any.

Conflicts may arise when different values are given for the same profile attribute. For example, in one of the fragments the value given for certain attribute (say, music preference) is x and on some other fragment the same attribute may have different value, say y. Hence, during integration, the integrator should deal with such type of conflicts.

3. *Developing Profile Representation tool:* Profile data are interrelated and have complex structures that require an appropriate representation tool. The current profile representation tools such as CC/PP do not meet some of the profile representation requirements that have been described in section 2.2. For example, CC/PP has only two levels for structuring profiles, whereas most of the profiles require more than two levels, see figure 2.1 for profile structuring. This, thus, calls upon inventing a new profile representation tool as per the requirements of the profile information.

# Reference


[1]. *Alessandra A., Claudio B., Nicolo C., Dario M., Daniele R.,* "***Integrated Profile Management for Mobile Computing***".

http://www.dimi.uniud.it/workshop/ai2ia/cameraready/agostini.pdf, visited on November 23, 2004.

[2]. *Girma Berhe, Lionel Brunie, Jean-Marc Pierson,* "***Realization of distributed content adaptation with service-based approach for pervasive systems***", 2004.

http://liris.cnrs.fr/publis/LIRIS-RR-037.pdf, visited on November 18, 2004.

[3]. *Tao GU, Hung Keng Pung, Da Qing Zhang, Xiao Hang Wang,* "***A Middleware for Building Context-Aware Mobile Services***", 2004.

http://citeseer.ist.psu.edu/659291.html, visited on November 20, 2004.

[4]. *Albert Held, Sven Buchholz, Alexander Schill,* "***Modeling of Context Information for Pervasive Computing Applications***", Proc. of the World Multiconference on Systemics, Cybernetics …, 2002

http://wwwrn.inf.tu-dresden.de/~buchholz/hades/SCI2002-paper512JH.pdf, visited on November 4, 2004.

[5]. *Sven Buchholz, Thomas Hamann, Gerald Hübsch,* "***Comprehensive Structured Context Profiles (CSCP): Design and Experiences***".

http://wwwrn.inf.tu-dresden.de/~buchholz/cormorant/CoMoRea04.pdf, visited on December 2, 2004.

[6]. ***Webnox corp. dictionary - meaning of ubiquitous computing***

[7]. *Khalil El-Khatib, Gregor v. Bochmann, and Abdulmotaleb El Saddik,* "**A Distributed Content Adaptation framework for Content Distribution Networks",** 2004.
http://beethoven.site.uottawa.ca/dsrg/PublicDocuments/Publications/ElKh04c.pdf, visited on December 5, 2004.

[8]. *Andrew S. Tanenbaum*, Maarten Van Steen, "***Distributed Systems***", Pearson Education Asia, 2002.

[9]. *Zhijun Lei, and Nicolas D. Georganas*, "***Context-based Media Adaptation in Pervasive Computing***". http://citeseer.ist.psu.edu/446978.html, visited on December 20, 2004.

[10]. *Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy*, "***Modeling Context Information in Pervasive Computing Systems***". http://citeseer.ist.psu.edu/henricksen02modeling.html, visited on December 10, 2004.

[11]. *John Shelley*, "***How to Use XML"***, Babani Computer Books, November 2002.

[12]. *Elliotte Rusty Harold, W. Scott Means*, "***XML in a Nutshell"***, January 2001.

[13]. *Giovanni Vigna*, "***Mobile Agents: Ten Reasons for Failure"*,** 2004. http://doi.ieeecomputersociety.org/10.1109/MDM.2004.1263077, visited on March 22, 2005.

[14]. *Claudia Raibulet and Claudio Demartini*, "***Mobile Agent Technology for the Management of Distributed Systems***", 2002. http://www.terena.nl/conferences/archive/tnc2000/proceedings/1A/1a2.html, visited on March 20, 2005.

[15]. *Arkady Zaslavsky*, "***Mobile Agents: Can They Assist with Context Awareness?"*,** Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM'04). http://csdl.computer.org/comp/proceedings/mdm/2004/2070/00/20700304.pdf, visited on March 27, 2005.

[16]. *Michael N. Huhns and Munindar P. Singh*, "***Agents on the Web***", 1997. http://www.bookrags.com/sciences/computerscience/agents-on-the-web-csci-04.html, visited on March 29, 2005.

[17]. *Uche Ogbuji*, "***An introduction to RDF"***, Dec 2000

http://www.ibm.com/developerworks/library/w-rdf/, visited on March 25, 2005.

[18]. *Ganesh Sivaraman*, "***Composite Capability/Preference Profile - RDF based profiler for mobile devices",*** 2000.

www.tml.hut.fi/Opinnot/Tik-111.590/2000/Papers/Rdf.html, visited on January 2, 2005.

[19]. *W3C Recommendation*, "***Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0***", 15 January 2004.
http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/, visited on February 10, 2005.

[20]. *Tom Clements*, "**Overview of SOAP**", January 2002

*http://java.sun.com/developer/technicalArticles/xml/webservices/*, visited on March 20, 2005.

[21]. **"SOAP Version 1.2 Part 0: Primer**", W3C Recommendation, June 2003

http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, visited on April 2, 2005.

[22]. *Tom Sheldon and Big Sur*, "***The Encyclopedia of Networking and Telecommunications***", 2001.

http://www.tomsheldon.com/, visited on April 06, 2005.

[23]. *Robert E. McGrath*, "***Caching for Large Scale Systems***", D-Lib Magazine, January 1996.

http://www.dlib.org/dlib/january96/ncsa/01mcgrath.html, visited on March 10, 2005.

[24]. *Girma Berhe, Lionel Brunie, Jean-Marc Pierson*, "***Modeling Service-Based Multimedia Content Adaptation in Pervasive Computing***", Proceedings of the 1st conference on Computing frontiers, pages 60 – 69, Ischia, Italy, 2004.

# Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: _____

Signature: _____

Date: _____

Confirmed by advisor:

Name: _____

Signature: _____

Date: _____

Place and date of submission: Addis Ababa, November 2005.