An Adaptable Context-Management Framework for Pervasive Computing

by

Jared A. Zebedee

A thesis submitted to the
School of Computing
in conformity with the requirements for the
degree of Master of Science

Queen's University
Kingston, Ontario, Canada
September, 2008

Abstract

Pervasive Computing presents an exciting realm where intelligent devices interact within the background of our environments to create a more intuitive experience for their human users. We demonstrate enabling context-awareness through our creation of a standardized context-management framework. Our framework moves towards device intelligence by supporting context-awareness.

Context-awareness is what gives devices the ability to understand and exchange information about each other. Context information is used to determine device purpose, capabilities, location, current state, and other properties.

Several elements are required in order to achieve context-awareness, including a suitable ontology, a context model, and a middleware platform upon which to implement the context model. In this work, a complete context-management framework is presented and evaluated. We propose our own ontology specification and context model, and implement a middleware using the Web Services Distributed Management (WSDM) interoperability standard.

Acknowledgements

I wish to extend my sincerest thanks to my supervisor, Dr. Patrick Martin, for his expertise, guidance, and support throughout this work. I would also like to thank Cyrus Boadway and Wendy Powley for their outstanding support and advice. Additionally I wish to acknowledge my friends, family and Queen’s University staff and student colleagues who have provided unending encouragement and interest during this research. Finally, I wish to submit a special thanks to Kirk Wilson, Ph.D., CA, Inc. for his expert direction, assistance and enthusiasm which were invaluable to me throughout this endeavour.

Table of Contents

List of Tables

List of Figures

Glossary of Acronyms

ACME	Adaptable Context-Managed Environment
ACMF	Adaptable Context-Management Framework
CoBrA	Context Broker Architecture
CME	Context Managed Environment
CSB	Computer Science Building
GED	Global Element Declaration
HTTP	Hypertext Transfer Protocol
J2EE	Java 2 Platform, Enterprise Edition
J2ME	Java 2 Platform, Micro Edition
MANET	Mobile Ad-Hoc Network
mDNS	Multicast Domain Name System
OASIS	Organization for the Advancement of Structured Information Standards
OSGi	Open Services Gateway initiative (name now obsolete)
OWL	Web Ontology Language
OSI Model	Open Systems Interconnection Basic Reference Model
RMD	Resource Metadata Descriptor
RPSD	Resource Properties Schema Document
SOAP	Simple Object Access Protocol
SOUPA	Standard Ontology for Ubiquitous and Pervasive Applications
TCP/IP	Transmission Control Protocol / Internet Protocol
URI	Universal Resource Identifier

Wi-Fi	Wireless Fidelity
WSDL	Web Services Description Language
WSDM	Web Services Distributed Management
XML	Extensible Markup Language
XSD	XML Schema Document
Zeroconf	Zero Configuration Networking

Chapter 1

Introduction

Mark Weiser, a well-known pioneer in pervasive (or ubiquitous) computing [26], described pervasive computing as the third wave (or era) of computing, with the first wave being the mainframe era, and the second wave being the PC era [28]. He described pervasive computing as a state where devices are so pervasive and critical to our activities that they are taken for granted and effectively disappear into the background [29].

Weiser predicted that the crossover point between the PC era and pervasive computing era would occur sometime between 2005-2020 [28]. We are currently in a state of transition between these two eras. It is only recently that technological advances have produced devices small and sophisticated enough to provide the necessary hardware infrastructure for creating pervasive environments.

Given that the hardware is now available, our attention turns to software and the issue of what differentiates ordinary computing devices from pervasive devices. In general terms, a pervasive device has the ability to process and share information about itself and its surrounding environment, that is, to be context-aware. This context-awareness acts as a cushion between the technology and the user which allows us to interact with devices in a more intuitive way.

Context has a variety of definitions in the pervasive computing literature (Chen & Kotz [7], daCosta, Yamin & Geyer [13], Strang & Linnhoff-Popien [27]). Dey [15] describes context as information that can be used to characterize the situation of an entity, where an entity can be a person, location or object relevant to the interaction between a user and an application. Following from this definition, we can say that a system or application is context-aware if it uses context to provide information or services relevant to the user's task.

For instance, a smart thermostat's context information should include details about its location, functionality, and information on how to access its temperature controls. Upon entering the vicinity of a context-aware thermostat, a context-aware PDA would be able to detect its presence, determine that it is a smart thermostat, and obtain information on how to access and manipulate the current room temperature.

1.1 Motivation

In order to achieve context-awareness in pervasive computing environments, there must be some means of managing the context information. While various context management solutions such as CoBrA [8] and PersonisAD [5] have been proposed, a standardized solution has not been successfully established. A standardized solution is necessary in order to facilitate compatibility between different hardware manufacturers and application providers. Standardized device hardware and communication infrastructures are rapidly becoming commonplace; that is, devices can currently discover and communicate with one another using established communications standards.

However, high-level information exchange is still mostly proprietary and application-specific. A context-management framework is needed in order to solve this problem.

For instance, a PDA and smart thermostat can establish a wireless connection using Wi-Fi, and exchange data using the TCP/IP communications protocol. However, at the application level, there is no standard interface and a proprietary interface is often implemented instead. This interface may not even include a context-management mechanism. For example, Proliphix Inc. offers thermostats featuring "an embedded Web server with advanced IP networking technology and a browser-based Graphical User Interface (GUI)" [25]. With implementations of this type, there is no context-awareness, which forces users to access devices manually through proprietary interfaces.

These types of interfaces lead to scenarios where different devices exchange information in different ways. In contrast, a context-management framework adds structure to application-level communication, and enables context-awareness across all devices.

1.2 Problem

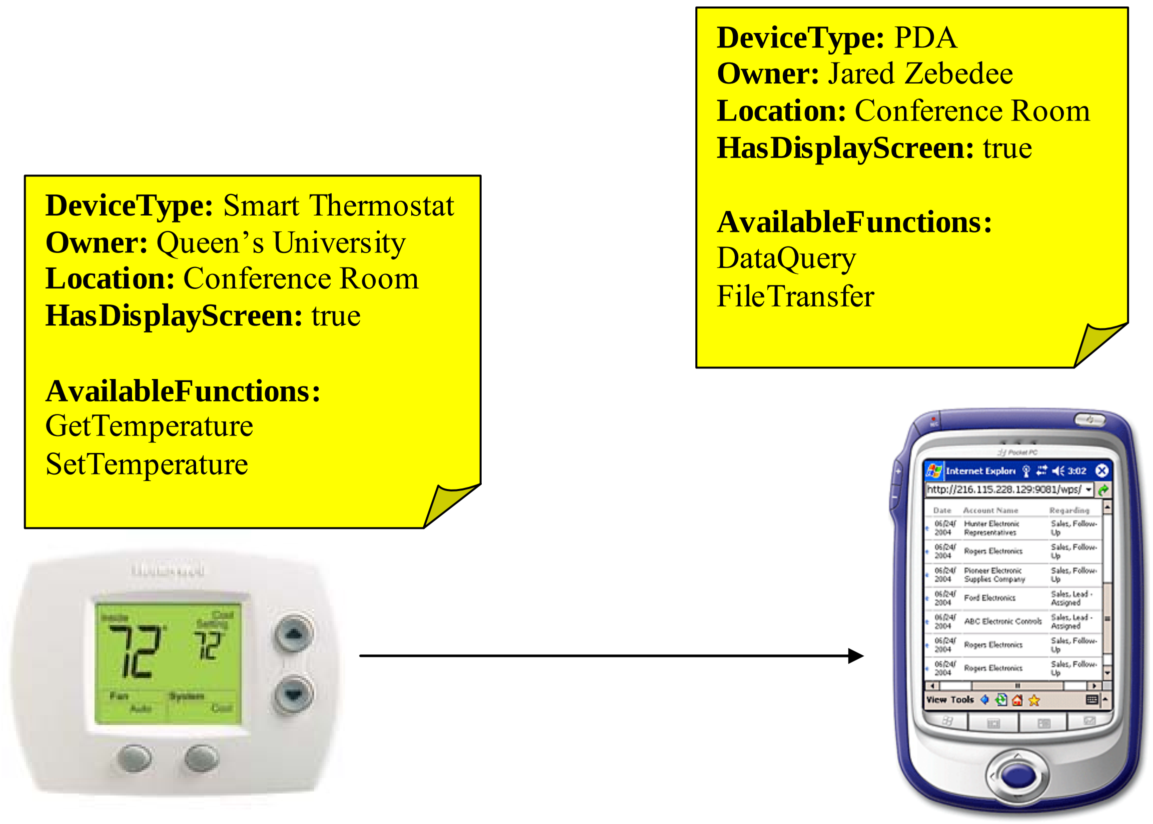
Devices in pervasive environments must be able to interact with only minimal human assistance. They must be able to autonomously connect and exchange information with each other. As mentioned previously, devices can currently connect using existing communications standards such as Wi-Fi and Bluetooth and are able to exchange data using protocols such as TCP/IP. The problem is that devices are currently unable to adaptively exchange information relevant to their context.

An adaptable, standards-based context-management framework is needed in order to help realize the pervasive computing paradigm. A proprietary design may produce an excellent framework that does its job very well, but without standards its usefulness is limited to the devices produced by a single manufacturer or use cases conceived by an individual developer.

Consider the following two scenarios, which are based on the smart thermostat example. In the first scenario, the thermostat has no context-management framework. The thermostat therefore has no explicit way of sharing context information with the outside world. Thus, another device such as a PDA may be able to detect that there is a device nearby but cannot ascertain its purpose or function. In this situation, the onus is on the user to determine that the nearby device is in fact a smart thermostat and how to access it through the PDA. This requires that the user have some prior knowledge about the existence of the smart thermostat and how it is accessed (through a Web interface, for example).

In the second scenario, the thermostat incorporates a context-management framework which enables it to store and share context information about itself. This allows other devices, such as nearby PDAs, to ascertain the thermostat's purpose, current state, and available functions. As shown in Figure 1.1, both the thermostat and PDA have context information associated with them. In this situation, the user does not require any prior knowledge about the thermostat, or even awareness of its existence. The information exchange between devices takes place autonomously, in the background, and without the need for user assistance. Should the user wish to adjust the temperature, the

PDA can provide access to the thermostat's functions in a standard way which is also facilitated through the context-management framework.



DeviceType: Smart Thermostat
Owner: Queen's University
Location: Conference Room
HasDisplayScreen: true

AvailableFunctions:
GetTemperature
SetTemperature

DeviceType: PDA
Owner: Jared Zebedee
Location: Conference Room
HasDisplayScreen: true

AvailableFunctions:
DataQuery
FileTransfer

Figure 1.1 Pair of context-aware devices

1.3 Research Statement

The goal of this research is to investigate an adaptable context-management framework (ACMF) to support context-awareness in pervasive computing environments. We propose a framework that intelligently integrates several existing technologies and standards. The motivation is to create a model that is easy to understand and can be used across disparate devices, software platforms and physical environments.

We recognize that security and privacy are important considerations in pervasive environments. However, we do not address these issues in our framework because our objective is to demonstrate enabling context-awareness. We feel that incorporating security falls outside the scope of this research and such an attempt would result in an insufficient treatment of the subject. Instead, our framework focuses on adaptability and can later be extended in order to address the issues of security and privacy.

To demonstrate our framework we present a simulated pervasive environment which incorporates a device discovery and communication mechanism, a simple context model, a set of context-exchange protocols and a middleware platform.

The device discovery and communication mechanism enables devices to see each other and exchange raw data. We assume that pervasive devices support TCP/IP which handles communication below the application layer of the OSI model [18]. We simulate a self-configuring TCP/IP network, and therefore assume that some form of ad-hoc networking is used, such as Zeroconf (Zero Configuration Networking) [10]. Zeroconf provides DNS (Domain Name System) service discovery [11] which is one method of enabling devices using TCP/IP to see each other on the network.

The context model consists of an inheritance hierarchy which incorporates a master ontology and guidelines for specifying device properties. Devices within the pervasive environment must follow the context model and derive their context properties from the master ontology. For this research, we have created a sample master ontology containing definitions of devices and elements that are likely to be encountered in an academic environment. The range of devices that the framework can support is limited only by the scope of the master ontology (which can be expanded to suit any environment or scenario). The context model also defines how context information is exchanged between devices.

The middleware platform consists of the software and logic necessary to implement the context model, and handles all device communication at the OSI application layer. For this we have selected WSDM (Web Services Distributed Management) [19] technology, an interoperability standard that facilitates distributed management of resources. This turns out to be an excellent fit for device context management. Our context-management framework treats each pervasive device as a WSDM “Resource” that can be remotely managed.

1.4 Thesis Organization

The remainder of the dissertation is organized as follows. Chapter 2 provides an overview of pervasive computing in general, and discusses related work in the areas of context, context management, and ad-hoc networking. Chapter 3 describes our ACMF in detail. Chapter 4 discusses the WSDM standard and how the Apache Muse middleware platform was used to implement our ACMF. An overall summary is presented in Chapter 5 along with a discussion of future work.

Chapter 2

Background and Related Work

This chapter is divided into three sections. The first section (2.1) provides background in the area of pervasive computing, and discusses the role of context-awareness in pervasive computing environments. Section 2.2 examines two existing context-management frameworks and an ontology model. Finally, Section 2.3 presents two existing technologies relating to ad-hoc networking and device discovery.

2.1 Pervasive Computing and Context-Awareness

In contrast to traditional desktop computing, where a user deliberately interacts with a single device for a specific purpose, someone engaged in pervasive computing interacts with one or more computing devices in their surrounding environment without needing prior instruction on how to operate them or, in some cases, without even realizing that the interactions are occurring.

Context-awareness is a key prerequisite to achieving pervasive computing. Context-aware devices have the ability to store, process and disseminate information about themselves and their environment. This ability allows devices to infer their own capabilities, purpose, operating state and environmental circumstances, which in turn reduces the need for human attention and involvement in operating them. In a context-aware environment, computing devices are able to sense and respond to their users' needs automatically.

2.2 Context Management

In order to achieve context-awareness, there must be some mechanism to manage the context information. At the device level, we call this a context manager. In the broader scope of managing context across a pervasive computing environment, the term "context-management framework" is used.

A context-management framework is typically made up of a context model and a context manager middleware. A context model provides the overall framework structure, and a context manager middleware is software which implements the model (using communications protocols that specify how interactions take place between devices). Each environment which implements a context-management framework must also define an ontology which provides a shared vocabulary for all devices using the framework.

Anagnostopoulos, Tsounis and Hadjiefthymiades [2] focus on the use of ontologies for describing the communication schemes between entities in pervasive computing environments. They are similar to our approach in that entities exchange data and semantics to allow the interactions. Their model is, however, not implemented.

Christopoulou, Goumopoulos and Kameas [12] describe an ontology-based system for context modeling, management and reasoning. It is intended to be used in building context-aware applications. Their work focuses on the rule engine used in the system. Our work differs in that we concentrate on providing a standardized framwork to discover, store and exchange contexts.

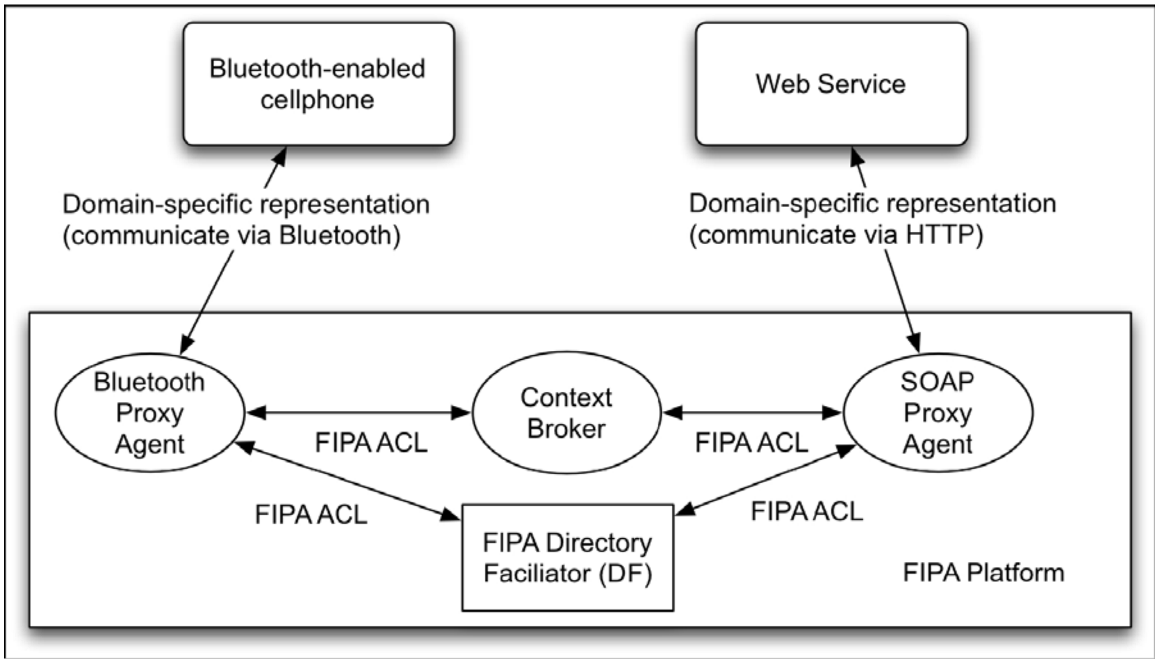
Da Rocha and Endler (2005) [14] are concerned with the efficient evolution of the context model and dealing with the heterogeneity of pervasive computing environments. Their approach, like our own, identifies roles (context provider, context consumer and context service) in their model. It also uses event-based communication which is similar to the WSDM notifications employed in our implementation. A key difference is that our work is based on accepted standards while da Rocha and Endler implement their context management on a non-standard research prototype.

The remainder of this section examines some existing work in the area of context management. The CoBrA [8] and PersonisAD [5] context-management frameworks and the Delivery Context Ontology [31] are discussed.

2.2.1 CoBrA (Context Broker Architecture)

CoBrA is a context-management framework that was developed in 2004 by Dr. Harry Chen as part of his PhD research at the University of Maryland, Baltimore County [8]. CoBrA uses a centralized approach to managing context and incorporates a collection of intelligent software agents as its context manager middleware. A principal component of CoBrA is the "context broker" agent which acts as a hub, providing a shared context model and disseminating context information among all devices and agents in its environment. While the broker approach provides a consistent context model, it also creates a server-centric architecture, where devices in the environment must rely on a centralized server in order to achieve pervasive functionality.

CoBrA agents follow the FIPA (Foundation for Intelligent Physical Agents) Agent Management Specification [17] as a standardized means of communication. As Figure 2.1 illustrates, communication between devices is facilitated by their respective agents, as well as the central context broker agent. Communication between agents takes place using the FIPA ACL (Agent Communication Language).


Bluetooth-enabled cellphone
Web Service
Domain-specific representation (communicate via Bluetooth)
Domain-specific representation (communicate via HTTP)
Bluetooth Proxy Agent
Context Broker
SOAP Proxy Agent
FIPA ACL
FIPA ACL
FIPA ACL
FIPA ACL
FIPA Directory Faciliator (DF)
FIPA Platform
Figure 2.1 CoBrA Agent Communication Model [8]

CoBrA uses OWL (Web Ontology Language) to define its ontologies for representing context and modeling. The main ontology associated with CoBrA is SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications). SOUPA [9] is a comprehensive ontology expressed using the Web Ontology Language (OWL), which is based on XML. It is made up of "modular component vocabularies that represent intelligent agents with associated beliefs, desires, and intentions, time, space, actions and events, user profiles, actions, and policies for security and privacy."

As shown in Figure 2.2, SOUPA includes a collection of ontology documents and their interrelationships. The ontology documents are divided into two sets, SOUPA Core and SOUPA Extension. Each document declares the vocabulary for a specific entity such as an agent, device or person.

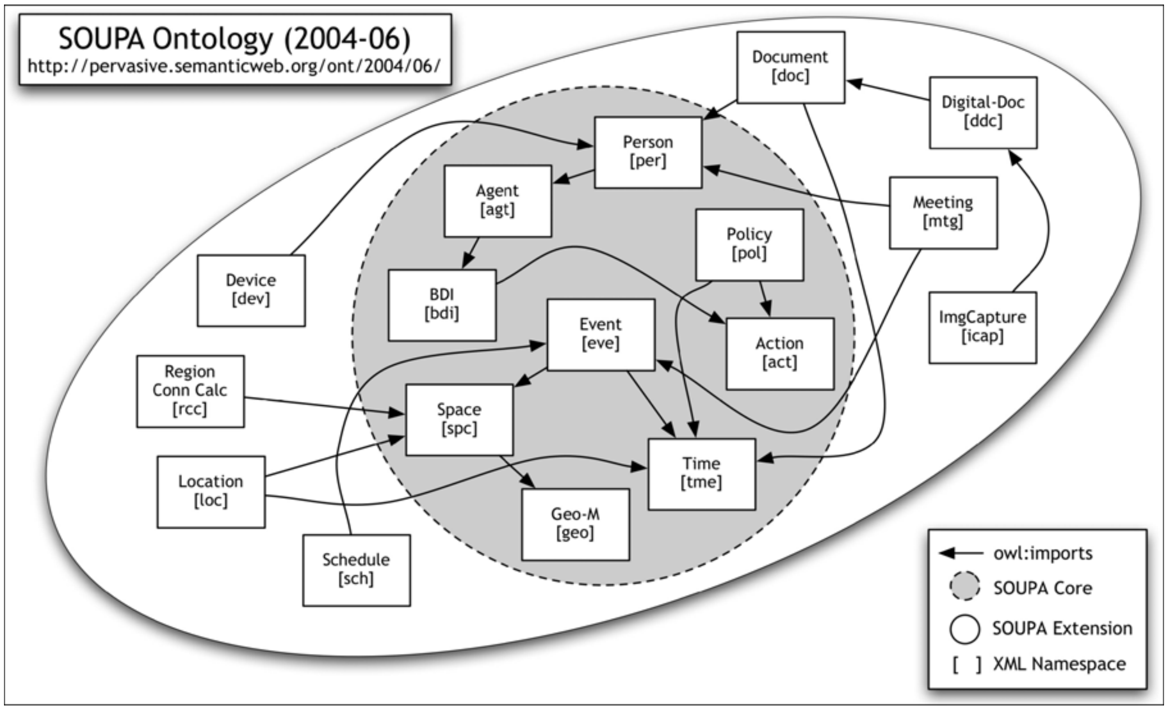


Figure 2.2 SOUPA [9]

CoBrA uses a server-centric architecture requiring context broker servers, whereas our framework does not require any sort of dedicated servers. SOUPA defines a large and complex vocabulary spanning a wide range of entities including people, space and time, geography, policies, locations and several others. Our ontology model uses a simpler, lightweight approach which allows ontologies to be built to suit their environments while avoiding the large vocabulary and complex interrelationships found in SOUPA.

2.2.2 PersonisAD

The PersonisAD [5] (Distributed, Active, Scrutable Model Framework for Context-Aware Services) framework is built on a consistent mechanism for scrutable modeling of people, sensors, devices and places.

The framework includes a context model and a set of operations which facilitate application-model interaction (illustrated in Figure 2.3).

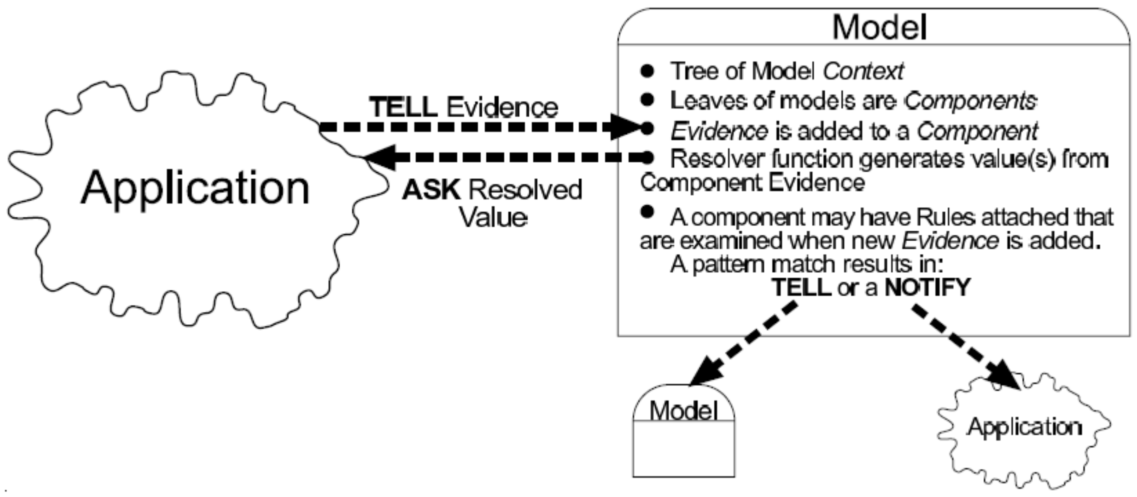

Application
TELL Evidence
ASK Resolved Value
Model
Tree of Model Context
Leaves of models are Components
Evidence is added to a Component
Resolver function generates value(s) from Component Evidence
A component may have Rules attached that are examined when new Evidence is added.
A pattern match results in:
TELL or a NOTIFY
Model
Application

Figure 2.3 PersonisAD interaction between application and model [5]

The context model is organized as a tree containing context attributes (components) of the entities being represented. The general mechanism of PersonisAD is to collect evidence, which is the data used to calculate or resolve component values. Evidence for a location component might include data from GPS and RFID sensors, for example. A GPS sensor would supply latitude/longitude coordinates while an RFID sensor would provide building/room names. A resolver function consults evidence and returns component values which can very depending on the type of request.

PersonisAD applications interact with the context model trees using a set of simple operations including “tell”, “ask” and “notify”. The “tell” operation is used to supply a model with evidence for a component. Applications use the “ask” operation to retrieve component values which are generated by a resolver function using available component evidence. The “notify” operation serves to notify applications when new component evidence is added.

Context information is stored on “model servers” which PersonisAD applications locate using Bonjour [4] service discovery. Communication between applications and model servers uses JSON (JavaScript Object Notation) over HTTP. The interactions between applications and model servers are illustrated in Figure 2.4.

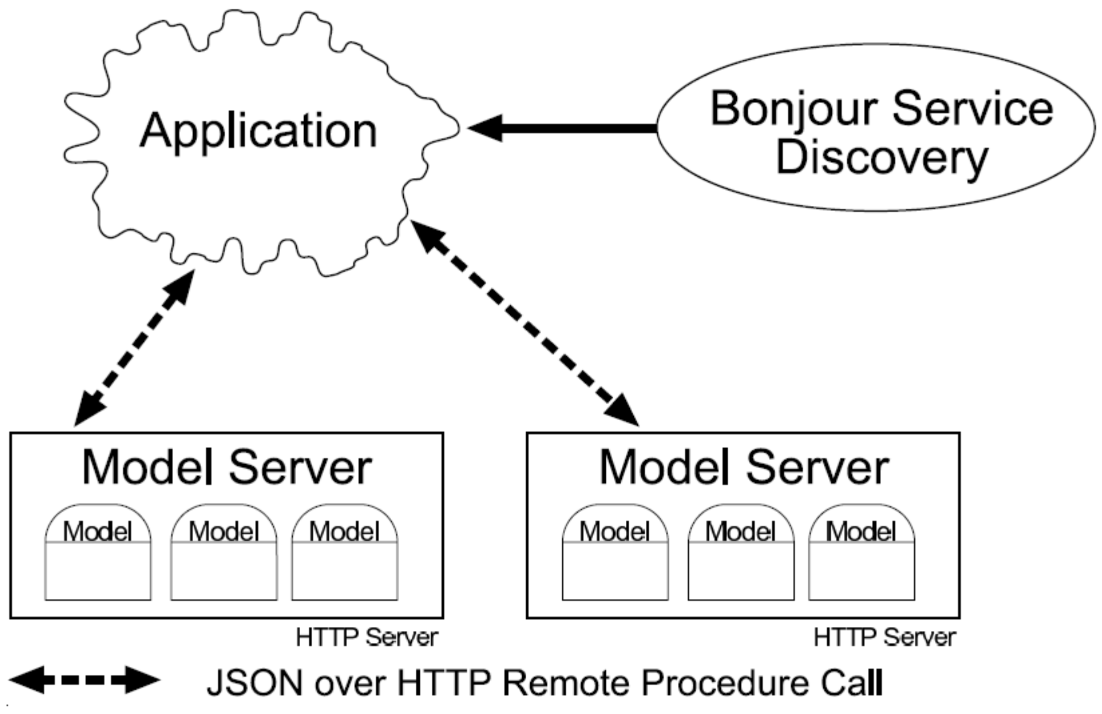
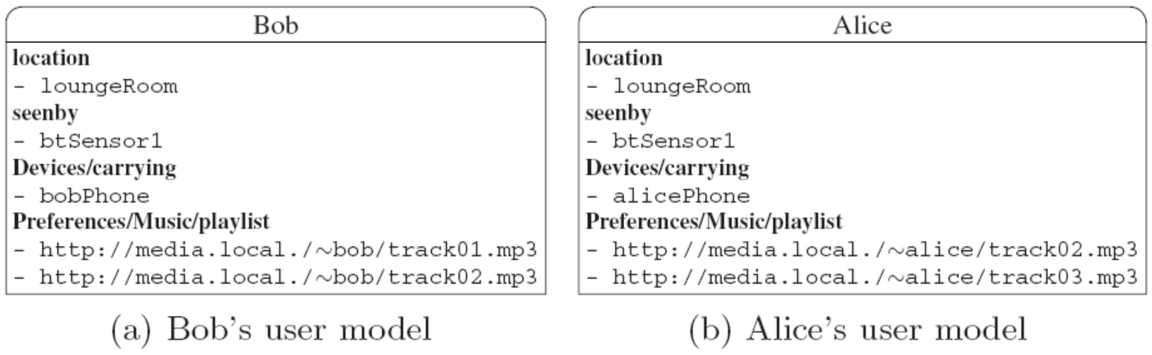


Figure 2.4 PersonisAD interaction between application and model servers [5]

PersonisAD applications use information from models combined with rules to provide context-aware services. One such example is the MusicMix application built on PersonisAD which selects and plays music according to the preferences of people detected within a specified listening area. A pair of simplified user models is illustrated in Figure 2.5. In this example, the MusicMix application would determine that Bob and Alice are located in the ‘loungeRoom’ listening area and have a common music preference of ‘track02.mp3’. The application could then adjust the playlist in the lounge room to accommodate both Bob’s and Alice’s music preferences.


Bob
location
- loungeRoom
seenby
- btSensor1
Devices/carrying
- bobPhone
Preferences/Music/playlist
- http://media.local./~bob/track01.mp3
- http://media.local./~bob/track02.mp3
(a) Bob's user model
Alice
location
- loungeRoom
seenby
- btSensor1
Devices/carrying
- alicePhone
Preferences/Music/playlist
- http://media.local./~alice/track02.mp3
- http://media.local./~alice/track03.mp3
(b) Alice's user model
Figure 2.5 PersonisAD simplified user models [5]

PersonisAD has the same general goal as our framework, namely context management. Its focus, however, is different. PersonisAD focuses on the timeliness of context information and uses a rule-based approach. It also models a wider range of entities including people and locations. Our framework, on the other hand, focuses on providing a flexible and adaptable context management scheme based on appropriate standards. PersonisAD's dependence on model servers is also server-centric (similar to CoBrA's context broker approach) whereas our framework is not.

2.2.3 Delivery Context Ontology

The Delivery Context Ontology is a W3C Working Draft providing a "formal model of the characteristics of the environment in which devices interact with the Web or other services. [31]" The ontology is a class hierarchy formally specified in OWL (Web Ontology Language) containing a large number of disjoint classes. Each class in the ontology is associated with properties defined by a set of fields listed in Table 2.1.

Field Name	**Description**
Name	Formal name of the property
Type	Type of the property (either Datatype or Object).
Description	Description of the property
Restriction(s)	Property restrictions, including cardinality and valid values.
Subproperty(ies)	Subproperties of the property

Table 2.1 Delivery Context Ontology, Property Fields

Each class in the ontology represents a specific entity-type such as device, location or unit of measure. For example, the “Camera” class represents a camera device. Its properties are listed in Table 2.2. Unfortunately, we had already completed our ACMF (which incorporates its own ontology specification) when the Delivery Context Ontology working draft was first published.

Property Name	**Description**
aspectRatio	Camera’s default aspect ratio
cameraEnabled	Indicates whether camera is enabled
cameraResolution	Rolution in pixels
imageFormatSupport	Image formats supported
pixelAspectRatio	Aspect ratio associated with a single pixel
videoFormatSupport	Video formats supported

Table 2.2 Delivery Context Ontology Camera Class Properties

The Delivery Context Ontology bears many similarities to our Domain Ontology Specification (Section 2.2.3). The concept of a class hierarchy representing entities which contain properties is apparent in both models. The fields used to represent properties themselves are also very similar.

2.3 Ad-Hoc Networking and Device Communication

An important aspect of pervasive computing is that devices have the ability to discover and communicate with each other, without the need for an underlying network infrastructure. This section discusses MANETs, followed by an overview of Zeroconf.

2.3.1 MANETs

The term MANET refers to any network made up of mobile nodes that use a wireless interface to exchange data without the need for an underlying network infrastructure [6]. MANETs are critical to pervasive computing because devices must be able to communicate with each other in varying environments, many of which will not be equipped with the hardware necessary to supply a network infrastructure. For example, two users walking down the street may wish to exchange contact information using their PDAs. These users should not have to rely on the presence of a network infrastructure to perform a simple data exchange between their devices.

An important aspect of MANET is its ability to adaptively form networks based on available hardware capabilities and user requirements. For example, consider a user who has a PDA with Bluetooth and a notebook computer with Bluetooth and Wi-Fi. She wishes to download a file to her PDA from a colleague's PDA that is only equipped with Wi-Fi. Using MANET technology, a network is spontaneously formed between the three devices. As depicted in Figure 2.6, the PDAs can communicate with each other using the notebook computer as a waypoint (note the absence of a router).

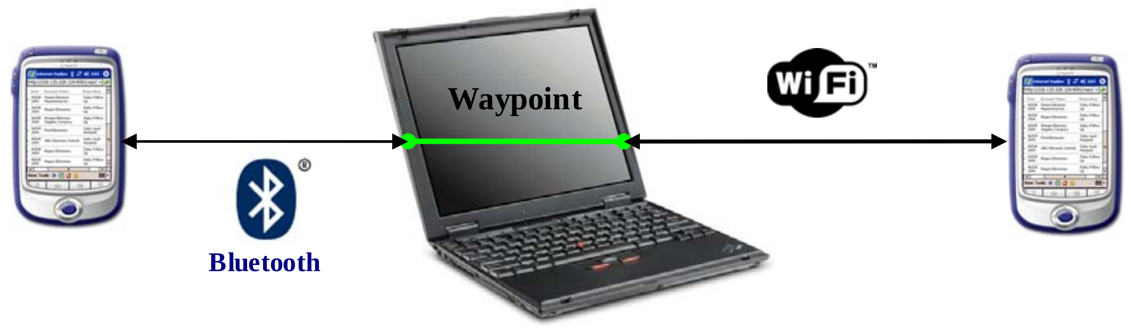


Figure 2.6 Simple MANET example

Another important aspect of MANET is that these network configuration details are invisible at the application layer. In this example, the file transfer application would see all devices on the network the same way, regardless of their interface type.

2.3.2 Zeroconf

Zeroconf is a set of techniques used to create usable IP networks without configuration or special servers” [10]. This is really a specific type of ad-hoc network that utilizes the TCP/IP protocol. Using Zeroconf with Wi-Fi or other wireless standards such as Bluetooth, it is possible to create MANETs.

Zeroconf greatly simplifies networking for the end-user by taking care of all the background details, which is a goal consistent with the pervasive computing paradigm. It is made up of a combination of three technologies, namely link-local addressing, Multicast DNS, and DNS Service Discovery [11].

Link-local addressing allows devices to select their own IP address without the need for a DHCP server. An IP address is chosen at random in the range between 169.254.1.0 to 169.254.254.255 (providing a total of 65,024 possible addresses). The address is then tested on the network to ensure that it is not in already in use. If the address is already taken, then another random IP address is chosen. The process repeats until an unused address is found. Since Zeroconf is designed for small ad-hoc networks where fewer than 2% of the available IP addresses have been assigned, the link-local addressing method usually succeeds in obtaining an IP address within the first one or two tries.

Multicast DNS (mDNS) allows devices to specify their own unique name without the need for a DNS server. While link-local addressing is a good first step, human users prefer to use names rather than numeric addresses when referring to computers and devices. Multicast DNS lets device owners specify a unique name that maps to their device’s IP address. Even if the IP address changes, the mDNS name remains the same.

Multicast DNS functions similarly to standard DNS, with some important differences. Instead of relying on centralized DNS name servers, mDNS shares the responsibility across all devices on the network. Upon entering the network, an mDNS enabled device probes the system to ensure its name is unique, then announces itself by multicasting its mDNS information. Multicast DNS names typically use the top-level domain name ".local" in order to distinguish themselves from standard DNS names.

DNS Service Discovery rounds out the Zeroconf technology suite by providing a distributed service discovery mechanism. This is the mechanism that allows users (or devices) to discover what services are available on the network without having any prior knowledge of them (in terms of names, IP addresses, or other such information). An important distinction is that DNS Service Discovery identifies services, rather than devices. A single device with one IP address and unique name may offer one or more services. This service-based approach offers greater flexibility than would simply being able to discover devices, since users are more interested in what a device can do for them than technical details.

Apple Inc. has developed a Zeroconf implementation known as Bonjour [4], which enables automatic discovery of devices on IP networks. Bonjour ensures that devices are able to do three essential things: Allocate IP addresses without a DHCP server, translate between names and addresses without a DNS server, and locate or advertise services without using a directory server. Bonjour is currently available on both Macintosh and Windows platforms.

Chapter 3

Adaptable Context-Management Framework

In this chapter we discuss our Adaptable Context-Management Framework (ACMF) and how it serves as a foundation for creating context-managed environments (CMEs). An adaptable context management system is able to automatically adjust to changes in its environment, for instance users disconnecting and reconnecting to the pervasive environment or a user moving around within the environment. An adaptable context management system supports context-awareness through the automatic discovery of changes in the environment and enhances usability by insulating users from the changes by automatically adjusting to them. An adaptable context management system is able to distribute processing and communication tasks based on device capability levels, for instance allowing more powerful devices to act as proxies for limited devices.

We envision a CME as a physical space (domain) containing one or more discrete smaller spaces (regions) where various context-aware electronic devices may move about freely and interact with one-another in useful ways while requiring minimal conscious human effort. Devices within a CME must have some means of storing and sharing context information.

Our framework consists of an entity specification which provides a logical representation of physical spaces and devices, a context model which provides a specification for storing context information, and context-exchange protocols which specify how context information is exchanged between devices. The framework's components are illustrated in Figure 3.1.

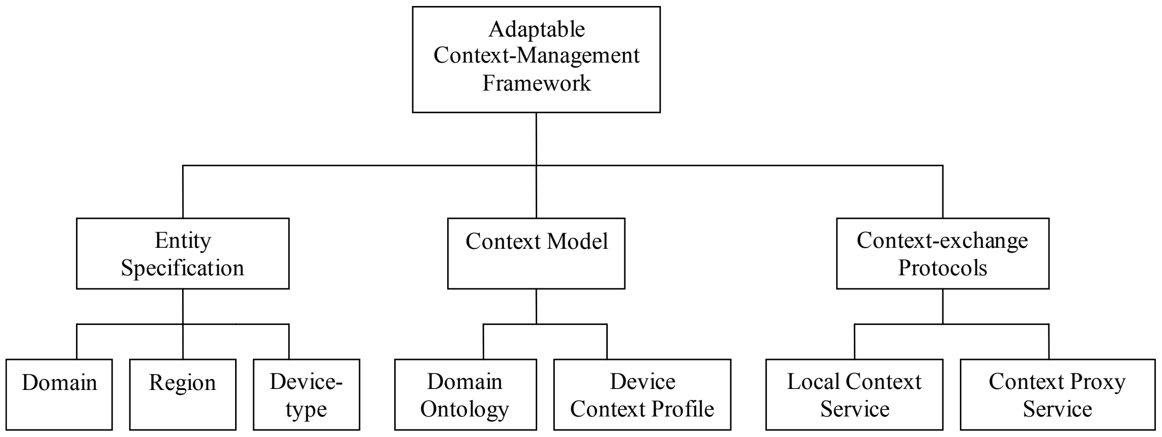


Figure 3.1 Adaptable Context-Management Framework

Figure 3.2 illustrates how the framework's components fit together, showing two physical devices that have context profiles corresponding to a common domain ontology. A middleware platform serves as the mechanism for implementing context-exchange protocols.

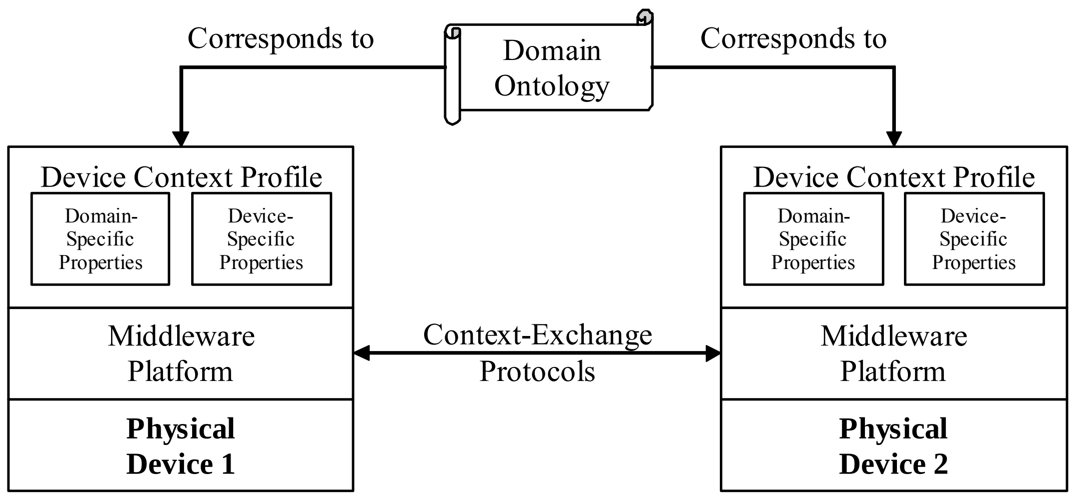


Figure 3.2 ACMF components

3.1 Entity Specification

The entity specification defines three types of entities (domain, region and device-type) that are used to logically represent physical spaces and devices. A CME must consist of exactly one domain, a set of one or more regions and a set of one or more device-types. A domain is a logical representation of a physical space (such as a building) containing regions and device-types. Each region represents a discrete physical space that exists within a domain (such as a room in a building) and each device-type represents a specific type of device (such as a laser printer) that at any point in time resides in a region within the domain.

We portray our framework using an example called the Computer Science Building (CSB) scenario. In this scenario we use the entity specification to represent a university computer science building. The domain represents the building itself, with regions representing its rooms and device-types representing the types of devices that can be found within the building. The CSB scenario's entities are listed in Table 3.1. Entities are used when creating a domain ontology and device context profiles as discussed in the following section.

Entity Type	**Name(s)**
Domain	Computer Science Building
Region	Conference Room Lecture Hall Main Office Lab
Device-type	Data Projector Laser Printer PDA Smart Thermostat

Table 3.1 Computer Science Building CME Entity List

3.2 Context Model

The context model consists of a domain ontology specification and a device context profile specification which together provide a way to specify the context of the entities in a CME.

A domain ontology specifies a single CME (domain) in terms of its regions and device-types. A device context profile is a collection of properties that describe a particular device's contextual state. Devices within a domain use context profiles to share their context information with one another. Each device must have a context profile schema (device schema) from which it generates its context profile. Each device schema must in turn be based on a domain ontology.

The context model is illustrated in Figure 3.3 which shows how a domain ontology represents a physical space while a device context profile represents a physical device. It also illustrates the relationships between a domain ontology schema, domain ontology, device schema, and device context profile. The remainder of this section describes how domain ontologies and device context profiles are specified.

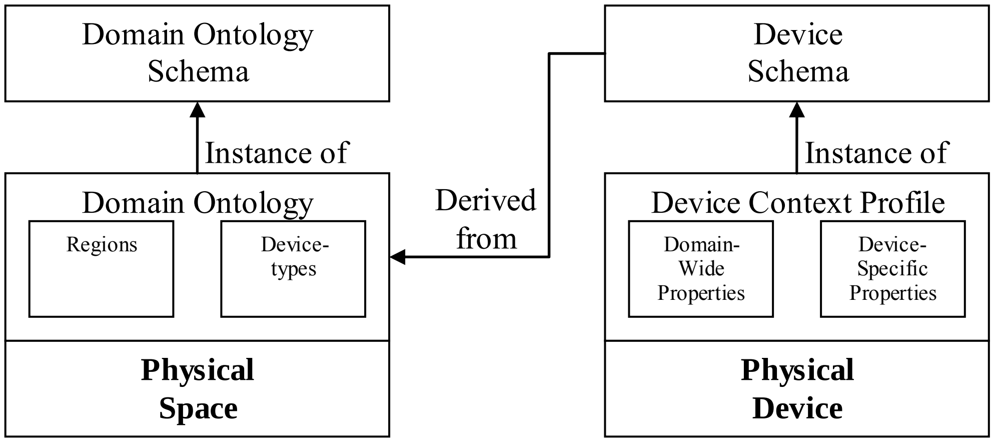


Figure 3.3 Context model

3.2.1 Domain Ontology Specification

Domain ontologies provide a common vocabulary and semantic structure to be used by all devices within a CME (domain). They are defined as XML instance documents. The framework includes a master domain ontology schema (included in Appendix A) which can be used to create domain ontologies. A domain ontology explicitly defines a domain, which contains a set of one or more regions and a set of one or more device-types. Domains and regions are specified as simpleType XML Elements of type “string”. Device-types are more complex, each having a unique name and a set of properties. The framework’s master domain ontology schema is illustrated in Figure 3.4 using Altova XMLSpy Content Model View [1].

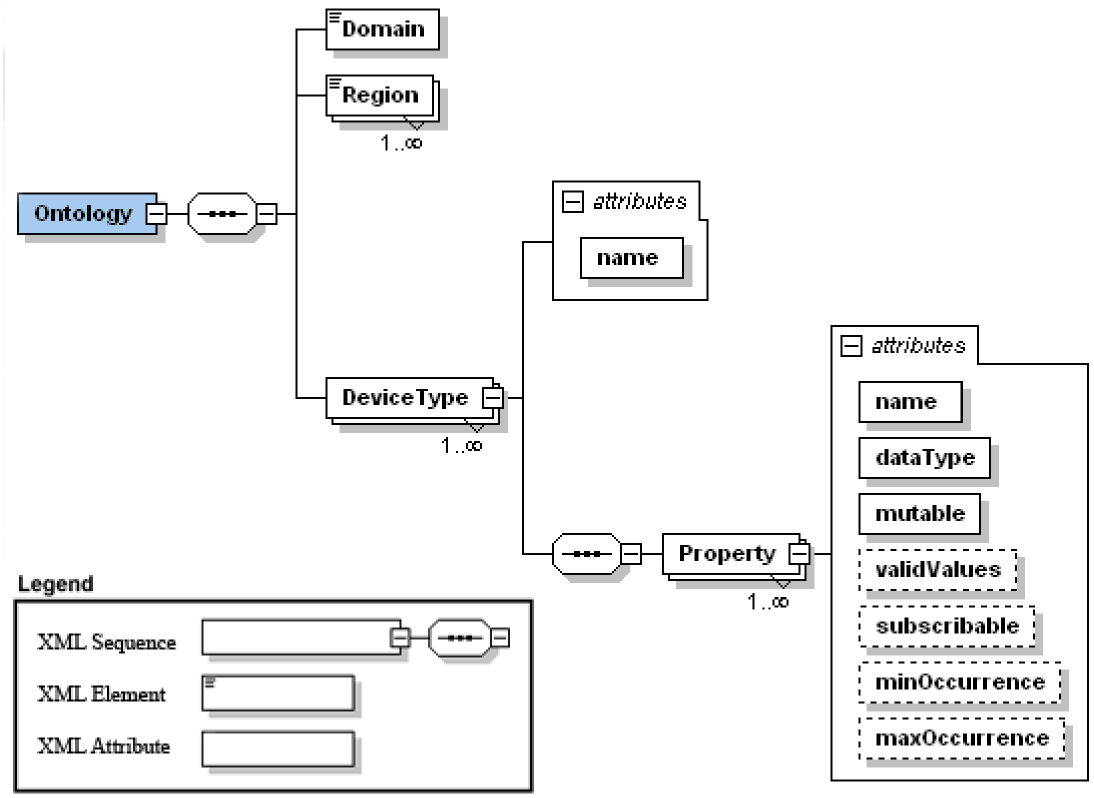


Figure 3.4 Master domain ontology schema

The “Domain” Element specifies the domain being described, and should be given a name that appropriately reflects it (e.g. “ComputerScienceBuilding”). The “Region” Element specifies the region(s) that exist within the domain being described. The ”DeviceType” Element represents the possible device-type(s) that may exist within the domain being described. It contains an XML “name” attribute representing the device-type’s name and a set of additional “Property” elements representing its properties. Each Property Element in turn contains a set of attributes which contain property-specific parameters. Each Property must include a “name”, “dataType”, and “mutable” parameter. The remaining parameters are optional (as shown by the dotted outline). These parameters are used to qualify each property and are listed in Table 3.2. They are explicitly declared in the domain ontology schema found in Appendix A. Whenever an optional parameter is omitted, its default value is assumed.

Parameter Name	**Type**	**Description**	**Allowed Values**	**Req'd**	**Default Value**
name	string	Name of the property	{any}	Yes	n/a
dataType	property Type	The property's data type	"boolean", "integer", "string"	Yes	n/a
mutable	boolean	Indicates whether the property's value may change over time	"true", "false"	Yes	n/a
ValidInteger-Values	integer-List	Provides a set of allowable integer values to constrain the property	{any}	No	{}
ValidString-Values	string-List	Provides a set of allowable string values to constrain the property	{any}	No	{}
subscribable	boolean	Indicates whether the property supports notifications[1]	"true","false"	No	"false"
minOccurrence	nonNegativeInteger	Minimum number Of instances (values)	{integer >= 0}	No	1
maxOccurrence	allNNI	Maximum number Of instances (values)	{integer >= 1}, "unbounded"	No	1

Table 3.2 Device-type Property Parameters

All domain ontologies must include the "PervasiveDevice" device-type which contains domain-wide properties (properties common to all devices within a domain). Furthermore, this device-type must include the "DeviceType" and "Location" properties which are respectively constrained to the domain's available device-types and regions using the "ValidStringValues" parameter. All other device-types should contain device-specific properties (properties unique to a specific device-type). Each device-type (other than "PervasiveDevice") is a collection of the properties necessary to represent one specific type of physical device (such as a laser printer).

[1] Notifications are discussed in Section 3.3.3.

Our CSB scenario includes the “ComputerScienceBuilding” ontology which defines the “ComputerScienceBuilding” domain along with its regions, the required “PervasiveDevice” device-type as well as the scenario’s four available device-types. Tables 3.3 and 3.4 list the “PervasiveDevice” and “LaserPrinter” device-types’ properties and their parameters. The complete “ComputerScienceBuilding” ontology XML instance document is included in Appendix A.

Property Name	**Parameters**
UniqueName	dataType=”string”, mutable=”false”
DeviceType	dataType=”string”, mutable=”false”, ValidStringValues=”Data_Projector Laser_Printer PDA Smart_Thermostat”
Owner	dataType=”string”, mutable=”true”
Location	dataType=”string”, subscribable=”true”, mutable=”true” ValidStringValues=”Conference_Room Lecture_Hall Main_Office Lab”
HasDisplayScreen	dataType=”boolean”,mutable=”false”

Table 3.3 CSB Scenario PervasiveDevice Device-Type Properties

Property Name	**Parameters**
CanPrintInColour	dataType=”boolean”, mutable=”false”
IsOnline	dataType=”boolean”, mutable=”true”
PaperRemaining	dataType=”integer”, mutable=”true”

Table 3.4 CSB Scenario LaserPrinter Device-Type Properties

Each ontology document should be accompanied by a prose description document in order to help clarify domain semantics. For example, the “ComputerScienceBuilding” ontology does not describe the semantics behind the “ValidStringValues” specified for the “Location” property (nor would it be appropriate). In this case we have included a prose description document in Appendix A which contains a floor plan of the building depicting the room locations and their corresponding “Location” property values.

3.2.2 Device Context Profile Specification

A device context profile is a collection of property values that describes a device's contextual state. In order to generate a context profile, a device must have a corresponding context profile schema (device schema). Domain ontologies contain the property names and parameter information that is used to create device schemas.

Device Schema Specification

A device schema must correspond to a domain ontology and can only be used within the domain defined by that ontology. Devices may have schemas for multiple domains and upon moving between domains must switch schemas accordingly. If a device has no schema for a particular domain, it simply disables its context-management features. Domain determination occurs through some implementation-specific means (using RFID room tags, for example). In the simplest cases such as with the CSB scenario, devices are designed to be used in a single domain and therefore have only one device schema and no means of domain-detection.

A device schema defines a collection of domain-wide and device-specific properties which is a subset of those contained in the corresponding domain ontology. Each device schema forms a class hierarchy. The top level (root node) is a type defining the entity being described and must be named "Device". The "Device" node contains a set of types which represent the device-types found in the corresponding ontology. Device-types are declared as GEDs (global element declarations) in XML Schema, and device properties as XML child elements within the type definitions of the GEDs. Each device property element includes values for the property parameters introduced in Section 3.2.1.

Device Schema Derivation

Device schemas are derived from domain ontologies according to the following mapping. Each “DeviceType” Element in an ontology maps to an element declaration representing a device-type in the corresponding device schema, according to the following rules:

1. The element must be a Global Element Declaration (GED).

2. The value of the element’s “name” attribute must correspond to the “name” attribute of the DeviceType Element.

3. The element must be declared as a complex type consisting of an XML Schema sequence. It is sufficient that the complex type be defined as a local, anonymous type on the element.

The “PervasiveDevice” device-type is required and must be mapped to all device schemas. All other device-types are optional and should be mapped based on which properties will be supported by the device for which a schema is being created.

Each ontology “Property” Element maps to an element declaration representing a device property in the device schema, according to the following rules:

1. The element must be declared as a child of the element which represents its device-type (this is why elements representing device-types must be declared as complex types as stated in rule #3 above).

2. The value of the element’s “name” attribute must correspond to the "name” attribute of the Property Element.

For example, each “Property” Element which is part of the “PervasiveDevice” DeviceType in the ontology must appear as a child of the “PervasiveDevice” element in the device schema. Each “Property” Element contains a set of Attributes which map to attributes in the corresponding schema element declaration. The mapping of “Property” Element Attribute values is listed in Table 3.5 and an example mapping of a “DeviceType” and “Property” from an ontology to a device schema follows in Figure 3.5.

Ontology “Property” Element Attribute	**Corresponding Schema element attribute**
name	name
dataType	type
mutable	meta:mutable
subscribable	meta:subscribable
ValidStringValues	meta:ValidStringValues
ValidIntegerValues	meta:ValidIntegerValues

Table 3.5 Mapping of “Property” Element Attribute Values

Ontology “DeviceType” and “Property” Elements

```
<DeviceType name=”PervasiveDevice”>
  <Property name="Location" dataType="string" meta:subscribable="true"
meta:mutable="true"    meta:ValidStringValues="Conference_Room Lecture_Hall Main_Office
Lab"/>
</DeviceType>
```

Corresponding Device Schema Declarations

```
<xs:element name=”PervasiveDevice”>
  <xs:complexType>
    <xs:sequence>
      <xs:element name=”Location” type=”xs:string” meta:mutable=”true”
      meta:subscribable=”true” meta:ValidStringValues=”Conference_Room Lecture_Hall
Main_Office Lab”/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 3.5 Example mapping from ontology to device schema

Figure 3.6 illustrates an example device schema for the CSB scenario’s Laser Printer device, shown using Altova XMLSpy Content Model View [1]. The complete device schema document and a sample corresponding instance document are included in Appendix A.

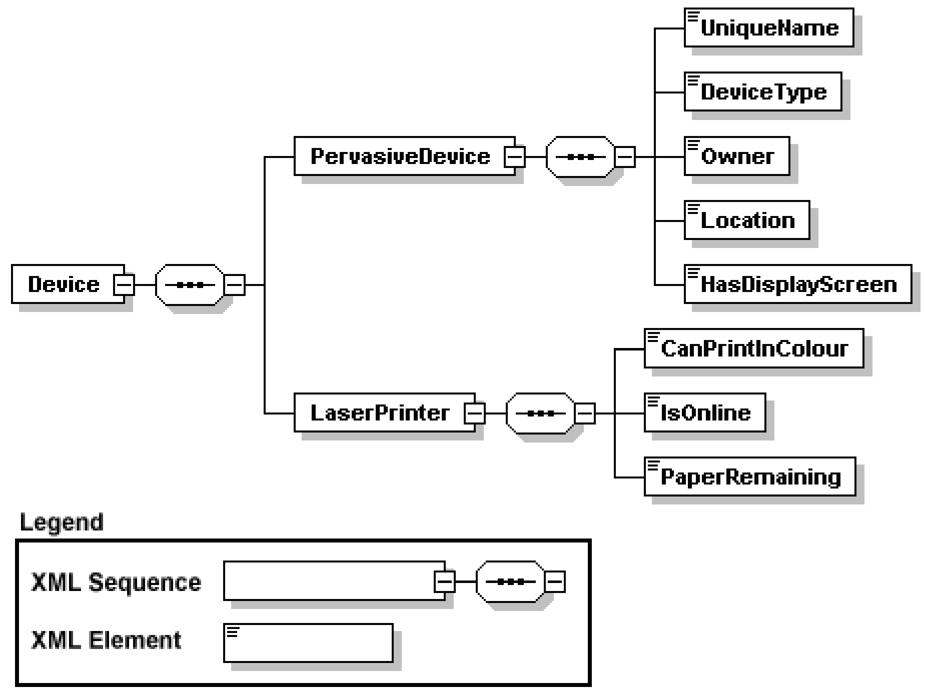


Figure 3.6 Laser printer device schema

3.3 Context-Exchange Protocols

This section presents a set of protocols which govern how context information is exchanged between devices in CMEs. CMEs must mandate some form of MANET as discussed in Section 2.3.2. Each environment must also provide a master ontology as described in Section 3.2.1.

3.3.1 Device Roles

Devices must support either a server or client role, or a combination of both. The server role defines the elements necessary for a device to share its context information, while the client role defines the elements necessary for a device to access the context information of other devices.

Devices supporting the server role must provide a device schema conforming to the context model specification described in Section 3.2.2. The device schema must be derived from an ontology which corresponds to the environment in which that device will be used. The device must also advertise its universal resource identifier (URI) using whatever MANET implementation the environment supports. The URI is a unique identifier used to locate a specific device or service on the network. For example, in an environment supporting Zeroconf the URI would be advertised using DNS service discovery. Each device supporting the client role must support the MANET protocol mandated by its intended operating environment. All device interactions are client-initiated, with the exception of notifications which are server-initiated.

3.3.2 Device Interaction States

From the client role perspective, devices move through a four-state "Interaction" sequence. These states are "Initialized", "Device Aware", "Context Aware" and "Interacting" which are illustrated using a UML State Diagram in Figure 3.7.

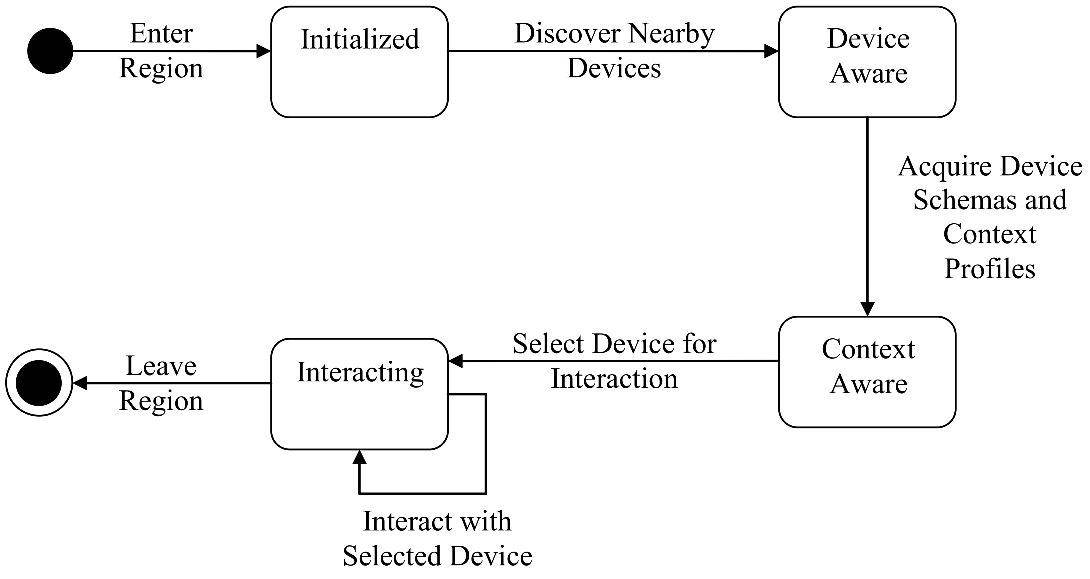
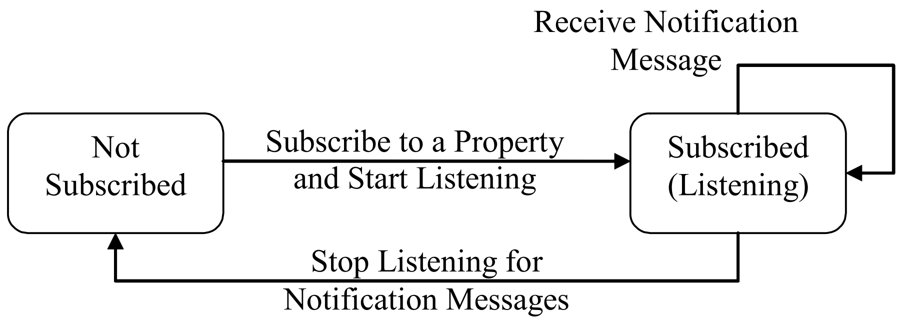

Enter Region
Initialized
Discover Nearby Devices
Device Aware
Acquire Device Schemas and Context Profiles
Context Aware
Select Device for Interaction
Interacting
Leave Region
Interact with Selected Device

Figure 3.7 Device interaction state sequence diagram

In addition to the "Interaction" state sequence, devices concurrently maintain a "Subscription" state sequence corresponding to whether they are actively listening for property change notification messages (defined later in this section). The two subscription states are "Subscribed" and "Not Subscribed". These states are illustrated using a UML state diagram in Figure 3.8.


Receive Notification Message
Not Subscribed
Subscribe to a Property and Start Listening
Subscribed (Listening)
Stop Listening for Notification Messages
Figure 3.8 Device subscription state diagram

3.3.3 Local Context Service

The transitions in the device interaction state sequence depend on a set of functions which are listed in Table 3.6. Together these state transitions and functions form the local context service which enables a device to share its context information. Any device supporting the server role correspondingly supports the local context service. In this section we discuss how the local context service functions are used in each of the transitions between device interaction states.

Function	**Parameters**	**Description**
Discover	None	Retrieve device URI
Request device schema	None	Retrieve device schema
Request properties	Property name(s)	Retrieve property value(s)
Subscribe to property	Property name	Receive notifications when property value(s) change

Table 3.6 Local Context Service Functions

1. Device Discovery

The first step in the device interaction process is device discovery, where a client retrieves the URIs of all devices within its wireless range. Devices located in the same region as well as devices in other regions may be detected during this step. Devices not supporting the server role may also be detected but will not provide a URI and are thus ignored. Figure 3.9 is a standard UML sequence diagram illustrating a scenario where a PDA (client) discovers a nearby Laser Printer (server).

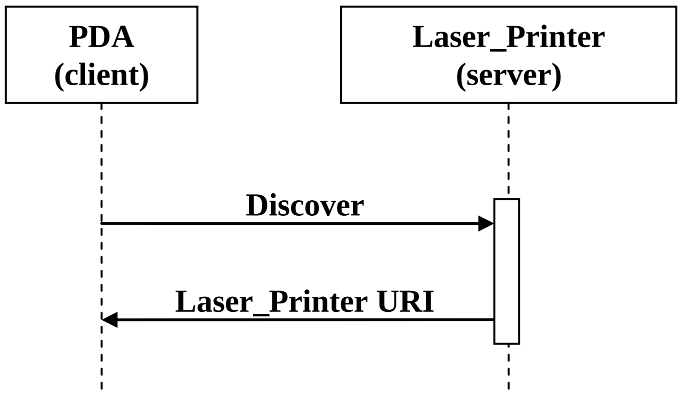


Figure 3.9 Device discovery process

2. Device Schema and Context Profile Acquisition

Following device discovery, a client typically retrieves device schema and context profile information from newly discovered devices. Device schemas are used to interpret device context profile information. This approach allows devices to interact even in cases where the client does not have access to the domain ontology. Using the device schema information, the client issues a request to retrieve the values of all properties that comprise the context profile. It is not necessary to retrieve all of the properties; this flexibility allows clients to retrieve as little or as much information from other devices as desired.

Figure 3.10 is a UML sequence diagram illustrating how the PDA in our scenario retrieves device schema and context profile information from the laser printer.

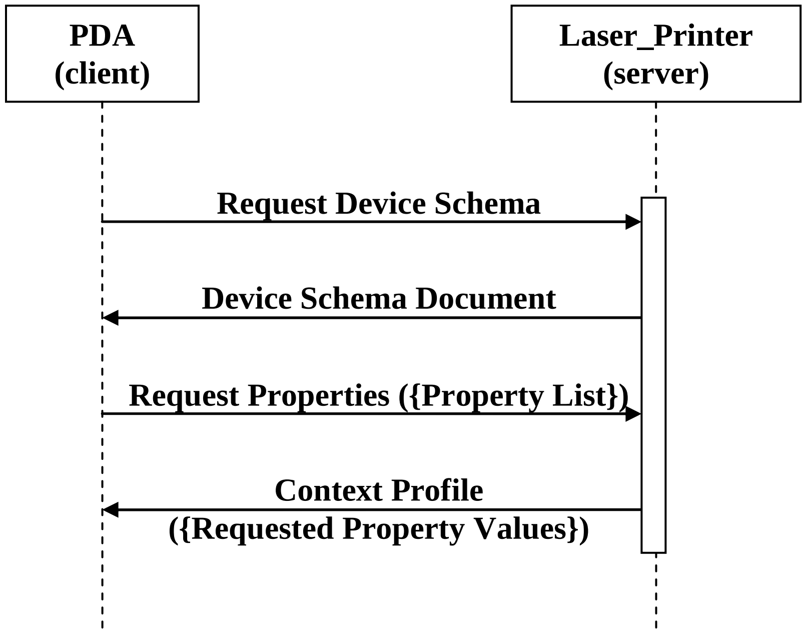


Figure 3.10 Device schema and context profile acquisition

The context profile data retrieved by the client in this scenario is listed in Table 3.7.

Property Name	**Value**
UniqueName	Conference_Room_Printer
DeviceType	Laser_Printer
Owner	IT_Department
Location	Conference_Room
HasDisplayScreen	True
CanPrintInColour	False
IsOnline	True
PaperRemaining	506

Table 3.7 Laser Printer Context Profile

3. Device Selection

Upon obtaining the context profiles of nearby devices, the client typically analyses the context information in order to select a desired device with which to interact. Of particular importance is the “Location” property which indicates a device’s current region. A client will typically only be interested in devices located within the same region and can thus ignore detected devices in other regions.

While our protocols specify the exchange of device schema and context profile information, the mechanism for device selection is left as an implementation decision at the device level. This allows the greatest amount of flexibility as device selection criteria will vary depending on the device. For example, the needs of a stationary Desktop PC will differ significantly from those of a PDA. In some cases device selection will be user-delegated.

For example, in our CSB scenario the PDA (client) has been programmed to interact only with devices located in the same room (region), and to alert the user when new devices become available (such as when entering a new room). Continuing the scenario, a student picks up the PDA and walks into the conference room. Upon detecting that it has entered that room (through some implementation-specific means), the PDA's user interface alerts the student that a new device is available, and hides any devices that may have been present at the student's previous location. Next, the student indicates that she would like to interact with the Laser Printer, which completes the device selection stage.

4. Device Interaction

At this stage a device has been selected and its device schema and context profile have been retrieved. The client device may now proceed to interact directly with the selected server device. Possible device interactions include additional property retrieval, property subscription, and application-level interactions. We present a property retrieval example followed by a property subscription example. Application-level interactions (such as printing a document) are not strictly context-related but are included in the following examples to help illustrate context-related interactions.

Property Retrieval

In this example the student has chosen to use her PDA to interact with a nearby Laser Printer (as illustrated during the device selection stage). The student wishes to print a large 500 page document. She issues a command to send her document to the Laser Printer (an application-level interaction). The PDA retrieves the "PaperRemaining" device-specific property value from the Laser Printer, and compares it with the size of the student's document. It turns out that someone has printed a document sometime after the PDA first retrieved the Laser Printer's context profile, and its "PaperRemaining" value has decreased from 507 to 482. The PDA notifies the student with a dialogue box stating that she must add more paper before her document can be printed. The student then restocks the printer and instructs the PDA to proceed. The PDA checks the Laser Printer's "PaperRemaining" property a final time before sending the document to be printed. These steps are illustrated in Figure 3.11 using a UML sequence diagram.

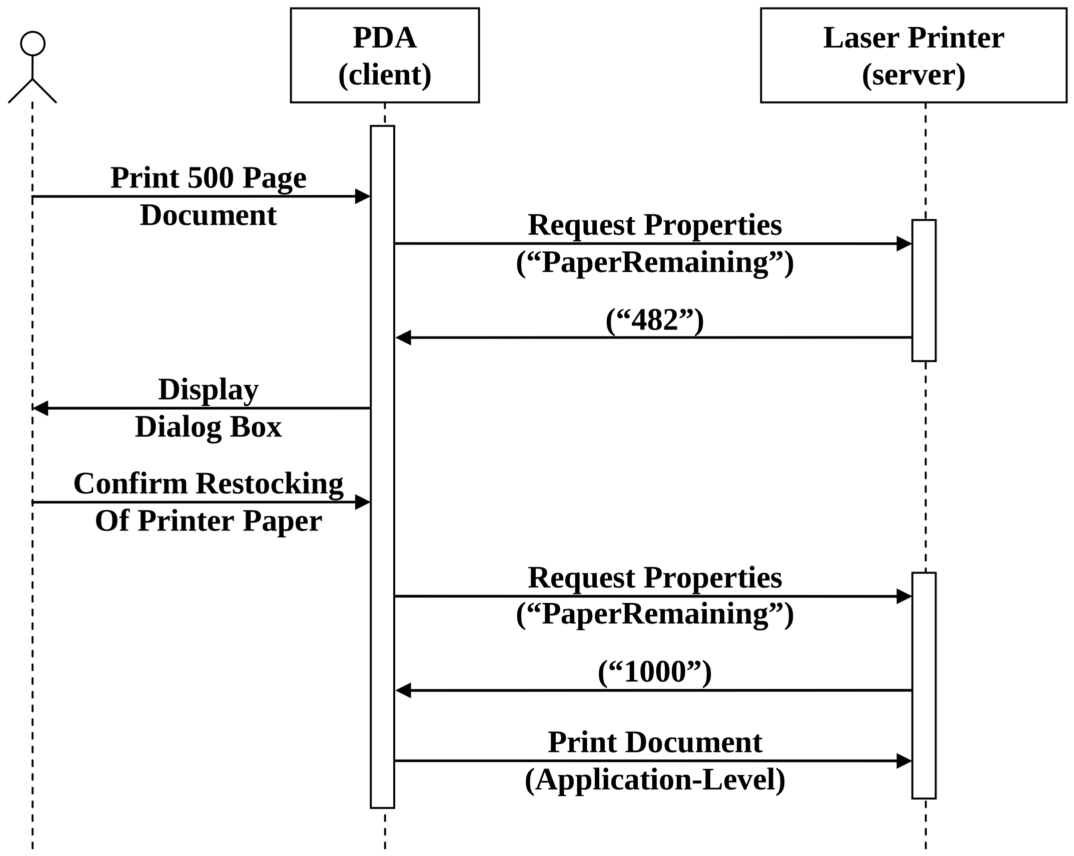

PDA
(client)
Laser Printer
(server)
Print 500 Page Document
Request Properties ("PaperRemaining")
("482")
Display Dialog Box
Confirm Restocking Of Printer Paper
Request Properties ("PaperRemaining")
("1000")
Print Document (Application-Level)

Figure 3.11 Device interaction example

Property Subscription

Property subscription allows a device to receive notification messages whenever a specific property value changes. This is particularly useful when used with the "Location" property because it enables a device to become aware of other devices' movements between regions. Properties that support notification events are identified by the "subscribed" parameter.

When subscribing to a property, the client must provide a URI specifying where the notification messages should be sent. There must be a listener at that location which will accept and process the notification messages. Subscription lifetime is left as an implementation decision but for simplicity we recommend that a subscription should be destroyed after one or more failed attempts to deliver a notification message to a client. This simple policy is based on the assumption that failed delivery would indicate that either the client device has left the environment or is no longer listening for notification messages pertaining to the subscribed property. It is permissible to implement a more robust subscription lifetime policy as needed.

In this example, the student wishes to receive notification messages whenever the Laser Printer goes offline. She uses her PDA's client interface to request a subscription. The PDA must first retrieve the Laser Printer's schema document to determine which of its properties support notifications. Upon confirmation that the "IsOnline" parameter does in fact support them, the PDA begins listening for the notifications, and sends a subscription request to the Laser Printer, specifying the URI of the open port as the subscription destination. Once the subscription is created, the Laser Printer sends notification messages back to the PDA whenever the value of the "IsOnline" property changes, as long as the PDA continues to accept them. If the PDA stops accepting the messages, the Laser Printer destroys the subscription and stops sending the notification messages. These steps are illustrated in Figure 3.12 (UML sequence diagram).

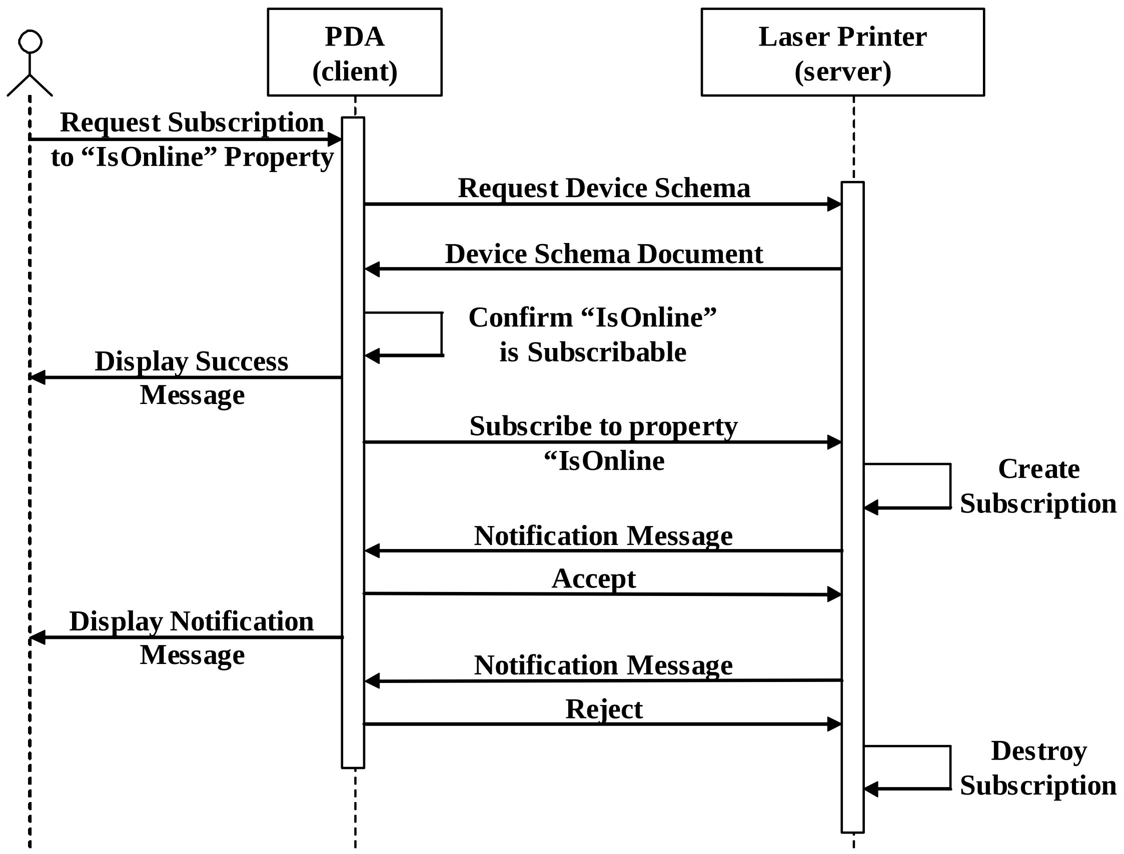


Figure 3.12 Subscription example

3.3.4 Context Proxy Service

The context proxy service enables a device to host and share the context information of other devices. It is used in situations where devices are limited in terms of resources such as storage, processing, bandwidth or battery power. A limited device can submit a copy of its context profile to a more robust device which will in turn relay it to other devices. The context proxy service works as an extension to the local context service and any device supporting the former must also support the latter.

We introduce two additional device roles, the proxy provider role and the proxied device role, in order to help specify the context proxy service. A device is a proxy provider if it supports the context proxy service. All proxy providers are also servers because they must support both the context proxy and local context services. A device is a proxied device if it uses a proxy provider to host its context information. A proxied device must still support the local context service but is able to conserve resources by ignoring other devices' discovery requests. State sequences representing the proxy provider and proxied device roles respectively are shown in Figures 3.13 and 3.14.

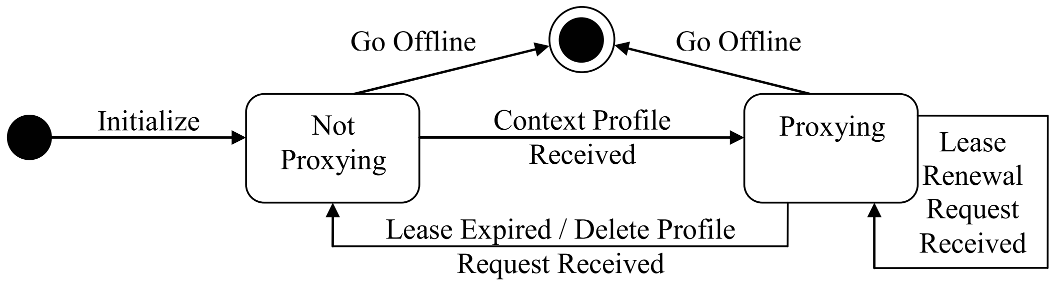

Go Offline
Go Offline
Initialize
Not Proxying
Context Profile Received
Proxying
Lease Renewal Request Received
Lease Expired / Delete Profile Request Received

Figure 3.13 Proxy provider state sequence

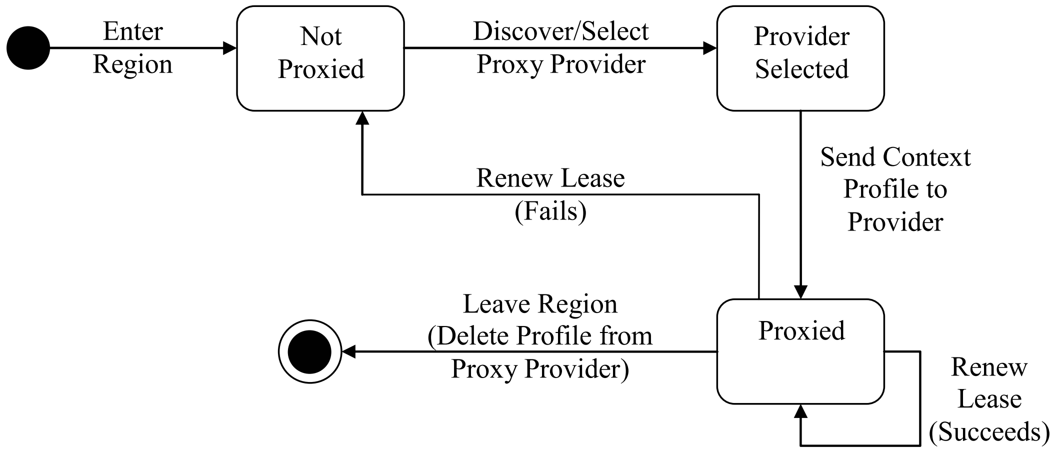

Enter Region
Not Proxied
Discover/Select Proxy Provider
Provider Selected
Send Context Profile to Provider
Renew Lease (Fails)
Leave Region (Delete Profile from Proxy Provider)
Proxied
Renew Lease (Succeeds)

Figure 3.14 Proxied device state sequence

These state sequences depend on an extension to the device schema specification, a set of context proxy service functions and a set of device interactions.

1. Device Schema Extension

The context proxy service uses an extended version of the device schema specification which allows a proxy provider to piggyback/host the context profiles of other devices alongside its own. The extension is specified as a device-type called "ProxyProvider". This device-type represents a logical/virtual entity (as opposed to a physical device) which acts as a container for other devices' context profiles. Any device which acts as a proxy provider must include the "ProxyProvider" device-type in its schema. Since device schemas must correspond to a domain ontology, all domains allowing the use of proxy providers must include the "ProxyProvider" device-type in their ontologies. The device schema for the Laser Printer introduced in the "ComputerScienceBuilding" scenario is illustrated in Figure 3.15, extended to include the "ProxyProvider" device-type.

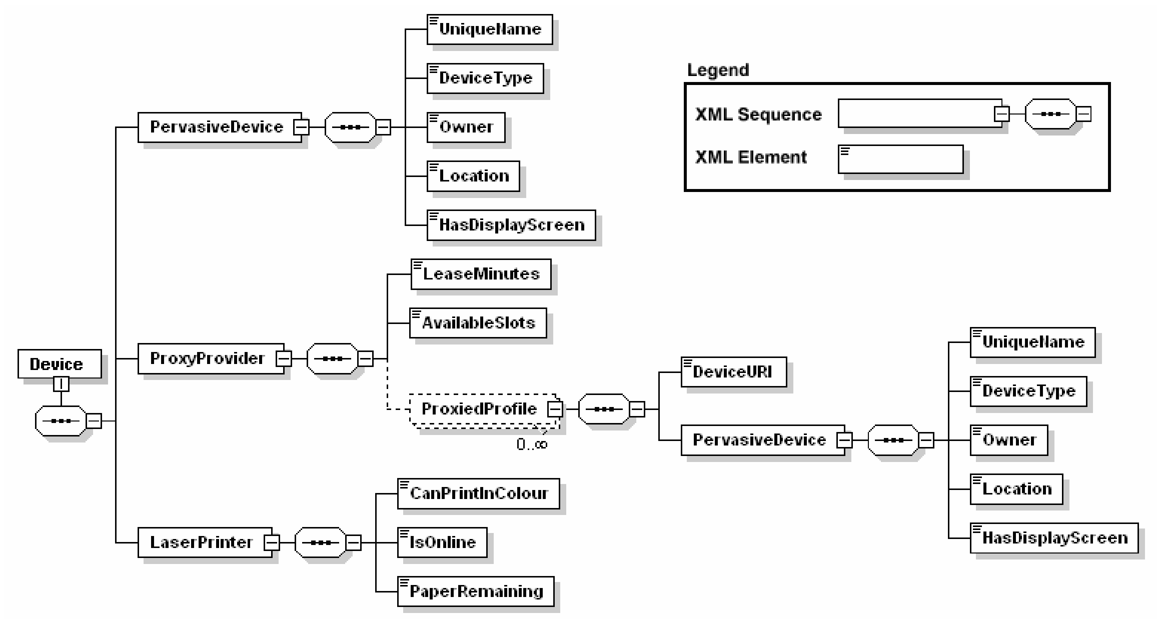


Figure 3.15 LaserPrinter device schema with ProxyProvider

As with all other device-types, "ProxyProvider" is declared as an XML sequence. It contains two XML Element declarations ("LeaseMinutes" and "AvailableSlots") which represent device properties, and an XML Sequence declaration ("ProxiedProfile") which represents the container that is used to manage the context profiles of proxied devices.

The “LeaseMinutes” property represents the minimum amount of time in minutes that the proxy provider will guarantee to retain a proxied profile. If a proxied device does not renew its lease within this time, the proxy provider may delete (discontinue hosting) its copy of that device’s profile. “LeaseMinutes” should be set high enough to minimize unnecessary lease renewals, yet low enough to ensure that lease expiry occurs within a reasonable amount of time. A typical “LeaseMinutes” value would be 60 minutes.

The “AvailableSlots” property represents the number of profiles that the proxy provider is currently willing to accept for hosting. A value of zero indicates that the proxy provider is not accepting profiles.

Each instance of the “ProxiedProfile” sequence contains a single proxied context profile. The sequence has a lower cardinality of zero and an unbounded upper cardinality which means in schema terms that “ProxyProvider” may contain zero or more instances of “ProxiedProfile” at any given time. “ProxiedProfile” in turn contains one property declaration, “DeviceURI”, and one XML sequence declaration, “DeviceProperties”. “DeviceProperties” represents the proxied profile itself, and “DeviceURI” represents the URI of the device to which the profile belongs.

“DeviceProperties” contains a set of domain-wide properties which must match those found in the proxy provider’s “PervasiveDevice” device-type declaration. These properties are ultimately derived from the domain ontology as with all devices. Device-specific properties are not included because they typically change more frequently and are intended to be accessed directly rather than via proxy. Domain-wide properties provide client devices with sufficient information to facilitate device selection. Since domain-wide properties are not proxied, there is no need to proxy device schemas.

2. Context Proxy Service Functions

The context proxy service includes three new device interaction functions which are listed in Table 3.8. These functions facilitate the establishment, maintenance and removal of context profiles stored on a proxy provider.

Function	**Parameters**	**Description**
Send Profile	DeviceURI, {ContextProfile}	Transfer URI and profile to proxy provider
Renew Lease	DeviceURI	Request renewal of proxied profile lease
Delete Profile	DeviceURI	Remove profile from proxy provider

Table 3.8 Context Proxy Service Functions

3. Device Interactions

The context proxy service involves two sets of interactions and a context profile retention (lease) policy. The first set specifies the interactions between a proxied device and proxy provider, including how a proxied device selects a proxy provider and uses it to host its context profile. The second set specifies the interactions between a client device and a proxy provider, including how a client device discovers proxied devices and retrieves their context profiles. The context profile retention policy specifies proxy provider requirements for retaining proxied device profiles.

Interactions between Proxied Device and Proxy Provider

A proxied device must implement some means of discovering and selecting a suitable proxy provider in order to satisfy cases where there is more than one proxy provider available on the network. This is left as an implementation decision at the device level. Typically the proxied device will use a selection algorithm to choose a proxy provider after discovering all available devices and retrieving their schemas in order to identify which ones support the "ProxyProvider" device-type. An example of a simple algorithm is to select the first available proxy provider.

Once a proxy provider has been selected, the proxied device must retrieve the LeaseMinutes property to determine how often it will need to renew its lease, and the "AvailableSlots" property to ensure that there is at least one slot available. The proxied device must then transfer its context profile and periodically renew its lease with the provider. Upon receiving a context profile, the proxy provider adds it to the ProxiedProfile sequence as specified in the device schema extension. Before leaving a region or going offline, a proxied device should remove its profile from the provider. These steps are illustrated in Figure 3.16 and Figure 3.17 using a laser printer as the proxy provider and a smart thermostat as the proxied device.

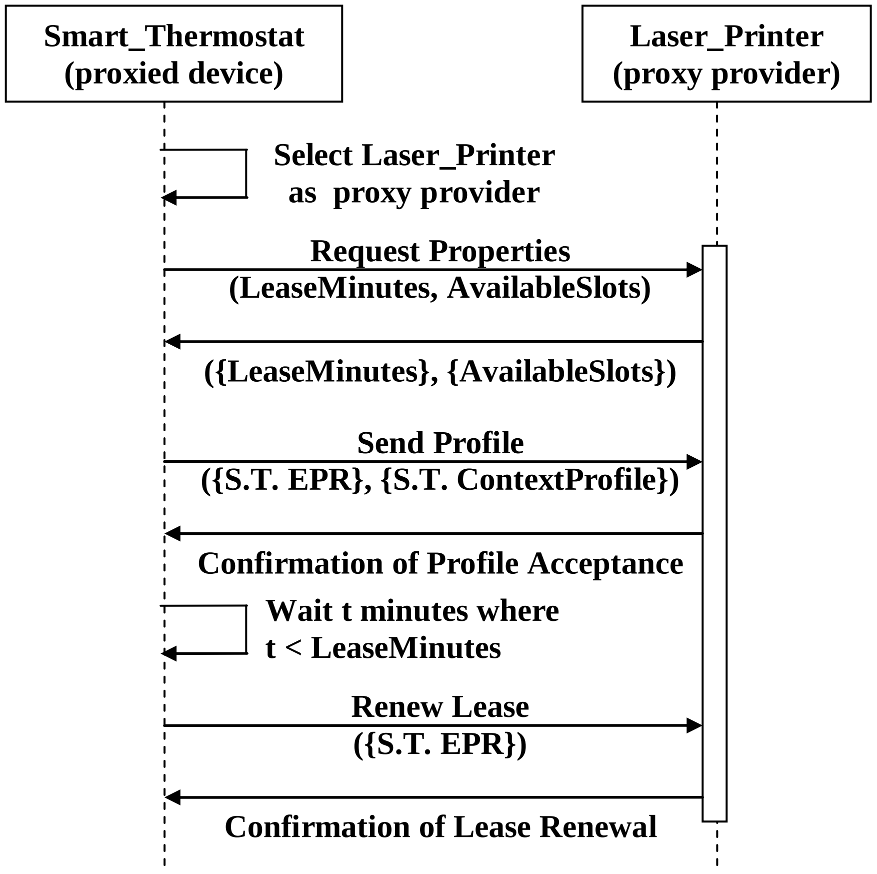

Smart_Thermostat
(proxied device)
Laser_Printer
(proxy provider)
Select Laser_Printer as proxy provider
Request Properties (LeaseMinutes, AvailableSlots)
({LeaseMinutes}, {AvailableSlots})
Send Profile ({S.T. EPR}, {S.T. ContextProfile})
Confirmation of Profile Acceptance
Wait t minutes where t < LeaseMinutes
Renew Lease ({S.T. EPR})
Confirmation of Lease Renewal

Figure 3.16 Proxied context profile insertion and lease renewal

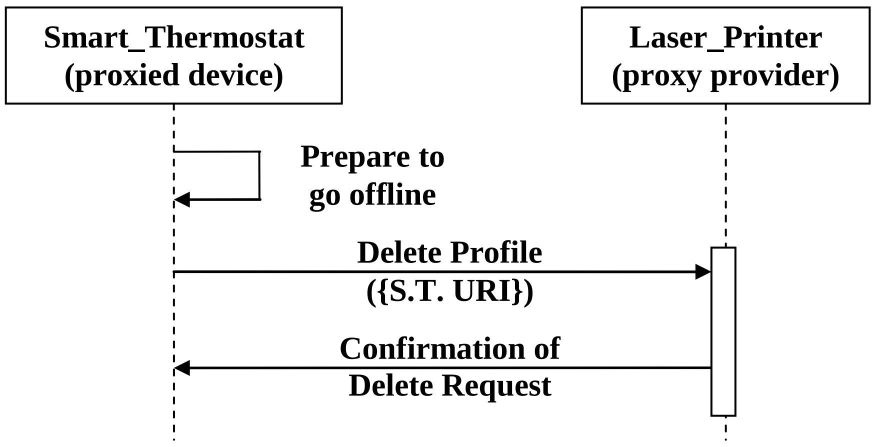

Smart_Thermostat (proxied device)
Laser_Printer (proxy provider)
Prepare to go offline
Delete Profile ({S.T. URI})
Confirmation of Delete Request

Figure 3.17 Proxied context profile removal

Interactions between Client and Proxy Provider

The process of discovering and acquiring the context profiles of proxied devices is somewhat different than that of non-proxied devices. In order to ensure that proxied devices are discovered, client devices interacting within a domain supporting the context proxy service must include the following additional steps when performing device schema and context profile acquisition as specified in Section 3.3.3.

For each discovered device which includes the “ProxyProvider” device-type in its schema, the client retrieves the ProxiedProfile sequence using the Request Properties function. This proxy provider responds by returning any proxied context profiles it is hosting as well as their associated device URIs. Figure 3.18 illustrates how a client obtains proxied device URI and context profile information from a proxy provider.

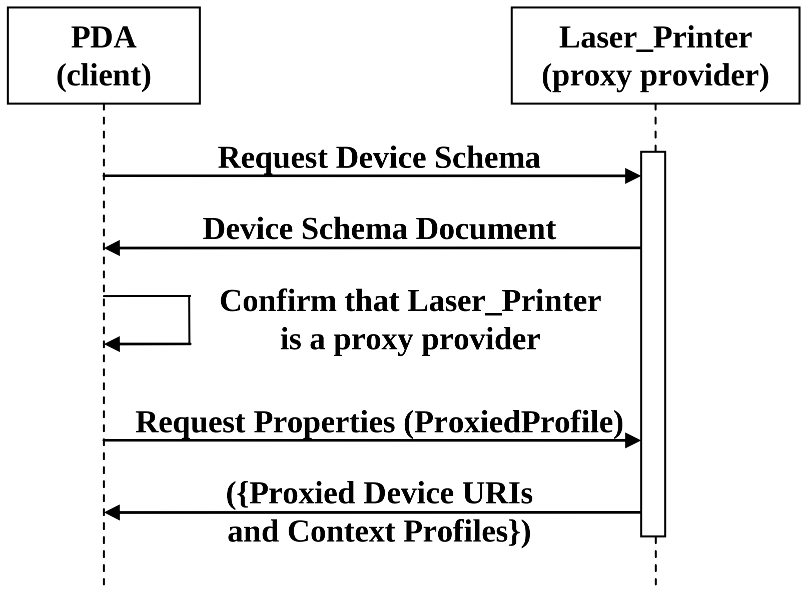

PDA
(client)
Laser_Printer
(proxy provider)
Request Device Schema
Device Schema Document
Confirm that Laser_Printer is a proxy provider
Request Properties (ProxiedProfile)
({Proxied Device URIs and Context Profiles})

Figure 3.18 Proxied device URI and context profile acquisition

Context Profile Retention Policy

Under normal circumstances, a proxied device will request its proxied context profile to be deleted before leaving a region or becoming unavailable. However, in the event of a communication error or device failure, the proxy provider may never receive a deletion request. The profile retention policy resolves this situation using a timed lease technique.

A proxy provider must retain a proxied profile as long as it continues to receive renewal requests from the appropriate proxied device, as long as the requests are received within the amount of time specified by the “LeaseMinutes” property. If a proxy provider does not receive an update request within the specified time, it may assume that the proxied device has either become unavailable or no longer requires its services. In the event that a proxy provider becomes unavailable, any associated proxied devices will detect the failure during their next scheduled update interval. The affected proxied devices must then select a new proxy provider.

Chapter 4

Implementation

We have created of a set of mappings between our ACMF and corresponding components of the WSDM (Web Services Distributed Management) Standard which allows our framework to be implemented using an existing WSDM-based middleware platform such as Apache Muse [3]. This chapter begins with an introduction to WSDM, followed by a discussion of the mappings. Finally, we discuss our ACME (Adaptable Context-Managed Environment) simulation that we created as a proof-of-concept using Apache Muse. Muse is a Java-based implementation of WSDM which we found to be highly suitable for this application.

4.1 The WSDM Standard

4.1.1 Background

WSDM (Web Services Distributed Management) [19] is a standard specified by the Organization for the Advancement of Structured Information Standards (OASIS) which provides remote manageability for networked resources. WSDM is based on the Web Services Architecture [30], an existing open standard providing remote application communication through the Web. The purpose of WSDM is to provide a universal management interface for any type of resource.

WSDM is built around the concept of a "Manageable Resource" (MR) which is formally defined in the WSDM MUWS (Management Using Web Services) specification

[20]. A Manageable Resource is a logical entity which represents some real-world resource (which can be anything from a software application to a physical device) in a standardized form that can be exposed to external entities. Manageable Resources are discussed in detail in Section 4.1.2. The WS-Resource specification [22] composes a resource and a basic Web service. WSDM in turn adds a manageability interface, resulting in a Manageable Resource which is accessed through a WSDM endpoint. Figure 4.1 illustrates how a typical real-world resource such as a laser printer can be represented as a Manageable Resource which is in turn accessed by a WSDM client (also referred to as a “ManageabilityConsumer”).

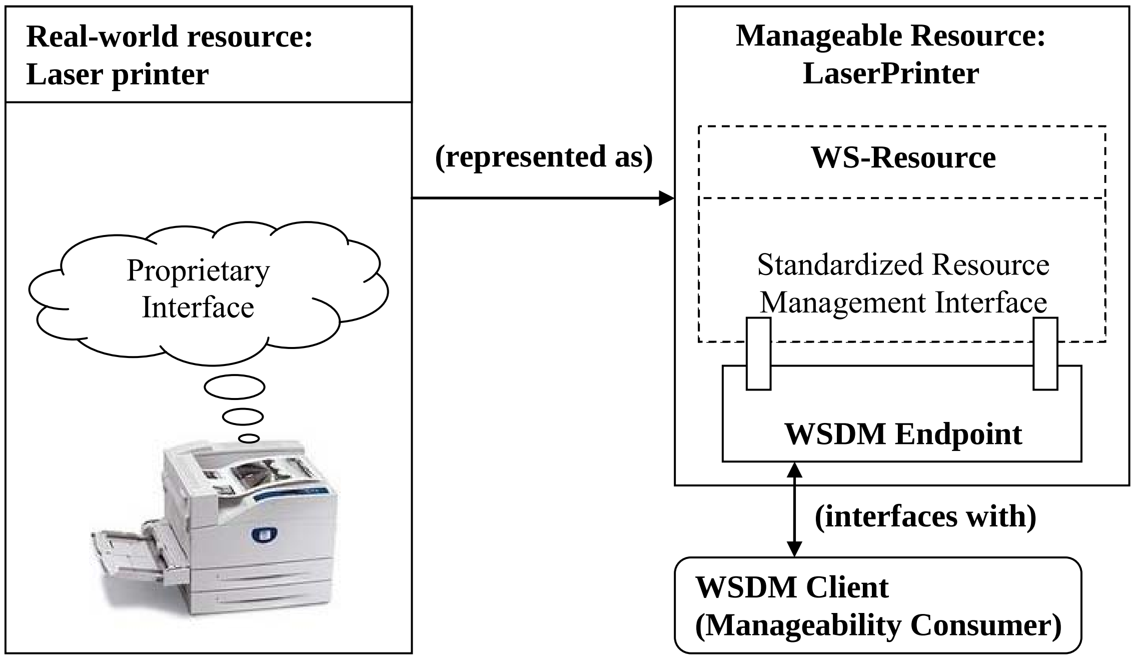


Figure 4.1 Laser Printer Represented as a Manageable Resource

WSDM enables remote clients (Manageability Consumers) to access and manage heterogeneous resources in a standard way, regardless of implementation or platform. Our ACMF implementation uses WSDM to access and manage the context-related properties of pervasive devices, thus moving towards context-awareness in pervasive environments.

4.1.2 Manageable Resources

WSDM Manageable Resources support a high-level facility called Manageability Capabilities which are composed of Resource Properties [23], Operations, and Management Events. Resource Properties are abstracted representations of attributes typically belonging to the real-world resource that is being managed. Operations are commands that invoke specific functions on the resource. Management Events are asynchronous notifications that resources can generate on their own, as opposed to relying on Manageability Consumers to request the information. They are defined using Topics, which associate resource state changes to the generation of notification messages.

All of the above elements are defined for a Manageable Resource using four XML-based document types, namely the WSDL (Web Services Description Language) document, the Resource Properties Schema document, the Resource Metadata Descriptor document and the TopicSpace document. The WSDL document is an XML instance document that describes the services offered by a WSDM Manageable Resource. It is the top-level document and includes references to the Resource Properties Schema document and Resource Metadata Descriptor document. A Manageable Resource's Operations are also declared in the WSDL document. The Resource Properties Schema document is an XML Schema document (XSD) that provides the Resource Property declarations for a Manageable Resource. The Resource Metadata Descriptor document is an XML instance document that is used to declare static metadata information that Manageability Consumers can use to determine the semantics of a Manageable Resource's capabilities.

The TopicSpace document is an XML instance document that is used to declare a Manageable Resource's available Topics.

Manageability Capabilities are sets of Resource Properties, Operations and Management Events (specified as Topics, and hereafter referred to as “Events”). They are used to define grouped representations of related functions. WSDM includes predefined Manageability Capabilities, such as the “Identity” capability, which is required for all Manageable Resources. Developers can also create their own custom Manageability Capabilities. Resource Properties primarily reflect the state of the real-world resource being managed. Operations are used to invoke functions on the real-world resource being managed. Operations are defined as MEPs including an input message (request), output message (response) and fault message (in case the Operation does not complete as expected). Events, which provide notification of specific occurrences taking place on a Manageable Resource, are defined in terms of Topics, Operations and MEPs. Events provide a “push” style information exchange that is not available directly via Resource Properties or Operations.

As with traditional Web services, the WSDL document is an XML document that describes the services offered by a WSDM Manageable Resource. It is the top-level document and includes references to the Resource Properties Schema document and Resource Metadata Descriptor document. A Manageable Resource’s Operations, MEPs and Topics are also declared in the WSDL (it is also permissible to declare Topics in a separate XML document if an appropriate reference is placed in the WSDL). WSDM includes a set of predefined Operations, MEPs and Topics that may be used, but developers are free to add their own as well. A sample WSDL document depicting a declaration of the predefined “GetResourcePropertyDocument” Operation is shown in Figure 4.2.

```
...
<wsdl:operation name="GetResourcePropertyDocument">
<wsdl:input
wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/GetResourcePropertyDocument/GetResourcePropertyDocumentRequest"
name="GetResourcePropertyDocumentRequest"
message="tns:GetResourcePropertyDocumentRequest" />
<wsdl:output
wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/GetResourcePropertyDocument/GetResourcePropertyDocumentResponse"
name="GetResourcePropertyDocumentResponse"
message="tns:GetResourcePropertyDocumentResponse" />
<wsdl:fault name="ResourceUnknownFault"
message="tns:ResourceUnknownFault" />
<wsdl:fault name="ResourceUnavailableFault"
message="tns:ResourceUnavailableFault" />
</wsdl:operation> ...
```

Figure 4.2 WSDL document fragment

The Resource Properties Schema document is an XML Schema document (XSD) that provides the Resource Property declarations for a Manageable Resource. As with Operations and MEPs, WSDM includes a set of predefined Resource Properties that may be used or extended. Figure 4.3 shows part of a sample Resource Properties Schema document depicting the declaration of custom Resource Properties.

```
...
<xsd:element name="UniqueName" type="xsd:string"/>
<xsd:element name="DeviceType" type="xsd:string"/>
<xsd:element name="Owner" type="xsd:string"/>
...
```

Figure 4.3 Resource Properties Schema Document fragment

The Resource Metadata Descriptor document declares static metadata information which Manageability Consumers can use to determine the semantics of a Manageable Resource’s capabilities. It contains metadata elements describing Resource Properties that have been declared in the Resource Properties Schema document. These metadata elements may declare associations between Manageability Capabilities and Resource Properties, as well as various Resource Property semantics such as modifiability and permitted values. Figure 4.4 shows part of a sample Resource Metadata Descriptor document depicting the declaration of a Resource Property metadata element for a Resource Property that is read-only and is part of a custom Manageability Capability called “DeviceContext”.

```
...
<rmd:Property path="LaserPrinter:UniqueName"
        modifiability="read-only">
        <muws-p2-xs:Capability>
                http://largo.cs.queensu.ca/LaserPrinter/DeviceContext
        </muws-p2-xs:Capability>
</rmd:Property>
...
```

Figure 4.4 Resource Metadata Descriptor document fragment

Figure 4.5 gives an overall picture of a WSDM Manageable Resource. The arrows indicate which Manageable Resource components are declared in each document type.

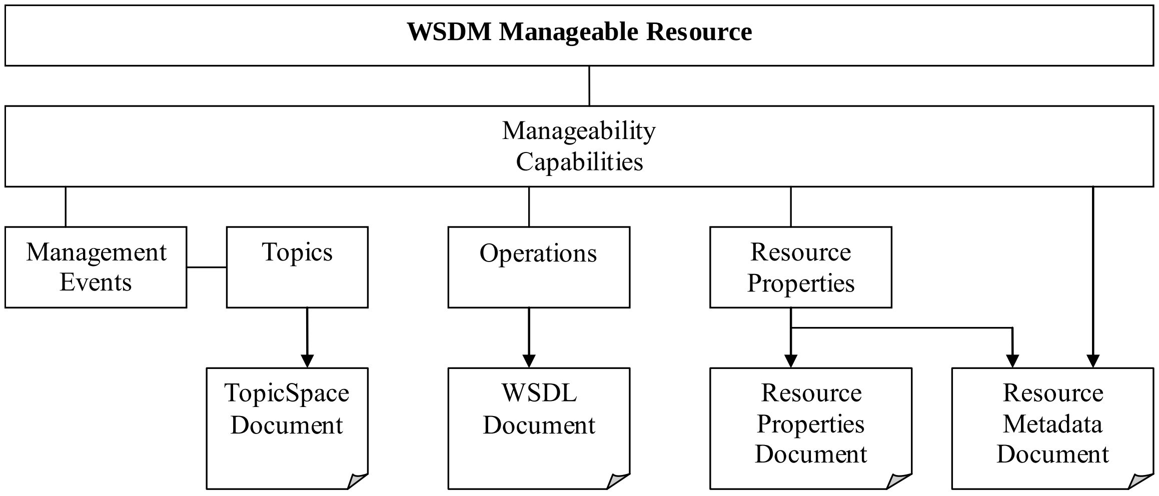


Figure 4.5 WSDM Manageable Resource breakdown

4.1.3 Manageability Capabilities

As mentioned in Section 4.1.2, Manageability Capabilities are sets of Resource Properties, Operations and Management Events (specified as Topics, and hereafter referred to as "Events"). They are used to define grouped representations of related functions. Although they provide these groupings, the functions themselves are still accessed independently.

Developers are free to create their own custom Manageability Capabilities, but WSDM also includes several predefined ones which are designed to be applicable to various types of resources. The "Identity" capability, for instance, is a special Manageability Capability that must be exposed by every Manageable Resource. Its purpose is to establish uniqueness between entities and it contains only one element, the predefined "ResourceId" Resource Property. It does not contain any Operations or Events.

A custom Manageability capability typically contains one or more custom Resource Properties, Operations or Events. For example, a custom Manageability Capability called "PrinterManager" might include Resource Properties for managing the amount of ink and paper remaining, Operations for printing test pages and running diagnostics, and Events for notifying users once their print jobs have been completed.

4.1.4 Resource Properties

Resource Properties are one of the fundamental components of WSDM Manageable Resources. They are pieces of information that reflect attributes regarding the state of the real-world resource being managed. They may also be used to represent metadata, manageability information or other data. In fact, there is no constraint on the context of data represented by Resource Properties. For example, a laser printer might have a “TonerLevel” Resource Property reflecting the amount of toner remaining.

Resource Property declarations must include a data type parameter and cardinality constraints. These parameters are specified along with each Resource Property in the Resource Properties Schema document. The data type is specified using the “type” parameter and can be a custom type or a primitive XML schema datatype such as “integer” or “string” [32]. Cardinality is specified using the “minOccurs” and “maxOccurs” parameters and governs the number of instances of a property that are allowed. The minimum cardinality is usually “0” or “1” while the maximum is usually set to “1” or “Unbounded” (when there is no upper limit).

Finally, Resource Properties have the following additional parameters which are specified in the Resource Metadata Descriptor document:

- **mutability:** Expresses how the value can change over time. This parameter can be set to “constant” (the value must not ever change), “mutable” (values may be added, removed or changed at any time), or “appendable” (new values may be added at any time, but once added may not be changed or removed).
- **modifiability:** Indicates whether a Manageability Consumer can modify the value. This parameter can be set to “read-only” or “read-write”.
- **subscribability:** Expresses whether the associated Resource Property can be the target of a subscription (which triggers notifications when the value changes). This parameter can be set to “true” or “false”.
- **ValidValues:** Allows an enumeration or range of permissible values to be specified.

- **staticvalues:** Allows specification of a minimum set of values that the associated Resource Property must contain.
- **initialValues:**Allows specification of the set of values that the associated Resource Property must contain when the Manageable Resource first becomes available.

Resource Properties are usually associated with Manageability Capabilities. For example, the predefined “Caption” and “Version” Resource Properties are both part of the predefined “Description” capability. This is not a requirement, however.

4.1.5 Operations

Operations are used to invoke functions on the real-world resource being managed. Operations are defined as MEPs including an input message (request), output message (response) and fault message (in case the Operation does not complete as expected). The input message includes function parameters, and the output message contains results data generated by the resource, such as confirmation of successful completion. If an Operation does not complete successfully, a fault message is returned instead of the response message. As with Resource Properties, Operations may be associated with Manageability Capabilities although this is not a requirement. For example, the predefined “GetResourceProperty” Operation is part of the WSDM stack and does not have any associated capabilities.

4.1.6 Events

Events provide a means of being notified of specific occurrences taking place on a Manageable Resource, such as a Resource Property value change. They are defined in terms of Topics, Operations and MEPs. Manageability Consumers can use the "Subscribe" Operation to select Topics which represent occurrences such as a change to a specified Resource Property. Whenever an occurrence takes place, a Manageable Resource will generate Events and send Notification messages to all Manageability Consumers that have subscribed to the Topic representing the occurrence. Events therefore provide a "push" style information exchange that is not available directly via Resource Properties or Operations.

4.1.7 Implementation and Scalability

WSDM implementations are the middleware applications that form Manageable Resources. Apache Muse is a Java-based implementation of the WSDM standard. In March 2007, Apache Muse version 2.2.0 became available for J2EE and OSGi-based platforms. J2ME Foundation Profile support for mobile/limited devices was also added in this version [3]. The ACMF implementation created for this research was built using Apache Muse and is discussed in Section 4.2.

In addition to the Apache Muse implementation, IBM offers a WSDM tooling supplement for its Eclipse TPTP (Test & Performance Tools Platform) product [16]. The WSDM tooling component of TPTP assists developers with building WSDM Manageable Resources.

WSDM Manageable Resource implementations are highly scalable. According to the WSDM MUWS specification, "The WSDM architecture provides appropriate coverage from low-end manageability of small devices like mobile phones, to high-end manageability of very capable components like application servers and business processes" [20]. The minimum requirement of a Manageable Resource is to expose the "Identity" capability. This makes the implementation flexible enough to support very limited devices while allowing more complex devices to expose as many capabilities as needed. Figure 4.6 illustrates how a mobile phone, laser printer, and high-end server can implement varying levels of WSDM functionality depending on their requirements and capabilities. Note that in the diagram, all devices support the "Identity" capability.

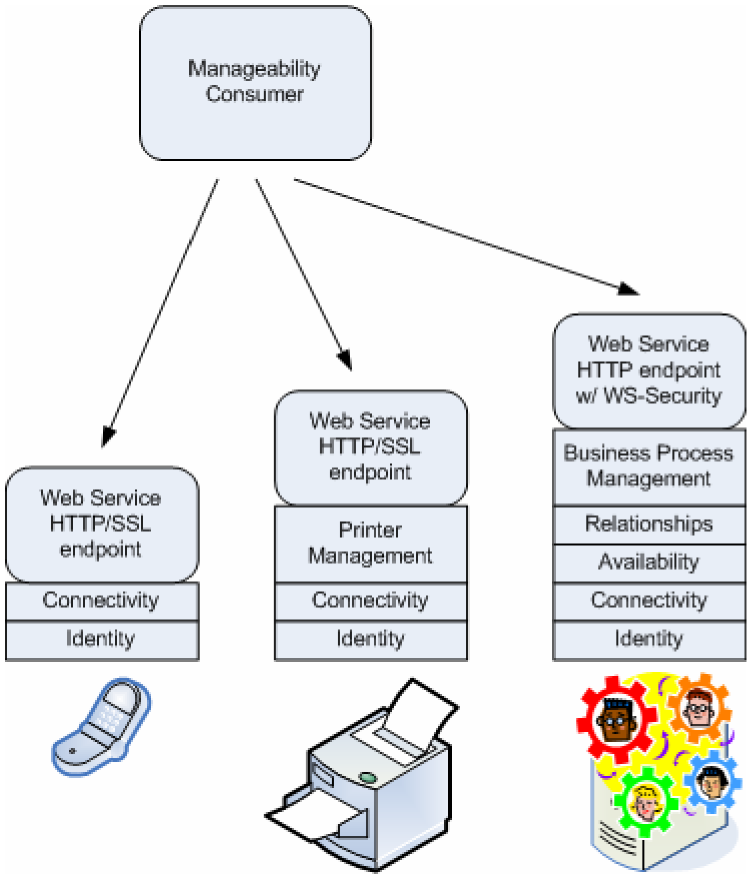

Manageability Consumer
Web Service HTTP/SSL endpoint
Connectivity
Identity
Web Service HTTP/SSL endpoint
Printer Management
Connectivity
Identity
Web Service HTTP endpoint w/ WS-Security
Business Process Management
Relationships
Availability
Connectivity
Identity

Figure 4.6 Low-end to high-end manageability [20]

4.2 ACMF-WSDM Mappings

In order to map ACMF to WSDM we define two sets of mappings. One set maps our context model to the WSDM Manageable Resource (MR). The other set maps our context-exchange protocols to WSDM Operations. This section includes a discussion of the context model mappings followed by the context-exchange protocol mappings.

4.2.1 Context Model Mappings

A WSDM MR is specified using a WSDL document, a Resource Properties Schema (RPSD), a Resource Metadata Descriptor (RMD) document and a TopicSpace document. WSDM Resource Properties are declared in the RPSD, and Topics are declared in the TopicSpace document. Manageability Capabilities are associated with Resource Properties in the RMD document. The WSDL document specifies the Web service interface used to access the MR and includes several standard definitions as well as references which tie in the other three WSDM MR documents. These documents can be derived from an ACMF device schema using templates we provide in Appendix B.

Domain Ontology Mapping

Domain ontologies are not directly mapped because there is no suitable corresponding WSDM component. The framework does not generally require domain ontologies to be accessible as they are only needed when creating new devices and can be made available to designers through application-specific means.

Device Schema Mapping

The ACMF device schema is used to specify a WSDM MR by mapping to the four MR documents. Specifically,

1. Each ACMF device-type is mapped to a WSDM Manageability Capability. MC's are defined through a series of steps described in Section 2.8 of the WSDM MUWS Primer [21]. The principal steps in this process are:

   a. The MC is given a URI so that it can be identified. The URI is then added as an instance value of the MR's ManageabilityCapability Resource Property.

   b. All Resource Properties to be associated with the MC are declared as GEDs in a schema document (our schema documents containing related properties are used here).

   c. A topic must be created in the Topics Topic for events related to the MC.

   d. Entries are added to the RMD document to associate each Resource Property with the MC.

2. Each ACMF device property maps to a WSDM Resource Property. If a device property is subscribable, it maps to a WSDM Topic as well.

ACMF device properties are grouped by device-type when mapped to WSDM Resource Properties. Before being referenced in the RPSD, each group of properties is first mapped to its own separate XML Schema document having a namespace which reflects the corresponding device-type name (step 1.b above). For example, the CSB "LaserPrinter" contains two device-types ("PervasiveDevice" and "LaserPrinter") and thus has two groups of properties which map to two separate Schema documents. We preserve our Resource Property groupings by defining a Manageability Capability for each group.

The RPSD contains a single Resource Property declaration which references all the device properties declared in the other schema documents, thereby consolidating them into a single document. The namespace of the RPSD matches the namespace of the MR itself.

Table 4.1 lists the parameters of ACMF device properties and to which MR document each one maps.

ACMF device schema property parameter	**Maps to (MR document)**
Name	RPSD
Type	RPSD
minOccurs	RPSD
maxOccurs	RPSD
Mutable	RMD
Subscribable	TopicSpace
ValidStringValues	RMD
ValidIntegerValues	RMD

Table 4.1 ACMF Device schema property parameter mappings

Table 4.2 lists the set of WSDL, RPSD, RMD and TopicSpace documents that are produced by mapping the "LaserPrinter" device from our "ComputerScienceBuilding" scenario. The complete documents are included in Appendix C.

LaserPrinterDevice.wsdl	Specification of the Web service, references to schema & metadata documents
PervasiveDevice.xsd	Domain-wide property declarations
LaserPrinter.xsd	Device-specific property declarations
LaserPrinterDevice_RPSD.xsd	Resource Properties Document declaration
LaserPrinterDevice_TopicSpace.xsd	Topic declarations
LaserPrinterDevice.rmd	Metadata values

Table 4.2 Laser Printer WSDM MR documents

Context Profile Mapping

Device context profiles (device schema instances) are represented in WSDM using the Resource Properties document that is produced by instantiating the RPSD. A sample "LaserPrinter" Resource Properties document is included in Appendix C.

4.2.2 Context-Exchange Protocol Mappings

We have defined a set of mappings between WSDM Operations and the ACMF context-exchange protocol functions specified in Sections 3.3.3 and 3.3.4. The mapping of local context service functions to WSDM Operations is summarized in Table 4.3. The “Discover Device” operation is implemented using an underlying network facility (such as Bonjour [4]) and so is not mapped to a WSDM Operation. The “Request Device Schema” operation is implemented by simply accessing schemas directly via their URIs. The other two local context service functions map directly to WSDM Operations.

Local Context Service Function	**WSDM Operation**
Discover	n/a
Request device schema	n/a
Request properties	GetMultipleResourceProperties
Subscribe to property	Subscribe

Table 4.3 Mappings between local context service functions and WSDM Operations

There are no suitable built-in WSDM Operations for the functions required by the context proxy service (‘send profile’, ‘renew lease’, ‘delete profile’) and they must therefore be implemented as custom Operations. Appendix C includes WSDL and XML Schema document examples containing the necessary declarations for these custom Operations. These declarations are only needed when specifying devices that support the proxy provider role as described in Section 3.3.4.

4.3 ACME Simulation

Our ACME simulation uses the mappings described in Section 4.2 to create a simulated pervasive environment as proof-of-concept. The simulation is made up of three main components including a MANET network environment simulator and pervasive device simulators for both the server and client roles. The simulation's architecture is illustrated in Figure 4.7.

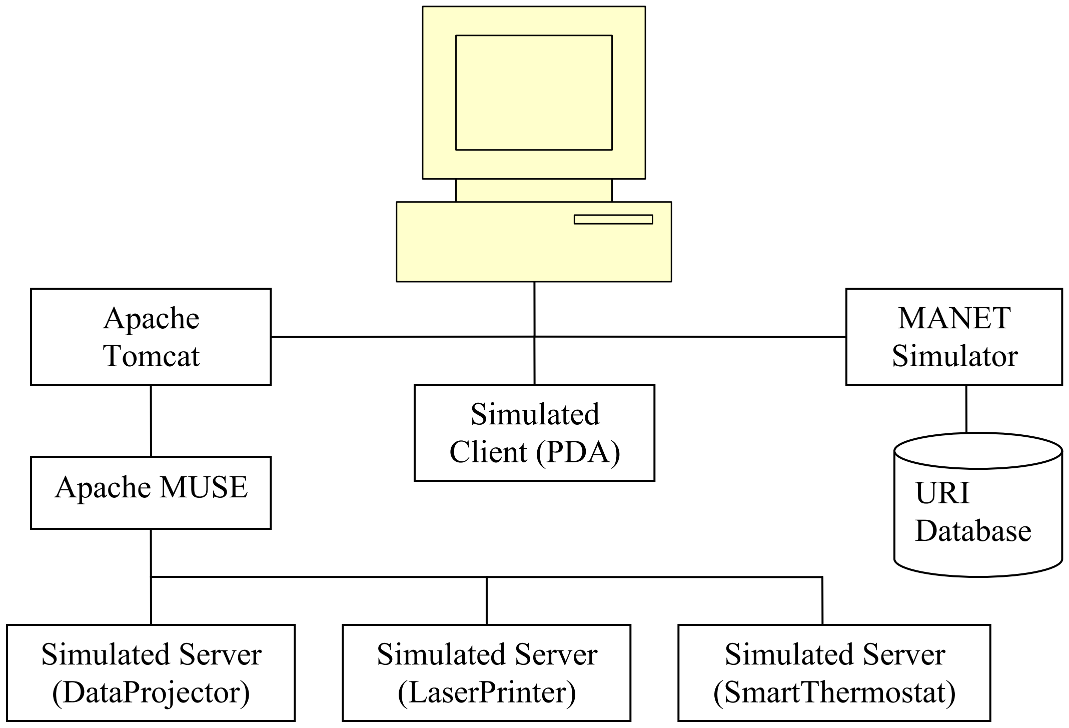

Apache Tomcat
MANET Simulator
Simulated Client (PDA)
Apache MUSE
URI Database
Simulated Server (DataProjector)
Simulated Server (LaserPrinter)
Simulated Server (SmartThermostat)

Figure 4.7 ACME simulation architecture

4.3.1 MANET Simulator

Our ACMF is designed to work with any MANET implementation (such as Bonjour [4]) that provides a self-configuring TCP/IP network. It is also possible to use the framework with a simulated MANET environment. This can be accomplished by using a TCP/IP network and a DHCP server combined with some means of managing the URIs of devices available on the network. For example, a Web service with a predetermined URI could be used by all devices on the network to maintain a list of URIs for all connected devices.

Our ACME simulation includes a MANET simulator that uses a database to store the URIs of the simulated servers. The simulated servers update the URI database when they go online or offline, and the simulated client uses it to obtain the servers' URI information. The URI database is implemented using PostgreSQL [24] and the MANET simulator is implemented as a Java class which uses JDBC connectivity to access the database.

4.3.2 Simulated Servers

Each simulated server is a WSDM MR implemented using Apache Muse Version 2.2.0. These MRs respond to requests from clients, behaving as if they were implemented on actual physical devices. Each MR has a GUI interface which provides real-time information about its current state. The ACME simulation includes three simulated servers: DataProjector, LaserPrinter and SmartThermostat. The LaserPrinter simulated server supports the ProxyProvider role and is thus able to manage other devices' context profiles as described in Section 3.3.4.

4.3.3 Simulated Client

The simulated client is a Java program that interacts with the simulated servers. It is meant to represent a PDA featuring an interactive GUI interface which demonstrates device discovery and context information retrieval. The simulated client also has the ability to make changes to the property values on other devices. For example, the client can operate the data projector's lamp by modifying its 'LampActive' Resource Property value. This is an application-level feature which was included as an added advantage of using WSDM technology; it is not strictly part of the context management framework. Such functionality is desirable in a pervasive environment where devices should be able invoke operations provided by other devices.

4.3.4 Use Cases

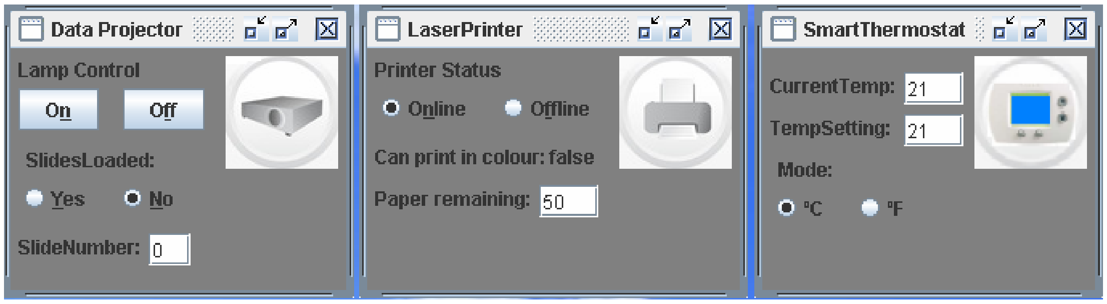
This section discusses the set of use cases that we ran against the ACME simulation. The cases are divided into two sections, “Local Context Service Use Cases” and “Context Proxy Service Use Cases”, which demonstrate the operation of each service respectively and as described in Sections 3.3.3 and 3.3.4. These use cases serve to validate our framework implementation.

Local Context Service Use Cases

These use cases demonstrate the functions that make up the local context service. Each use case covers a specific function, including device discovery, device schema and context profile acquisition, device selection, and device interaction. We also include cases which demonstrate devices entering/leaving an environment, as well as property subscription and notification.

Use Case 1: Initialization (devices enter environment)

This use case demonstrates the initialization of our ACME simulation. Upon initialization, the three simulated servers enter the environment and begin “broadcasting” their availability by adding their address into the URI database via the MANET simulator. Figure 4.8 illustrates the initial state of the three simulated servers and Table 4.4 lists the contents of the URI database after their initialization.



SlidesLoaded: Yes [✓ No]

SlideNumber: 0

Online [✓] Offline

Can print in colour: false

Paper remaining: 50

CurrentTemp: 21

TempSetting: 21

Mode: [✓ °C] °F

Figure 4.8 Initial state of simulated servers

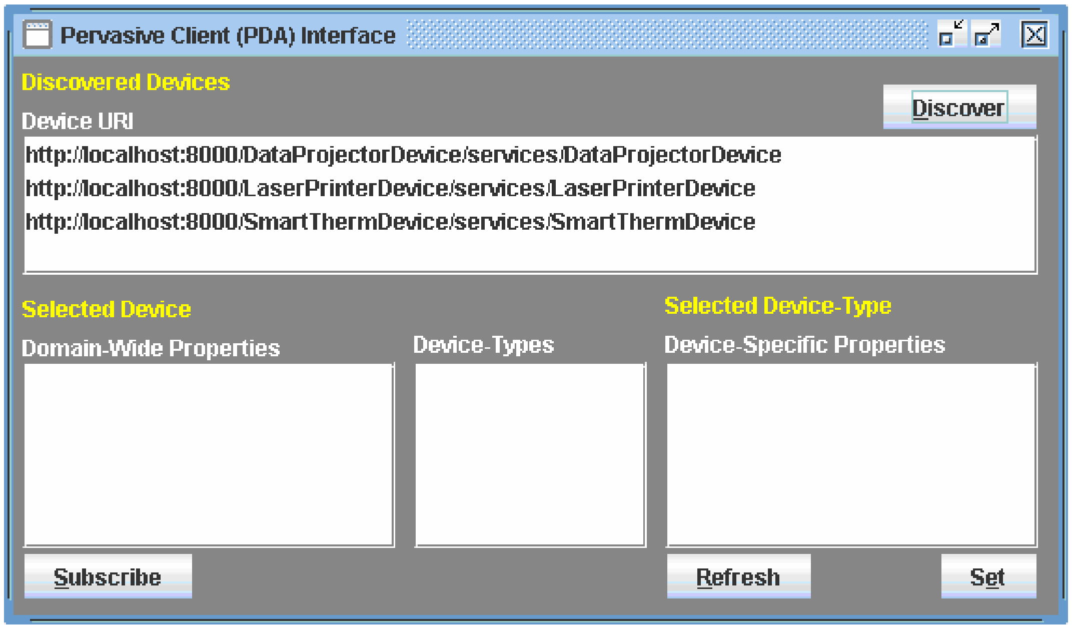
URI
http://localhost:8000/DataProjectorDevice/services/DataProjectorDevice
http://localhost:8000/LaserPrinterDevice/services/LaserPrinterDevice
http://localhost:8000/SmartThermDevice/services/SmartThermDevice

Table 4.4 Initial URI database contents

The presence of the three simulated servers' address information in the URI database indicates that the devices have entered the environment and initialized successfully.

Use Case 2: Device Discovery

This use case demonstrates discovery of the simulated servers by the simulated client. The client retrieves the contents of the URI database in order to determine what devices (if any) are present in the environment. The retrieved URIs are then displayed in the simulated client's GUI interface. Figure 4.9 illustrates the simulated client's state after discovery of the three simulated servers. The simulated servers' URIs appear in the "Device URI" list box, indicating successful discovery.



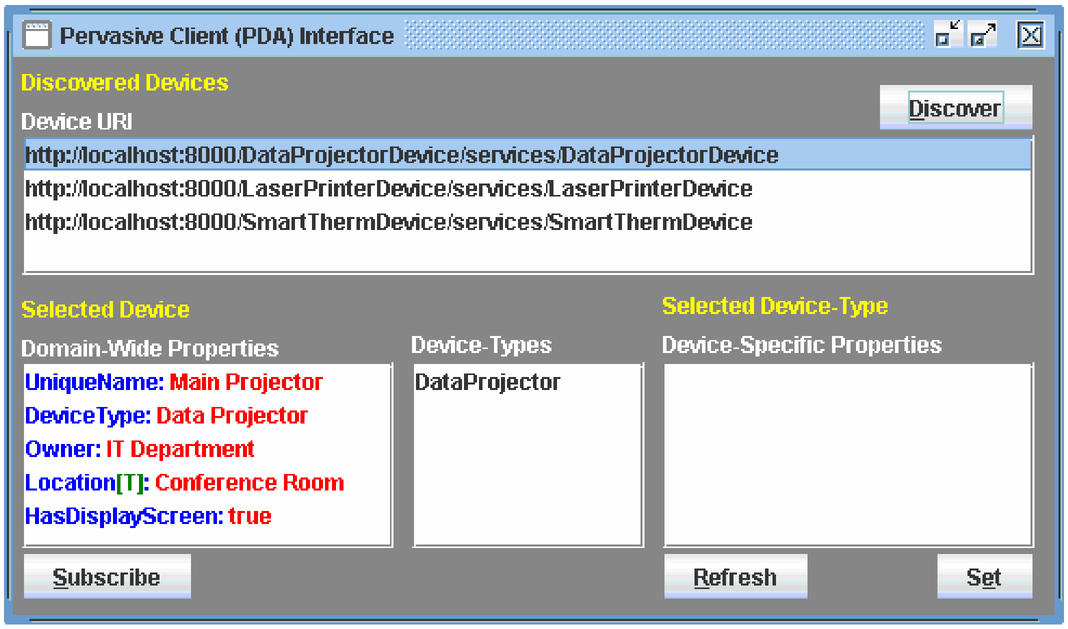
http://localhost:8000/DataProjectorDevice/services/DataProjectorDevice
http://localhost:8000/LaserPrinterDevice/services/LaserPrinterDevice
http://localhost:8000/SmartThermDevice/services/SmartThermDevice

Figure 4.9 Simulated client after device discovery

Use Case 3: Device Schema & Context Profile Acquisition

This use case demonstrates device schema and context profile acquisition which is shown by retrieving and displaying a discovered device's context profile.

During device discovery, the simulated client retrieves the device schema and context profile for all discovered devices. When one of the discovered devices is selected, the simulated client responds by displaying its device schema and context profile, effectively demonstrating that device schema and context profile acquisition was successful. Figure 4.10 illustrates the client's state after selecting the Data Projector device. The Data Projector's context profile is displayed, which includes a list of domain-wide properties and their values, as well as a list of supported device-types.



http://localhost:8000/DataProjectorDevice/services/DataProjectorDevice
http://localhost:8000/LaserPrinterDevice/services/LaserPrinterDevice
http://localhost:8000/SmartThermDevice/services/SmartThermDevice

UniqueName: Main Projector
DeviceType: Data Projector
Owner: IT Department
Location[T]: Conference Room
HasDisplayScreen: true

DataProjector

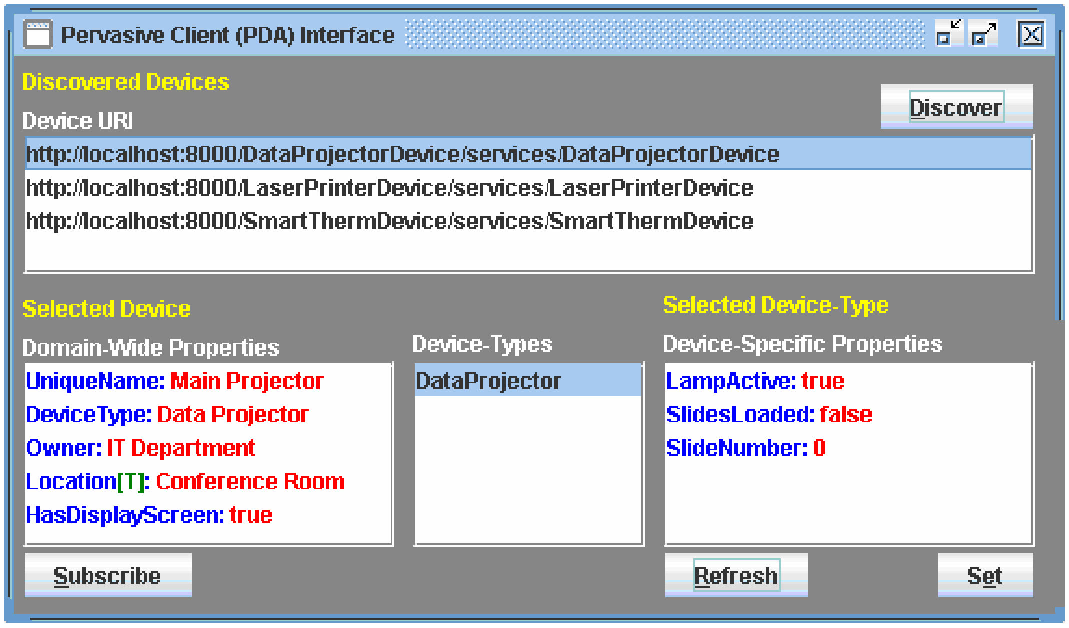
Figure 4.10 Simulated client with data projector context profile

Use Case 4: Device Selection & Interaction

This use case demonstrates device selection and interaction. In our ACME simulation, device selection is user-delegated which means that the individual using the simulated client manually selects a device with which to interact. In this use case, the

Data Projector device has been selected. Device interaction between the PDA (client) and the Data Projector (server) is demonstrated by retrieving device-specific property values.

Figure 4.11 illustrates the state of the simulated client after the Data Projector device has been selected and its device-specific property values have been retrieved. The "Device-Specific Properties" list box displays the values of the "LampActive", "SlidesLoaded" and "SlideNumber" properties.



http://localhost:8000/DataProjectorDevice/services/DataProjectorDevice
http://localhost:8000/LaserPrinterDevice/services/LaserPrinterDevice
http://localhost:8000/SmartThermDevice/services/SmartThermDevice

UniqueName: Main Projector
DeviceType: Data Projector
Owner: IT Department
Location[T]: Conference Room
HasDisplayScreen: true

DataProjector

LampActive: true
SlidesLoaded: false
SlideNumber: 0

Figure 4.11 Simulated client with data projector device-specific properties

Figure 4.12 illustrates the state of the simulated Data Projector, which matches the values of properties that were retrieved by the client. The lamp is active (LampActive = true), there are no slides loaded (SlidesLoaded = false) and the current slide number is 0 (SlideNumber = 0).

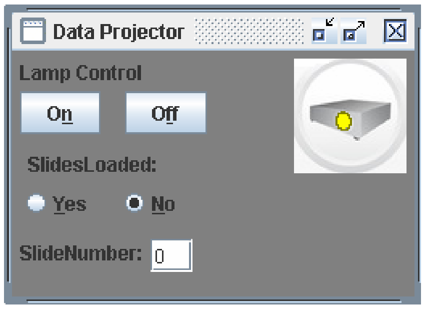


Figure 4.12 Data projector device state

Use Case 5: Device leaves environment

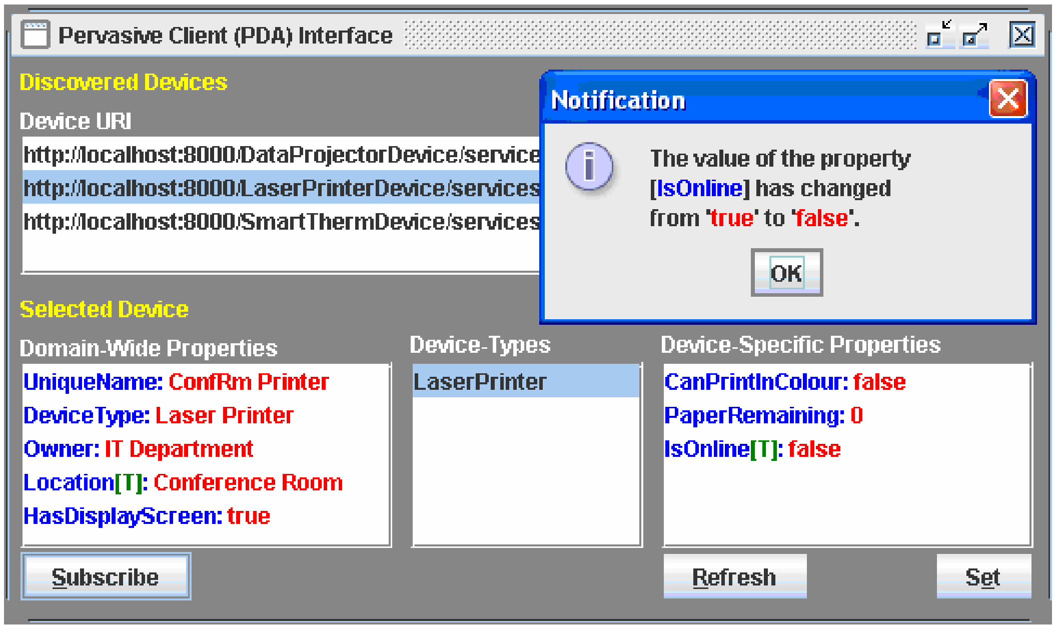
This use case demonstrates what occurs when a device leaves the environment. In this case, the simulated Data Projector removes itself from the URI database and then shuts down. This is illustrated in Table 4.5 which shows the contents of the URI database after the Data Projector has left. The Data Projector's URI is no longer present in the database. In a true MANET-type environment, a device's URI would no longer be broadcast after it had left an environment. The URI database simulates this behaviour.

URI
http://localhost:8000/LaserPrinterDevice/services/LaserPrinterDevice
http://localhost:8000/SmartThermDevice/services/SmartThermDevice

Table 4.5 URI database contents after Data Projector has left

Use Case 6: Subscription & Notification

This use case demonstrates property subscription and notification. The simulated client indicates subscribable properties by appending the suffix "[T]" to the property name. In this case, the client has subscribed to the Laser Printer's "IsOnline" property and receives a notification message when the Laser Printer goes offline. The client displays the notification as a pop-up window which is illustrated in Figure 4.13.


Pervasive Client (PDA) Interface

Discovered Devices

Device URI

http://localhost:8000/DataProjectorDevice/service
http://localhost:8000/LaserPrinterDevice/services
http://localhost:8000/SmartThermDevice/services

Notification

The value of the property [IsOnline] has changed from 'true' to 'false'.

Selected Device

Domain-Wide Properties

UniqueName: ConfRm Printer
DeviceType: Laser Printer
Owner: IT Department
Location[T]: Conference Room
HasDisplayScreen: true

Device-Types

LaserPrinter

Device-Specific Properties

CanPrintInColour: false
PaperRemaining: 0
IsOnline[T]: false

Figure 4.13 Simulated client with notification pop-up

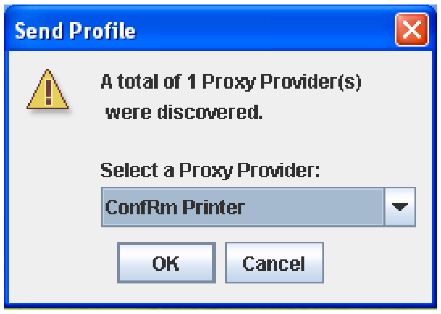
Context Proxy Service

The following use cases demonstrate the operation of the context proxy service and how it functions as an extension to the local context service. Each use case covers a specific proxy-related activity, including proxied profile storage, retrieval, renewal, expiry and deletion. Proxied profile storage, renewal and deletion employ the context proxy service’s “Send Profile”, “Renew Lease” and “Delete Profile” functions, while proxied profile retrieval relies on the local context service’s “Get Profile” function. Proxied profile expiry is accomplished without invoking functions from either service.

These use cases are demonstrated using an enhanced version of the ACME simulation where the Laser Printer acts as a proxy provider and the Smart Thermostat acts as a proxied device.

Use Case 7: Proxied Profile Storage

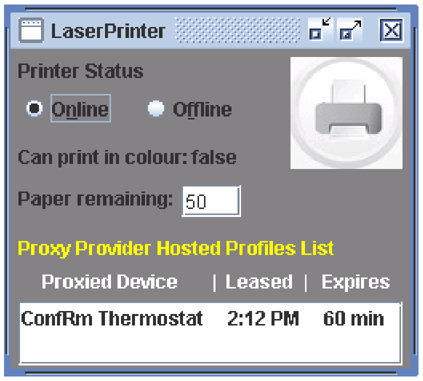
In this use case, the Smart Thermostat (proxied device) stores its context profile on the Laser Printer (proxy provider). When the thermostat's "Send Profile" button is clicked, it discovers all available proxy providers and displays a dialog box which allows one to be selected manually (more sophisticated devices would automatically select a provider). Once a proxy provider has been selected, the thermostat sends its context profile to it using the "Send Profile" function. Figure 4.14 illustrates the thermostat's dialog box confirming proxy provider selection, and Figures 4.15 and 4.16 show the state of both simulated devices after the context profile transfer has taken place. The Laser Printer GUI displays its current status as a Proxy Provider, which includes a list of devices for which it is hosting profiles. This listing contains the device name, the time the lease was obtained, and the time (in minutes) remaining before the lease expires. The Smart Thermostat GUI correspondingly displays its status as a Proxied Device. In this case, the printer indicates that it has granted a lease to the "Confrm Thermostat" device at 2:12 PM, with 60 minutes remaining on the lease. Similarly, the thermostat indicates that it has obtained a lease from the "ConfRm Printer" at 2:12 PM, with 60 minutes remaining on the lease. The lease data matches on both devices, indicating successful completion of this use case.


Send Profile

A total of 1 Proxy Provider(s) were discovered.

Select a Proxy Provider: ConfRm Printer

Figure 4.14 Smart thermostat proxy provider selection dialog box


LaserPrinter

Printer Status

● Online ○ Offline

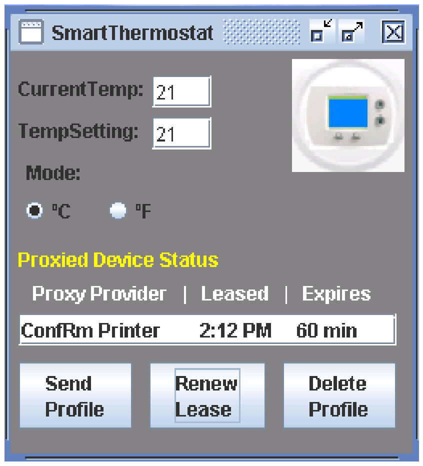
Can print in colour: false

Paper remaining: 50

Proxy Provider Hosted Profiles List

Proxied Device	Leased	Expires
ConfRm Thermostat	2:12 PM	60 min

Figure 4.15 Laser printer status after receiving context profile


SmartThermostat

CurrentTemp: 21

TempSetting: 21

Mode:

● °C ○ °F

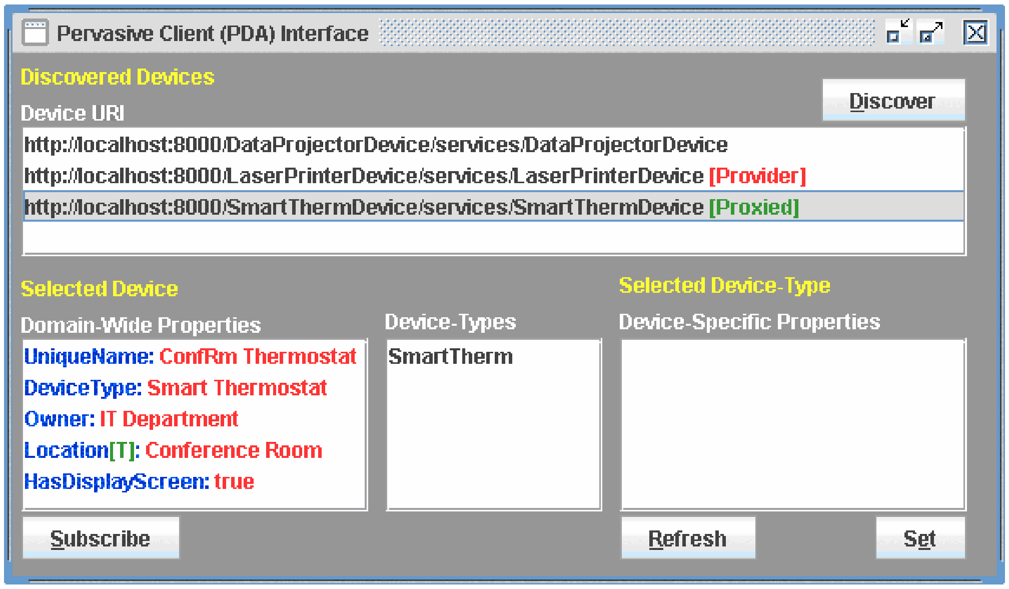
Proxied Device Status

Proxy Provider	Leased	Expires
ConfRm Printer	2:12 PM	60 min

Figure 4.16 Smart Thermostat status after sending context profile

Use Case 8: Proxied Profile Retrieval

This use case demonstrates discovery and retrieval of context profile information from a proxied device. Figure 4.17 illustrates the status of the simulated client after discovering and retrieving the Smart Thermostat's context profile. The GUI indicates that a device was discovered through another device as a proxied device by appending "[Proxied]" to the device's URI. Similarly, "[Provider]" is appended to the URI of any device which is identified as a proxy provider. In this case the Laser Printer is identified as a proxy provider, and the Smart Thermostat is discovered and identified as a proxied device. Additionally, the GUI displays the thermostat's context profile, which was obtained via the printer. The discovery and retrieval of the thermostat's context profile via the laser printer indicates successful completion of this use case.



http://localhost:8000/DataProjectorDevice/services/DataProjectorDevice
http://localhost:8000/LaserPrinterDevice/services/LaserPrinterDevice [Provider]
http://localhost:8000/SmartThermDevice/services/SmartThermDevice [Proxied]

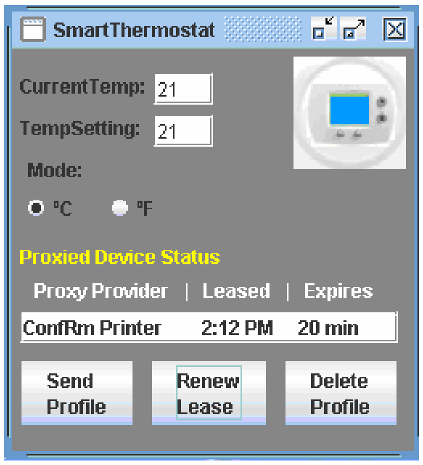
UniqueName: ConfRm Thermostat
DeviceType: Smart Thermostat
Owner: IT Department
Location[T]: Conference Room
HasDisplayScreen: true

SmartTherm

Figure 4.17 Simulated client After successful proxied profile retrieval

Use Case 9: Proxy Lease Renewal

This use case demonstrates renewal of a proxy lease which is about to expire. As the Smart Thermostat's lease with the Laser Printer nears expiry, it sends a renewal request using the "Renew Lease" function. Figure 4.18 illustrates the thermostat's state before the request. The thermostat's GUI displays that there are 20 minutes remaining before the lease expires. Figure 4.19 illustrates the state after the request has been sent. The thermostat's GUI now displays that there are 60 minutes remaining on the lease. The lease has been renewed which indicates successful completion of this use case.

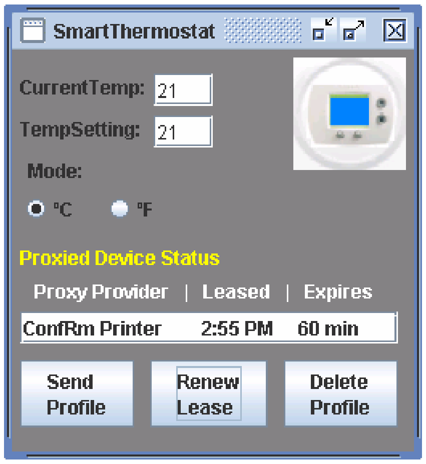


CurrentTemp: 21
TempSetting: 21
Mode:
● °C ○ °F

Proxied Device Status

Proxy Provider | Leased | Expires
ConfRm Printer 2:12 PM 20 min

Figure 4.18 Smart thermostat before lease renewal



CurrentTemp: 21
TempSetting: 21
Mode:
● °C ○ °F

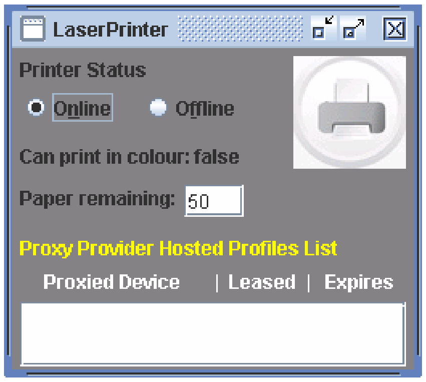
Proxied Device Status

Proxy Provider | Leased | Expires
ConfRm Printer 2:55 PM 60 min

Figure 4.19 Smart thermostat after lease renewal

Use Case 10: Proxy Lease Expiry

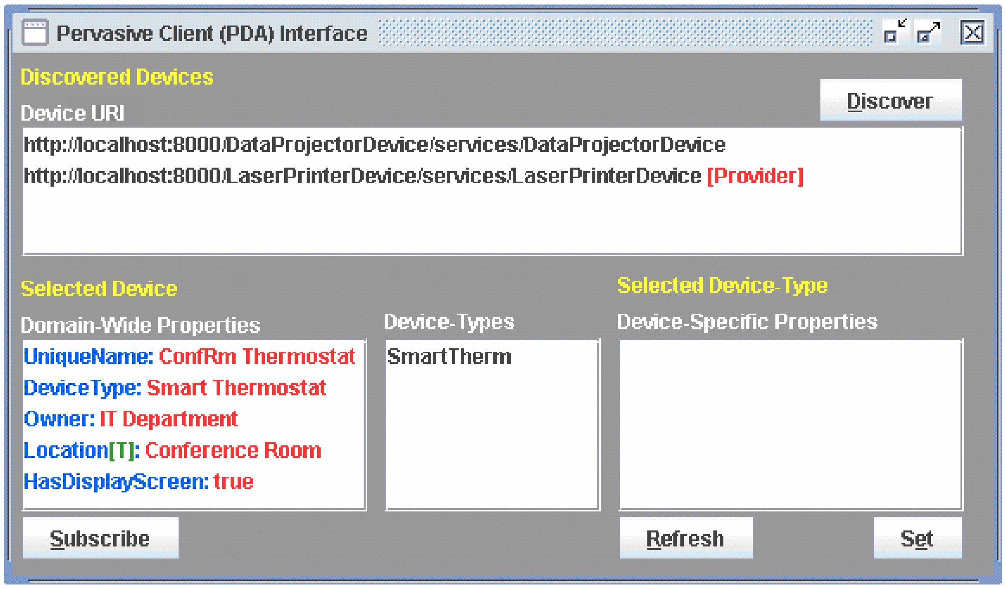
This use case demonstrates how a Proxy Provider will stop hosting a proxied device's profile if the lease expires. Figure 4.20 illustrates the Laser Printer's status as a Proxy Provider after the Smart Thermostat's lease has expired. The thermostat's profile no longer appears in the printer's Proxied Device list. Figure 4.21 illustrates how the Simulated Client is no longer able to discover the thermostat once the printer has stopped hosting its profile. The removal of the thermostat's profile from the printer demonstrates successful completion of this use case.


LaserPrinter
Printer Status
Online
Offline
Proxy Provider Hosted Profiles List
Proxied Device | Leased | Expires

Can print in colour: false

Paper remaining: 50

Figure 4.20 Laser printer status after smart thermostat lease expiry



http://localhost:8000/DataProjectorDevice/services/DataProjectorDevice
http://localhost:8000/LaserPrinterDevice/services/LaserPrinterDevice [Provider]

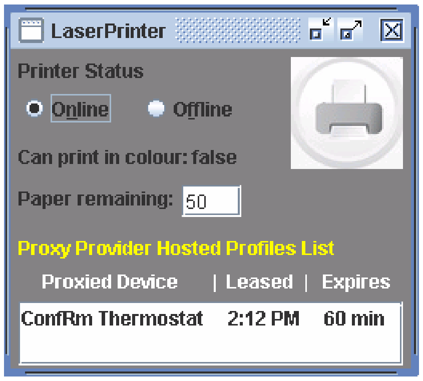
UniqueName: ConfRm Thermostat
DeviceType: Smart Thermostat
Owner: IT Department
Location[T]: Conference Room
HasDisplayScreen: true

SmartTherm

Figure 4.21 Simulated client status after smart thermostat lease expiry

Use Case 11: Proxied Profile Deletion

This use case demonstrates how a Proxied Device can manually delete its profile from a Proxy Provider without waiting for its lease to expire. Figure 4.22 illustrates the Laser Printer's status while it is hosting the Smart Thermostat's context profile. The thermostat appears in the printer's Proxied Device list. Figure 4.23 illustrates the printer's status after the thermostat has issued a "Delete Profile" request. The thermostat is no longer listed as a Proxied Device. Consequently, the Simulated Client is no longer able to discover the Smart Thermostat (refer to Figure 4.21). The removal of the thermostat's Proxied Profile from the printer demonstrates successful completion of this use case.


LaserPrinter

Printer Status

Online [✓] Offline

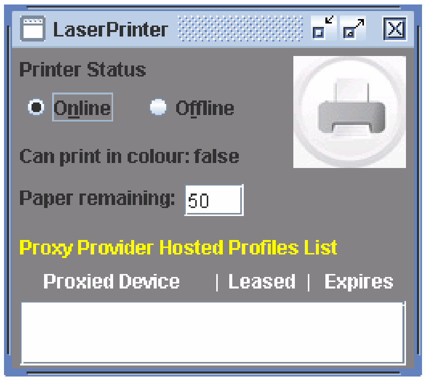
Can print in colour: false

Paper remaining: 50

Proxy Provider Hosted Profiles List

Proxied Device	Leased	Expires
ConfRm Thermostat	2:12 PM	60 min

Figure 4.22 Laser printer status before smart thermostat profile deletion


LaserPrinter

Printer Status

Online [✓] Offline

Can print in colour: false

Paper remaining: 50

Proxy Provider Hosted Profiles List

Proxied Device	Leased	Expires

Figure 4.23 Laser printer status after smart thermostat profile deletion

Chapter 5

Conclusions and Future Work

Pervasive computing environments that are made up of hundreds of devices and within which intelligent devices interact in the background on behalf of their users will soon become commonplace. Context-awareness is a key property that differentiates pervasive computing applications from the more traditional applications. A device is context-aware if it is able to adapt its behaviour to the current context. We submit that a context management system that is able to support the automatic discovery, retrieval and exchange of context information by devices while accommodating the heterogeneity present in a pervasive computing environment is needed to provide context-awareness.

5.1 Thesis Contributions

This thesis examines key issues surrounding pervasive computing, context-awareness and context management. We discern that interoperability supported by standardized context management is a central challenge in this area. As a solution we present an adaptable context management framework, called the ACMF (Adaptable Context Management Framework).

Our ACMF includes an entity specification, a context model and a set of context-exchange protocols, which together provide a means of context-enabled device interoperability.

Our ACMF adopts the autonomic computing paradigm, which ensures that systems built using our ACMF are aware of their surroundings and can automatically react to changes in order to effective execution. It enhances usability by insulating users from the frequent changes in context experienced by pervasive devices.

We describe how our ACMF can be mapped to the WSDM standard in order to produce a Web service based implementation of a context management system. This prototype is adaptable in that it can be used across disparate devices, software platforms and physical environments. It is also, to the best of our knowledge, the first implementation of a context management system based on standards.

5.2 Conclusions

Based on this research, we conclude that:

- Our context model is a viable construct against which diverse environments and devices can be modeled and represented.
- Our ACMF provides a viable solution to help provide context-management and device interoperability in pervasive computing environments.
- The WSDM standard and its Apache Muse implementation provide a suitable platform upon which to implement our ACMF, as demonstrated by our ACME simulation.

5.3 Future Work

Some possible directions for future research on our ACMF include the following. First, we plan to look at extending our context model to incorporate entities beyond devices. The context model in PersonisAD, for example, explores the modeling of people and places in addition to devices. Second, while our ACMF provides a solid foundation for context management, practical applications require inclusion of security standards. Future work will investigate the use of policies or other mechanisms for security integration. Third, our current prototype implementation of our ACMF is in a simulated environment using a proven standard (WSDM) and middleware platform (Muse). The next logical step is to move this implementation outside the laboratory and into the real world. Future work will involve creating an ontology to represent a real-world environment and programming devices with WSDM-capable middleware to support our current prototype.

As the trend toward ubiquitous computing continues context models and management frameworks will continue to advance and improve. Eventually we expect to see the convergence of these designs to a small number of universally accepted standards, much like what has happened with other technologies such as the Internet and World Wide Web. Only then will it be possible to fully enable context management on a larger scale across disparate systems and environments.

The notion of context has, until recently, been limited in scope and has focused on simple properties like location. As more sophisticated facilities for context management are developed we expect to see context encompass a wider range of concepts and data. Context will become more structured and context models will have to be expanded to accommodate the added complexity. Context will be viewed on the same level as application data and data queries will encompass both types of data.

Devices need to fade into the background in a pervasive computing environment. We therefore expect the autonomic computing paradigm to play an increasingly important role in these environments. Autonomic devices will have the ability to act on their own given high level guidance from their users. They will also be able to detect and adapt to changes in their environment.

The progression of embedding computing into our daily lives will necessitate the need for systems to actively handle and share ever-increasing amounts of sensitive information about our environments and the people that occupy them. As issues of security and privacy have already begun to surface in environments where computing systems are empowered with our personal data, ubiquitous systems will further the notion. Standards will need to be established that define strict parameters governing how personal data is processed and exactly what information is considered “public” or “free”. The importance of such standards will increase as data-sharing scopes widen among the technologies rapidly becoming embedded into our environments.

References

[1] Altova GmbH (2007). “Altova XMLSpy 2008 Enterprise Edition Content Model View”. Retrieved September 1, 2007 from http://www.altova.com/manual2008/XMLSpy/SpyEnterprise/index.html?contentmodelview.htm.

[2] Anagnostopoulos, C., Tsounis, A. & Hadjiefthymiades, S. (2005). Context Management in Pervasive Computing Environments. *Proceedings of International Conference on Pervasive Services,* pp.421-424, Santorini, Greece (2005).

[3] Apache Software Foundation (2007). “Apache Muse - A Java-based implementation of WSRF 1.2, WSN 1.3, and WSDM 1.1”. Retrieved May 1, 2007 from http://ws.apache.org/muse/.

[4] Apple, Inc. (2008) "Networking - Bonjour". Retrieved March 15, 2008 from http://developer.apple.com/networking/bonjour/index.html.

[5] Assad, M., Carmichael, D.J., Kay, J. & Kummerfeld, B. (2007). PersonisAD: Distributed, Active, Scrutable Model Framework for Context-Aware Services. A. LaMarca et al. (Eds.): *Proc. Pervasive 2007, LNCS 4480*, pp. 55-72, Springer (2007).

[6] Basagni, S., Conti, M., Giordano, S., & Stojmenović, I. (2004) *Mobile Ad Hoc Networking*. Hoboken: John Wiley & Sons.

[7] Chen, G. and Kotz, D. (2000). A Survey of Context-Aware Mobile Computing Research. Technical Report. UMI Order Number: TR2000-381., Dartmouth College.

[8] Chen, Dr. Harry Lik (2004). "An Intelligent Broker Architecture for Pervasive Context-Aware Systems". Retrieved May 1, 2007 from http://ebiquity.umbc.edu/get/a/publication/152.pdf.

[9] Chen, H., Perich, F., Finin, T., & Joshi, A. (2004). "SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications". Retrieved May 1, 2007 from http://ebiquity.umbc.edu/paper/html/id/168/.

[10] Chesire, S. and SteinBerg, D. H. (2005) *Zero Configuration Networking: The Definitive Guide*. Sebastopol: O'Reilly.

[11] Chesire, Stuart (2008). "DNS Service Discovery (DNS-SD)". Retrieved March 15, 2008 from http://www.dns-sd.org.

[12] Christopoulou, E., Goumopoulos, C. & Kameas, A. (2005). An ontology-based context management and reasoning process for UbiComp applications. *Proceedings of the 2005 Joint Conference on Smart Objects and Ambient Intelligence,* pp.265-270, Grenoble, France (2005).

[13] da Costa, C., Yamin, A. & Geyer, C. (2008). Toward a General Software Infrastucture for Ubiquitous Computing. *IEEE Pervasive Computing*, 7(1), 64 – 73.

[14] da Rocha, R. and Endler, M. (2005). Evolutionary and efficient context management in heterogeneous environments. *Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing,* pp.1-7, Grenoble, France (2005).

[15] Dey, A. (2001). Understanding and Using Context. *Personal and Ubiquitous Computing 7*, 4 - 7.

[16] Eclipse Foundation (2008). “Eclipse Test & Performance Tools Platform Project”. Retrieved August 30, 2008 from http://www.eclipse.org/tptp

[17] FIPA (2003). "FIPA Agent Management Specification". Retrieved March 15, 2008 from http://www.fipa.org/specs/fipa00023/SC00023K.html.

[18] ISO/IEC 7498-1 (1994). “Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model”. International Organization for Standardization, Geneva, Switzerland.

[19] OASIS (2006). "OASIS Web Services Distributed Management (WSDM) TC". Retrieved June 1, 2007 from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm#overview.

[20] OASIS (2006). “Web Services Distributed Management: Management Using Web Services (MUWS 1.1) Part 1”. Retrieved June 1, 2007 from http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf.

[21] OASIS (2006). "Web Services Distributed Management: MUWS Primer". Retrieved June 1, 2007 from http://www.oasis-open.org/committees/download.php/17000/wsdm-1.0-muws-primer-cd-01.doc.

[22] OASIS (2006). “Web Services Resource 1.2”. Retrieved June 1, 2007 from http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.

[23] OASIS (2006). “Web Services Resource Properties 1.2”. Retrieved July 1, 2007 from http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf.

[24] PostgreSQL Global Development Group (2007). "PostgreSQL: The world's most advanced open source database". Retrieved August 15, 2007 from http://www.postgresql.org.

[25] Proliphix, Inc. (2005). "Proliphix, Inc.'s Internet-Powered NT120e Thermostat Awarded Editor's Pick for April by CE Pro Magazine". Retrieved May 1, 2007 from http://www.proliphix.com/news_detail.aspx?news_id=4.

[26] Stanford University (2008). "Remembering Mark Weiser". Retrieved August 31, 2008 from http://library.stanford.edu/weiser.

[27] Strang, T. and Linnhoff-Popien, C. (2004). A Context Modeling Survey In J. Indulska & D. De Roure (Eds.) *Proc of First International Workshop on Context Modeling, Reasoning and Management (UbiComp 2004)*. Nottingham England.

[28] Weiser, M. and Brown, J. S. (1996). "The Coming Age of Calm Technology". Pp. 75-85 in *Beyond calculation: the next fifty years*. New York, NY: Copernicus.

[29] Weiser, Mark. (1991). The Computer for the 21'st Century. *Scientific American*, Sept. 1991, pp. 94-104.

[30] W3C (2004)."Web Services Architecture". Retrieved Jan 20, 2008 from http://www.w3.org/TR/ws-arch/.

[31] W3C (2008). "Delivery Context Ontology". Retrieved May 31, 2008 from http://www.w3.org/TR/2008/WD-dcontology-20080415/.

[32] W3C (2004). "XML Schema Part 2: Datatypes Second Edition". Retrieved June 1, 2007 from http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

Appendix A

Context Model XML Schema and Instance Documents

1. Master Domain Ontology Schema Document.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.cs.queensu.ca/OntologySchema"
  targetNamespace="http://www.cs.queensu.ca/OntologySchema"
  xmlns:meta="http://www.cs.queensu.ca/MetadataSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:annotation>
    <xs:documentation xml:lang="en">
      Schema for Creating Context-Managed Environment Ontologies
      Created by Jared A. Zebedee (2007)
    </xs:documentation>
  </xs:annotation>

  <xs:import namespace="http://www.w3.org/2001/XMLSchema"
  schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>

  <xs:import namespace="http://www.cs.queensu.ca/MetadataSchema"
  schemaLocation="http://www.cs.queensu.ca/MetadataSchema.xsd"/>

  <xs:simpleType name="propertyType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="boolean"/>
      <xs:enumeration value="integer"/>
      <xs:enumeration value="string"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="Ontology">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Domain" type="xs:string"/>
        <xs:element name="Region" type="xs:string" maxOccurs="unbounded"/>
        <xs:element name="DeviceType" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Property" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="name" type="xs:string" use="required"/>
                  <xs:attribute name="dataType" type="propertyType" use="required"/>
                  <xs:attribute ref="meta:mutable" use="required"/>
                  <xs:attribute ref="meta:subscribable" use="optional" default="false"/>
                  <xs:attribute ref="meta:ValidStringValues" use="optional"/>
                  <xs:attribute ref="meta:ValidIntegerValues" use="optional"/>
                  <xs:attribute name="minOccurrence" type="xs:nonNegativeInteger"
use="optional" default="1"/>
                  <xs:attribute name="maxOccurrence" type="xs:allNNI" use="optional"
default="1"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

2. ComputerScienceBuilding Ontology XML Instance Document.

```
<Ontology xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.cs.queensu.ca/OntologySchema
http://www.cs.queensu.ca/OntologySchema.xsd"
xmlns="http://www.cs.queensu.ca/OntologySchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">

  <!-- Domain Definition -->
  <Domain>ComputerScienceBuilding</Domain>

  <!-- Region Definitions -->
  <Region>Conference Room</Region>
  <Region>Lecture Hall</Region>
  <Region>Main Office</Region>
  <Region>Lab</Region>

  <!-- PervasiveDevice DeviceType Definition -->
  <DeviceType name="PervasiveDevice">
    <Property name="UniqueName" dataType="string" mutable="false"/>
    <Property name="DeviceType" dataType="string" mutable="false"
ValidStringValues="Data_Projector Laser_Printer PDA Smart_Thermostat"/>
    <Property name="Owner" dataType="string" mutable="true"/>
    <Property name="Location" dataType="string" subscribable="true" mutable="true"
ValidStringValues="Conference_Room Lecture_Hall Main_Office Lab"/>
    <Property name="HasDisplayScreen" dataType="boolean" mutable="false"/>
  </DeviceType>

  <!-- DataProjector DeviceType Definition -->
  <DeviceType name="DataProjector">
    <Property name="LampActive" dataType="boolean" mutable="true"/>
    <Property name="SlidesLoaded" dataType="boolean" mutable="false"/>
    <Property name="SlideNumber" dataType="integer" mutable="false"/>
  </DeviceType>

  <!-- LaserPrinter DeviceType Definition -->
  <DeviceType name="LaserPrinter">
    <Property name="CanPrintInColour" dataType="boolean" mutable="false"/>
    <Property name="IsOnline" dataType="boolean" mutable="true"/>
    <Property name="PaperRemaining" dataType="integer" mutable="true"/>
  </DeviceType>

  <!-- PDA DeviceType Definition -->
  <DeviceType name="PDA">
    <Property name="BacklightActive" dataType="boolean" mutable="true"/>
    <Property name="BatteryLevel" dataType="integer" mutable="true"/>
    <Property name="LowBattery" dataType="boolean" mutable="true"/>
  </DeviceType>

  <!-- SmartThermostat DeviceType Definition -->
  <DeviceType name="SmartThermostat">
    <Property name="CurrentTemperature" dataType="integer" mutable="true"/>
    <Property name="TemperatureSetting" dataType="integer" mutable="true"/>
    <Property name="IsCelcius" dataType="boolean" mutable="true"/>
  </DeviceType>
</Ontology>
```

3. Metadata Schema Document.
(Used by Ontology and Device Schemas)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.cs.queensu.ca/MetadataSchema"
  targetNamespace="http://www.cs.queensu.ca/MetadataSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">>

  <xsd:simpleType name="stringList">
    <xsd:list itemType="xsd:string"/>
  </xsd:simpleType>

  <xsd:simpleType name="integerList">
    <xsd:list itemType="xsd:integer"/>
  </xsd:simpleType>

  <xsd:attribute name="mutable" type="xsd:boolean"/>
  <xsd:attribute name="subscribable" type="xsd:boolean"/>
  <xsd:attribute name="ValidStringValues" type="stringList"/>
  <xsd:attribute name="ValidIntegerValues" type="integerList"/>
</xsd:schema>
```

4. ComputerScienceBuilding Prose Description Document

The ComputerScienceBuilding domain represents a hypothetical university computer science building comprised of one floor containing four device-types and four regions (rooms). The device types are Data Projector, Laser Printer, PDA and Smart Thermostat. The regions are Conference Room, Lecture Hall, Main Office and Lab. The building floor plan is illustrated in Figure A.1.

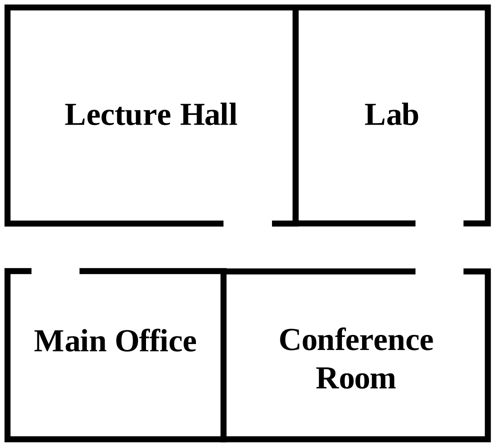


Figure A.1 ComputerScienceBuilding floor plan

5. Laser Printer Device Schema Document.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:meta="http://www.cs.queensu.ca/MetadataSchema"
xmlns="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDevice"
targetNamespace="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDevice"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      Laser Printer Device Schema
      Created by Jared A. Zebedee (2007)
    </xs:documentation>
  </xs:annotation>

  <xs:import namespace="http://www.cs.queensu.ca/MetadataSchema"
schemaLocation="http://www.cs.queensu.ca/MetadataSchema.xsd"/>

  <!-- Top-level Pervasive Device Definition -->
  <xs:element name="Device">
    <xs:complexType>
      <xs:sequence>

        <!-- PervasiveDevice device-type (required) -->
        <xs:element name="PervasiveDevice">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="UniqueName" type="xs:string" meta:mutable="false"/>
              <xs:element name="DeviceType" type="xs:string" meta:mutable="false"
meta:ValidStringValues="Laser_Printer"/>
              <xs:element name="Owner" type="xs:string" meta:mutable="true"/>
              <xs:element name="Location" type="xs:string" meta:mutable="true"
meta:subscribable="true" meta:ValidStringValues="Conference_Room Lecture_Hall Main_Office
Lab"/>
              <xs:element name="HasDisplayScreen" type="xs:boolean" meta:mutable="false"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <!-- /PervasiveDevice -->

        <!-- LaserPrinter Device Definition -->
        <xs:element name="LaserPrinter">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CanPrintInColour" type="xs:boolean" meta:mutable="false"/>
              <xs:element name="IsOnline" type="xs:boolean" meta:mutable="true"/>
              <xs:element name="PaperRemaining" type="xs:integer" meta:mutable="true"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <!-- /LaserPrinter -->
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- /Device -->
</xs:schema>
```

6. Laser Printer Context Profile XML Instance Document.

```xml
<Device xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDevice
http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDevice.xsd"
xmlns="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDevice"
elementFormDefault="qualified" attributeFormDefault="unqualified">

  <PervasiveDevice>
    <UniqueName>Conference_Room_Printer</UniqueName>
    <DeviceType>Laser_Printer</DeviceType>
    <Owner>Conference_Room_Manager</Owner>
    <Location>Conference_Room</Location>
    <HasDisplayScreen>true</HasDisplayScreen>
  </PervasiveDevice>

  <LaserPrinter>
    <CanPrintInColour>true</CanPrintInColour>
    <IsOnline>true</IsOnline>
    <PaperRemaining>500</PaperRemaining>
  </LaserPrinter>

</Device>
```

Appendix B

WSDM MR Creation Templates

1. WSDL Template.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="{domain namespace}/{device name}"
  xmlns:tns="{domain namespace}/{device name}"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdl-soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
  xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
  xmlns:wsrmd="http://docs.oasis-open.org/wsrf/rmd-1"
  xmlns:muws1="http://docs.oasis-open.org/wsdm/muws1-2.xsd"
  xmlns:muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd"

  xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
  xmlns:wsntw="http://docs.oasis-open.org/wsn/bw-2"
  xmlns:wst="http://docs.oasis-open.org/wsn/t-1"

  name="{device name}">

  <wsdl:types>

    <!-- Include WSDM Base Schemas -->

    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://www.w3.org/2005/08/addressing">
      <xsd:include schemaLocation="WS-Addressing-2005_08.xsd" />
    </xsd:schema>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://schemas.xmlsoap.org/ws/2004/09/mex">
      <xsd:include
        schemaLocation="WS-MetadataExchange-2004_09.xsd" />
    </xsd:schema>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://docs.oasis-open.org/wsrf/rp-2">
      <xsd:include schemaLocation="WS-ResourceProperties-1_2.xsd" />
    </xsd:schema>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://docs.oasis-open.org/wsrf/rmd-1">
      <xsd:include
        schemaLocation="WS-ResourceMetadataDescriptor-CD-01.xsd" />
    </xsd:schema>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://docs.oasis-open.org/wsdm/muws1-2.xsd">
      <xsd:include schemaLocation="WSDM-MUWS-Part1-1_1.xsd" />
    </xsd:schema>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://docs.oasis-open.org/wsdm/muws2-2.xsd">
      <xsd:include schemaLocation="WSDM-MUWS-Part2-1_1.xsd" />
    </xsd:schema>

    <!-- Include WSN Schemas -->

    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://docs.oasis-open.org/wsn/b-2">
      <xsd:include schemaLocation="WS-BaseNotification-1_3.xsd" />
    </xsd:schema>
    <xsd:schema elementFormDefault="qualified"
```

```
     targetNamespace="http://docs.oasis-open.org/wsn/t-1">
     <xsd:include schemaLocation="WS-Topics-1_3.xsd" />
   </xsd:schema>

   <!-- Import Resource-Specific Schemas -->

     <xsd:schema elementFormDefault="qualified"
targetNamespace="{domain name}/{device-type name}">
       <xsd:include schemaLocation="{schema location}" />
   </xsd:schema>

  </wsdl:types>

   <!-- WSDM Base Message definitions -->

  <wsdl:message name="GetMetadataMsg">
    <wsdl:part name="GetMetadataMsg" element="wsx:GetMetadata" />
  </wsdl:message>
  <wsdl:message name="GetMetadataResponseMsg">
    <wsdl:part name="GetMetadataResponseMsg" element="wsx:Metadata" />
  </wsdl:message>
  <wsdl:message name="GetResourcePropertyDocumentRequest">
    <wsdl:part name="GetResourcePropertyDocumentRequest"
      element="wsrf-rp:GetResourcePropertyDocument" />
  </wsdl:message>
  <wsdl:message name="GetResourcePropertyDocumentResponse">
    <wsdl:part name="GetResourcePropertyDocumentResponse"
      element="wsrf-rp:GetResourcePropertyDocumentResponse" />
  </wsdl:message>
  <wsdl:message name="GetResourcePropertyRequest">
    <wsdl:part name="GetResourcePropertyRequest"
      element="wsrf-rp:GetResourceProperty" />
  </wsdl:message>
  <wsdl:message name="GetResourcePropertyResponse">
    <wsdl:part name="GetResourcePropertyResponse"
      element="wsrf-rp:GetResourcePropertyResponse" />
  </wsdl:message>
  <wsdl:message name="InvalidResourcePropertyQNameFault">
    <wsdl:part name="InvalidResourcePropertyQNameFault"
      element="wsrf-rp:InvalidResourcePropertyQNameFault" />
  </wsdl:message>
  <wsdl:message name="GetMultipleResourcePropertiesRequest">
    <wsdl:part name="GetMultipleResourcePropertiesRequest"
      element="wsrf-rp:GetMultipleResourceProperties" />
  </wsdl:message>
  <wsdl:message name="GetMultipleResourcePropertiesResponse">
    <wsdl:part name="GetMultipleResourcePropertiesResponse"
      element="wsrf-rp:GetMultipleResourcePropertiesResponse" />
  </wsdl:message>
  <wsdl:message name="QueryResourcePropertiesRequest">
    <wsdl:part name="QueryResourcePropertiesRequest"
      element="wsrf-rp:QueryResourceProperties" />
  </wsdl:message>
  <wsdl:message name="QueryResourcePropertiesResponse">
    <wsdl:part name="QueryResourcePropertiesResponse"
      element="wsrf-rp:QueryResourcePropertiesResponse" />
  </wsdl:message>
  <wsdl:message name="UnknownQueryExpressionDialectFault">
    <wsdl:part name="UnknownQueryExpressionDialectFault"
      element="wsrf-rp:UnknownQueryExpressionDialectFault" />
  </wsdl:message>
  <wsdl:message name="InvalidQueryExpressionFault">
    <wsdl:part name="InvalidQueryExpressionFault"
      element="wsrf-rp:InvalidQueryExpressionFault" />
  </wsdl:message>
  <wsdl:message name="QueryEvaluationErrorFault">
    <wsdl:part name="QueryEvaluationErrorFault"
      element="wsrf-rp:QueryEvaluationErrorFault" />
  </wsdl:message>
  <wsdl:message name="SetResourcePropertiesRequest">
    <wsdl:part name="SetResourcePropertiesRequest"
      element="wsrf-rp:SetResourceProperties" />
  </wsdl:message>
  <wsdl:message name="SetResourcePropertiesResponse">
    <wsdl:part name="SetResourcePropertiesResponse"
      element="wsrf-rp:SetResourcePropertiesResponse" />
  </wsdl:message>
  <wsdl:message name="InvalidModificationFault">
    <wsdl:part name="InvalidModificationFault"
      element="wsrf-rp:InvalidModificationFault" />
  </wsdl:message>
  <wsdl:message name="UnableToModifyResourcePropertyFault">
    <wsdl:part name="UnableToModifyResourcePropertyFault"
      element="wsrf-rp:UnableToModifyResourcePropertyFault" />
  </wsdl:message>
  <wsdl:message name="SetResourcePropertyRequestFailedFault">
    <wsdl:part name="SetResourcePropertyRequestFailedFault"
      element="wsrf-rp:SetResourcePropertyRequestFailedFault" />
  </wsdl:message>

      <!-- WSN Message Definitions -->
      <wsdl:message name="SubscribeRequest" >
       <wsdl:part name="SubscribeRequest"
                 element="wsnt:Subscribe"/>
    </wsdl:message>

    <wsdl:message name="SubscribeResponse">
       <wsdl:part name="SubscribeResponse"
                 element="wsnt:SubscribeResponse"/>
    </wsdl:message>

    <wsdl:message name="SubscribeCreationFailedFault">
       <wsdl:part name="SubscribeCreationFailedFault"
             element="wsnt:SubscribeCreationFailedFault" />
    </wsdl:message>

    <wsdl:message name="TopicExpressionDialectUnknownFault">
       <wsdl:part name="TopicExpressionDialectUnknownFault"
             element="wsnt:TopicExpressionDialectUnknownFault" />
    </wsdl:message>

    <wsdl:message name="InvalidFilterFault">
       <wsdl:part name="InvalidFilterFault"
             element="wsnt:InvalidFilterFault" />
    </wsdl:message>

    <wsdl:message name="InvalidProducerPropertiesExpressionFault">
       <wsdl:part name="InvalidProducerPropertiesExpressionFault"
             element="wsnt:InvalidProducerPropertiesExpressionFault" />
    </wsdl:message>

    <wsdl:message name="InvalidMessageContentExpressionFault">
       <wsdl:part name="InvalidMessageContentExpressionFault"
             element="wsnt:InvalidMessageContentExpressionFault" />
    </wsdl:message>

    <wsdl:message name="UnrecognizedPolicyRequestFault">
       <wsdl:part name="UnrecognizedPolicyRequestFault"
             element="wsnt:UnrecognizedPolicyRequestFault" />
    </wsdl:message>

    <wsdl:message name="UnsupportedPolicyRequestFault">
       <wsdl:part name="UnsupportedPolicyRequestFault"
             element="wsnt:UnsupportedPolicyRequestFault" />
    </wsdl:message>

    <wsdl:message name="NotifyMessageNotSupportedFault">
       <wsdl:part name="NotifyMessageNotSupportedFault"
             element="wsnt:NotifyMessageNotSupportedFault" />
    </wsdl:message>

    <wsdl:message name="UnacceptableInitialTerminationTimeFault">
```

```
      <wsdl:part name="UnacceptableInitialTerminationTimeFault"
            element="wsnt:UnacceptableInitialTerminationTimeFault"/>
   </wsdl:message>

   <wsdl:message name="GetCurrentMessageRequest">
      <wsdl:part name="GetCurrentMessageRequest"
            element="wsnt:GetCurrentMessage"/>
   </wsdl:message>

   <wsdl:message name="GetCurrentMessageResponse">
      <wsdl:part name="GetCurrentMessageResponse"
            element="wsnt:GetCurrentMessageResponse"/>
   </wsdl:message>

   <wsdl:message name="InvalidTopicExpressionFault">
      <wsdl:part name="InvalidTopicExpressionFault"
            element="wsnt:InvalidTopicExpressionFault" />
   </wsdl:message>

   <wsdl:message name="TopicNotSupportedFault">
      <wsdl:part name="TopicNotSupportedFault"
            element="wsnt:TopicNotSupportedFault" />
   </wsdl:message>

   <wsdl:message name="MultipleTopicsSpecifiedFault">
      <wsdl:part name="MultipleTopicsSpecifiedFault"
            element="wsnt:MultipleTopicsSpecifiedFault" />
   </wsdl:message>

   <wsdl:message name="NoCurrentMessageOnTopicFault">
      <wsdl:part name="NoCurrentMessageOnTopicFault"
            element="wsnt:NoCurrentMessageOnTopicFault" />
   </wsdl:message>

  <!-- WSDM Base Operations -->
    <wsdl:portType name="{device name}ResourcePortType"
   wsrf-rp:ResourceProperties="tns:{Device name}ResourceProperties"
   wsrmd:Descriptor="{device name}MetadataDescriptor"
   wsrmd:DescriptorLocation="{device name}.rmd">
   <wsdl:operation name="GetMetadata">
     <wsdl:input
      wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata"
      name="GetMetadataMsg" message="tns:GetMetadataMsg" />
     <wsdl:output
      wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadataResponse"
      name="GetMetadataResponseMsg" message="tns:GetMetadataResponseMsg" />
   </wsdl:operation>
   <wsdl:operation name="GetResourcePropertyDocument">
     <wsdl:input
      wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/GetResourcePropertyDocument/GetResourcePropertyDocumentRequest"
      name="GetResourcePropertyDocumentRequest"
      message="tns:GetResourcePropertyDocumentRequest" />
     <wsdl:output
      wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/GetResourcePropertyDocument/GetResourcePropertyDocumentResponse"
      name="GetResourcePropertyDocumentResponse"
      message="tns:GetResourcePropertyDocumentResponse" />
     <wsdl:fault name="ResourceUnknownFault"
      message="tns:ResourceUnknownFault" />
     <wsdl:fault name="ResourceUnavailableFault"
      message="tns:ResourceUnavailableFault" />
   </wsdl:operation>
   <wsdl:operation name="GetResourceProperty">
     <wsdl:input
      wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/GetResourceProperty/GetResourcePropertyRequest"
      name="GetResourcePropertyRequest"
      message="tns:GetResourcePropertyRequest" />
     <wsdl:output
```

```
      wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/GetResourceProperty/GetResourcePropertyResponse"
      name="GetResourcePropertyResponse"
      message="tns:GetResourcePropertyResponse" />
     <wsdl:fault name="ResourceUnknownFault"
      message="tns:ResourceUnknownFault" />
     <wsdl:fault name="ResourceUnavailableFault"
      message="tns:ResourceUnavailableFault" />
     <wsdl:fault name="InvalidResourcePropertyQNameFault"
      message="tns:InvalidResourcePropertyQNameFault" />
   </wsdl:operation>
   <wsdl:operation name="GetMultipleResourceProperties">
     <wsdl:input
      wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/GetMultipleResourceProperties/GetMultipleResourcePropertiesRequest"
      name="GetMultipleResourcePropertiesRequest"
      message="tns:GetMultipleResourcePropertiesRequest" />
     <wsdl:output
      wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/GetMultipleResourceProperties/GetMultipleResourcePropertiesResponse"
      name="GetMultipleResourcePropertiesResponse"
      message="tns:GetMultipleResourcePropertiesResponse" />
     <wsdl:fault name="ResourceUnknownFault"
      message="tns:ResourceUnknownFault" />
     <wsdl:fault name="ResourceUnavailableFault"
      message="tns:ResourceUnavailableFault" />
     <wsdl:fault name="InvalidResourcePropertyQNameFault"
      message="tns:InvalidResourcePropertyQNameFault" />
   </wsdl:operation>
   <wsdl:operation name="QueryResourceProperties">
     <wsdl:input
      wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/QueryResourceProperties/QueryResourcePropertiesRequest"
      name="QueryResourcePropertiesRequest"
      message="tns:QueryResourcePropertiesRequest" />
     <wsdl:output
      wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/QueryResourceProperties/QueryResourcePropertiesResponse"
      name="QueryResourcePropertiesResponse"
      message="tns:QueryResourcePropertiesResponse" />
     <wsdl:fault name="ResourceUnknownFault"
      message="tns:ResourceUnknownFault" />
     <wsdl:fault name="ResourceUnavailableFault"
      message="tns:ResourceUnavailableFault" />
     <wsdl:fault name="UnknownQueryExpressionDialectFault"
      message="tns:UnknownQueryExpressionDialectFault" />
     <wsdl:fault name="InvalidQueryExpressionFault"
      message="tns:InvalidQueryExpressionFault" />
     <wsdl:fault name="QueryEvaluationErrorFault"
      message="tns:QueryEvaluationErrorFault" />
   </wsdl:operation>
        <wsdl:operation name="SetResourceProperties">
     <wsdl:input
      wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/SetResourceProperties/SetResourcePropertiesRequest"
      name="SetResourcePropertiesRequest"
      message="tns:SetResourcePropertiesRequest" />
     <wsdl:output
      wsa:Action="http://docs.oasis-open.org/wsrf/rpw-
2/SetResourceProperties/SetResourcePropertiesResponse"
      name="SetResourcePropertiesResponse"
      message="tns:SetResourcePropertiesResponse" />
     <wsdl:fault name="ResourceUnknownFault"
      message="tns:ResourceUnknownFault" />
     <wsdl:fault name="ResourceUnavailableFault"
      message="tns:ResourceUnavailableFault" />
     <wsdl:fault name="InvalidModificationFault"
      message="tns:InvalidModificationFault" />
     <wsdl:fault name="UnableToModifyResourcePropertyFault"
      message="tns:UnableToModifyResourcePropertyFault" />
     <wsdl:fault name="InvalidResourcePropertyQNameFault"
```

```
      message="tns:InvalidResourcePropertyQNameFault" />
     <wsdl:fault name="SetResourcePropertyRequestFailedFault"
      message="tns:SetResourcePropertyRequestFailedFault" />
   </wsdl:operation>

    <!-- wsntw:NotificationProducer operations -->
      <wsdl:operation name="Subscribe">
         <wsdl:input  wsa:Action="http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/SubscribeRequest"
                      name="SubscribeRequest"
               message="tns:SubscribeRequest" />
         <wsdl:output wsa:Action="http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/SubscribeResponse"
                      name="SubscribeResponse"
               message="tns:SubscribeResponse" />
         <wsdl:fault  name="ResourceUnknownFault"
                      message="tns:ResourceUnknownFault" />
         <wsdl:fault  name="InvalidFilterFault"
                   message="tns:InvalidFilterFault"/>
         <wsdl:fault  name="TopicExpressionDialectUnknownFault"
                   message="tns:TopicExpressionDialectUnknownFault"/>
         <wsdl:fault  name="InvalidTopicExpressionFault"
                      message="tns:InvalidTopicExpressionFault" />
         <wsdl:fault  name="TopicNotSupportedFault"
                      message="tns:TopicNotSupportedFault" />
         <wsdl:fault  name="InvalidProducerPropertiesExpressionFault"
                      message="tns:InvalidProducerPropertiesExpressionFault"/>
         <wsdl:fault  name="InvalidMessageContentExpressionFault"
                      message="tns:InvalidMessageContentExpressionFault"/>
         <wsdl:fault  name="UnacceptableInitialTerminationTimeFault"
                      message="tns:UnacceptableInitialTerminationTimeFault"/>
         <wsdl:fault  name="UnrecognizedPolicyRequestFault"
                      message="tns:UnrecognizedPolicyRequestFault"/>
         <wsdl:fault  name="UnsupportedPolicyRequestFault"
                      message="tns:UnsupportedPolicyRequestFault"/>
         <wsdl:fault  name="NotifyMessageNotSupportedFault"
                      message="tns:NotifyMessageNotSupportedFault"/>
         <wsdl:fault  name="SubscribeCreationFailedFault"
                      message="tns:SubscribeCreationFailedFault"/>
      </wsdl:operation>
      <wsdl:operation name="GetCurrentMessage">
         <wsdl:input  wsa:Action="http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/GetCurrentMessageRequest"
                      name="GetCurrentMessageRequest"
               message="tns:GetCurrentMessageRequest"/>
         <wsdl:output wsa:Action="http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/GetCurrentMessageResponse"
                      name="GetCurrentMessageResponse"
               message="tns:GetCurrentMessageResponse"/>
         <wsdl:fault  name="ResourceUnknownFault"
                      message="tns:ResourceUnknownFault" />
         <wsdl:fault  name="TopicExpressionDialectUnknownFault"
                      message="tns:TopicExpressionDialectUnknownFault"/>
         <wsdl:fault  name="InvalidTopicExpressionFault"
                      message="tns:InvalidTopicExpressionFault" />
         <wsdl:fault  name="TopicNotSupportedFault"
                      message="tns:TopicNotSupportedFault" />
         <wsdl:fault  name="NoCurrentMessageOnTopicFault"
                      message="tns:NoCurrentMessageOnTopicFault" />
         <wsdl:fault  name="MultipleTopicsSpecifiedFault"
                      message="tns:MultipleTopicsSpecifiedFault" />
      </wsdl:operation>

  </wsdl:portType>

   <!-- WSDM Base Operation bindings -->

  <wsdl:binding name="{device name}ResourceBinding"
    type="tns:{device name}ResourcePortType">
    <wsdl-soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
```

```
<wsdl:operation name="GetMetadata">
  <wsdl-soap:operation soapAction="GetMetadata" />
  <wsdl:input>
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:input>
  <wsdl:output>
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetResourcePropertyDocument">
  <wsdl-soap:operation
    soapAction="GetResourcePropertyDocument" />
  <wsdl:input name="GetResourcePropertyDocumentRequest">
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:input>
  <wsdl:output name="GetResourcePropertyDocumentResponse">
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:output>
  <wsdl:fault name="ResourceUnknownFault">
    <wsdl-soap:fault use="encoded"
      name="ResourceUnknownFault" />
  </wsdl:fault>
  <wsdl:fault name="ResourceUnavailableFault">
    <wsdl-soap:fault use="encoded"
      name="ResourceUnavailableFault" />
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="GetResourceProperty">
  <wsdl-soap:operation soapAction="GetResourceProperty" />
  <wsdl:input name="GetResourcePropertyRequest">
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:input>
  <wsdl:output name="GetResourcePropertyResponse">
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:output>
  <wsdl:fault name="ResourceUnknownFault">
    <wsdl-soap:fault use="encoded"
      name="ResourceUnknownFault" />
  </wsdl:fault>
  <wsdl:fault name="ResourceUnavailableFault">
    <wsdl-soap:fault use="encoded"
      name="ResourceUnavailableFault" />
  </wsdl:fault>
  <wsdl:fault name="InvalidResourcePropertyQNameFault">
    <wsdl-soap:fault use="encoded"
      name="InvalidResourcePropertyQNameFault" />
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="GetMultipleResourceProperties">
  <wsdl-soap:operation
    soapAction="GetMultipleResourceProperties" />
  <wsdl:input name="GetMultipleResourcePropertiesRequest">
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:input>
  <wsdl:output name="GetMultipleResourcePropertiesResponse">
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:output>
  <wsdl:fault name="ResourceUnknownFault">
    <wsdl-soap:fault use="encoded"
      name="ResourceUnknownFault" />
  </wsdl:fault>
  <wsdl:fault name="ResourceUnavailableFault">
    <wsdl-soap:fault use="encoded"
```

```
      name="ResourceUnavailableFault" />
  </wsdl:fault>
  <wsdl:fault name="InvalidResourcePropertyQNameFault">
    <wsdl-soap:fault use="encoded"
      name="InvalidResourcePropertyQNameFault" />
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="QueryResourceProperties">
  <wsdl-soap:operation soapAction="QueryResourceProperties" />
  <wsdl:input name="QueryResourcePropertiesRequest">
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:input>
  <wsdl:output name="QueryResourcePropertiesResponse">
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:output>
  <wsdl:fault name="ResourceUnknownFault">
    <wsdl-soap:fault use="encoded"
      name="ResourceUnknownFault" />
  </wsdl:fault>
  <wsdl:fault name="ResourceUnavailableFault">
    <wsdl-soap:fault use="encoded"
      name="ResourceUnavailableFault" />
  </wsdl:fault>
  <wsdl:fault name="UnknownQueryExpressionDialectFault">
    <wsdl-soap:fault use="encoded"
      name="UnknownQueryExpressionDialectFault" />
  </wsdl:fault>
  <wsdl:fault name="InvalidQueryExpressionFault">
    <wsdl-soap:fault use="encoded"
      name="InvalidQueryExpressionFault" />
  </wsdl:fault>
  <wsdl:fault name="QueryEvaluationErrorFault">
    <wsdl-soap:fault use="encoded"
      name="QueryEvaluationErrorFault" />
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="SetResourceProperties">
  <wsdl-soap:operation
    soapAction="SetResourceProperties" />
  <wsdl:input name="SetResourcePropertiesRequest">
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:input>
  <wsdl:output name="SetResourcePropertiesResponse">
    <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </wsdl:output>
  <wsdl:fault name="ResourceUnknownFault">
    <wsdl-soap:fault use="encoded"
      name="ResourceUnknownFault" />
  </wsdl:fault>
  <wsdl:fault name="ResourceUnavailableFault">
    <wsdl-soap:fault use="encoded"
      name="ResourceUnavailableFault" />
  </wsdl:fault>
  <wsdl:fault name="InvalidModificationFault">
    <wsdl-soap:fault use="encoded"
      name="InvalidModificationFault" />
  </wsdl:fault>
  <wsdl:fault name="UnableToModifyResourcePropertyFault">
    <wsdl-soap:fault use="encoded"
      name="UnableToModifyResourcePropertyFault" />
  </wsdl:fault>
  <wsdl:fault name="InvalidResourcePropertyQNameFault">
    <wsdl-soap:fault use="encoded"
      name="InvalidResourcePropertyQNameFault" />
  </wsdl:fault>
  <wsdl:fault name="SetResourcePropertyRequestFailedFault">
    <wsdl-soap:fault use="encoded"
    name="SetResourcePropertyRequestFailedFault" />
  </wsdl:fault>
</wsdl:operation>

   <!-- wsntw:NotificationProducer Operation bindings -->
     <wsdl:operation name="Subscribe">
      <wsdl-soap:operation
        soapAction="Subscribe" />
      <wsdl:input name="SubscribeRequest">
        <wsdl-soap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </wsdl:input>
      <wsdl:output name="SubscribeResponse">
        <wsdl-soap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </wsdl:output>
      <wsdl:fault name="ResourceUnknownFault">
        <wsdl-soap:fault
          use="encoded" name="ResourceUnknownFault" />
      </wsdl:fault>
      <wsdl:fault name="InvalidFilterFault">
        <wsdl-soap:fault
          use="encoded" name="InvalidFilterFault" />
      </wsdl:fault>
      <wsdl:fault name="TopicExpressionDialectUnknownFault">
        <wsdl-soap:fault
          use="encoded" name="TopicExpressionDialectUnknownFault" />
      </wsdl:fault>
      <wsdl:fault name="InvalidTopicExpressionFault">
        <wsdl-soap:fault
          use="encoded" name="InvalidTopicExpressionFault" />
      </wsdl:fault>
      <wsdl:fault name="TopicNotSupportedFault">
        <wsdl-soap:fault
          use="encoded" name="TopicNotSupportedFault" />
      </wsdl:fault>
      <wsdl:fault
        name="InvalidProducerPropertiesExpressionFault">
        <wsdl-soap:fault
          use="encoded"
          name="InvalidProducerPropertiesExpressionFault" />
      </wsdl:fault>
      <wsdl:fault name="InvalidMessageContentExpressionFault">
        <wsdl-soap:fault
          use="encoded" name="InvalidMessageContentExpressionFault" />
      </wsdl:fault>
      <wsdl:fault
        name="UnacceptableInitialTerminationTimeFault">
        <wsdl-soap:fault
          use="encoded" name="UnacceptableInitialTerminationTimeFault" />
      </wsdl:fault>
      <wsdl:fault name="UnrecognizedPolicyRequestFault">
        <wsdl-soap:fault
          use="encoded" name="UnrecognizedPolicyRequestFault" />
      </wsdl:fault>
      <wsdl:fault name="UnsupportedPolicyRequestFault">
        <wsdl-soap:fault
          use="encoded" name="UnsupportedPolicyRequestFault" />
      </wsdl:fault>
      <wsdl:fault name="NotifyMessageNotSupportedFault">
        <wsdl-soap:fault
          use="encoded" name="NotifyMessageNotSupportedFault" />
      </wsdl:fault>
      <wsdl:fault name="SubscribeCreationFailedFault">
        <wsdl-soap:fault
          use="encoded" name="SubscribeCreationFailedFault" />
      </wsdl:fault>
     </wsdl:operation>
     <wsdl:operation name="GetCurrentMessage">
      <wsdl-soap:operation
        soapAction="GetCurrentMessage"/>
```

```
          <wsdl:input name="GetCurrentMessageRequest">
            <wsdl-soap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
          </wsdl:input>
          <wsdl:output name="GetCurrentMessageResponse">
            <wsdl-soap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
          </wsdl:output>
          <wsdl:fault name="ResourceUnknownFault">
            <wsdl-soap:fault
              use="encoded" name="ResourceUnknownFault" />
          </wsdl:fault>
          <wsdl:fault name="TopicExpressionDialectUnknownFault">
            <wsdl-soap:fault
              use="encoded" name="TopicExpressionDialectUnknownFault" />
          </wsdl:fault>
          <wsdl:fault name="InvalidTopicExpressionFault">
            <wsdl-soap:fault
              use="encoded" name="InvalidTopicExpressionFault" />
          </wsdl:fault>
          <wsdl:fault name="TopicNotSupportedFault">
            <wsdl-soap:fault
              use="encoded" name="TopicNotSupportedFault" />
          </wsdl:fault>
          <wsdl:fault name="NoCurrentMessageOnTopicFault">
            <wsdl-soap:fault
              use="encoded" name="NoCurrentMessageOnTopicFault" />
          </wsdl:fault>
          <wsdl:fault name="MultipleTopicsSpecifiedFault">
            <wsdl-soap:fault
              use="encoded" name="MultipleTopicsSpecifiedFault" />
          </wsdl:fault>
         </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="{device name}ResourceService">
    <wsdl:port name="{device name}ResourcePort"
      binding="tns:{device name}ResourceBinding">
      <wsdl-soap:address
        location="{device URI}" />
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

2. RPSD Template.

```
<xsd:schema targetNamespace="{domain namespace}/{device name}"
  xmlns:tns="{domain namespace}/{device name}"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"

  <!-- WSDM Base Schema References -->
  xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
  xmlns:muws1="http://docs.oasis-open.org/wsdm/muws1-2.xsd"
  xmlns:muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd"

  <!-- WSN Schema References -->
  xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
  xmlns:wsntw="http://docs.oasis-open.org/wsn/bw-2"
  xmlns:wst="http://docs.oasis-open.org/wsn/t-1">

  <!-- Resource-Specific (Device-type) Schema References -->
  xmlns:{device-type prefix}="{domain namespace}/{device-type name}"
  {...}
  >

  <!-- Resource Properties Document Element References -->
  <xsd:element name="{device name}ResourceProperties">
```

```
   <xsd:complexType>
     <xsd:sequence>

       <!-- WSDM Base Resource Property References -->
       <xsd:element ref="wsrf-rp:QueryExpressionDialect" minOccurs="0"
maxOccurs="unbounded" />
       <xsd:element ref="muws1:ResourceId"/>
       <xsd:element ref="muws1:ManageabilityCapability" minOccurs="0"
maxOccurs="unbounded" />
       <xsd:element ref="muws2:OperationalStatus"/>

       <!-- WSN Resource Property References -->
       <xsd:element ref="wsnt:FixedTopicSet"/>
       <xsd:element ref="wst:TopicSet" minOccurs="0"/>
       <xsd:element ref="wsnt:TopicExpression" minOccurs="0" maxOccurs="unbounded"/>
       <xsd:element ref="wsnt:TopicExpressionDialect" minOccurs="0"
maxOccurs="unbounded"/>

       <!-- Resource-Specific Resource Property References -->
       <xsd:element ref="{device-type prefix}:{property name}"/>
       {... additional element references ...}
       </xsd:sequence>
     </xsd:complexType>
   </xsd:element>
  </xsd:schema>
```

3. RMD Template.

```
<?xml version="1.0"?>
<Definitions xmlns="http://docs.oasis-open.org/wsrf/rmd-1" >

  <!-- WSDM Base Schema References -->
  <MetadataDescriptor xmlns:wsrl="http://docs.oasis-open.org/wsrf/rl-2"
  xmlns:muws1="http://docs.oasis-open.org/wsdm/muws1-2.xsd"
  xmlns:muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd"
  xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
  xmlns:wst="http://docs.oasis-open.org/wsn/t-1"
  xmlns:myns="{domain namespace}/{device name}"

  <!-- Device-type schema references -->
  xmlns:{device-type prefix}="{domain namespace}/{device-type name}"
  {...}
  name="{device name}MetadataDescriptor"
  interface="myns:{device name}PortType"
  wsdlLocation="{domain namespace}/{device name} {WSDL location}" >

  <!-- Device-type Resource Property Metadata Declarations -->
  <Property name="{device-type prefix}:{property name}" mutability="{value}"
modifiability="read-only">
    <ValidValues>
      <{device-type prefix}:{property name}>{Property Value}</{device-type
prefix}:{property name}>
    </ValidValues>
    <muws2:Capability>{domain namespace}/{device-type name}Capability</muws2:Capability>
  </Property>

  {... additional property metadata declarations ...}

  <!-- Topic Metadata Declarations -->
  <Property name="wsnt:TopicExpression" mutability="constant" modifiability="read-only">
    <StaticValues>
      <wsnt:TopicExpression>{topic prefix}:{topic name}</wsnt:TopicExpression>
    </StaticValues>
  </Property>

  {... additional topic metadata declarations ... }

</MetadataDescriptor>
</Definitions>
```

4. TopicSpace Template

```
<wst:definitions  targetNamespace="{domain namespace}/{device name}TopicSpace"
     xmlns:tns="{domain namespace}/{device name}TopicSpace"
     xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
     xmlns:wst="http://docs.oasis-open.org/wsn/t-1">

  <wst:TopicSpace name="{device name}Topics"
   targetNamespace="{domain namespace}/{device name}TopicSpace">
   <wst:Topic name="{topic name}"
   messageTypes="wsrf-rp:ResourcePropertyValueChangeNotification" />
   {... additional topics ...}
  </wst:TopicSpace>
</wst:definitions>
```

5. Device-Type Schema Template

Note: A separate device-type schema document is required for each device-type that is referenced by the RPSD.

```
<xs:schema targetNamespace="{domain namespace}/{device name}"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name="{property name}" type="xs:{property data-type}"/>
 {...}

</xs:schema>
```

Appendix C

CSB Device WSDM MR Documents

1. LaserPrinter WSDL Document.

Note: In order to conserve space, we have included only the resource-specific schema references here. The entire document can be reconstructed using the template in Appendix B.

LaserPrinterDevice.wsdl:

```
<wsdl:definitions targetNamespace="{domain namespace}/{device name}"
xmlns:tns="{domain namespace}/{device name}"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdl-soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
xmlns:wsrmd="http://docs.oasis-open.org/wsrf/rmd-1"
xmlns:muws1="http://docs.oasis-open.org/wsdm/muws1-2.xsd"
xmlns:muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd"
xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
xmlns:wsntw="http://docs.oasis-open.org/wsn/bw-2"
xmlns:wst="http://docs.oasis-open.org/wsn/t-1"
name="{device name}">

  <wsdl:types>
    {...} (refer to WSDL template)

<!-- Include Resource-Specific Schemas -->

    <xsd:schema elementFormDefault="qualified"
targetNamespace="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDeviceTopic
Space">
      <xsd:include schemaLocation="LaserPrinterDevice_TopicSpace.xsd" />
    </xsd:schema>
    <xsd:schema elementFormDefault="qualified"
targetNamespace="http://www.cs.queensu.ca/ComputerScienceBuilding/PervasiveDevice">
      <xsd:include schemaLocation="PervasiveDevice.xsd" />
    </xsd:schema>
    <xsd:schema elementFormDefault="qualified"
targetNamespace="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinter">
      <xsd:include schemaLocation="LaserPrinter.xsd" />
    </xsd:schema>
    <xsd:schema elementFormDefault="qualified"
targetNamespace="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDevice">
      <xsd:include schemaLocation="LaserPrinterDevice_RPSD.xsd" />
    </xsd:schema>
  </wsdl:types>

{...}
</wsdl:definitions>
```

2. LaserPrinter Schema Documents.

Note : The LaserPrinter device has a total of three schema documents as shown.

PervasiveDevice.xsd :

```
<schema
targetNamespace="http://www.cs.queensu.ca/ComputerScienceBuilding/PervasiveDevice"
xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="UniqueName" type="string"/>
  <element name="DeviceType" type="string"/>
  <element name="Owner" type="string"/>
  <element name="Location" type="string"/>
  <element name="HasDisplayScreen" type="boolean"/>
</schema>
```

LaserPrinter.xsd:

```
<schema targetNamespace="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinter"
xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="CanPrintInColour" type="xs:boolean"/>
  <element name="IsOnline" type="xs:boolean"/>
  <element name="PaperRemaining" type="xs:int"/>
</schema>
```

LaserPrinterDevice_RPSD.xsd :

```
<xsd:schema
targetNamespace="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDevice"
  xmlns:tns="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDevice"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
  xmlns:muws1="http://docs.oasis-open.org/wsdm/muws1-2.xsd"
  xmlns:muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd"
  xmlns:context="http://www.cs.queensu.ca/ComputerScienceBuilding/PervasiveDevice"
  xmlns:laserprinter="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinter"

  xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
  xmlns:wsntw="http://docs.oasis-open.org/wsn/bw-2"
  xmlns:wst="http://docs.oasis-open.org/wsn/t-1">

  <!-- Import schemas for PervasiveDevice and LaserPrinter device-types -->

  <!-- Resource Properties Document Element Declarations -->
 <xsd:element name="LaserPrinterDeviceResourceProperties">
    <xsd:complexType>
      <xsd:sequence>
        <!-- Must be specified for all MRs -->
        <xsd:element ref="wsrf-rp:QueryExpressionDialect" minOccurs="0"
maxOccurs="unbounded" />
        <xsd:element ref="muws1:ResourceId"/>
        <xsd:element ref="muws1:ManageabilityCapability" minOccurs="0"
maxOccurs="unbounded" />
        <xsd:element ref="muws2:OperationalStatus"/>

        <!-- Must be specified if topics are supported -->
        <xsd:element ref="wsnt:FixedTopicSet"/>
        <xsd:element ref="wst:TopicSet" minOccurs="0"/>
        <xsd:element ref="wsnt:TopicExpression" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="wsnt:TopicExpressionDialect" minOccurs="0"
maxOccurs="unbounded"/>

        <!-- PervasiveDevice Capability Properties -->
        <xsd:element ref="context:UniqueName"/>
        <xsd:element ref="context:DeviceType"/>
        <xsd:element ref="context:Owner"/>
        <xsd:element ref="context:Location"/>
        <xsd:element ref="context:HasDisplayScreen"/>

        <!-- LaserPrinter Capability Properties -->
        <xsd:element ref="laserprinter:CanPrintInColour"/>
        <xsd:element ref="laserprinter:IsOnline"/>
        <xsd:element ref="laserprinter:PaperRemaining"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

3. LaserPrinter RMD Document.

```
<Definitions xmlns="http://docs.oasis-open.org/wsrf/rmd-1">
  <MetadataDescriptor xmlns:wsrl="http://docs.oasis-open.org/wsrf/rl-2"
  xmlns:muws1="http://docs.oasis-open.org/wsdm/muws1-2.xsd"
  xmlns:muws2="http://docs.oasis-open.org/wsdm/muws2-2.xsd"
  xmlns:wsnt="http://docs.oasis-open.org/wsn/b-2"
  xmlns:wst="http://docs.oasis-open.org/wsn/t-1"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"

  xmlns:myns="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrin
terDevice"

  xmlns:pervasive="http://www.cs.queensu.ca/ComputerScienceBuilding/Perv
asiveDevice"

  xmlns:laserprinter="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinter"

  name="LaserPrinterMetadataDescriptor"
interface="myns:LaserPrinterPortType"

  wsdlLocation="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDevice
http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDevice/LaserPrinterDevice.ws
dl" >

  <!-- PervasiveDevice Capability Resource Property Metadata Declarations -->
    <Property name="pervasive:UniqueName" mutability="mutable" modifiability="read-only">
      <muws2:Capability>
        http://www.cs.queensu.ca/ComputerScienceBuilding/
        PervasiveDeviceCapability
      </muws2:Capability>
    </Property>

    <Property name="pervasive:DeviceType" mutability="constant"
  modifiability="read-only">
      <ValidValues>
        <pervasive:DeviceType>Data_Projector</pervasive:DeviceType>
        <pervasive:DeviceType>Laser_Printer</pervasive:DeviceType>
        <pervasive:DeviceType>Smart_Thermostat</pervasive:DeviceType>
        <pervasive:DeviceType>PDA</pervasive:DeviceType>
      </ValidValues>

      <muws2:Capability>
        http://www.cs.queensu.ca/ComputerScienceBuilding/
        PervasiveDeviceCapability
      </muws2:Capability>
    </Property>

    <Property name="pervasive:Owner" mutability="mutable"
      modifiability="read-only">
      <muws2:Capability>
        http://www.cs.queensu.ca/ComputerScienceBuilding/
        PervasiveDeviceCapability
      </muws2:Capability>
    </Property>

    <Property name="pervasive:Location" mutability="mutable"
```

```
   modifiability="read-only">
     <ValidValues>
       <pervasive:Location>Boardroom</pervasive:Location>
       <pervasive:Location>Conference_Room</pervasive:Location>
     </ValidValues>
     <muws2:Capability>
       http://www.cs.queensu.ca/ComputerScienceBuilding/
       PervasiveDeviceCapability
     </muws2:Capability>
   </Property>

   <Property name="pervasive:HasDisplayScreen" mutability="constant"
modifiability="read-only">
     <muws2:Capability>
       http://www.cs.queensu.ca/ComputerScienceBuilding/
       PervasiveDeviceCapability
     </muws2:Capability>
   </Property>

   <!-- LaserPrinter Capability Resource Property Metadata Declarations -->

   <Property name="laserprinter:CanPrintInColour" mutability="mutable"
   modifiability="read-write">
     <muws2:Capability>
       http://www.cs.queensu.ca/ComputerScienceBuilding/
       LaserPrinterCapability
     </muws2:Capability>
   </Property>

   <Property name="laserprinter:IsOnline" mutability="mutable"
   modifiability="read-only">
     <muws2:Capability>
       http://www.cs.queensu.ca/ComputerScienceBuilding/
       LaserPrinterCapability
     </muws2:Capability>
   </Property>

   <Property name="laserprinter:PaperRemaining" mutability="mutable"
   modifiability="read-write">
     <muws2:Capability>
       http://www.cs.queensu.ca/ComputerScienceBuilding/
       LaserPrinterCapability
     </muws2:Capability>
   </Property>

   <Property name="wsnt:TopicExpression" modifiability="read-only"
   mutability="constant">
     <StaticValues>
       <wsnt:TopicExpression>pervasive:Location</wsnt:TopicExpression>
     </StaticValues>
   </Property>
 </MetadataDescriptor>
</Definitions>
```

4. LaserPrinter TopicSpace Document.

```
<wst:definitions
targetNamespace="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDeviceTopic
Space"
 xmlns:tns="http://www.cs.queensu.ca/ComputerScienceBuilding/LaserPrinterDeviceTopicSpace
"
 xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
 xmlns:wst="http://docs.oasis-open.org/wsn/t-1">
 <wst:TopicSpace name="PervasiveDeviceTopics"
targetNamespace="http://www.cs.queensu.ca/ComputerScienceBuilding/PervasiveDeviceTopicSpa
ce">
   <wst:Topic name="Location" messageTypes="wsrf-
rp:ResourcePropertyValueChangeNotification" />
 </wst:TopicSpace>
</wst:definitions>
```

5. Context Proxy Service Operations WSDL Document Fragment.

```
<wsdl:definitions
targetNamespace="http://www.cs.queensu.ca/SchoolOfComputing/{DeviceName}"
  xmlns:tns="http://www.cs.queensu.ca/SchoolOfComputing/{DeviceName}"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdl-soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:proxy="http://www.cs.queensu.ca/SchoolOfComputing/ProxyProvider"
  {...}
  name="{DeviceName}">

  {...}

  <wsdl:types>
    {...}
    <!-- Import Proxy Provider Schema -->
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://www.cs.queensu.ca/SchoolOfComputing/ProxyProvider">
      <xsd:include schemaLocation="ProxyProvider.xsd" />
    </xsd:schema>
  </wsdl:types>

  <!-- Proxy Provider Message definitions -->

  <wsdl:message name="SendProfileRequest">
    <wsdl:part name="SendProfileRequest" element="proxy:SendProfileRequest" />
  </wsdl:message>
  <wsdl:message name="SendProfileResponse">
    <wsdl:part name="SendProfileResponse" element="proxy:SendProfileResponse" />
  </wsdl:message>
  <wsdl:message name="SendProfileFault">
    <wsdl:part name="SendProfileFault" element="proxy:SendProfileFault" />
  </wsdl:message>

  <wsdl:message name="RenewLeaseRequest">
    <wsdl:part name="RenewLeaseRequest" element="proxy:RenewLeaseRequest" />
  </wsdl:message>
  <wsdl:message name="RenewLeaseResponse">
    <wsdl:part name="RenewLeaseResponse" element="proxy:RenewLeaseResponse" />
  </wsdl:message>
  <wsdl:message name="RenewLeaseFault">
    <wsdl:part name="RenewLeaseFault" element="proxy:RenewLeaseFault" />
  </wsdl:message>

  <wsdl:message name="DeleteProfileRequest">
    <wsdl:part name="DeleteProfileRequest" element="proxy:DeleteProfileRequest" />
  </wsdl:message>
  <wsdl:message name="DeleteProfileResponse">
    <wsdl:part name="DeleteProfileResponse" element="proxy:DeleteProfileResponse" />
  </wsdl:message>
  <wsdl:message name="DeleteProfileFault">
    <wsdl:part name="DeleteProfileFault" element="proxy:DeleteProfileFault" />
  </wsdl:message>

  <!-- Proxy Provider Operation definitions -->

  <wsdl:portType
    name="{DeviceName}ResourcePortType"
    {...}>

    <!-- ProxyProvider Operations -->

    <wsdl:operation name="SendProfile">
      <wsdl:input
wsa:Action="http://www.cs.queensu.ca/SchoolOfComputing/SendProfileRequest"
        name="SendProfileRequest"
        message="tns:SendProfileRequest"/>
      <wsdl:output
wsa:Action="http://www.cs.queensu.ca/SchoolOfComputing/SendProfileResponse"
        name="SendProfileResponse"
      message="tns:SendProfileResponse"/>
     <wsdl:fault   name="SendProfileFault"
      message="tns:SendProfileFault"/>
   </wsdl:operation>

   <wsdl:operation name="RenewLease">
     <wsdl:input
wsa:Action="http://www.cs.queensu.ca/SchoolOfComputing/RenewLeaseRequest"
      name="RenewLeaseRequest"
      message="tns:RenewLeaseRequest"/>
     <wsdl:output
wsa:Action="http://www.cs.queensu.ca/SchoolOfComputing/RenewLeaseResponse"
      name="RenewLeaseResponse"
      message="tns:RenewLeaseResponse"/>
     <wsdl:fault   name="RenewLeaseFault"
      message="tns:RenewLeaseFault"/>
   </wsdl:operation>

   <wsdl:operation name="DeleteProfile">
     <wsdl:input
wsa:Action="http://www.cs.queensu.ca/SchoolOfComputing/DeleteProfileRequest"
      name="DeleteProfileRequest"
      message="tns:DeleteProfileRequest"/>
     <wsdl:output
wsa:Action="http://www.cs.queensu.ca/SchoolOfComputing/DeleteProfileResponse"
      name="DeleteProfileResponse"
      message="tns:DeleteProfileResponse"/>
     <wsdl:fault   name="DeleteProfileFault"
      message="tns:DeleteProfileFault"/>
   </wsdl:operation>

  {...}
  </wsdl:portType>

  <!-- Proxy Provider Bindings -->

  <wsdl:binding name="{DeviceName}ResourceBinding"
   type="tns:{DeviceName}ResourcePortType">
   <wsdl-soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />

   <wsdl:operation name="SendProfile">
     <wsdl-soap:operation soapAction="SendProfile" />
     <wsdl:input name="SendProfileRequest">
      <wsdl-soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
       </wsdl:input>
       <wsdl:output name="SendProfileResponse">
      <wsdl-soap:body use="encoded"
encodingStle="http://schemas.xmlsoap.org/soap/encoding/" />
       </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="RenewLease">
     <wsdl-soap:operation soapAction="RenewLease" />
     <wsdl:input name="RenewLeaseRequest">
      <wsdl-soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
       </wsdl:input>
       <wsdl:output name="RenewLeaseResponse">
      <wsdl-soap:body use="encoded"
encodingStle="http://schemas.xmlsoap.org/soap/encoding/" />
       </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="DeleteProfile">
     <wsdl-soap:operation soapAction="DeleteProfile" />
     <wsdl:input name="DeleteProfileRequest">
      <wsdl-soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
       </wsdl:input>
       <wsdl:output name="DeleteProfileResponse">
      <wsdl-soap:body use="encoded"
encodingStle="http://schemas.xmlsoap.org/soap/encoding/" />
       </wsdl:output>
   </wsdl:operation>

   {...}
  </wsdl:binding>
  {...}
</wsdl:definitions>
```

6. Context Proxy Service Operations XML Schema Document.

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:proxy="http://www.cs.queensu.ca/SchoolOfComputing/ProxyProvider"
  xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  targetNamespace="http://www.cs.queensu.ca/SchoolOfComputing/ProxyProvider">

  <xsd:element name="SendProfileRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
     </xsd:element>

  <xsd:element name="SendProfileResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="SendProfileFaultType">
    <xsd:complexContent>
      <xsd:extension base="wsrf-bf:BaseFaultType"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="SendProfileFault" type="proxy:SendProfileFaultType"/>

  <xsd:element name="RenewLeaseRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
     </xsd:element>

  <xsd:element name="RenewLeaseResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="RenewLeaseFaultType">
    <xsd:complexContent>
      <xsd:extension base="wsrf-bf:BaseFaultType"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="RenewLeaseFault" type="proxy:RenewLeaseFaultType"/>
  <xsd:element name="DeleteProfileRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
     </xsd:element>

  <xsd:element name="DeleteProfileResponse">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="DeleteProfileFaultType">
    <xsd:complexContent>
      <xsd:extension base="wsrf-bf:BaseFaultType"/>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="DeleteProfileFault" type="proxy:DeleteProfileFaultType"/>

</xsd:schema>
```