Software Architecture
for Pervasive Computing

Sifei HE

Submitted for Master of Philosophy
from the
University of Surrey

UniS

Department of Computing
School of Electronics and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, UK

July 2006

© Sifei HE 2006
Summary

This dissertation studies software architecture for a Pervasive Computing environment. The concepts and characteristics of Pervasive Computing are described followed by a review of the software architectural design methodologies. Problems of component-based system modelling in Pervasive Computing are identified. A review of current research work in building software systems for Pervasive Computing is then conducted. The work focuses on defining a service oriented software architecture for applications development in a Pervasive Computing environment. The architecture emphasizes strong interoperability and semantic relationships between entities in a Pervasive Computing environment by considering a platform and protocol independent service description with added semantic annotations. A prototype has been implemented based on the DBE framework. On top of the prototype, two example services and a context aware application was built to demonstrate the system capability. Finally, conclusions and future work are addressed.

Key words: Pervasive Computing, Software Architecture, Service-Oriented Architecture, Ontology, Context-aware.
Acknowledgments

I would like to thank my supervisor, Prof Paul Krause for proposing this novel research area. His patience, encouragement and guidance have always been invaluable asset to my research. I would also like to thank my co-supervisor, Dr Roger Peel for his advice and comments on various areas of my research.

I gratefully acknowledge the funding support from Department of Computing, University of Surrey.

Finally, I would like to take this opportunity to thank my parents for their support and encouragement.
Contents

1 Introduction .......................................................................................................................... 1
  1.1 Introduction ................................................................................................................. 1
  1.2 Objectives .................................................................................................................. 5
  1.3 Structure of this report .............................................................................................. 5

2 Literature Review ................................................................................................................ 6
  2.1 Pervasive Computing .................................................................................................. 7
    2.1.1 The Environment ................................................................................................. 7
    2.1.2 Human-Centric ..................................................................................................... 9
    2.1.3 Heterogeneity ....................................................................................................... 9
    2.1.4 Context awareness ............................................................................................... 10
    2.1.5 Mobility ............................................................................................................... 11
    2.1.6 Adaptability ........................................................................................................ 12
    2.1.7 Privacy and Security .......................................................................................... 13
    2.1.8 Software Challenges in Pervasive Computing ................................................... 13
  2.2 Software Architecture .............................................................................................. 14
    2.2.1 What is Software Architecture ........................................................................... 14
    2.2.2 Component-based Architecture ......................................................................... 16
    2.2.3 Service Oriented Architecture ........................................................................... 18
    2.2.4 Web Service Architecture .................................................................................. 24
  2.3 Ontology modelling ................................................................................................. 31
  2.4 Combining SOA with Pervasive Computing ............................................................. 33
  2.5 Summary .................................................................................................................. 36

3 Related Work .................................................................................................................... 38
  3.1 Oxygen ..................................................................................................................... 38
  3.2 Aura .......................................................................................................................... 39
  3.3 One.World ................................................................................................................ 42
  3.4 GAIA ....................................................................................................................... 44
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5 Task Computing</td>
<td>47</td>
</tr>
<tr>
<td>3.6 Summary</td>
<td>50</td>
</tr>
<tr>
<td>4 Proposed service oriented architecture</td>
<td>53</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>53</td>
</tr>
<tr>
<td>4.2 A Layered Structure</td>
<td>56</td>
</tr>
<tr>
<td>4.3 System Architecture</td>
<td>57</td>
</tr>
<tr>
<td>4.4 Users, Tasks and Applications</td>
<td>58</td>
</tr>
<tr>
<td>4.5 Devices</td>
<td>59</td>
</tr>
<tr>
<td>4.6 Services</td>
<td>60</td>
</tr>
<tr>
<td>4.6.1 Service Description</td>
<td>61</td>
</tr>
<tr>
<td>4.6.2 Service Publication</td>
<td>62</td>
</tr>
<tr>
<td>4.6.3 Service Invocation</td>
<td>63</td>
</tr>
<tr>
<td>4.6.4 Service Update</td>
<td>63</td>
</tr>
<tr>
<td>4.6.5 Service Deletion</td>
<td>63</td>
</tr>
<tr>
<td>4.7 Services Discovery</td>
<td>64</td>
</tr>
<tr>
<td>4.8 Context</td>
<td>66</td>
</tr>
<tr>
<td>4.9 Repository</td>
<td>67</td>
</tr>
<tr>
<td>4.9.1 Service Repository</td>
<td>67</td>
</tr>
<tr>
<td>4.9.2 Personal Repository</td>
<td>69</td>
</tr>
<tr>
<td>4.10 Common Ontology</td>
<td>70</td>
</tr>
<tr>
<td>4.11 Summary</td>
<td>71</td>
</tr>
<tr>
<td>5 Implementation</td>
<td>73</td>
</tr>
<tr>
<td>5.1 The Service and Models</td>
<td>75</td>
</tr>
<tr>
<td>5.1.1 The Business Model</td>
<td>76</td>
</tr>
<tr>
<td>5.1.2 The Technical Model</td>
<td>77</td>
</tr>
<tr>
<td>5.1.3 The Ontology Definition Model</td>
<td>78</td>
</tr>
<tr>
<td>5.1.4 The Service Manifest Model</td>
<td>79</td>
</tr>
<tr>
<td>5.2 The Core Service</td>
<td>80</td>
</tr>
<tr>
<td>5.3 The Execution Environment</td>
<td>83</td>
</tr>
<tr>
<td>5.4 The Service Factory</td>
<td>87</td>
</tr>
<tr>
<td>5.4.1 BML Editor</td>
<td>88</td>
</tr>
</tbody>
</table>
List of Figures

Figure 2-1 An interface is a mediator between two components ........................................16
Figure 2-2 Software architecture describes a system's components and connectors. ...... 17
Figure 2-3 Technologies for Service Oriented Architecture ........................................ 19
Figure 2-4 The Service Oriented Architecture ................................................................20
Figure 2-5 Service Oriented interaction pattern ................................................................23
Figure 2-6 Communication Protocol in SOA ..................................................................24
Figure 2-7 WSDL document components .......................................................................25
Figure 2-8 WSDL Types example ....................................................................................26
Figure 2-9 WSDL abstract definitions .............................................................................27
Figure 2-10 WSDL concrete definitions ..........................................................................27
Figure 2-11 The SOAP message structure .......................................................................28
Figure 2-12 SOAP message over HTTP .........................................................................29
Figure 2-13 UDDI data structures and their relationships to the traditional phonebook...30
Figure 2-14 SOAP message to find a business in a UDDI registry ..................................30
Figure 3-1 An H2Is device prototype ...............................................................................38
Figure 3-2 Aura's Architecture .......................................................................................40
Figure 3-3 Aura's Contextual Information Service architecture ......................................40
Figure 3-4 Overview of One.World's architecture .............................................................42
Figure 3-5 Physical and Active spaces in GAIA ...............................................................45
Figure 3-6 Gaia Architecture .........................................................................................45
Figure 3-7 Gaia context infrastructure ............................................................................46
Figure 3-8 General architecture of Task Computing Environment ...................................48
Figure 3-9 Screenshot of STEER (A Task Computing Client) .........................................49
Figure 4-1 A layered view of the proposed architecture ..................................................56
Figure 4-2 Proposed system architecture for Pervasive Computing ..................................58
Figure 4-3 Relationships between service concepts .......................................................60
Figure 4-4 An example of the proposed service description .............................................62
Figure 4-5 Structure of the Service Repository ...............................................................68
Figure 4-6 Structure of the Personal Repository .............................................................69
Figure 4-7 A visualised SOUPA#person ontology ..........................................................71
Figure 5-1 Service Modelling Architecture, a MDA-based approach .................................. 75
Figure 5-2 BML modelling .................................................................................................. 76
Figure 5-3 SDL modelling .................................................................................................. 77
Figure 5-4 ODM modelling .................................................................................................. 78
Figure 5-5 Service Manifest ................................................................................................. 79
Figure 5-6 CoreAdapter code snippets ................................................................................. 80
Figure 5-7 Knowledge Base service interface code snippets .............................................. 80
Figure 5-8 KB Service ........................................................................................................ 81
Figure 5-9 SR Service ........................................................................................................ 82
Figure 5-10 The Execution Environment ........................................................................... 83
Figure 5-11 Servent Interface code snippets ........................................................................ 84
Figure 5-12 Create a Protocol Adapter code snippets ......................................................... 85
Figure 5-13 FADA Web-based Administration ..................................................................... 86
Figure 5-14 DBE Studio Components ................................................................................ 88
Figure 5-15 BML Editor ..................................................................................................... 89
Figure 5-16 ODM Editor ..................................................................................................... 90
Figure 5-17 SDL Editor ..................................................................................................... 91
Figure 5-18 The BML Data Editor ..................................................................................... 92
Figure 5-19 SDL2JAVA Compiler ..................................................................................... 93
Figure 5-20 SSL2SDL Compiler ....................................................................................... 93
Figure 5-21 The Service Manifest Creator ......................................................................... 94
Figure 5-22 The Service Exporter ...................................................................................... 95
Figure 6-1 The DBE Studio work flow ............................................................................... 97
Figure 6-2 The Context Service Model .............................................................................. 98
Figure 6-3 The PrintingService Model .............................................................................. 99
Figure 6-4 PrintingService Ontology ................................................................................. 100
Figure 6-5 Obtain self location code snippets .................................................................... 101
Glossary of Terms

ADL: Architecture description language.

BML: Business Modelling Language.

CBSE: Component Based Software Engineering.

COM: Component Object Model.

CORBA: Common Object Request Broker Architecture.

DBE: Digital Business Ecosystem.

ExE: Execution Environment.

EvE: Evolutionary Environment.


IDL: Interface Description Language.


JVM: Java Virtual Machine.

MDA: Model Driven Architecture.

MEP: Message Exchange Pattern.

MOF: Meta Object Facility.

ODM: Ontology Definition Model.

ORB: Object Request Broker.

OWL: Web Ontology Language.

OWL-S: OWL-based Web Service Ontology.
POJO: Plain Old Java Object.

SDL: Service Description Language.

SF: Service Factory.

SM: Service Manifest.

SMB: Server Message Block.

SMID: Service Manifest ID.

SOA: Service Oriented Architecture.


SSDP: Simple Service Discovery Protocol.

SSL: Semantic Service Language.

UDDI: Universal Description, Discovery and Integration.

UPnP: Universal Plug and Play.

WSDL: Web Service Definition Language.

XDB: XML Database.

XMI: XML Metadata Interchange.

XML: Extensible Markup Language.

XSL: Stylesheet Language for XML.

XSTL: XSL Transformations.
Chapter 1

1 Introduction

1.1 Introduction

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." Pioneered by [62], Pervasive Computing has the potential to dramatically change our future lives. Also called ubiquitous computing\(^1\), the general idea is to seamlessly assist users in completing their tasks by integrating indistinguishable devices that coordinate themselves non-intrusively throughout our working and living environment.

Classic technologies such as paper and electronic motors have already been pervasively used so that people sensuously do not feel the existence of technologies behind them. Pervasive Computing aims to enhance such experience by enabling people to shift their focus from the computing technologies to the tasks themselves. For example, Bob would like to print some presentation slides while he is at a conference. Instead of identifying the appropriate printer at a nearby location, moving files between devices and handling the data format, Bob simply instructs the software to "print the slides".

Driven by Moore's Law, it is reasonable to believe that the variety of devices will further increase over the next several years. Apart from conventional PDAs, mobile phones, desktops and servers, the Pervasive Computing environment also consists of sensors, actuators, appliances with embedded processor etc. Essentially anything in this environment is a kind of "device" regardless its physical form.

Context plays a vital role in the domain of Pervasive Computing. Although many definitions co-exist, the On-line Dictionary of Computing [18] defines context as "that

\(^1\) For the purpose of this report, we treat terms Pervasive Computing and ubiquitous computing synonymously.
which surrounds, and gives meaning to something else". One of the fundamental differences between traditional applications and Pervasive Computing applications is the requirement for Pervasive Computing applications to gather and understand their contexts in order to assist their users. Considering location as a kind of context, a smart home application is able to switch on house lights when the presence of occupants is detected. However, the occupant's location can be retrieved from a wide range of sources such as a face/voice recogniser, a fingerprint device, RFIDs, a personal scheduler, a computer login event or some high level contexts which are inferred from previous locations. Given such a diversity of devices, it is infeasible to assume that all context sources can be known in advance. Therefore, acquiring contextual information without adequate system support becomes impossible. Researchers have been working on how to provide contextual information. For example: Glenn and Peter [32] proposed a single service interface that provides applications with contextual information in a unified manner.

In the scope of this report, we define Pervasive Computing as environments encompassed by an enormous number of devices which are connected using wired or wireless connection to provide intelligent services for consumption by other parties. Dissimilar technologies behind devices are hidden to consumers by only exporting functionalities through their concrete software services.

Software development for Pervasive Computing becomes highly difficult not only because of the variety of integrated devices but also because of the lack of information about the runtime environment due to the mobility requirements imposed. Also, contexts and resources may change from time to time. As a consequence, it is not always clear at design time which particular resource/device is to be used to complete a task.

Furthermore, indistinguishable devices are usually resource limited by their physical form factor. The software infrastructure must be able to bridge the gap between application requirements and the capabilities of devices. For example, a thermal sensor only generates discrete levels of electric signals; it is the concrete temperature reading service that provides a temperature reading through low level signal-temperature conversion.

Ever since the software crisis, software engineers have been focused on designing complex software systems from independent, reusable collections of functional entities. These entities may already be available from a range of sources such as previous
implementations or a third party. While other entities may have to be created on the fly, the system must be capable of assembling all entities in a normative mechanism. One example is the well-known concept of Component-Based Software Engineering (CBSE) [14]. By separating the system into reusable software components, a new application can be built on top of many existing components. On the other hand, components can be easily replaced and customised to provide better functionality whilst the application remains unaltered. Today, the concept of CBSE has been widely realised by using technological approaches such as Common Object Request Broker Architecture (CORBA) [69], Microsoft Component Object Model (COM) technology [16], the .NET platform [24] as well as the Java 2 Enterprise Edition (J2EE) [39].

As previously mentioned, devices in a Pervasive Computing environment are likely to be heterogeneous, requiring special considerations in terms of their interoperability. An additional consideration is that software in the Pervasive Computing environment will be running distributively across many environments to achieve its objective. An architectural solution must coordinate the access to different devices in a unified methodology. One way to design such architecture is to consider the usage of software services.

With the introduction of Web Services over the last few years, there has been increased interest in Service Oriented Architecture (SOA). A service has been defined as a unit of work done by a service provider to achieve desired end results for the service consumer [37]. An SOA is essentially a collection of services and interactions between these services. The system benefits from natural evolution through the addition of new services. This offers some great advantages to the Pervasive Computing domain. First, it allows services to be implemented independently. As the number of devices in a Pervasive Computing environment grows, many of them will provide similar or even identical functionality whereas the actual implementations can be fundamentally different. By only exposing a well-defined service interface, the desired functionality can be reached without knowledge of the underlying technology. Secondly, it increases the system availability. In a Pervasive Computing Environment, people should be able to move freely from one place to another where computation support is always available. SOA achieve this by allowing services to be consumed in a natural way regardless of their physical locations. For example, an office printer can also offer a printing service at any other remote location where a similar service does not exist, as long as the service is
accessible. Thirdly, it gives better interoperability via so called late binding and loose coupling. In contrast to some of the current distributed systems, applications in a Pervasive Computing environment typically have less information about devices to which they will be up against. SOA overcomes the problem by employing descriptive messages delivered through the service description. A service consumer can therefore dynamically engage a service by interpreting those messages at the runtime. As we can see, the consumer does not depend directly on the service’s implementation but only on the service interface. Along with many other advantages, we believe that service-oriented software architecture is a promising solution for Pervasive Computing.

However, SOA does not restrict the terms and vocabularies to be used in describing a service. Taking account of the heterogeneous pervasive environment, it becomes reality that services from different providers may choose to use different terms to describe the same meaning. For example, a mobile operator in one country may use the term "mobile_phone service" while others may use "cell_phone service" to describe the services they offer. Although we as humans can understand these concepts are equivalent very quickly, machines do not handle it well. This particular weakness in supporting application level interoperability often leads to an ad-hoc approach such as implementation or platform dependent service descriptions which do not scale up in Pervasive Computing. Therefore, a common vocabulary is needed.

Ontology is a formal specification of a conceptualization [94]. It provides a common vocabulary to which both parties involved in an interaction must commit. The use of shared ontologies in SOA guarantees the consistency of service descriptions which is a key element to the ontology-based service discovery mechanism. Additionally, an ontology also specifies the relationships that hold among the concepts in that vocabulary. Reusing the mobile operator example above; if a relationship "same as" is drawn between "mobile_phone service" and "cell_phone service", both services will satisfy a service request which is intended to look up a "mobile_phone service".

In addition to the service description, ontologies can be also used in modelling contextual information [35] [99] and various other pieces of the system. As a conclusion, we propose a service-oriented, ontology based software architecture in attempting to solve some of the problems in Pervasive Computing. The hypothesis is that,
"By abstracting devices and context sources into discoverable services, we can provide
an architecture that supports the dynamic nature of human activity. As a consequence,
it is anticipated that the Service Oriented Architecture is a natural choice for software
development in the Pervasive Computing environment."

Building on top of SOA, we need not repeat our research on enabling technologies such
as OWL[25], WSDL [26], SOAP [30], WS-* standards which industries are trying to
solve. Instead, we can focus on the unique challenges raised by Pervasive Computing
such as: application level interoperability; context modelling; service matching; and,
smart environment design.

1.2 Objectives

The major objectives of this research are as follows:

- To survey the current research directions in Pervasive Computing system design.
- To study the service oriented architecture (SOA) and ontology-based system
  modelling and its integration with SOA.
- To design a generic service oriented software architecture for large-scale usage in
  the Pervasive Computing environment.
- To investigate the applicability, interoperability and scalability of this architecture.
- To study the performance and possible application areas of the architecture.

1.3 Structure of this report

The rest of the report is organised as follows: Chapter 2 presents a literature review on the
Pervasive Computing as well as software architecture. Chapter 3 discusses a number of
related works that attempted to build system software for Pervasive Computing. An initial
definition and analysis of our proposed software architecture is given in Chapter 4. We
introduce the preliminary implementation in Chapter 5 and demonstrate it in use in
Chapter 6. Chapter 7 summarises the current research work and makes some proposals for
future work.
Chapter 2

2 Literature Review

As envisioned by Mark Weiser [62] in 1991, Pervasive Computing has the potential to change our lives. The essential idea is to create an environment saturated with devices and computation capability, yet gracefully integrated with human users to support them in completing their tasks [60].

The advances of mobile technology over the past several years have created many critical elements for Pervasive Computing. Commercial products such as PDAs, mobile phones, the 802.11 wireless network, smart appliances and sensors are spreading across the widest possible market place. Given this trend, it is no doubt that device diversity will be further increased in the future. Moreover, these devices and technologies are often embedded in places, such as living rooms and kitchens, where computing is not normally expected to happen in today's human life. According to [62], technology must vanish into the background and seamlessly provide computing support whenever is needed. Thus, Pervasive Computing shifts human focus away from the technologies to the tasks that we, as humans, want to perform. Distributed systems and mobile computing are two early efforts which tried to create conceptual frameworks and mechanisms to enable computing access over distributed and heterogeneous systems. They provide the fundamental elements in building Pervasive Computing system.

One of the key challenges in Pervasive Computing is to build software applications that provide adequate task support even when people move across physical or Pervasive Computing environments. The implication is that, applications must not assume a fix set of devices with which they will interact. Instead, devices and services embedded in a remote environment are often unknown in advanced and must be discovered before they can be invoked. Therefore, a software architecture for pervasive computing must facilitate dynamic service discovery and interactions at runtime.
In this chapter, the characteristics of Pervasive Computing are studied followed by an overview of software architecture. In particular, Service Oriented Architecture (SOA) is discussed with a brief example of how Web Service architectures embodied the SOA vision. Ontology is also introduced in terms of service and context modelling. Finally, a summary is drawn.

2.1 Pervasive Computing

Early work on the foundation of Pervasive Computing started in 1990s when the Active Badge system [76] was developed at Olivetti Research lab. Since then, many projects have been developed to advance the vision from different perspectives such as location-aware [59] and context aware systems [9]. Recent effort has been focusing on providing a distraction free [22], human-centric [56], active space [49] or task computing [79] environment which in turn serves as the infrastructure for developing software applications for Pervasive Computing.

Pervasive Computing aims to provide seamless computing support whenever and wherever we need. Software applications must face the challenge where heterogeneous devices across several environments are potentially useful to the tasks that a user needs to perform. Another vision of Pervasive Computing is that the technology vanishes. People have already been pervasively using classic products such as car and paper without a deep knowledge of how they are made. Software in Pervasive Computing aims to realise the vision by hiding the complex device technology, and adapt their behaviours according to the contextual information in the current environment.

2.1.1 The Environment

Proposed by [91], Pervasive Computing involves integrations of computing nodes and the physical world. Computing often takes place in some discrete environments that are logical containers for people, devices, services and applications. The boundaries of an environment are largely determined by the physical place such as a restaurant or a car park. Although physical boundaries change less frequently, the logical boundaries of an environment can be dynamically expanded to embrace multiple physical places or be shrunk to a specific sub-place. A boundary should not by any mean constrain the
interoperations between environments. As with the Internet, the creation of a global Pervasive Computing environment is realised by connecting many local environments which may have totally different knowledge about their own domain.

In daily life, people can be associated with a "home" environment and are free to move across any other environments where different tasks are performed. However, the computing support should remain at the same level compared to what it was regardless of the environment a user resides in. Heterogeneous computing technologies in each environment are naturally perceived without annoyance.

Embedded in physical places, devices in an environment are not limited to traditional computing hardware, but may include a lamp, a microwave or sensors. Essentially, anything that provides useful services to the users and other entities within the environment is a kind of device. For example, a thermometer offers a temperature reading service which could be used to control the in house air condition system. To be accessible, devices are inter-connected by the underlying communication network and expose their functionalities over the network.

A service has been defined as a unit of work performed by a service provider to achieve desired end results for the service consumer [37]. The functionality of a service is exposed through a well-defined interface. Given the increase in device variation, it is infeasible to build pervasive applications that directly interact with the required devices. A service thus, is a mediator between applications and devices. We will discuss the concept of a service later in section 2.2.3.

In the Pervasive Computing environment, a task represents a user's intention. For example, a user wants to print a document and the task here is to "print a document". Tasks are often carried out by the appropriate applications which were designed to accomplish the tasks on behalf of users. Therefore, interactions between users and the environment are virtually provided by the applications which should perceive the change and reconfigure themselves effectively while a user is moving across the environment boundaries.
2.1.2 Human-Centric

As argued by Project Oxygen [55], computation has centred about machines, rather than people for forty years. Humans are forced to interact with them on their terms, by manipulating their keyboards or mice. Apart from the effective strategies for HCI design [7], applications must be able to adapt to the current environment so that users are free from discomfort when performing their tasks.

While Weiser's original vision of "technology disappearing" [62] is desirable, practical systems should maintain a balance between proactivity and "user distraction" [60]. For example, incorrect context aware adaptations can annoy a user and thus destroy the user fidelity. Application developers must, therefore, decide the human's role in resolving uncertainty.

Another issue is the way we interact with the environment. Speaking a common language all over the world is not easy. Translators are often employed to present common meanings in different languages to all parties involved in a conversation. The same principle applies to the Pervasive Computing environment where devices and services are developed in different technologies and programming languages. If applications are forced to interact with other entities in a way that they are not designed to, their users will end up with no task support. Software architecture must bridge the gap between application knowledge and knowledge in the target environment.

2.1.3 Heterogeneity

Driven by Moore's Law, it is reasonable to believe that the variety of devices will further increase over the next several years. Devices in Pervasive Computing are no longer limited to a "computer"; they can be anything that is useful. Essentially every physical entity in an environment is a kind of "device" regardless of its physical form. Besides, notable technology improvements such as wireless networking, processing capability, storage capacity and high-quality displays indicates that hardware for many applications are disappearing [78].

The biggest impact of device heterogeneity is that, applications for Pervasive Computing can no longer assume a fixed set of devices with which they will interact. Since
movement is a nature of our daily life, applications must cope with any environment where devices are typically unknown at the time they are developed. Devices from different vendors are likely to use disparate technologies while similar functionalities are provided. It is impossible to develop an application that is capable of interacting with every type of device.

Apart from heterogeneous devices, services in the Pervasive Computing environment are also diverging from each other. For example, a fax machine may offer a printing service that has some distinct attributes comparing with a laser printer in the same environment. Therefore, service discovery must be able to reflect this polymorphism by returning all service instances of type "Printing" as a result for a query of "Printing service" [72].

Given the technology trend, it is infeasible to assume a single platform that will accommodate all devices and services. Software architecture for Pervasive Computing must be flexible enough so that, interoperations between applications, services and devices are not restricted to a single programming language, communication protocol or system platform.

### 2.1.4 Context awareness

The term context has been defined in many different ways. If we refer to "context" in general knowledge, it is the general conditions or circumstances in which an event or action occurs [98]. The meaning in general gives the basic idea of what context is, but it lacks clarity if we refer to Pervasive Computing. By context, we refer to "any information that can be used to characterise the situation of an entity, where an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves." [6].

According to [6], context awareness is "the use of context to provide task-relevant information and/or services to a user, where relevancy depends on the user's task". Pervasive Computing aims to provide seamless task support whenever it is needed. Efficient use of context information thus enables software applications to better assist their users.

As an example, early work on an Active Badge location system has shown the technology
for locating people and equipment indoors. It uses location as the main context to find
where the user is and what the surrounding contexts are. It also uses the context
information: who they are with, where they are and time of the day to provide location-
based contextual information to a user [76]. Another similar work on location information
is ParcTab, a context sensitive application that helps users to find available local
resources. [77].

In Pervasive Computing where applications may run in different environments, the kind
of device to provide a particular type of context is not restricted. To use a certain context,
applications must be able to communicate with various context sources. While direct
context acquisition from low level devices is convenient in some cases, it does not scale
well across different domains. For example, a person's presence can be detected by a
motion sensor, or a finger print device on wall. Applications that use motion sensors will
not be able to extract the presence of a person from the finger print device simply because
they involve totally different hardware. Therefore, a software architecture must facilitate
context provisioning through a common way. Another concern is the format that is used
to represent a context. Context information obtained from different sources can be
expressed in different formats. For example, a person's location can be expressed in terms
of longitude, latitude and altitude; or his/her relationship to another entity such as in a
room, or near another person. For most of the cases, it is up to the application to interpret
the acquired context in whatever the way that is meaningful. However, it is useful to have
some formal definitions about the data contained in a context in order to gain better
understanding.

2.1.5 Mobility

Computing is no longer limited to a single place in Pervasive Computing. Recognising
dynamism as a systematic nature, users in Pervasive Computing may have to work from
one place to another in an ad hoc basis. Upon arriving, applications are completely blind
to the target environment. Meanwhile, mobile devices and services can be dynamically
added and removed from an environment without explicit notification. A software
architecture for such environments should be designed on the assumption that users,
applications, devices and services are highly dynamic and unpredictable [91].
In Pervasive Computing, the user data may span over several environments along with the application. On resuming an application, persistent data must be restored and made accessible. However, the unpredictable Pervasive Computing environment provides no guarantee that the storage device/service can be obtained. Context File Systems (CFS) [13] introduced the concept of context-based mount points and assumed that the virtual folder contains only the files relevant in the current situation. In many cases, this assumption can not be made when historic data is valuable to the current executing task. Therefore, providing a consistent file system for mobile applications becomes an essential element in architectural design for Pervasive Computing.

2.1.6 Adaptability

The mobility constraints introduced in the previous section indicates that availability of devices, service and context in an environment will consistently change as they move in and out of that environment. Applications should adapt to change when the involved entities are part of the execution. Three adaptation strategies have been proposed in [60]. First, user guided adaptation. Second, reservation-based adaptation and third, corrective action recommendation based adaptation. For example, consider a mobile instant messenger application that is composed by a text input component, a display component and a network component. At the time of the composition, keyboards, speech recognition and handwriting pads can all be used to realise a text input component. For the first strategy, every available component is presented to the user regardless the quality and functional aspects of a certain component. For the second strategy, an application can explicitly ask the environment to guarantee a keyboard; other types of devices are simply ignored. And for the third strategy, a prioritised component list is suggested to the user.

Applications in a Pervasive Computing environment do not rely on a single adaptation strategy. In PCOM [11], component reselection is supported by applying a user defined strategy. Aura [22] and Gaia [49] provides context-aware adaptation support at the system level. And Task Computing Environment [80] combines simple context filtering and manual adaptation. While corrective action recommendation is conceptually promising, correctness has to be proven by conducting experiments over a period of time.
2.1.7 Privacy and Security

A Pervasive Computing environment attempts to embrace the surrounding physical environment as if it were part of the integrated computing experience. Through embedded devices, applications and the environment are designed to adapt their behaviours according the contextual information. Despite the "intelligent" adaptation, this feature creates a heavy impact on the privacy of users. First, sensitive data could be revealed. In environments like hospitals, patient records must not be accessed unless necessary. Second, personal information must be securely accessed with authorisation. Furthermore, in certain situations, peace should be maintained where no computation is allowed.

Security in Pervasive Computing environment is separated into two categories: physical security and environmental security. As the boundaries may expand, intruders must be prevented from accessing multiple remote physical places where the computation occurs. On the other hand, a security policy for entities in one environment may not prevent attack which was originated from another environment.

2.1.8 Software Challenges in Pervasive Computing

Developing adequate software for the usage in Pervasive Computing poses a number of challenges. The dynamic Pervasive Computing environment accommodates hundreds or thousands of devices that will provide diverse functionalities. An application must be able to capitalize on such versatility and continuously support users in completing their tasks.

Devices in Pervasive Computing are manufactured by vendors in dissimilar technology. To interact with these devices, an agreement must be established on what can be used during the communication. Conventional software development tends to hardcoded the agreement at the design time. However, as the type of devices increase, it is infeasible to assume a single agreement between one application and many devices. Although runtime negotiation is promising, how this feature could be implemented remains unknown.

Context awareness lets applications provide task-relevant information to a user. However, context acquisition is difficult in a dynamic environment where context sources are potentially unknown. Context may be represented in different ways; understanding the formats and relationships between these formats is the essential for context-aware
adaptation. Humans tend to recognise differences very quickly while software applications do not. This particular weakness has to be compensated before Pervasive Computing is widely accepted by the user communities.

Adaptation plays a crucial component in fulfilling the Pervasive Computing promises. Different strategies can be applied when change is detected. In some cases, an "intrusion" requiring input or arbitration from the user is unavoidable. Even though a balance between proactivity and transparency during the adaptation is defined by a system architect, it is difficult to apply the balance to application development.

2.2 Software Architecture

Ever since the "software crisis"[29] was perceived, there has been some significant advancement in stressing the importance of structured system design. Core technology such as functional development, object-oriented analysis & design and component-based software engineering have been inspiring new approaches toward a software architectural-based system modelling.

The foundation of modern software architecture was built in [15] where the architectural model was divided into three components: elements, form and rationale. Architectural patterns and abstractions were then introduced in [52]. Architecture description languages (ADLs) have been proposed to support architecture-based software development in a number of works such as Darwin [43] that distinguishes components and connectors. ACME [23] supports mapping of architectural specifications from one ADL to another. A recent survey [64] has presented a definition and a classification framework for ADLs.

2.2.1 What is Software Architecture

Software systems are becoming increasingly complex; it is no longer practical to address system problems at the algorithmic and data structure levels. When a software system is constructed in a modular basis, the overall structure and interconnections between modules presents the basic architectural impression.

The word architecture in the building and construction industries is used to define the style and method of design and construction. If we consider software development as a
sequence of constructions, the definition of software architecture is analogous with the architecture that was introduced. In short, software architecture is "the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." [47].

Over the past decade, system architects have come to realize that software architecture is a critical success factor for system design and development. A good architecture ensures the correctness, performance, reliability, portability, scalability of the system. Reusable architectural style has made the development of new products much easier than before.

The architectural system design has been addressed in different level of abstractions. At the lowest level, functional design separates a piece of program into several small entities with each provides a discrete functionality. Object orientation extends the vision by packaging functions and data together into objects. Each object has several properties and methods to access these properties. A program's overall functionality is achieved through a set of interoperations between several objects. However, to use an object, one must know the identity of that object. Whenever an object changes its identity, all other objects have to be modified. In parallel to the object oriented design, layered system was introduced which essentially divides the system into several logical layers with each provides services to the layer above. The approach supports incremental development and reuse in different layers. Layered systems fall short in two deficiencies. First, not all systems are easily modelled in a layered basis and even if they do, performance may be lost. Second, it is not easy to find the right layer for a system.

With the rapid spread of computer networks, distributed systems have been introduced to abstract the interactions between many computers. A range of network topologies are used in distributed architectures, such as ring, star, tree or peer to peer. As a simple example, we can consider the client-server architecture. A server has a number of processes that provide services to other processes. A client communicates with a server and accesses the services. The server does not know in advance how many and which clients will access its services. In contrast, a client must know the address of a server (or locate the server at runtime) before accessing the required service. In the case of object-oriented techniques, client-server architecture allows objects to be accessed over a network. However, as the size and complexity of distributed software system increases, so
design needs to go beyond object-orientation. Components are then introduced as a coarse-grained software entity that is reusable and composable.

### 2.2.2 Component-based Architecture

Component-based architecture is a natural evolution from the object-oriented paradigm. "A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties." [14].

A component is a self-contained unit that interacts with other components only through well-defined interfaces. As demonstrated in Figure 2-1, the interface is isolated from any component who wants to use the interface (a client) or wants to implement the interface (a provider). Technically speaking, an interface should specify a set of operations and the semantics of each operation. An interface specification can be viewed as a contract between a client and a provider. In this case [14], the contract should specify what has to be done before using the interface and what has to be implemented to meet the service promised by the interface. In a more concrete operational level, pre and post conditions can be applied to each operation.

An independent interface decouples clients and providers. With the pre and post conditions fully specified and satisfied, components are substitutable and thus reusable. Considering the example in Figure 2-1, Component A requires certain functionality that is characterised in the interface. Both components B1 and B2 provide that functionality through implementing the interface. Then, B1 and B2 are substitutable. The same functionality may be required again by future components A1...An, component B1 and/or B2 can be reused to fulfil their obligations.
Shaw and Clements [53] defined a connector as a mechanism that mediates communication, coordination, or cooperation among components. As shown in Figure 2-2, a connector focuses on interactions while a component focuses on functionalities. In a distributed environment where components are distributed over computing nodes, connectors may also represent communication protocols such as RPC, RMI or SQL link between component and persistent database entities.

A component system architecture expresses the concerns of a set of platform decisions and a set of frameworks for component interactions and composition [14]. The essence of a platform is an infrastructure for component planting. The kind of components to be planted is affected by the platform decisions. A framework governs the mechanisms for which components are connected and composite. The mechanisms are enforced by one or more policies which were set by the framework. The exact policies are not restricted.

Components are fundamental elements in architectural design for large scale component-based software system. The overall functionality is a collaborative effort between many composable and reusable components. As proposed in [21], component based design for Pervasive Computing has been exploited in a number of projects. In PCOM [11], applications are dynamically composed of components — either local or remote. Dependencies between components are captured in contracts. Strategies are applied in cases where the environment changes. Sparkle [63] uses downloadable components to promote dynamic component composition. Small components are downloaded from the network on demand at runtime, and may be cached for future use.

Component-based architecture is the basic step toward an integrated architecture for Pervasive Computing. However, the general purpose of a software component is a unit of composition. They are mainly built for software system, not for human. From a system architect point of view, they are too small to abstract the interactions between users and their surrounding Pervasive Computing environments. Additionally, interactions with heterogeneous devices requires tremendous amount of connectors that are impractical to implement in the Pervasive Computing environment. Inspired by the web services
community, we believe that service oriented architectural design has the potential to formulate the interaction patterns and styles involved between human and software entities in our daily life.

2.2.3 Service Oriented Architecture

The evolution of technologies in smart environments has changed the way we interact with multiple devices (mobile phones, PDAs, wearables, laptops, GPS terminals, sensors, RFIDs) [76] that are creating new opportunities for Pervasive Computing. Moreover, the massive diffusion of the Internet has created a nearly ubiquitous information and communications infrastructure which enables services to be accessed from almost every computer [66]. From a user perspective, devices are becoming increasingly irrelevant to the task he or she needs to perform. What matters are the services that a device can offer. For example, we talk to friends through a mobile phone service, not the handset; we print documents through a printing service, not the printer box. In a Pervasive Computing environment, users are free to move while a similar degree of computing support is always expected at any place. However, the enormous amount of devices of which the environment is comprised often provides thousands of services at the same time. The implied problem is that a Pervasive Computing application does not know in advance what devices and services will be available and how they can be accessed to achieve its goal. One way to cope with this problem is to consider the use of Service-Oriented Architecture.

Service-Oriented Architecture (SOA) has been defined as:

- A way of designing a software system to provide services to either end-user applications or other services through published and discoverable interfaces. [3]
- An architectural style whose goal is to achieve loose coupling among interacting software agents. [37]
- A set of components which can be invoked, and whose interface descriptions can be published and discovered. [38]

The concept of SOA is a natural evolution of ideas that were proposed in the early 1980s when Object Oriented Programming was introduced, and developed into the 1990s in major software development communities such as COM/DCOM [16], CORBA [69] and Java [39]. Recently, SOA has been further realised by emerging technology standards like
Web Services [20] and XML. Figure 2-3 shows that different technologies can be used to implement Service Oriented Architecture.

![Diagram of Software Architecture and Service-Oriented Architecture](image)

Figure 2-3 Technologies for Service Oriented Architecture.

A key notion in SOA is the term Service. From a general point of view, a service is anything that is done which benefits another by one or more entities. A service does not have to be technology-oriented, but may have many non-functional elements that can be realised through a technological solution [86].

Practically, service in SOA has been defined as "a course-grained, discoverable software entity" in [3] or "an abstract resource that represents a capability of performing tasks" in [20]. Although the word "service" means different things to different people, for the purpose of this thesis, we will use a definition of service taken from ICSOC 2005 [91] "Services are autonomous, platform-independent computational entities". In contrast to the CBSE where components are intended to be reused at the programming level, services in SOA are designed to be assembled and reused within and across organizational boundaries as long as the service agreement is conformed. Therefore, a service can have multiple implementations that are interchangeable. For instance, if one printing service implementation failed, it can be replaced by another without the client's knowledge.

A service is described by its **service description**. This characterises the service in terms of its capability, behaviour, interface, transport, pre/post condition and non-functional attributes. It is common that a service oriented architecture relies solely on syntactic-
based service descriptions which describe services by their respective service interface and a textual description. Service semantics in this approach are implicitly coded into the operation name and depend on an operation behaviour specification which is defined by standard organisation. However, the approach has a significant flaw where building mutual consensus for every service interface takes a long time and can not adapt to continually appearing new services. Approaches such as adapter chains [82], attacked the problem by composing interface adapter chains from an adapter repository. Recent works on the Semantic Web [7] and Ontology Web Language (OWL) [25] have been focused on explicitly describing the semantic for services and the Web based on RDF [19] that provides the basic ontological primitives.

![Service Oriented Architecture Diagram](attachment:service_architecture.png)

**Figure 2-4 The Service Oriented Architecture**

Although the notion of service plays the key, the basic Service-Oriented Architecture is not only about services but also a coherent framework of the relations between three independent elements: the service provider, the service consumer and the service registry. The interactions involved between these elements are expressed in three correlative operations: publish, find and bind.

Once implemented, a service is made discoverable through its service description which is published to the service registry which potential consumers have access to. The consumer looks for a required service by querying the registry with a set of criteria. The service registry matches the criteria against all available services, and returns services that were matched. To use a service, both service consumer and provider must agree on a service level agreement that regulates the way of which a service is invoked. The agreement is often included in the service description that a consumer must interpret after which the desired service is selected. The consumer then binds to the service provider and executes the service. Figure 2-4 shows a high level view of Service Oriented Architecture.
The usage of a service registry supports dynamic binding to services which is particularly important in cases where service availability or a client’s requirements for a service may change.

2.2.3.1 Service Provider vs. Consumer
Let us first consider the service provider and the service consumer. A service is implemented by one or more service providers who should also supply the service description although this is not compulsory. A service provider can be a mainframe system, a software component or any other software entity that carries out incoming requests and optionally sends the response back to the consumer in a concerted format that is specified in the service description.

A service consumer is an application, service, or other type of software entity that is asking for a service. It engages the service by sending a request message over a transport layer to the service implementation. The exact details of the service interface are revealed upon examination of the service description.

Service orientation facilitates the loosely-coupled through the separation of the service consumer and provider. The service provider does not assume a particular service consumer with which they will interact whereas the service consumer is not bound to a particular service provider until they need to do so. Consequently, both provider and consumer are substitutable as long as they comply with the service agreement imposed by the service description.

2.2.3.2 Service Registry
The above two components need to be augmented with a mechanism to support the discovery of services. To build an integrated SOA, the services must be published to a known place where clients may search for them. Such a place is referred to as service registry.

The service provider publishes a service to the registry through which the service description is published. The client looks for the required service by supplying a set of criteria to the registry where published services are matched against with them. The exact matching mechanism is an implementation choice and directly affected the way a service is described. Finally, the matched service(s) are returned to the client.
SOA promotes the dynamic environment where services could be added, removed and updated at anytime. It is possible to have a service consumer, provider and registry reside in a single node, distributed on a local area network or even across several networks. Adequate modifications must be made to ensure the consistency of services information contained in one or more service registries. While assumption is made that the location of a service registry is known to the client prior to the interaction, a service registry can be implemented as a special service and broadcast its presence to both service consumer and provider over the network. The service registry can also have one or more repository components that conserve supplemental information about each service such as functional/non-functional properties.

2.2.3.3 Publish, Find and Bind

A SOA is a collaborative system by which entities are glued together using the publish-find-bind interaction pattern. To be accessible, a service must be published by means of its service description being published to the service registry so that it can be discovered and invoked by a service consumer. A service consumer can find a required service by querying the service registry. The find operation includes three steps. First, a service consumer supplies a set of criteria which describes the preferred service in terms of its input, output, functional/non-functional properties etc, to the service registry. Second, the service registry performs service matching by considering the supplied criteria against the available services. Third, matched services are returned. After that, the service consumer binds to a selected service provider and consumes the service. To support service life cycle management, further interactions such as service release, destroy, remove and update are required. From an architecture point of view, each of these interactions improves system integrity. Due to the size of this report, we will not discuss them in further detail. Figure 2-5 illustrates a sequence diagram of the interaction pattern.

2.2.3.4 Communication Protocol

One of the key ingredients behind SOA is the ability to invoke services regardless of their proprietary implementation. SOA is not bound to any specific communication protocol. It only requires an agreement on the protocol that is obeyed by all parties involved in the communication. With increasing complexity, services may use standard communication techniques both at the physical and network level (e.g., Ethernet and IP) and several types of communication protocols above that to interchange data with other clients that use the
same technology. Given the evolutionary nature, the communication protocol must be platform-independent, extensible, and as lightweight as possible.

2.2.3.5 Communication Protocol

One of the key ingredients behind SOA is the ability to invoke services regardless of their proprietary implementation. It only requires an agreement on the protocol that is obeyed by all parties involved in the communication. With increasing complexity, services may use standard communication techniques both at the physical and network level (e.g., Ethernet and IP) and several types of communication protocols above it to exchange data. Given the evolutionary nature, the communication protocol must be platform-independent, extensible, and as lightweight as possible.

A communication protocol in SOA encompasses the message protocol and transport mechanisms. Figure 2-6 depicts these two components. A message protocol is an agreed-upon format for transmitting data between parties. The open XML standards have been widely accepted as the lingua franca for information and data encoding [30]. Building a message protocol using XML is thus a natural choice for SOA. The transport mechanisms
specify rules used to send data in the form of message units between services. Examples include: HTTP, FTP and SMTP. By a clear separation of message layer and transport layer, SOA does not force the usage of a specific high-level protocol for each service. In other word, any message protocol $MP$ can be conveyed over any transport $TP$ as long as they are supported by both parties. The required communication protocol is explicitly specified in the published service description. To invoke the service, a consumer should follow the concerted message protocol and send the message over a transport that is acceptable by the provider.

To reach a broader vision of Service Oriented Computing (SOC) [58], many problems must be addressed on top of the basic SOA such as service composition, service management, choreography and orchestration [10], coordination, security and other general concerns that apply to all components in service architecture [57]. Both academics and industry have been working on the above issues and proposed some emerging standards such as WS-Coordination [31], OASIS WS-Security [67], WS-cdl [65], BPEL4WS [89]. For the purpose of this report, we will not discuss them in further details.

### 2.2.4 Web Service Architecture

Service Oriented Computing [58] envisions a computing paradigm that utilizes services as fundamental elements for developing applications/solutions. One of the most prominent service architecture been proposed is the Web Service Architecture [20].

According to W3C, "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network." [20]. The architecture is
Chapter 2. Literature Review

divided into three main areas—service descriptions, communication protocols and service discovery. In typical web services architecture [12], a service provider has a service that is useful to other systems. The provider creates a WSDL service description that defines the service interface, that is, the operations of the service and the input and output messages. A binding description that describes how to send each message over a transport (SOAP over http in particular) is then added. The WSDL service description is published to one or more discovery agencies such as Universal Description, Discovery and Integration (UDDI) which associates service to their business and taxonomy. Flexible service discovery is performed using UDDI including keywords, category and business based query. Finally, service requesters find services via discovery agencies and use the WSDL description to interact with the corresponding service provider.

2.2.4.1 Web Service Definition Language (WSDL)

For a client to use a service, the precise service interface must be described. CORBA's IDL is one of the many examples that define interface in a platform independent manner. WSDL extends the concept in IDL and adds message format and transport independence.

WSDL [25] is an XML format developed by IBM and Microsoft for describing web services as a collection of endpoints that a client can exchange messages. In a WSDL document, the abstract definition of endpoints and messages is separated from their concrete network and data format bindings. Messages represent the abstract descriptions of the data being exchanged, and port types represent abstract collection of operations which make use of messages. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports defines a service.

![Figure 2-7 WSDL document components.](image-url)
WSDL makes use of the following eight elements in the definition of web services:

- **Definitions.** The root element of all WSDL documents. It defines the name of the web service, declares namespaces and contains all other elements;
- **Type.** The data type definition using some type system (such as XSD).
- **Message.** An abstract but typed definition of the data being exchanged.
- **Operation.** An abstract description of an action supported by the service.
- **Port Type.** An abstract collection of operations supported by one or more endpoints.
- **Binding.** A concrete protocol and data format specification for a particular port type.
- **Port.** A single endpoint defined as a combination of a binding and a network address.
- **Service.** A collection of related endpoints.

Any typed communication must agree on a common vocabulary [30]. Instead of reinventing a new type definition language, WSDL uses XML schemas (XSD) as the canonical type system. However, extensibility elements may be added under the *types* element to identify the type system being used other than XSD. Figure 2-8 shows how to define a complex data type *TradePriceRequest*.

```
<types>
  <schema ...
  <element name="TradePriceRequest">
    <complexType>
      <all>
        <element name="tickerSymbol" type="string"/>
      </all>
    </complexType>
  </element>
  ...
</types>
```

*Figure 2-8 WSDL Types example.*

The *message* element is formed by one or more message *part*. It describes the abstract contents of a message to be used in one or more operations. Each part is associated with a type using message type attribute. Parts are considered to be abstract definitions of the message contents. Message binding maps these abstract definitions to their concrete format although the resulting format may be similar or identical to the abstract definition. Figure 2-9 shows an external data type *TradePriceRequest* is used by message *GetLastTradePriceInput* and one data type defined in XSD (*xsd:string*) is used by message *GetLastTradePriceOutput*. Both messages appear in the operation *GetLastTradePrice*. 

26
Chapter 2. Literature Review

The *operation* and *portType* elements aggregate messages to define interactions. Each operation presents a message exchange pattern (MEP) that is supported by an endpoint. Four transmission primitives are defined by WSDL although only *request-response* and *one-way* are defined by WDSL bindings. The *input* and *output* elements specify the abstract message format for the request and response, respectively. A *binding* specifies concrete message format and protocol details for messages and operations within a particular *portType*. There can be any number of bindings for a specific *portType*. WSDL bindings are not limited to SOAP, but can use any transport such as HTTP GET&POST or MIME. Individual endpoint is defined by a *port* element as a union of a binding and a network address. Finally, a *service* groups a set of related ports. The approach allows new endpoints to be added incrementally, without affecting existing endpoints or their users. Figure 2-10 shows SOAP binding for *StockQuoteService*.

![Figure 2-9 WSDL abstract definitions](image_url)

![Figure 2-10 WSDL concrete definitions](image_url)
WSDL provides a formal definition of a consumer-provider interaction. It specifies the abstract information of a service and then binds it to the concrete message and transport protocol. As a result, consumer can use WSDL as input to dynamically generate service requests at runtime. However, WSDL does not tell the non-functional properties of a service, nor does it define semantic meaning of a service.

2.2.4.2 Simple Object Access Protocol (SOAP)

Initially created by Microsoft and later becoming a collaborative effort from W3C, SOAP[17] is an XML-based protocol for messaging and making remote procedure calls in a decentralized, distributed environment. SOAP defines a framework for describing a message which can be exchanged over any underlying transports protocol, such as: HTTP, SMTP and MIME.

As depicted in Figure 2-11, a SOAP message has a very simple structure: an envelope containing two child elements; an optional header; and a required body. The envelope defines a set of namespaces that are used by the rest of the SOAP message. Each envelope must contain exactly one body element which can have as many valid XML syntax as required. A header element contains any complementary information relevant to process a SOAP message, such as authentication, transaction and payments information. Processors in the SOAP processing chain may modify or ignore items from the header.

As a messaging protocol, SOAP also defines a framework for its message processing and a data encoding specification for encoding or decoding application data types into XML format when they are to be exchanged using SOAP. Transport protocols for exchanging
message are not concerned by SOAP. However, SOAP over HTTP is by far the most widely accepted protocol for web service communication. It is very nature to implement request-response MEP using SOAP over HTTP as the later is a request-response based protocol. A SOAP request message is posted to a HTTP server over an HTTP request and a SOAP response message is carried by an HTTP response. Figure 2-12 illustrates HTTP request and response that contain a SOAP message. The $\textit{SOAPAction}$ HTTP header indicates the intended purpose of this SOAP HTTP request. Body of the SOAP response message contains any response data. In this case, a $\textit{GetLastTradePrice}$ request is send and a response containing price information is returned.

![Figure 2-12 SOAP message over HTTP](image)

SOAP has been implemented on a number of programming languages, including C#, Java, Perl and PHP. Assuming they conform to the SOAP specification, messages can be exchanged between services that are implemented in different languages.

2.2.4.3 Universal Discovery Description and Integration (UDDI)

The Universal Discovery Description and Integration [90] is a centralised approach that defines a registry to publish and find information about services and businesses. The publicly accessible UDDI Business Registry (UBR) is operated by several companies such as IBM\(^1\) and Microsoft\(^2\). It is also possible to operate a private UBR to streamline access to an organisation's internal services.

UDDI defines two components about the structure and operation of a registry. First, an XML schema defines four key data structures: $\textit{businessEntity}$, $\textit{businessService}$, $\textit{bindingTemplate}$ and $\textit{tModel}$. Second, a set of interfaces to interact with the registry.

\(^1\) IBM UDDI Business Registry: https://uddi.ibm.com/ubr/
\(^2\) Microsoft UDDI Business Registry: http://uddi.microsoft.com/
As illustrated in Figure 2-13, *BusinessEntity* describes simple information about a business including name, description, services offered and contact information. A *businessEntity* may have more than one *businessService* provide detail on each service being offered. Each service can have multiple *bindingTemplates* that describe different technical entry points for the service (email address, URL etc.). Finally, *tModel* (technical model) represents a technical specification a service uses. Each data structure above is assigned a unique key (UUID) that might be cross-referenced when it is registers. To effectively locating a business or service, UDDI pre-registers three standard taxonomies (NAICS, UNSPSC and ISO 3166) as tModels which can be cross-referenced by both *businessEntity* and *businessService* using the *categoryBag* element. The idea behind tModel is that, a technical specification can be cross-referenced by several services.

UDDI interface consists of two subsets: the inquiry interface for locating businesses, services, bindings or tModels and the publisher interface for manipulating UDDI data in the registry. To save spaces, interface details are not discussed at here. A UBR is itself a SOAP Web Service [48]. Any interface can be accessed either via a Web site or by making SOAP requests that conform to the UDDI API specification. XML fragment in Figure 2-14 demonstrates how to find a business in a UDDI registry using SOAP.

```
<Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  <Body>
    <find business xmlns="urn:uddi-org:api" generic="1.0">
      <name>GetQuote Company</name>
    </find business>
  </Body>
</Envelope>
```

*Figure 2-14 SOAP message to find a business in a UDDI registry.*

Web service is a coherent framework of WSDL, SOAP and UDDI which embodies the abstract definition of service description, communication protocol and service discovery.
Chapter 2. Literature Review

To invoke a service, the service provider first publishes its service (WSDL) to UBR alone with other data that is required. The service consumer find a service by issue a SOAP message to UBR which will return a service list by generate a SOAP response. After the consumer has learned the technical details of a service from its tModel (typically points to an external WSDL file), it generates service proxies to invoke a service.

2.3 Ontology modelling

The emerging Pervasive Computing technology aims to provide seamless support to its users at anytime from anywhere. Applications are no longer coupled to devices; they are viewed as entities that perform tasks on behalf of users. To work in a highly dynamic environment, applications must recognise different services in a Pervasive Computing environment and subsequently make use of them. The implied problem is how services can be modelled in a way that they can be understood by both humans and software applications. Furthermore, different types of contextual information need a formal semantic definition so that they can be used by pervasive applications to better adapt their behaviours. One way to model such complex information is to use ontology.

An Ontology is a formal specification of a conceptualization. The term "Ontology" is borrowed from philosophy where an ontology is a systematic account of existence [94]. In the programming literature, ontology defines computer understandable concepts and relationships among them. A common ontology provides the vocabulary to which both parties involved in an interaction must be committed.

In a Pervasive Computing environment, services are described by their service descriptions which can use any arbitrary terms, ie. a print service and a printing service. This is not a big issue as human can understand this very quickly. However, the same term may have different meanings in different contexts, and different terms may be used for a service that has the same meaning. It is difficult to recognise these concepts by a software entity unless a formal definition of concepts is given. Different services may have a number of properties that distinct from or interrelates with other properties and concepts. For example, WSDL abstract definition in Figure 2-9 specifies a StockQuoteService service which doesn't mean anything to a software application other than a line of text. If the message is associated with a \texttt{NASDAQ\_StockQuoteService} concept
that is derived from a generic StockQuoteService concept in a StockQuote ontology, an application would "know" the semantic meaning of the service. The same approach can be applied to message elements in Figure 2-9 where a message GetLastTradePriceOutput can be associated with a NASDAQ_StockPrice concept.

Ontology-based modelling has been proved to be a promising solution in the semantic web [7] community where it is used to model web services by providing additional vocabularies along with the formal semantics. One of the collaborative efforts is the establishment of the Web Ontology Language (OWL) for computer applications that need to process the information instead of just presenting information to humans [25]. By incorporating semantic web technology and ontology into Pervasive Computing, service discovery is no longer limited to finding a specific type of service but is rather capable of retrieving services that are conceptually similar to what application intended to lookup. Thus improving the degree of supports available to the users.

Contextual information in Pervasive Computing is increasingly being used by applications to provide users with adequate task support. This information is often acquired from different sources which exhibit themselves by various means. To handle such diversity, Ontologies have been used to model contextual information in a number of projects. CoBrA [35] designed an OWL based ontology for context representation, reasoning, sharing knowledge and user’s privacy control in a broker-based intelligent meeting room environment. An RDF-based ontology for representing raw sensor-based context information to be used on mobile devices is proposed in [71]. The ontology structure permits the use of composited context and constructing context hierarchies. CONCON [99] proposed an OWL encoded ontology for context modelling and reasoning based on a layered approach. The upper layer consists of some generic concepts about the basic contexts. Any domain-specific ontology can be added through extensibility provided by the upper layer. Although OWL as an emerging technology standard has drawn many attentions, alternative solutions, including Knowledge Interchange Format (KIF) [54], Cyc [26] and Process Specification Language (PSL) [82] have been developed to support knowledge interchange among machines or software entities. For the purpose of this thesis, we will not discuss them in details.
Finally, the Semantic Web in UbiComp Special Interest Group\(^1\) has defined a standard ontology to support knowledge representation and communication interoperability in building Pervasive Computing applications. This project's goal is not aimed to construct a comprehensive ontology library, but to provide useful ontologies that can be reused by individual applications.

### 2.4 Combining SOA with Pervasive Computing

From a Pervasive Computing perspective, SOA is an architectural style that utilises a collection of loosely coupled services as fundamental building blocks for software development which supports users in completing their tasks. A service is a conceptual abstraction that must be realised by one or more service providers which are free to use any technology they wish, as long as the service commitment is satisfied. Devices in a Pervasive Computing environment are likely to be manufactured by different vendors which undoubtedly will use distinct technology. It is impracticable to develop a pervasive application that is able to communicate with any unknown device due to the mobility requirement imposed. The view of service has the potential to glue what application needs (the required functionality) with the capability a device can offer by transforming device into one or more concreted software services that expose the complete device functionality through a well-defined interface.

Apart of the service interface, a service is further described by the required message protocol to which a client must follow in order to invoke the service, and one or more transport mechanisms to convey the messages. Being supplied from different vendors, heterogeneous devices in a Pervasive Computing environment differ from not only technology but also their service implementations. Therefore, it is not always clear how to interact with the required service in advanced. Depending on the service description which is written in a well-known standard basis, i.e. XML, pervasive applications (can be other services) in SOA are developed as independent of the service implementations and communication protocols. Service interface details are only revealed at the runtime and thus enable late binding. As a consequent, a pervasive application can still work in an

---

\(^1\) UbiComp Special Interest Group: http://pervasive.semanticweb.org/
uncharted environment without knowing what language the service is coded in or what platform the service runs on.

Pervasive application severely relies on contextual information obtained from disparate sources which have their own proprietary APIs, data models and communication protocols. SOA unify the way on which a context source is accessed through a new layer of abstraction called context service. Sources are grouped by the context they provide. Each group represents a context service that has a standard interface. The only difference between a service and context service is that the context service provides only contextual information whereas a service generally presents a complete business functionality. Therefore, context service can also be published to the service registry in the same way as a service is published. In a sense, they are equally treated as the fundamental element to form a SOA for Pervasive Computing.

Users and devices in a Pervasive Computing environment may move from place to place at anytime. The implication is that service availability varies from one environment to another. In consequence, software architecture must perceive the change as continue to facilitate adequate user support. SOA directly addresses the problem from two perspectives. First, a service registry acts as meeting point for pervasive applications and available services in the current environment. Services are updated either explicitly by their provider or implicitly through leasing. The updated service descriptions are published to reflect such change. Second, the abstraction nature of the service decides that service providers are substitutable. Pervasive application simply invokes a service across environments regardless the service provider. If one provider becomes not available, it can be replaced by another one as long as it embodies the service.

Ranging from tiny sensors to mainframe system, devices in Pervasive Computing environment provides a variety of information and services. Finding the most suitable service in a Pervasive Computing environment is challenging. First, it is difficult to define the meaning of a suitable service. Second, the lack of a universal service property standard poses notable problems when matchmaking takes place. And third, context awareness cause reconsideration of the criteria used in service matching. SOA does not restrict the way in which service discovery is conducted. For example, UDDI [90] in web service architecture is a centralised approach that matches services at syntactical level,
JINI [44] use java interface or java attributes for service matching in one or more Jini lookup service and UPnP matches services in a peer to peer manner through service template that consists of service type and location. In a service-rich environment, it is unlikely to have a single prefect matched service for each discovery request. A service matching algorithm therefore, must return a prioritised service list whose priority is calculated based on the similarity to the request. Satyanarayanan in [60] proposed three alternative adaptation strategies: application adaptation, reservation-based guarantee and adaptation using corrective actions to minimize user distractions. Using the prioritised service list, pervasive application recommends a set of options to the user who is likely to select the right one.

Using the discovered service description, a service consumer constructs a request message with the data format specified, then binds the message and sends it over a transport. The service provider executes the desired process and optionally returns a message whose format is also specified in the service description. Once all services are deployed, the required functionalities to perform a task on behalf of the user can be easily accessed through the standard service interface, message protocols and transport mechanisms. Such loosely coupled SOA enables an evolutionary system by mean of incrementally adding new services. However, it is difficult for users and client programs to know them automatically. Suppose a printing service is offered by a wireless printer, when a newly added printer calls itself a print service, association can't take place if wireless camera requests a printing service [91]. One way to overcome such obstacle is to build common concepts, vocabularies and relationships among past, current and future services, for example, a relationship "equivalent to" between service concept print and printing or a "subClassOf" between concept digitalPrint and print. Existing work such as OWL [25] SOUPA [36] have already focused on defining concepts (classes), relationships between classes and properties those classes may have. Leverage from ontology modelling, service matching algorithm will be able to find services that are conceptually similar to the required one but syntactically different. Same approach can be applied to context service where different terms are used. Other than concept modelling, the use of description logic in ontology enables high level context inference through many low level context sources [99]. Moreover, the use of ontology increases the system interoperability as we can define concept translations across several environments.
Although SOA was initially developed for enterprise scale environment, the loosely-couple and late binding concepts have the potentials to dramatically change the way of software development in Pervasive Computing. Therefore, this research's hypothesis is:

"By abstracting devices and context sources into discoverable services, we can provide an architecture that supports the dynamic nature of human activity. As a consequence, it is anticipated that the Service Oriented Architecture is a natural choice for software development in the Pervasive Computing environment."

2.5 Summary

In this chapter, we presented an overview on some important characteristics of Pervasive Computing. A global Pervasive Computing system can be divided into many structured virtual environments which are conceptually matched to a physical environment. People, applications, devices and services may move from one environment to another. The implication of this is the presence of a highly dynamic environment where no assumption can be made on the identity of any specific device that an application will interact with.

A review on the concepts and practices of software architecture was then conducted. Problems of component-based architecture were identified followed by an introduction of the concepts and interactions of the Service Oriented Architecture (SOA). As a de facto standard for realising SOA, the Web Service architecture was also investigated. Despite its platform, communication and message protocol independency, an SOA does not specify the terms and vocabularies for knowledge sharing among parties involved in an interaction. Ontology-based modelling as a promising solution was proposed. In particular, ontologies can be used to specify the semantic meaning and relationships between services or contexts.

One of the challenges in building pervasive applications is to provide task assistance whenever is needed and wherever is needed. A structured environmental view helps separating architectural design from global ad-hoc interaction, thus promoting a more practical model which concerns inter-environment interaction. However, heterogeneous devices in each environment raise significant problems for connector-based interaction modelling. In particular, it is impractical to implement connectors for each device. On the
Chapter 2. Literature Review

other hand, interaction in SOA is characterised in a protocol independent basis. Once the service description has been analysed, the service consumer dynamically constructs a request message with the data format specified, binds the message over a transport mechanism and sends it to the service provider. SOA thus enables protocol independent communication between application and devices in a Pervasive Computing environment.

Context-aware applications provide task relevant information to the users and adapt themselves accordingly. Meanwhile, context sources are not limited to a specific device and the format of a particular context is not constrained. Context acquisition becomes the key in realising context-awareness. In the Service Oriented Architecture, irrelevant context sources can be grouped by the context they provide. Each group presents a unique context provisioning service which has a common service interface. The unified view of service greatly simplified the context extraction process.

Intrusion is unavoidable; the system architect must define the roles of humans in resolving uncertainty and provide a balance between proactivity and transparency. However, it is difficult to apply this principle to application development. Service discovery in SOA returns a prioritised list of candidate services which makes corrective action recommendation easier for an application. Finally, we concluded that Service Oriented Architecture is a natural choice for Pervasive Computing.
Chapter 3

3 Related Work

3.1 Oxygen

The Oxygen project [55] [56] carried out at MIT argued that computation has forced humans to interact with devices using a keyboard and mouse in a machinelike way. Oxygen therefore envisioned a new computing paradigm where computation will be human-centred, freely available whenever we need it.

The Oxygen project relies on a collection of technologies. The first is a new collection of computational devices: Enviro21s (E21s), embedded in homes, offices, and cars that sense and affect a user's immediate environment. Second, reconfigurable handheld devices: Handy21s (H21s), which rely on software to reconfigure themselves to support multiple communication protocols or to perform a wide variety of useful functions. Third, the dynamic self-configuring networks: N21s, to form collaborative regions between H21s and E21s. Other interrelated technologies include Oxygen software technologies, Oxygen perceptual technologies and Oxygen user technologies.

1 The prototype is based on a Compaq iPaq with a 200Mhz StrongArm processor, extended by a custom BackPAQ sleeve. The BackPAQ contains a digital camera, an accelerometer, a FPGA, an audio codec and headset jack, and two PCMCIA slots. Oxygen software runs under Linux on the H21.
Oxygen’s software architecture configures currently available software services to achieve the desired goals. The architecture heavily relies on three concepts. **Abstraction**, serves as the container to provide adaptive interface to observe and influence the component’s behaviour. **Specification** exposes features of abstraction to other system components and **persistent object stores with transactional semantics** which provides generic store for code, data objects, and specifications.

Overall, the Oxygen project serves as an umbrella to cover many research disciplines. The approach depends on the availability of the reconfigurable handheld devices H21s. Although, a prototype had been developed as illustrated in Figure 3-1, and there are real commercial products available on the marketplace today such as the Sony-Ericsson P910i[^1], Windows Mobile system[^2] and Symbian OS[^3]. The H21s, E21s, N21s aggregation are still far from widely accepted by the community. Oxygen assumed E21s as well as H21s are universal communication and computation devices so that the same software can run on both devices. However, the heterogeneous Pervasive Computing environment often implies that independent implementations of H21s and E21s can vary from one to another. This poses many challenges for software development. For example, how would software on an ARM-based H21 device running Linux access an E21 device which has an Intel-based processor running Microsoft Windows? Furthermore, Oxygen’s E21s connect directly to a greater number and wider variety of sensors and appliances. How would Oxygen application distinguish, recognise, search and use these devices is still missing.

### 3.2 Aura

Project Aura at CMU [22] aims to minimize user distractions and create an environment that adapts to the user's context. Aura proposed a new architectural framework that solves two problems of Pervasive computing. First, it allows work to be preserved by attempting to reconfigure the new environment and resume the task started elsewhere early on when moving between different environments. Second, it tackles the problem of computation adaptation in the presence of dynamic resource variability.

The key ingredients of Aura's architectural framework are task manager (Prism) that represents user tasks as collections of services, context observer that allows the task to be configured in a way that is appropriate to the environment, and environment manager that coordinate resource monitoring and adaptation. The services needed to support a user’s task are carried out by service suppliers which are typically implemented as wrappers of the traditional applications. Figure 3-2 shows Aura’s architecture from different perspectives. Finally, interactions between components are carried out by explicit connectors that hide details of interaction mechanism.

*Prism* describes a task as a coalition of abstract services, such as "edit text" or "play video". Upon receiving events from *Context Observer* and indications from the user, Prism coordinates the migration, suspension, instantiation and part adjustment of the task when constraints specified in task descriptions are altered by context changes.

*Context Observers* provide information about the physical context and generates events back to *Prism* and the *Environment Manager* in a unified manner. As illustrated in Figure 3-3, Aura's contextual information service a database-like interface where queries are

---

**Figure 3-2 Aura’s Architecture**

**Figure 3-3 Aura’s Contextual Information Service architecture.**
decomposed to operate on individual service provider and results are synthesized. The architecture provides information on four classes of entities: people, devices, physical spaces, and networks as shown. Further information about the Contextual Information Service can be found in [32].

An Environment Manager coordinates the service suppliers by providing a registry to which service suppliers is registered. Distributed file access in Aura's architecture is encapsulated in Environment Manager. The choice of such access is depended on certain implementation issues. Service suppliers aim to provide abstract services that fulfil a task’s requirements. Aura uses XML-based language for the service description. It assume a shared vocabulary and interpretation is committed by a given service type.

Interactions in Aura are characterised using connectors. Latest progress in Aura [42] defined the interaction protocol of each connector type, discussed the contents and format of the exchanged message. Implementation of each connector type has a specific low-level interaction mechanism. Such as local method calls, RPC, DCOM or RMI. A service request is assigned a handle for the appropriate connector to reach the supplier.

As one of the pioneer projects, the architecture provides an abstract system model for Pervasive Computing. An implementation of such an architecture may depend on the local environment. Building on top of middleware technology such as CORBA or RMI, Aura is limited to deployment in a fairly resource-rich environment such as offices or Airports. Pervasive Computing Environment anticipates much smaller and resource-constrained devices to be embedded throughout the environment. Typically, these devices do not support such technology. Although Aura mentions a shared vocabulary and interpretation is assumed by a given service type, how these would be enforced is still unknown. For example, CORBA’s IDL does not specify what terms to be use in order to describe the attributes. Ad-hoc or arbitrary service description often indicates that the semantics of each service is hard coded into the program itself rather than explicitly specified which allows knowledge sharing among independent applications and other system entities. Aura also employs a private interaction protocol for its connectors which make inter-system interaction more difficult. Aura assumed the user only interacts with a single environment at given time and location whereas in Pervasive Computing, many environments may co-exist in a single physical place to support different user groups.
3.3 One.World

One.World [73] [75] is a new architecture to provide a comprehensive framework for building pervasive applications. The author argued that, an application's location and execution context constantly changes as people move through the physical world and users expect that their devices and applications will just plug together to share information. Thus, the architecture is based on three principles: expose change, compose dynamically, and facilitate sharing between applications and devices.

![Figure 3-4 Overview of One.World's architecture](image)

The system is structured into an n-tier system that separates the kernel from its users. Components in the foundation services directly address the three principles. Figure 3-4 shows an overview of One.World's architecture. First, One.World assumes that all code runs in a virtual machine. Second, all data are represented as tuples and third, all communication in One.World, whether local or remote, is through asynchronous events. Finally, environments host running applications, save persistent data, and facilitate the composition through environment nesting.

The design of environment nesting provides an outer environment the ability to interpose on an inner environment's interactions. Reusable functionalities can therefore be abstracted and placed into an outer environment whereas applications that rely on those

---

1 The implementation does not rely on a particular Virtual machine although One.World's prototype heavily relies on many features offered by JVM such as: reflection and object serialization.
functionalities are placed inside an inner environment. More importantly, an environment can group application functionality and persistent data within the same container, which can be migrated in a single operation.

Inside the tuples is a common data model, including the type system. Tuples have a fixed set of fields with specific types and the overall tuple has a type. One.World also has a special tuple type named DynamicTuple from which fields can be dynamically typed, added and removed.

Communication in One.World is based on asynchronous events which are represented by tuples with an additional event handler (source) field. In response to an event, a new event is sent to the event handler referenced by the original event's source field. Application functionality is implemented by exchanging events through importing, exporting event handlers as well as exposing the event handlers for linking. Imported and exported event handlers are typically declared in a component's constructor although they can be linked or unlinked at anytime. Events sent to the imported handlers are processed by the exported event handler once a link has been established. The connection can be terminated by an unlink command.

On top of foundation services, One.World introduces several system services that try to provide adequate system support for its common application needs. Several services, utilities and applications have been built based on One.World's architecture, such as a replication service, a user-application managing application named Emcee as well as a text-audio messaging system named Chat. The biology laboratory assistant has also been ported to One.World's architecture in the Labscape project [45] at University of Washington. For the purpose of this report, we will not discuss One.World's system services in further detail.

As evaluated in [74], the biggest limitation in One.World is that, tuples are implemented in Java classes, using public fields to represent a tuple's values. Although the approach provides some conveniences for applications, it also poses a considerable problem for services such as discovery, which process many different types of data for many different applications and must access the corresponding class files, in addition to the actual tuples. Furthermore, One.World is built on a virtual machine. Either Java or Microsoft .Net only provides limited functionality in their mobile version. One.World's architecture heavily
relies on reflection and object serialisation, making it impractical to be deployed to resource-constraint devices.

"The fundamental problem is that we've taken a single-node programming methodology—a programmatic data model, which expresses data schemas in the form of code, and applied it to a distributed system." By programming everything into code, One.World provides only minimum interoperability. Exclusively building on its own networking protocols in the form of remote events has made One.World harder to integrate with outside applications which aren’t written in Java. As suggested in [74], modern distributed systems must be compatible with Internet protocols first and offer additional capabilities second.

3.4 GAIA

Developed at the University of Illinois at Urbana-Champaign, GAIA [49] is a middleware infrastructure to develop applications for ubiquitous computing. It is argued in [50] that specialized embedded devices make computation spread across the physical space. The physical space as a whole, is perceived as computing system. GAIA proposed a model named Active Space which converts a physical space into a computing system. The result is a standard programming interface that is independent of the underlying environment. Figure 3-5 illustrates the relations between physical and Active spaces in GAIA.

A component based meta-operating system, GAIA OS, has been developed to support application access to and operation of the resources contained in the space. GAIA also implements a bootstrap mechanism that initiates the execution of the Gaia kernel in any arbitrary physical space using configuration file. Figure 3-6 shows Gaia architecture with three major building blocks: Gaia kernel, Application framework and the applications.

The Gaia kernel consists of a Component Management Core (CMC) and an interrelated set of basic services used by all Gaia applications. The implementation of CMC is a Unified Object Bus (UOB) [51] which coordinates all Gaia components and applications. On top of the CMC, Gaia presets five basic services. The event manager implements a decoupled event channel which forwards supplier's events to one or more consumers registered with the channel. A default set of event channels about Gaia system events are predefined to be consumed by interested Gaia components.
Contexts in Gaia are presented as first-order predicates with four arguments. An atomic context predicate is defined as Context(<ContextType>, <Subject>, <Relater>, <Object>) where the context type refers to the context the predicate is describing; the subject is the person, place, or thing with which the context is concerned; the relater associate the subject with the object using logic comparison (=, >, <), verb or preposition and the object is a value associated with the subject. Gaia integrates ontology to specify the structure of the context predicates including validity of the arguments. However, types of arguments are not restricted. Each context type is mapped to a class in the ontology. Furthermore, first order predicate allows high level context inference based on other contexts from a wide spectrum of context source.

The context infrastructure in Gaia [4] consists of several components. Query and optionally event channel are supported by Context Providers which provide the current contextual information from range of sources. Context Providers advertise the kind of contexts they provide to the Context Provider Lookup Service. Upon a query is received
from an application, the lookup service checks whether a context provider is available and return a satisfied provider's reference. Context Synthesizer use rule based logic to deduce high level context from various context provider. Probability measurement is introduced if the deducted information is uncertain. Other components include an Ontology Server, a Context History Service. Figure 3-7 presents an overview of Gaia context infrastructure.

Figure 3-7 Gaia context infrastructure.

The presence service in Gaia maintains soft-state information about both digital and physical entities. In particular, a digital entity periodically sends heartbeats to notify its presence and physical entities are detected using a sensor based approach. The Space repository in Gaia lets applications browse and retrieve software and hardware entities on the basis of specific attributes.

The Context File System (CFS) [13] aims to make personal data automatically available to applications when a user enters an active space through the creation of a virtual directory hierarchy where contexts are presented as directories and path components represent context types and values.

Overall, GAIA attempts to extend traditional middleware architecture to manage a ubiquitous computing space. As stated in [49] "Gaia's main contribution is not its individual services but rather the functionality it provides as the result of the interaction of these services."

However, the biggest flaw in Gaia lies on its distributed object based approach. Relying on CORBA technology and RPC, Gaia is not suitable for resource constrained devices. Second, service discovery in Gaia is based on the CORBA Trader service and a query is
constructed in the CORBA Trader Constraint Language which only supports a limited set of property types. Furthermore, these types address interoperability at the programming level (i.e. double or boolean) rather than at the semantic level where a StockQuote and a TemperatureReading may have the same output parameter type double.

Gaia uses ontologies to address several issues in the system, one of which is modelling context as first order predicate. However, the lookup service does not take account of the semantic relationships between context classes. For example, context class Location could be the same as context class Place where both represent an entity's position. A subclasses of Location can also provide the contextual information in a more precise manner.

Finally, all communications between Gaia entities takes place through CORBA. Although this is an effective solution for some pervasive environments like a meeting room, it does not scale well over the Internet. In particular, an address in IIOP is based on the server's port name and IP address in a way that is firewall unfriendly and harder to handle than a message-oriented protocol such as HTTP.

3.5 Task Computing

Fujitsu Laboratories of America and the MINDSWAP\(^1\) at the University of Maryland have defined Task Computing \[79\] as the computation needed to fill the gap between the tasks that users want to perform and the services that constitute available functionality. A conceptual Task Computing Framework (TCF) has been proposed in \[79\]. Embodiment of such framework is a Task Computing Environment (TCE, see Figure 3-8) which includes, at the least a Task Computing Client (TCC), a Semantically Described Service (SDS), a Semantic Service Discovery Mechanism (SSDM), and optionally a Service Control (SC).

More than 30 types of services such as "view on projector", "print" and "play(video)" have been deployed in a conference room. Each of these pervasive services has an UPnP device, a web server which provides the Semantic Service Description (SSD) that is

---

\(^1\) Maryland Information and Network Dynamics Lab Semantic Web Agents Project (MINDSWAP).
http://www.mindswap.org/
encoded in OWL-S language [90], and optionally a web service which offers the service exposed by SSD.

Figure 3-8 General architecture of Task Computing Environment.

A TCC presents an environment where user can create and execute tasks from combination of the available basic services. First, a client discovers the available services in where it is physically located using the discovery engine. Second, the client retrieves the SSD for each found service and feeds them into the inference engine (IE). Third, the client asks its IE to compute all possible compositions. Finally, results from IE are used by the client, which presents the composition page to the end user.

A Task Computing Client embodiment: Semantic Task Execution Editor (STEER) has been implemented to offers a web-based interface for presenting what user can do in the current context. For each UPnP devices found, STEER makes a specific UPnP call (getDescriptionURL(j)) to get the URL of its SSD and then retrieves the SSD. Once the execute button is clicked, the Execution Engine handles the execution of the composition and makes the UPnP and Web Service call. As shown in Figure 3-9, the possible service compositions are categorized based on their semantic inputs and outputs. The user chooses two services from the menu and executes the composition.
Service Controls such as the swirl on the upper right corner in Figure 3-9 is the "White Hole" where user can drag and drop objects such as files, URLs, contacts and schedules to create services that provide semantic versions of the objects dropped into it. On the bottom in Figure 3-9 is a service management GUI by PIPE, which enables user to adjust various service behaviours. Technical details about how these service controls work can be found at [100]. Utilities tool such as OntoLink has been developed to support generate SSD from WSDL. OntoLink also supports mapping between different ontology elements using XSLT. Such as a Name property of a Person concept defined in ontology A is the concatenation of FirstName and LastName properties that are defined in ontology B.

As evaluated in [79], implementation of the task computing environment crucially depends on Web ontologies written in the OWL language. The formal ontology allowed TCC to put services in the right place. However, it is the extra-ontological metadata such as comments and term labels embedded that give user confidence to make the final decision.
One of the main problems in Task Computing is that there is no context and adaptability support at all. Apart from the physical location where a Task Computing Client is situated, no other context is considered during the discovery phase. The discovery engine simply returns all services available with their SSDs, some of them are certainly irrelevant to the task. Furthermore, services not in the current user context may also offer functionalities that are essential to the task.

STEER incorporate UPnP discovery protocol, that is: SSDP [33], to search available services over the network. SSDP replies on multicast UDP which could cause channel congestion if numerous devices presence. As stated in [79], it is feasible to have every service announce itself to the client in a limited service environment. However, the heterogeneous pervasive environment requires more sophisticated discovery mechanism such as a distributed repository/broker based architecture. Furthermore, SSDP only supports service type matching whereas other discovery mechanisms such as UDDI do service matching at a syntactic level. Practically, neither one of the above will work solely in the pervasive environment.

3.6 Summary

In this chapter, five related research projects were briefly discussed with an emphasis on their architectural design patterns. In essence, each project attempted to build a software system for Pervasive Computing from a different perspective.

Oxygen attempted to abstract the computing system in three different components: E21s, H21s and N21s. Oxygen software architecture has three abstract concepts: Abstraction to influence component behaviour; Specification exports features of Abstraction to other components; and, Persistent object stores with transactional semantics which provides persistent storage for a specification. Oxygen depends on the aggregation of E21s and H21s which is far from widely acceptable by the community. Moreover, Oxygen assumed E21s and H21s are universal devices connected by network N21s. Although it is conceptually convenient, devices in heterogeneous Pervasive environment are typically supplied from various vendors. How an Oxygen application would distinguish, recognise, search and use these devices is still missing.
Aura attempted to minimize user distractions through explicitly specified user Tasks. The architecture consists of a task manager (Prism), context observer, environment manager and service suppliers. Interactions between Aura's components are carried out by connectors. Building on top of CORBA or RMI, Aura is limited to be deployed in a sophisticated environment such as office or airport. Connectors in Aura employ private protocols which make Aura components almost impossible to interact with other systems. Aura assumed a shared vocabulary is committed by a given service type. How this is enforced is still unknown.

One.World assumed a virtual machine on top of heterogeneous devices. Data and events are represented as Tuples. All communications are through asynchronous events and environments host running applications, save persistent data and facilitate composition through environmental nesting. The biggest limitation of One.World is that, tuples are implemented in Java classes, which posed considerable problems for its system services. One.World relies on object serialisation and reflection which are not well supported in mobile clients. The private asynchronous communications made One.World harder to integrate with the rest of the world. In short, One.World programming everything, but the interoperability of the system is moderated.

Gaia attempted to extend the traditional middleware solution to a Pervasive Computing environment. Gaia encompassed a component management core, space repository, event manager, context file system, presence service and context service. Entities are retrieved based on a set of attributes. However, queries in the CORBA Trader service only support a limited set of property types at the programming level. Ontology is used for describing various concepts in the Gaia environment. In particular, each type of context is mapped to a class (concept) in the context ontology but the relationships between concepts are not explicated. The underlying technology of Gaia is based on CORBA and RPC which makes it difficult to deploy in a resource constrained environment. Although CORBA’s success is undeniable, it’s tight coupling and connection oriented features analysed in [1] are inadequate for the dynamic nature of human activity in Pervasive Computing.

Task Computing defined a task as one or more semantic service compositions. The approach is to expose functionalities as Semantic Web services and advertise them via UPnP so that, mobile clients can discovery and present all possible compositions to the
end user. The main problem in Task Computing is that, there is no context and adaptation support at all. Service Discovery simply returns all services reside in the current physical environment where UPnP can reach. In some situations, Task Computing can hardly benefit from crucial services such as banking which are not located in the current context.

To meet those weaknesses in this chapter we need to provide an environment which delivers task supports with respect to the dynamic nature of human activity. Works in this chapter have demonstrated the idea of service abstraction from a different perspective. Both Context providers in Gaia and Service Suppliers in Aura indicated the usage of service can greatly simplified the interaction between heterogeneous entities. Meanwhile, Semantic Service in Task Computing provided an example of ontology based service annotation. System services in One.World have shown that a layered system is well suited for developing applications in a Pervasive environment. On the other hand, Abstraction and Specification in Oxygen was the first step toward platform, protocol independent service description. However, none of them explicitly used service as the fundamental element in constructing a Pervasive Computing system. Leveraging from the DBE project, we will demonstrate our service oriented architecture for Pervasive Computing environment in the next two chapters.
Chapter 4

4 Proposed service oriented architecture

4.1 Introduction

In this chapter, we will introduce our proposed service oriented software architecture for the Pervasive Computing environment. It is designed to be used in an inter-environment situation where devices and resources are highly unpredictable. The architecture is also flexible enough to facilitate person to person communication through a natural transition.

It is anticipated that in a Pervasive Computing environment, even an ordinary person will be surrounded by a large number of networked devices, which may not appear to be distinguishable. Without a unified mechanism, location, interaction and access to these heterogeneous devices become impossible. We envision that by transforming physical devices into software services, applications which carry out tasks on behalf of the users can easily access the required resources (e.g. various devices) through these concrete services without any prior knowledge about the resource themselves as long as both ends commit on the same semantic.

Software service is one of the fundamental elements that enable dynamic interactions among applications, devices and the environment. The hypothesis behind this is that, by using software services, better interoperability can be achieved. The idea is to create a platform- and protocol-independent service description for each service implementation and publish it to the service repository. Functional and non-functional properties are also published as part of the service context to the repository. The latter allows a service provider and/or service monitoring process to dynamically update these properties at runtime without altering the service itself. Properties are also used for service discovery purposes.

Because of the dynamic nature, it is unlikely that there will be a common agreement on a programming language or one set of semantic definitions for shielding up of all elements
in a Pervasive Computing environment. To tackle this problem, we prepare to encode knowledge of each service using explicit references to one or more ontologies it commits to. Particularly, when a service is being published, the associated ontologies are also explicitly published. To discover a service, the service consumer must specify the ontology it refers to in constructing the query. This is one of the key elements to enable our ontology based service recommendation. As a consequent, a service consumer will be able to "get" and "talk" to the desired service provider even if their implementations are independently designed. In other words, both ends share the same knowledge of a particular domain.

Ontology provides a formal representation of vocabularies (i.e. classes and properties) and their associated semantics (the relations between different classes and properties). It is reasonable to assume that ontologies for devices, services and contexts in the Pervasive Computing environment will be gradually maintained by their vendors and organisations, which may also create an oligopoly of ontologies by forming consortia. However, we will develop and later use our own ontologies (and tailored ontologies from others) throughout the experiments while the above assumptions remain.

Context plays a crucial role in the future of Pervasive Computing systems. As discussed in chapter 2, a Pervasive Computing environment is a highly dynamic space where people, devices and resources may come and go at anytime. Software that assists users in completing their tasks must be able to adapt to changes in context in a natural way. To provide effective context awareness support, we intend to build a set of context services that will offer some high level contextual information. A context service is similar to an ordinary software service which can be published to the service repository. As already mentioned in the previous paragraphs, ontologies will be used to ensure a common understanding of concepts and vocabularies at both ends. In our architecture, each type of context service is mapped to a service ontology and each type of context is mapped to a context ontology. Relationships between concepts are also specified.

Clearly, reliance on the request-response message exchange pattern is not efficient for service consumers who only need to adapt their behaviours when certain states are reached. We intend to develop an event notification mechanism which allows service consumers to subscribe to a number of event providers using the specified rules. As with
a service provider, an event provider publishes its service description to the service repository. Once an event provider has been located, a subscription is sent to the event pool which periodically extracts information from all subscribed providers. When satisfaction of the specified rules is detected, a message is sent back to the subscriber.

A service repository maintains all the published services in which it resides. It also maintains a knowledge map that specifies the relationships between concepts that are referred to by any published service. The map is dynamically updated once a concept or relationship is changed. The event pool is embedded into the service repository since an event provider is generally perceived as a service provider.

Pervasive computing often implies that people are surrounded by many devices that are providing similar or even identical services. To discover, match and select the most appropriate one, we propose a matching algorithm based on the ontology of service types, inputs and outputs, as well as contextual information. By using a weighted list of requirements, our matching algorithm will calculate the degrees of match and provide a prioritised services list.

The rest of the chapter is organised as follows. Section 4.2 presents a high level layered structure of our proposed architecture, followed by an architectural view of the proposed system in Section 4.3. Each part of the architecture is then discussed in the following sections. In particular, we proposed a semantic service matching algorithm in Section 4.7. Finally, a summary is given in Section 4.11.
4.2 A Layered Structure

Figure 4-1 shows a layered view of our proposed architecture. The bottom layer is a Pervasive Computing environment that embraces devices, services and applications. Unlike One.World [74], a Pervasive computing environment does not assume a common execution environment across all devices, services and applications. For the development of service oriented architecture, we assume that the entire environment is fully covered by wired and/or wireless network, which provides the communication infrastructure for other entities in a Pervasive Environment.

The device layer defines a placeholder for various devices contained in an environment. Devices may be manufactured from different vendors which can use whatever the technologies they wish to. A device can be any kind of physical entity and it does not have to be accessible over the network. However, the service provided by a device should be published to the service repository so that, it is discoverable by the service consumer. To focus on constructing an integrated SOA for Pervasive Computing, we assume that devices in a Pervasive environment are connected to the underlying network and are subject to providing one or more services that can be accessed through an interface.

On top of the device layer, service is the fundamental element for building Pervasive applications. A service represents an autonomous, platform-independent computational entity (that is coarser than a component) which must be implemented by one or more service providers. A service implementation encapsulates proprietary device interaction protocol and only exposes a standard interface over the network. We assume that the
services of each device will be developed by its vendor, in accordance with the proposed architectural style. For this context, vendors need not concern how the service is invoked. Instead, the vendor must specify a service description, and the service should be published once the device becomes available. We also assume that each device has the capability to execute the service implementation. If not, a third party host who has such capability should be provided.

The application layer consists of several Pervasive applications whose aims are to assist users in complete tasks. In contrast to Aura [22] and Task Computing [79] where a task is represented as a coalition of services or a set of semantic service composition, we assume that a task is captured by the application that a user initiated. For example, a task "edit documents" is captured by starting a word processing application. The task also can be captured by the actions performed inside an application. For example: a click on the "print..." button indicates a "print" task is raised. Therefore, the required resources to perform a task are well-known to the application developer in an abstract level.

4.3 System Architecture

The layered view serves as the principle guideline for abstracting a Pervasive Computing environment. However, Figure 4-1 did not reflect any concrete component in our system architecture, nor did it reflect the interactions between these components. To enable the interactions in the service oriented architecture, we need an architectural framework which specifies not only the required features and interactions, but also the role of each entity in the Pervasive Computing system.

Figure 4-2 shows our proposed system architecture in detail. The architecture consists of various components. In the Pervasive Environment, Devices layer is a collection of physical entities that embedded into our living environment. Services represent the capabilities offered by each device and export them as standard interfaces which are accessible through a generic protocol. Service Repository embodies the reception to a Pervasive Environment. At the user's side, a user is shielded up to a number of applications. User's tasks are captured by the applications he or she initiated. A Personal Repository contains enough information to the application demands. From a logical perspective, an environment may have more than one repository for specific purposes.
The boundaries of a User Environment may embrace a number of Pervasive Environments for task execution. In a person to person scenario, each User Environment becomes a Pervasive Environment through a natural transition by which applications embody the services and personal repository become Service Repository.

![Figure 4-2 Proposed system architecture for Pervasive Computing.](image)

### 4.4 Users, Tasks and Applications

Pervasive computing aims to support a user in the completion of their tasks whenever it is needed. Typical example tasks include but are not limited to: making a presentation, editing a file, printing a document and synchronising a contact list etc. To carry out these tasks, applications should be able to gather the appropriate services/resources on behalf of a user and consequently, shift user focus to the task itself. A task can be executed in a device, in a single environment or across several environments which follows the user movements. Because of the dynamic nature, applications should also take advantages of the newly available resources to best satisfy the user requirements.

To narrow down, we **assume** that applications for different tasks are initiated by their users. Much like the situation today where an application is developed for a specific purpose, an application in a Pervasive Computing environment is developed to complete a specific task. Therefore, it is anticipated that the required resources to perform a task are well-known to the application developer at the semantic level. However, no assumptions can be made on any specific device that an application will interact with during runtime.
Upon executing a task, the application searches for the available services in the current environment through sending one or more queries to the Service Repository. Since the required resources are well-known at the semantic level, a query can be constructed by specifying a number of semantic criteria. The repository matches all published services against the criteria and then returns a service list. Depending on the execution modes, the application dynamically constructs a message, binds it to the transport and sends it to the best matched service provider.

As we described in Section 2.2.3, from a user's perspective, devices are becoming irrelevant to the task itself as long as a service fulfil its promises. In our architecture, applications separate users from the devices and the use of services greatly simplified the access to heterogeneous devices in each environment.

4.5 Devices

Devices are the basic physical entities in a Pervasive Environment. In our architecture, devices are not limited to servers, desktops, laptops or hand-held computing devices, the concept of device may also embrace plasma screens, microwaves, varies type of sensors or even cars, essentially every physical object is a kind of device. These devices are often embedded or integrated to the physical environment where they reside, and provide some degree of functionalities to be used.

To focus on system applicability, we assume that each networked device satisfies the following two conditions:

- Each device has networked controllable interfaces that can be accessed by software;
- The capabilities of each device can be mapped to one ore more services.

Upon entering an environment, each device should be connected to the underlying network and expose its functionalities to the public unless otherwise specified. The exact mechanism for device configuring is an implementation issue. In one environment, this can be manual configuration, while in another environment; it may be UPnP, Jini or both.
4.6 Services

In our architecture, a service represents a computational entity and is the fundamental building block in constructing the service oriented architecture for a Pervasive Environment. The complexity and extent of a service implementation may vary from one entity to another. In other words, the service granularity is unconstrained. However, in a Pervasive Environment, a service typically represents some functionality that may be provided by one or more devices. For example, a printing service is provided by one or more printers. An environment may have many devices that provide numerous services to the users: the more it has, the richer the support that is offered.

A service exports its functionality to a conceptual service definition which is interpreted by service providers that intend to implement such service. The service implementation has a service description and a number of properties that characterise the service. For example, a bubble jet printer may not offer the same quality and speed of the printing service as a laser jet printer. Therefore, the concrete service descriptions must preserve the differences between each service implementation. In our architecture, this is achieved by using an XML-based service description similar to WSDL.

A service implementation has a number of functional/non-functional properties which should be specified by the service provider. Any other domain specific properties can be
added as long as they are understandable by the service consumer. However, an arbitrary service property description does not restrict the vocabulary of tags corresponding to the information relevant for the service. In our architecture, this is tackled by encoding knowledge of each property into the description by using explicit references to the relevant ontological definitions. Figure 4-4 shows an example property "speed" of the Printing Service.

In Pervasive Environment, the technical details of a service are only exposed through the standard service interface which forms part of the service description. An application (or service consumer) inspects the retrieved service descriptions at runtime, bind to a particular service provider and execute the service by sending a sequence of service requests over a transport to the service implementation. Figure 4-3 illustrated the relationships between different service concepts.

Due to the heterogeneous Pervasive Environment, a service may be implemented by various vendors which are unlikely to observe the same rule. In a component based software system, this is addressed by employing a number of connectors; each of them implements a specific low-level communication mechanism. As the number of devices grows, it is unrealistic to expect that interactions with potentially unknown devices can be abstracted by connectors. In our architecture, interactions are enforced to use a platform, transport independent message protocol such as SOAP. In particular, both messages and operations are described in abstract formats and assigned explicit references to the ontologies they refer to. Once they are inspected by a service consumer, these abstract components can be binded to any transport type which is prescribed in the binding information embedded in the service descriptions.

4.6.1 Service Description

In our architecture, a service implementation is described by using a platform independent XML-based description. However, an XML service description is meaningless if the service consumer does not know its semantics. For example, a WSDL document only provides formal descriptions of a service in programming level, eg. both temperature and stockquote are expressed in terms of a double type. If any application and service are going to interact sensibly in a Pervasive Environment, they must agree not only on the
mechanics of the interaction, eg. message format, data type and communication protocol, but they must also agree on meaning and purpose of the interaction. In other word, they share a common knowledge of the service.

In our day-to-day life, interactions often occur under implied agreements on the semantics and contexts that are considered to be relevant to the interactions themselves. For example, conversation of a specific topic between two persons implies an agreement on the meaning of that topic is observed. Although we humans are generally flexible enough to cope with the situation, machines are far from as simple as humans in dealing with ambiguity.

![Figure 4-4 An example of the proposed service description.](image)

To overcome the problem, we intend to build a set of ontological definitions that are explicitly referenced by the service description at the various sections. As doing so, a service description is grounded with the knowledge that is shared among all entities involved in an interaction. Figure 4-4 shows a segment of the proposed service description. It must be noticed that the namespace and other required components are omitted for space saving purpose.

### 4.6.2 Service Publication

A service is useless if it is not discoverable by other entities in an environment. In our proposed architecture, each device has one or more services that were developed by the vendors. Upon arrival, we assume the services are published to the service repository by
any mean. This may be done by manually configuration, self-publishing or third party elements.

The service repository periodically announces its presence to the environment in which it resides and exposes a set of standard interfaces for service manipulation. The announcement is sent to any communication channel that is attached to the repository which may be helpful in assembling a greater amount of devices. Once a service is published, a unique ID is returned for future reference.

4.6.3 Service Invocation

A service is described by a platform-independent service description which characterise a service in terms of abstract message and operation, and concrete binding. To invoke a service, an application needs to send a query to the service repository so that a collection candidate services is returned. Either user or application must select one service from the collection and inspect its service description. After that, one or more messages are constructed according to the format in a service description, bind the messages over a transport and send them to the service implementation. The interaction pattern is exactly the same as the one that is defined in a generic SOA.

4.6.4 Service Update

Services and devices are subjected to consistently changes in a Pervasive Computing Environment. Any sophisticated service implementation should reflect these changes by explicitly update the service properties or even the service description itself if necessary. The service repository has a set of standard interfaces for service updating and the update process also can be triggered by explicitly queries from the service repository such as the leasing mechanism. The unique ID that was given during service publication is required to update the same service along with other security concerns.

4.6.5 Service Deletion

Devices and Services may become unavailable at anytime without notification. Both active deletion and passive timeout policies should be implemented by the service repository so that a service provider can remove a service which is no longer available.
Meanwhile, non-responding services are wiped to increase the precision of the service discovery and save the resources.

4.7 Services Discovery

To utilise resources of an unvisited environment in delivering user supports, services and applications should be discovered on the fly. From an application point of view, the discovery process can be abstracted in three steps. First, an application queries the service repository for services (find). Second, an application or a user selects a service to use from the available services (select) and third, an application invokes the service (use). In a typical Pervasive Computing scenario, thousand of services may appear in an environment which makes syntactic-based service discovery very difficult to pursue a high precision. Besides, services may use different terms or types to represent the same meaning which is unlikely to be matched by a syntactical approach.

In our architecture, the required services to perform a task are well-known in advance to the application developer at the conceptual level since a task is intended to be captured by the application, and the service description is annotated with the ontological definition. Inspired from the standard tree-based matching technique in [46] [40], we propose an ontology-based multi-attributes service matching algorithm. The design philosophy behind the algorithm is to create a prioritised service list for corrective action recommendation. The matching algorithm is introduced as follow. Before starts, we assume that each discovery request is annotated with the ontological definitions.

To find a service, an application sends a find message to the service repository. We denote this message by a functional definition:

$$\text{Results } R = \text{find}(O_s, O_{in}, O_{out}, [\text{props}])$$

Where $O_s$, $O_{in}$, $O_{out}$ are Service Ontology, Input Ontology and Output Ontology respectively. $[\text{props}]$ is a list of properties preferred. Each property is grounded with an ontological definition although this is not compulsory. Where it is not grounded, the matching algorithm tries to match it at the syntactical level which may increase or decrease the precision. Implementation of the algorithm provides no warranty on this feature.
The service repository receive the query \textit{find}, starts the matching process from:

1. Obtain an ontological hierarchy $HO_s$ for concept $O_s$. The hierarchy includes superclasses, subclasses and related classes of the concept that is specified by $O_s$.

2. For each class $Hc$ in $HO_s$, the semantic similarity between $Hc$ and $O_s$ is calculated where the similarity is the ontological distance between each $Hc$ and $O_s$. The distance are further described in the following cases:

   1) If $Hc$ is identical or equivalentClass to $O_s$, the distance is 0, in other word, perfect matched in concept. The equivalentClasses of $O_s$ are denoted by $O_s'$ onward.

   2) If $Hc$ is subclass of $O_s$, or $O_s'$, the distance is $X$ where $X = N * w(N)$. $N$ is the number of paths away from $O_s$ or $O_s'$ in $HO_s$, $w()$ is a weight function whose purpose is undefined at the moment. For simplicity, we will assume for the present that $w(N) = 1$

   3) If $Hc$ is superclass of $O_s$, or $O_s'$, or is subclass of a superclass of $O_s$, or $O_s'$, the distance is $Y$ where $Y = f(N')$. $N'$ is the number of paths away from $O_s$ or $O_s'$ in $HO_s$ and $f()$ is a non-linear function which is subjected to modification from each implementation. In general, we believe, $Y \geq 2X$ when $N' = N$.

4) Repeat the previous three steps for $O_{in}$, $O_{out}$.

5) At this moment, three similarity annotated ontological hierarchies have been obtained. They are: $HO_{i}'$, $HO_{in}'$, and $HO_{out}'$.

3. The matching process consults the service pool, finds services whose types are referred to any concept in $HO_{i}'$, the intermediate result is $S'$. A semantic filter is then applied to $S'$. For each service in $S'$, $O_{in}$ and $O_{out}$ are validated against the $HO_{in}'$, and $HO_{out}'$. Any service with a non-existence $O_{in}$ or $O_{out}$ will be omitted. The similarity of $O_s$, $O_{in}$ and $O_{out}$ for each service in $S'$ are then normalised by a weighted function:

$$R' = \text{sim}O_s * w1() + \text{sim}O_{in} * w2() + \text{sim}O_{out} * w3()$$

Where $w1()$, $w2()$, $w3()$ are the weight functions and $\text{sim}O_s$, $\text{sim}O_{in}$, $\text{sim}O_{out}$ are the similarities of each remaining service against the request in $S'$. 65
4. Similar approach applies for each property whenever it is semantically annotated. Finally, the results are normalised.

The ontology based service discovery allows a more flexible process which finds not only syntactically matched service, but also those conceptually similar services even if they were not in mind from a developer's perspective.

4.8 Context

In Pervasive Computing, context can be defined as information that is relevant to the task execution. This information can include locations, services, tasks that are being executed, user intentions. It must also include information that characterise the physical environment where the task execution is located (e.g. lighting, temperature, noise levels, movement) since this may change the way a task is executed.

As one can see, the types of context are essentially unlimited and contexts may be characterised in different formats. The problem is further complicated by the growing demand of devices in the Pervasive Environment since the same context may be provided by a wide range of devices. Contexts in such environment are increasingly difficult to be extracted and modelled.

In our architecture, we attempt to model each type of context as an abstract context service, while different implementations of a particular service are anticipated. The resulting structure is a homogeneous framework with integrated context service support. Practically, context service becomes the fundamental building block in constructing a Pervasive Computing environment.

As we have introduced in section 4.6.1, ontologies are built for each type of context service presents and each type of context offered is also modelled by the ontology. Context service provider shares the same role of a generic service provider which should publish its service description to the service repository. Same as the others, the service description of a context service implementation must be annotated by explicit referencing to the context ontologies. Both SOUPA[36], CONON[99] have built various ontologies for modelling context information, leveraging from existing works thus enable us to concentrate on other system issues.
In our architecture, context acquisition is no harder than a generic service access. An application consults the service repository for available context services, once a prioritised service list is returned, the application simply selects one and engages the service. The application specific context-aware adaptation behaviours are not a concern of the architecture. We aim to support a limited range of context types and it is foreseeable that adding new types of context is a fairly straightforward process.

4.9 Repository

The idea behind a Service Repository is similar to a reception desk that is found in almost every place in the real world. Consider an example where people go to a large organisation for a business service; the service supplier is unknown until one has talked with the receptionist. In a Pervasive Environment, applications need to know the environment before they can interact with the virtual or physical entities present.

We propose to build two distinct repositories: the Service Repository for a Pervasive Environment and the Personal Repository for a person.

4.9.1 Service Repository

To carry out the tasks, an application needs to know the services present in a Pervasive Environment. A Service Repository embodies the reception to the environment: it stores information about all services and their properties in the environment and let applications search for the desired services on the basis of a flexible query. The number of repositories is unlimited in a Pervasive Environment; however, they must periodically broadcast their existence over the network so that both devices and application is aware of.

Insides a Service Repository, there are three main components: a service pool that stores service descriptions for each published service, a knowledge map which specifies the referred concepts from all published services and relationships among these concepts, and an event pool that periodically extracts information from all subscribed providers and regenerate events to the subscriber.

A service pool is a container for the service descriptions that were published by the service providers. Each published service description and its service provider is assigned
a unique ID for later reference. Function and non-functional properties of each service are also saved for later use in the service discovery phase. A unique ID must be assigned to each set of properties so that, mutual reference between service description, service provider and properties are preserved. The service pool implements the service matching algorithm and exposes a set of standard interface for service manipulation.

A knowledge map saves the concepts and relationships among them in an inter-connected format which is naturally inferred by using the ontology reasoner, for example, Jena\(^1\). Once a concept or a relationship has been altered, the map must be updated to reflect this change. Implementation of a knowledge map should provide a set of interface for querying the concepts and relationships.

An event pool serves as an event broker for subscribers and event providers. An event provider is simply a service provider whose service is an event source. An application searches for a required event source by sending a query whose format is exactly the same as a query for service discovery. Once an event source is located, a subscription is sent to the event pool with the event provider’s reference and a number of rules for event generation. The pool periodically pulls information from all subscribed providers and save it in the database system (for static information) or as key-value pair (for dynamic information). Upon satisfaction of the specified rules are detected, a message is sent back to the subscriber along with the actual event itself.

A Service Repository is self-published as a service that has four set of interfaces to different entities in the environment. A service is published through the service manipulation interface that is also used for service updating and removal. A service consumer finds available services through the service discovery interface. The knowledge

discovery interface is exported to find the concepts and relationships among the published services. And the management interface is provided for administration purpose.

### 4.9.2 Personal Repository

A Personal Repository provides information about the person and keeps the historical records of the visited environment for future reference. In our daily life, a user is often associated with a few "Home" environments where services are always available to the user wherever he or she is located. These services are changed less frequently than others in a Pervasive Environment and are kept in the Personal Repository for consumptions. In contrast to the Service Repository, a user must explicitly notify the existence of a personal repository to the application since a personal repository does not broadcast its presence unless necessary.

![Diagram of Personal Repository](image)

Figure 4-6 Structure of the Personal Repository.

Recognise movement as natural, users in a Pervasive Computing environment are free to move without boundaries. The architecture must define a persistent data storage for saving user's and application's data and making these data available in different environments. On the other hand, personal information is treated as extremely sensitive data which should not be revealed to others without proper security constrains. As a consequence, leveraging the arbitrary storage service from a Pervasive Environment is potentially vulnerable to the attackers and the reliability is not guaranteed. In our proposed architecture, a personal storage service is published to the Personal Repository for authorised entities. The exact implementation of such storage service is a personal choice, for example: SFTP, SMB or web service, but it must conform to the security standards. Figure 4-6 shows the basic structure of the Personal Repository.

One of the main reasons for building a Personal Repository is the support for inter-person collaboration. In a person to person scenario where both ends need to share information
on an ad-hoc basis, each Personal Repository can be naturally converted to a Service Repository by which applications embody the services and presence of the Personal Repository is broadcasted.

### 4.10 Common Ontology

The use of Ontologies has spread widely in many areas such as knowledge and data management or the semantic web. Humans interact with different entities based on the assumption that they know exactly what various entities do and they understand the relationships between those entities. For example, to print a document, one needs to know exactly what a printer does, and the relationships between a printer and its paper feeder. As one can see, the essence of such an assumption is an accurate conceptual model of the entities in the environment.

In a Pervasive Environment, devices and services may enter the environment at any time; these new entities must have a common understanding of the various terms and concepts used in the current environment so that, no misunderstanding occurs during an interaction.

In our architecture, ontologies are proposed to model a number of key concepts in the environment. First, each service type is assigned with a service concept that is represented by an ontological class. Second, messages and operations used in an interaction are mapped to message and operations classes respectively. Third, contexts are mapped to context classes. Fourth other entities such as Person, date, events and devices may be mapped to the corresponding concepts if that is required. Relationships among these service concepts are drawn so that they are semantically annotated.

We envision that ontologies will be gradually provided by device/service vendors or standard organisations which are likely to provide a common upper ontology for domain specific concepts much like the industrial standards or RFCs are being recommended by today. However, we would reuse as much the available ontology as possible today such as OWL-S [93], CorBA[35] and SOUPA [36], so that we can focus on our system experiments. Figure 4-7 illustrated a visual annotation of the SOUPA:person ontology.
4.11 Summary

In this chapter, we proposed a service oriented software architecture for Pervasive Computing environment where the precise devices and services needed to perform tasks are highly unpredictable. The architecture is flexible enough to be converted into an inter-person scenario where applications for each user become the services offered by a person.

First, we made a few assumptions on the basic system level to ensure our focus is not shifted away from the integrated service oriented architecture for Pervasive Computing. Based on these assumptions, we presented a layered structure that intends to split the system into four coherent layers.

However, the layered structure does not provide the necessary system components for discussion. We then presented a concrete system architecture for Pervasive Computing. The system is divided into two loosely-coupled environments: a user environment and a Pervasive environment. Interactions between these environments are through platform-form and transport independent messaging protocol. Taken from a SOA, we applied the publish-find-bind interaction pattern throughout the architecture. We also use service to hide complexity behind each environment. Particularly, devices, context sources and events sources are converted to one or more software services that expose the functionalities only through a semantic annotated XML based service description.
After that, various components in our proposed architecture were discussed with a strong emphasise on the interoperability issues that is faced by every entity in Pervasive Computing. In the heart of a successful SOA, we proposed an ontology-based service matching algorithm by considering multiple service properties. The underlying design philosophy is to provide a corrective action recommendation to the applications and their users.

Before a summary is drawn, we also introduced the role of ontology in our system architecture and envisioned that, ontology for various entities in our system architecture will be gradually provided by their vendors and organisations. We also concluded that maximised ontology reuse helps narrow down the focus to the future system experiments.
Chapter 5

5 Implementation

In this chapter, we present a Java-based system implementation of our proposed architecture. The system is based on the DBE\(^1\) project which is supported by the European Commission's 6th Framework Programme for research and development in Information Society Technologies.

The implementation is divided into three coherent components: Core Service, Execution Environment (ExE) and Service Factory (SF). The Core Service is a set of infrastructural DBE services that provides the operational support at the lowest level for the ExE. A DBE service is an atomic service unit in the ExE to which the service can be published. Consequently, the Core Service is an integrated component to the ExE. The Execution Environment is a distributed virtual machine that allows services to be published, discovered and consumed over the peer to peer FADA network. The Service Factory is a set of tools that collectively supports DBE service development. It consists of models editors, publication wizards, tools, java development environment and a repository for publishing and retrieving service models and data. Some of these components themselves are implemented as DBE services which can be consumed from the ExE.

The implementation is guided by two principles. First, there should be separation of concerns on both server and client side. Particularly, the service implementation is not limited to reside in the ExE. A service provider may choose to publish only the service proxy which knows its own communication protocol, to the DBE ExE. On the other hand, the ExE has its own proxy which can be obtained by the DBE service consumer as required. Second, there should not be single point failure inside the ExE. DBE employed a decentralized peer-to-peer network of nodes called: FADA (Federated Advanced Directory Architecture) which provides a leased data repository that can purge itself of

---

\(^1\) The Digital Business Ecosystem (DBE) is an Internet-based software environment in which business applications can be developed and used.
stale data. Inside the FADA network, any registered data can be searched starting from any interconnected node.

To isolate programmers from unnecessary P2P network coding complexity, the Execution Environment offers a transparent proxy toolkit to which applications are coded as Plain Old Java Object without any knowledge of distributed programming. The proxy toolkit provides a programmatic interface to search and invoke the service registered.

The Execution Environment is mostly written in Java which provides maximised portability although a few components do require platform dependent library support. The Execution Environment has been released as an open source project \textit{SWALLOW}\textsuperscript{1} and is currently at version 0.2.0. The installation package is 46,471,262 bytes (approximately 45 MB) including CoreService, Servent, FADA and XDB persistent storage.

The Service Factory is released as an open source project \textit{DBEStudio}\textsuperscript{2} and is currently at version 0.2.0. Built on the eclipse platform, the DBE Studio is a collection of tools developed as Eclipse Plug-Ins aim to facilitate the DBE service analysis, define, develop and deploy onto the DBE Execution Environment. The installation package is 28,350,193 bytes (approximately 27MB) including plug-ins and their library supports. To provide graphical model editor, the DBE studio requires \textit{GEF 3.1.x}, \textit{EMF 2.1.x} and \textit{Eclipse 3.1.x} support. The model compilers currently work with \textit{Java2 SDK 1.4.x} whilst migrating to Java 5.0 is underway.

\textsuperscript{1} \url{http://swallow.sourceforge.net}
\textsuperscript{2} \url{http://dbestudio.sourceforge.net}

74
5.1 The Service and Models

Comparable to functions in conventional software system, a service is an autonomous, platform-independent computational entity that is deployed to the Execution Environment through the service publishing interface. In the context of DBE, a service is characterised by two correlated models: the business model and the technical model that describe a service from both business and technical perspective.

A model is "description of (part of) a system written in a well-defined language. A well-defined language is a language with well-defined form (syntax) and meaning (semantics), which is suitable for automated interpretation by a computer"[2]. Essentially, a model defines what elements can exist in a system while a language defines what elements can be used in a model. The language itself can be described by the meta-model that is also written in a well-defined language.

To effectively describe a service, the DBE service modelling architecture redraw from [84] in Figure 5-1 is divided into four layers. The highest abstraction is OMG's MOF framework that consists of the description of the structure and semantics of metamodels. Lower than MOF is the BML/SDL Meta-model framework for BML/SDL models that can be reused for other services and the BML/SDL Instance/Data that describes the features of a particular service.
Recognise the dynamic nature of human activity, the Model Driven DBE service Architecture makes service interoperations easier among different platforms. Given a consistent metamodel, new services can be derived from existing models which were mutually agreed by industrial partners. Furthermore, the construction of a semantic discovery engine for which services and models can be searched is assured by the model's consistency.

5.1.1 The Business Model

A Business Model has been defined as "viewpoint on the business and its environment that focuses on the scope and goals of the business, and the terminology, resources, facts, roles, policies, rules, processes, organizations, locations, and events of concern to the business. The Business Model is a model of the business in its environment, by the business, in the language of business people, for business (not necessarily IT) purposes" [85]. A business Model may also include parts of the business that are not subject to system automation but that are interested by those who design and build the IT system. As a consequence, the role of business model in describing a pervasive service is to define the business parameters and requirements that are concerned by the service provider. The contents (BML data) of a particular business model are an instance consideration which will be specified at runtime.

![Figure 5-2 BML modelling.](image)

As redraw from [95] in Figure 5-2, the Business Model of the service (BML model) is written in Business Modelling Language. The language is defined by the BML metamodel which is written in meta-language that refers to the OMG's MOF [68]
architecture. The abstract and concrete syntax of the BML model is out of the scope of this thesis. It is currently developed and maintained by OMG and the DBE’s BML team.

5.1.2 The Technical Model

While the business model addresses the business concerns, the semantic service model describes the service from the candidate service consumer’s perspective. As part of the DBE modelling framework, the semantic service model is written in the Semantic Service Language (SSL) that is part of the BML and is used to define semantic service description models for capturing the semantics of specific types of business offerings [95]. In other words, it is the technical part of the Business model. Later in the service development, a service developer will be able generate the technical model of the service from an existing BML model, particularly, the semantic service model.

![Figure 5-3 SDL modelling.](image)

Similar to the BML model, the Technical Model of the service (SDL model) is written in Service Description Language (SDL) that is defined by the SDL metamodel. The SDL Model defines the technical aspects of the services, i.e. service interface and the sequences of messages. Furthermore, SDL adopts ontologies in defining message semantics. Using MDA tools and principles, technology or protocol specific interface
(SOAP, WSDL, XMI or Java) code can be automatically generated from the platform independent SDL model. Figure 5-3 demonstrated the SDL concepts and relationships to other DBE components.

The syntax and concrete specification of the SDL model is out of the scope of this thesis. However, an XML schema of the SDL metamodel (SDL schema in short) has been provided to validate the SDL Models by the DBE team. The SDL schema is also used by the SDL editor in the Service Factory.

5.1.3 The Ontology Definition Model

The use of ontologies as an important mechanism for knowledge sharing and semantic interoperability was identified from the very beginning of the DBE project. Business Model Ontologies and Service Ontologies are recognised as the mechanisms that will capture the semantics of the businesses and their services accordingly [95]. While ontology based approach is desirable for the purpose of semantic interoperability, service providers are allowed to describe the products (services) in their own way, for instance: define their own ontology models, make references to the domain ontology and/or adopt multiple ontologies.

![Figure 5-4 ODM modelling.](image)
The Ontology Definition Model (ODM) has been developed in DBE and is thereby used for the representation of domain specific ontologies throughout the implementation. Entities defined by ODM can be referenced by elements from both BML model and SDL model. Taken from [95], Figure 5-4 shows ODM in the context of OMG's MOF framework and its relations to W3C's Semantic Web Architecture.

5.1.4 The Service Manifest Model

The service exposes a programmatic interface whose sole purpose is to make functionality available externally, hiding the technical details of the underlying implementation from the service consumers. The implementation of the interface is located on a network-accessible platform within an independent service container. To ensure a service is made discoverable, the service description, which provides details necessary to interact with the service, is published to the Execution Environment. This description, like the BML/SDL data, is defined by the service manifest model.

A service manifest is a computationally independent service container, unlike the web service architecture where only technical interface is published, the service manifest embedded models as well as data. As illustrated in Figure 5-5 (picture adopted from [84]), the service manifest mainly contains the BML model, BML data, SDL model and the reference to a service implementation. Similar to the SDL, service manifest model is written in XML-like syntax. The structure of the service manifest is provided by the DBE team who has also supply the service manifest schema.
Written in a meta-language that refers to MOF, all metamodels can be naturally streamed into XMI documents for data exchange. The same principle applied to the model instances which are defined by the metamodel and instance level data representation.

5.2 The Core Service

The Core Service consist the minimum required services to bootstrap the DBE Execution Environment. As one of the central module, every core service must implement the CoreAdapter interface that will grant access to the entire ExE.

```java
public interface CoreAdapter extends Adapter {
    void init(ServentContext context);
}
```

Figure 5-6 CoreAdapter code snippets.

A fundamental service is the Knowledge Base service (KBServe), which is used to store information among DBE services while maintaining independency. The service consumer can send queries via the specific KBServe interface to search and manage the service information stored.

```java
public interface KBI {
    public String getServiceID();
    public boolean isAlive();
    // Session Management
    public String createNewSession();
    public void closeSession(String sessionID);
    // Model Management
    public void storeModel(String id, String data);
    public String retrieveModel(String id);
    public void deleteModel(String id);
    public void listModels();
    // Query interface
    public Collection submitQuery(String session, String query);
    public Collection searchByKeywords(String sessionID,
                                        String metamodelName, Collection keywords);
    public void storeUsageData(String id, String data);
    public String getUsageData(String id);
    public java.util.Collection queryUsageData(String xQuery);
    public void deleteUsageData(String id);
}
```

Figure 5-7 Knowledge Base service interface code snippets.
Shown in Figure 5-8 (taken from [96]), implementation of the KBService is built on top of the underlying DBE knowledge base peer infrastructure that is able to fully support all DBE metamodels and provides basic functionality for processing, storing and retrieving models following the above metamodels. The components that comprise the basic KB peer infrastructure are the KB manager, the Model Manager, the Recommender, the Usage Manager, the Knowledge Access Module, the MDR manager and the DB manager which are also shared by other DBE Core Service. Details of these components are briefly documented in [96] and are out of the scope of this thesis.

The KBService accepts queries that can be either in natural language or row format from multiple DBE components and tools. The query is then executed on the low level persistent storage that is implemented by using Berkeley XML database (XDB). XMI compliant documents that represent different DBE models are also accepted by the KBService which redirects them to the metamodel repository and finally store in the XDB. Replication between different KB peers is achieved by the Replication Manager that exploits the ExE's FADA peer 2 peer networks. Utilities include service information, session management, usage information etc, are exported as programmatic interfaces of which the KBService proxy is consisted. Despite a core service has all access to the ExE, the KBService is deployed as if it is a standard DBE service.
The Semantic Registry Service (SRService) is responsible for advertise, publish, search and retrieve the service manifest (SM) that is bundled by business models, semantic service models, data instances, service interface description models and other information. Similar to the KBService, The SRService is built on top of the DBE knowledge base peer infrastructure. Figure 5-9 taken from [96] illustrates the structure of the SRService.

As explained in section 5.1, the SM is a logical container that presents both technical details and business information. These information are assembled into an XML document and is published in the Semantic Registry. To effectively perform the semantic service discovery, an incoming SM document is decomposed into several components by the SM toolkit. Each of these components represents a model or instance data element within the original SM document. As a SM is written in XMI format, these components are stored in the metadata repository, processed by the model manager and well organised in the database. Same to KBService, the SRService exports functionalities as service interface and is deployed to the ExE as if it is a standard DBE service.

Both KBService and SRService provide dual ways of access: the programmatic interface and the graphical user interface. Although service manifests can be created for these
services, hence make them semantically discoverable, the current implementation does not provide such facility. Since Core Services consistently exist at any DBE node, it is unnecessary to follow the "discovery then consumer" pattern.

Other Core Service includes an accounting service that intercepts and records the service usage data, an identity service that provides a security framework for other DBE services and some other essential components which are more tightly integrated with the Execution Environment.

5.3 The Execution Environment

Pervasive Computing requires a container to provide platform independent application environment for the heterogeneous services. To fulfil the promises of ubiquity, even a single application environment would contains thousands of services that dynamically collaborate to deliver better user supports. The DBE Execution Environment (ExE) was designed to satisfy such demanding requirements by providing an architecture where pervasive services can be deployed, discovery, and consumed at anytime.

Multiple ExEs live in the parallel world where each of them is a self-controlled domain. Different domains are interconnected through the Federated Advanced Directory Architecture (FADA) P2P network. A goal of the ExE is to hide all the complexity details of distributed system programming from the ordinary developer. As a consequence, the ExE architecture is divided into the upper layer that provides unique access interface and the lower layer which handles networking issues as shown in Figure 5-10.
ServENT (Server and cliENT) is a server component which provides the runtime environment for DBE Services and access points for consumers to execute deployed services [96]. At the heart of a ServENT is a web service runs under Jetty, a 100% Java HTTP Server and Servlet Container [41] with typically 350KB payload and scales up to thousands of simultaneous connections.

Once been developed, a service can be deployed to any arbitrary ExE via the service deployment interface provided by the ServENT. The deployment interface implementation takes a DAR file (DBE Archive) that contains essential service components that is to be deployed. After receiving the DAR, the ServENT extracts the service manifest which is mandatory for the first time service deployment. The SM is then saved to the Semantic Registry which returns a unique SMID reference. Upon receiving this SMID, service implementation is extracted from the DAR and is installed within an exclusive execution container (a new Class loader). The service is then registered to the FADA network using the SMID reference that was returned by the SRIService. The usage of a unique SMID enables P2P service lookup within the FADA network. Non first time deployment follows a similar approach except that the new service is installed and registered using the existing SMID after which the old service is stopped and removed from the ExE as well as its own execution container.

```java
public interface Servent {
    void deploy (InputStream is) throws ServiceException;
    void undeploy (String id) throws ServiceException;
    void startService(String id) throws ServiceException;
    void stopService(String id) throws ServiceException;
    List getServices() throws ServiceException;
}
```

Figure 5-11 Servent Interface code snippets,

To execute a particular service, one must first find the corresponding service manifest reference ID. As explained in section 5.2, this can be done through the SRService. However, a ServENT does not offer direct access to the services in FADA network. Such facility is provided by a DBE toolkit bundler which assists DBE developers in the implementation of both server-side and client-side applications by means of the Proxy Framework and the Abstract Protocol Adapter (APA) framework.
Chapter 5. Implementation

The proxy framework wraps around the functionality provided by FADA toolkit and offers a simpler interface to the FADA lookup, retrieve mechanism. This eliminates the needs for understanding distributed system programming to the client application developer. The proxy framework also supports multiple client side User Interfaces (UIs). Once a service proxy objects has been retrieved, the service UI factory object is created at client side during runtime. The client may initiate different service UIs according to the contexts. Invocations to the service proxy object is forwarded to the workspace object which is part of the APA framework. The workspace object delegates the invocations to the concrete protocol adapter and provides session handling and other utilities to promote the dynamic service invocation.

```java
public static PAInvoker createPA(Map serviceConfig, Map clientConfig) throws ProtocolAdapterException {
    ProtocolAdapterInvoker pa;
    String protocol = (String) serviceConfig.get(SERVICE_PROTOCOL);
    // SOAP
    if (protocol.equals(SOAP)) {
        pa = new SOAPProtocolAdapter(serviceConfig, clientConfig);
    } // OBJECT SERIALIZATION
    else if (protocol.equals(OBJECT_SERIALISATION)) {
        pa = new SerializationProtocolAdapter(serviceConfig, clientConfig);
    } // Protocol not found
    else {
        throw new ProtocolAdapterException("Unsupported Protocol " + protocol);
    }

    return pa;
}
```

Figure 5-12 Create a Protocol Adapter code snippets.

The Abstract Protocol Adapter (APA) framework creates an extensible abstract protocol layer for components that require protocol independent communication. The exactly protocol adapter is selected at runtime rather than compile time by the encapsulated properties within the service proxy object. In other word, specified by the users. At the moment, DBE implementation only supports SOAP and object serialisation over HTTP.

Federated Advanced Directory Architecture or "FADA" in short, is a virtual Lookup Server in the sense that different Lookup Servers (FADA nodes) will work together to provide the LookupServer functionality from any entry point. Also, any of these Lookup Servers will cooperate with the rest to find implementations of services [61]. Unlike Jini, FADA does not use JAVA RMI to communicate between peers, Instead, the FADA core
resembles the standard Java RMI, modified to work transparently over HTTP, making it scalable over the Internet. As a P2P network, FADA make no assumption on the network latency. Instead, it uses mathematical tools such as Kalman filter to model the delay and adjust itself adaptively. A FADA node can join or leave the federation network at anytime, once a node has been connected, it will be automatically reached by other nodes in the network. As a consequence, FADA network topology is neither known nor can be forced beforehand. Tradeoffs have been made in which flexibility and reliability is more important than performance over the unpredictable Internet.

The FADA network holds proxies for registered services. A proxy is a Java class that knows how to communicate with the real service implementation and is downloadable at the runtime by client. Unlike Java RMI, FADA acts as its own HTTP server to host the class behaviour codes.

Both graphical and web-based management interfaces have been provided to query the status of a FADA node. Further details of the FADA implementation are out of the scope of this thesis, and can be found in Core FADA document [61].
Overall, FADA and the ServENT constitute a virtual environment where services are deployed, discovered and consumed. Each environment is a self-contained domain that aggregates to become a global service bus. A service is initiated within its own container that is provided by the ExE. The Execution Environment hide distributed services from the client application developer as if all services were deployed locally. To execute a service, the remote service proxy is retrieved and public method can be called. The ExE is not developed to replace the legacy system. In contrast, service implementation can reside on the original node while providing a service adapter to the front end. The Execution Environment design is essential to the Pervasive Computing where heterogeneous device collaborations are promoted.

5.4 The Service Factory

The Service Factory (SF) is an integrated Development Environment (IDE) where service providers with or without adequate technical skills can model and develop their services. One of the main advantages in DBE service modelling is to adopt Model-Driven-Architecture as its kernel which will significantly minimise the code that a service provider has to produce. As explained in section 5.1, A Service in the DBE context can be described from two coherent models which capture both business and technical concerns.

The service modelling architecture can be mapped into Service Factory using MDA approach. The results are three main activities that would be carried out by the service provider. They are: Business Modelling perspective, Service Modelling perspective and Service implementation perspective. Each of these activities has its corresponding editor and tools in the Service Factory.

The Service Factory has been released as an open source project "DBE Studio" that was developed as Eclipse [26] plug-ins. Eclipse is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software [26]. Delivering multi platform supports to the service providers, Eclipse has unique plug-in framework allowing unlimited extension to the eclipse's micro kernel architecture.

The DBE Studio has a number of visual editors with each representing a model in the
service modelling architecture. Compilers are provided to generate SDL model from the SSL model that is encapsulated in BML model, and the platform specific codes from the SDL model. The DBE studio works in associated with the Execution Environment to deploy, search and retrieve services. Tailored from [96], Figure 5-14 shows the component view of the DBE Studio.

Realising a DBE service requires a number of phases. Transitions between phases are supported by automated models or codes generation. DBE Studio was design to provide seamless support from model creation right through to service deployment. The development processes flow start from business modelling.

5.4.1 BML Editor.

As shown in Figure 5-15, the BML editor is used by a specialized business analyst on behalf of the service provider for modelling business concerns and service semantics according to their metamodels. The editor uses UML-like graphical notation for creating models in a drag and draw basis. Upon opening the business analysis perspective, the BML editor automatically connects with the specified DBE ServENT and lookup for the appropriate KBService. Existing BML models are retrieved and list at the left panel.
Both business models and service semantics can take advantages from domain specific ontologies that had been created by domain specialist during the modelling process. The editor fetches existing ODMs from the knowledge base and lists them in the right panel. The Semantic Description tab holds the visualised service semantic model. A service profile is a model for describing semantically a class of services [95]. Some service domains have been defined although it is not compulsory to apply one or more domain to the service profile. Functionalities and attributes can be added by mouse right click on the element. Wherever a type is required, ontology can be associated by applying a ConceptID that references to any predefined class in ODM. To reduce the overhead, BML provides a number of ready-to-use primitive types.

The Organization, Process, Event, Location and Motivation tabs hold corresponding visual models of the business concerns respectively. Restrictions imposed by the metamodel are ensured by the BML editor that, only valid elements or associations can be added. The modelling technique is out of the scope of this thesis.
5.4.2 ODM Editor.

The ODM editor is a visual tool based on an UML-like graphical notation that enables domain specialist to deploy specific ontologies in order to describe the business requirements of the service. Classes, Properties and Associations are modelled as diagram by which visual ODM elements can be selected and placed inside the diagram editor. Upon creation, the editor generates a unique ID for each ODM element. Ontologies annotation in BML editor is implemented by means of referencing the unique ODM element ID. By default, the ODM editor saves ontologies in the local file system as well deploy them to a specific DBE knowledge base. Before the ontology deployment, it is compiled to an XMI document which is interchangeable across platforms. The properties sheet is located at the bottom of the Ontology Analysis Perspective and provides an alternative way to edit the properties and values of a selected element. Any change in the properties sheet is synchronised to the diagram and Model Tree View Explorer and vice versa. Figure 5-16 shows the ODM editor in DBE Ontology Analysis Perspective.

![Figure 5-16 ODM Editor.](image-url)
5.4.3 SDL Editor.

A Service is described from two coherent views, the business view and the technical view. In DBE context, the technical view of a service is written in SDL (Service Description Language). The SDL editor presents a simple tree based graphic interface to which service developer can create/modify/update the SDL model. The property sheet is located at the bottom of the SDL editor. Provided by the DBE team, the SDL metamodel (schema) is used to validate against a SDL model using a simple validation tool inside the SDL editor. SDL only has limited number of primitive types, however, a service developer can define unlimited number of ComplexType to represents the data structure. Figure 5-17 shows the SDL editor in DBE Service Development Perspective.
5.4.4 The BML Data Editor

Instance data such as the BML data represents the concrete attributes of each element in their associated model. Attributes were attached to the elements in service modelling phase where each attribute was associated to a type such as a String or Integer. Values of these attributes are undetermined until the realised service is ready to be deployed. In the context of DBE, BML data template can be generated from the corresponding BML model that was stored in DBE knowledge base. The BML Data Editor in Figure 5-18 offers a simple tree base property editor for which values can be assigned to the given attributes. The service specific BML data will be published to the SR as part of the service manifest for semantic service discovery.

![Figure 5-18 The BML Data Editor.](image)

5.4.5 SSL2SDL & SDL2JAVA Compilers.

The DBE Service architecture insensitively adopts the concepts of Model Driven Architecture which separates business and application logic from underlying platform technology. Platform independent BML model can be realised on virtually any platform through the MDA tools. Mapping into the Service Development process, the BML model, particularly, the SSL model encapsulated within the BML model, can be transformed into the SDL model through the SSL2SDL compiler. Similar approach is carried out on which technology specific codes such as WSDL or Java Interface can be generated from the
SDL model. During the compilation, DBE knowledge base is optionally consulted if any ODM element is ontologically annotated. As a consequence, SSL2SDL and SDL2JAVA compilers are integrated as part of the DBE Studio release. Both compilers appear in the context menu as illustrated in Figure 5-19 and Figure 5-20.

**5.4.6 The Service Manifest Creator**

To make a service discoverable, the Service Manifest (SM) must be published to the semantic registry. As explained in section 5.1.4, a SM is a virtual container encapsulating both service models and data. To create a SM, DBE studio has included an eclipse plug-in named Service Manifest Creator as illustrated in Figure 5-21. The SM Creator can retrieve service's BML and SDL from the semantic registry or select them from the local file system. Along with other service information, the SM Creator is capable of saving a modified SM model to both local file system and the semantic registry. In case a SM has
been saved to the SR, the SMID field will be synchronized to the new SMID value that was returned by the SRService.

![The Service Manifest Creator](image)

**Figure 5-21 The Service Manifest Creator.**

### 5.4.7 The Service Exporter.

Once the business logic has been implemented in the DBE service development perspective, it is required to place all necessary files into the appropriate subdirectory on which the service deployment component of the DBE studio depends. To deploy a service to the DBE execution environment, all files must be packaged into a DBE archive (the DAR file) which is essentially a compressed file that contains implementation of the service, the service UI package, the service supporting libraries, the service deployment configuration file and other optional information.

To deploy a DAR file to the DBE execution environment, an eclipse plug-in, the Service Exporter was developed as part of the DBE Studio tools. The Exporter is a wizard based tool where the service developer can specifies the service name, service adapter, the service UI factory (if applicable) and other deployment specific settings. Once all the information has been entered, the exporter generates the deployment configuration and the DAR file which is sent to the ServENT via the service deployment interface.
5.5 Service & Application Programming Model

In the context of DBE, all services have their own service adapter and service interface. The service interface is an abstract programmatic service description that encapsulates the supported operations. Generated from SDL model, the service interface is delivered to the client application by other means. To invoke a particular service, the client application performs a semantic service discovery that is supported by the core SRService. The result is a ranked list of candidate services, each with a unique service manifest ID. The client application selects the best appropriate service and obtains a dynamic service proxy from the FADA network. Details of the distributed proxy lookup process are hidden from the client application as if it is a local call. Service invocation is therefore, a local method call on the service proxy object.

Alternatively, a client may use the workspace object that dynamically invokes a remote service provided that both operation and FADA service indexing information (SMID) was obtained in advanced. The dynamic invocation is delegated to the Abstract Protocol Adapter which selects the best possible protocol implementation to convey the invocation. Currently, the framework only supports SOAP and object serialisation.

While a service interface is an abstract specification, it is implemented by the service adapter who acts as the main bridge between execution environment and the concrete service implementation. Inside the adapter, the service provider can initialise or release...
any resource prior to or after the service instance is created as well as accepts incoming service invocations and delegates them to the service implementation which may or may not reside in the same execution container. Furthermore, a service adapter is not restricted to communicate with a single service implementation, it may invoke unlimited number of other services to achieve the desired objective even if they are outside the DBE network.

As explained in section 5.3, the proxy framework enables polymorphic interface for different clients. To discovery, fetch and initiate the user interface, the proxy framework provides UI factory interface that must be realised by the service provider who has provided one or more user interface implantation. A concrete UI factory response to the inbound query with a list of supported UI types and is responsible of creating a specific user interface as required by the client.

All user interfaces support a unique way of interface initiation while the remote communication is delegated to the internal workspace object during the creation of the instance. Additionally, classes of UI factory and UI implementation will be downloaded from the FADA code base at runtime.

5.6 Summary

In this chapter, implementation of the proposed architecture is presented and then its components discussed in details. As illustrated in Figure 5-1, the service modelling framework is derived from the OMG's MDA approach. To provide a reliable service execution environment, the implementation uses exclusive Classloader for each service instance. Given the diversity in a Pervasive Computing environment, the implementation does not restrict the location and communication protocol of which the service adapter communicates. To ensure the availability, services are registered to the FADA peer to peer network that forms a virtual lookup server from the client point of view. Several components have been provided to create the service factory that helps to model and develop the service. In particular, the business model is edited by the BML editor; domain specific ontologies can be created by ODM editor. The platform specific codes are generated from the SDL model which was generated from the BML model. The service exporter creates a DAR archive that encapsulates all necessary files and deploys it to the specified ServENT.
Chapter 6

6 Demonstration

In this chapter, two example services are presented to demonstrate the capability of the proposed architecture. The service development is fully compliant with DBE Studio workflow in Figure 6-1 (taken from [96]). The goal is to answer the question of whether abstracting devices and context resources into discoverable services has resulted in a system architecture that supports the dynamic nature of human activity in a Pervasive Computing environment. A critical discussion of the implementation is given at the end.

![DBE Studio workflow](image)

Figure 6-1 The DBE Studio workflow.

6.1 Example Services

Service is the atomic working unit in our proposed architecture. The DBE project has released the Execution Environment and Service Factory which provide all the necessary
Chapter 6. Demonstration

supports to make a service live in the Pervasive Computing environment. On top of it, a Context Service and a Printing DBE Service has been implemented. The ontology model was developed using ODM editor. The Business Models were visually developed using the BML editor. Ontological annotations were applied by referencing to the selected elements in the Ontology viewer panel. To simplify the process, business concerns were minimised where possible. The SDL models were generated from the BML model and platform specific codes were automatically generated from the SDL models.

6.1.1 The Context Service

In Pervasive Computing environment, the use of context could provide better tasks support. However, the kind of devices to provide a particular context is not restricted. One way of providing convenient access to the context resources is to aggregate a context service that hiding device dependent complexity behind the client applications.

A simple context service has been developed to encapsulate several context sources. The service exposes only three operations: getDate, getTemperature and getLocation to the execution environment. The first two operations take no inputs and simply return the current Date or Temperature. The third operation takes a consumer's reference as its input and returns the consumer's current location. For demonstration purpose, implementations were written in plain Java and no device level communication is involved, neither are other services. Any real world context service would require additional procedures to acquire raw data from device layer involving vendor specific configuration and protocol. Figure 6-2 shows the business model and technical model of the context service.

Figure 6-2 The Context Service Model.
6.1.2 The Printing Service

To simulate a Pervasive Computing environment where similar functionality can be provided by multiple devices, we have developed a Printing Services model which can be realised by various service providers. Each realisation of such model is a single Printing Service that represents a physical printing device. We have identified a number of common operations that should be implemented by each service instance. However, certain operations may not be supported due to device dependent limitations which are not concerned in the scope of this thesis.

![Service Profile]

Table: Service Profile

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invoke()</td>
<td>Take a recognised document and delegate the printing task to a physical printer.</td>
</tr>
<tr>
<td>isColor()</td>
<td>Simple operation to tell whether colour printing is supported.</td>
</tr>
<tr>
<td>getPrinter()</td>
<td>Return the associated printer object whose type is defined by ontological class Printer.</td>
</tr>
<tr>
<td>getServiceStatus()</td>
<td></td>
</tr>
<tr>
<td>getServiceContact()</td>
<td></td>
</tr>
<tr>
<td>getServiceProvider()</td>
<td></td>
</tr>
<tr>
<td>getName()</td>
<td></td>
</tr>
<tr>
<td>getType()</td>
<td></td>
</tr>
</tbody>
</table>

The business model is shown in Figure 6-3 (for demonstration purpose, other business elements are ignored). Some of the operations are annotated by ontologies which are defined in Figure 6-4. The invoke operation takes a recognised document and delegate the printing task to a physical printer. By recognised, we strictly mean a document whose type was annotated by the ontological class PrintingService_Input. The getPrinter operation returns the associated printer object whose type is defined by ontological class Printer. Each Printer has an "isLocatedAt" object property, denotes the physical location of the printer. Implementation of the Printer class should provide this information to the service consumer. In our case, the printer location is obtained from the Context Service we have developed. The isColor operation simple tells whether colour printing is supported. For demonstration purpose, we will not explain other operations in details.
6.2 An Example Application.

To illustrate the application development framework, we have developed a context aware printing application that will print a document from the nearest available printer. To support the semantic service discovery, a number of printing services have been deployed to the local Execution Environment. Each service has realised the Printing Service Model in from section 6.1.2.

First, the application obtains its own location by consuming the context service that was described in section 6.1.1. Second, it discovers the available printing services by consuming the core SRService described in section 5.2. Third, the application tries to obtain the location for each printing service. Finally, the application calculates the minimal distance between each printing service and its own thus find out the nearest available printer.
ClientHelper css = new 
ClientHelper(Serventlnfo.getInstance().getPrivateURL());
Class[] clazz = new Class[] {StringHolder.class, IntHolder.class};
StringHolder input = new StringHolder(this.toString());
IntHolder myLocation = new IntHolder();
String smid = "ContextService";
String operationName = "getMyLocation";
Workspace ws;
try {
    // Invoke a service without knowing interface in advance.
    ws = (Workspace)css.getProxy(null, new String[]{smid});
    ws.invoke(operationName, clazz, new Object[]{input, myLocation});
    System.out.println("Return Message from "+smid+": "+myLocation.value);
}catch (NoSuchServiceException e) {
    e.printStackTrace();
}catch (Exception e) {
    e.printStackTrace();
}

Figure 6-5 Obtain self location code snippets.

6.3 Limitations of the current architecture

Pervasive computing aims to provide adequate user supports in any circumstance. From a user's point of view, supports are perceived as the number of available software applications that similar to some ordinary tools which extend user's ability to perform various tasks. As we have addressed the problems in section 2.1.3, heterogeneous devices often invalidate the assumptions of which resources are well known in advanced to pervasive applications. We proposed a service oriented architecture that attempts to use service as the fundamental building block to provide computational user supports. The architecture relies on the richness of the services deployed in the Pervasive Computing environment. The more services are deployed, the better user supports can be provided.

The concept of service has provided a unique view to the applications that are increasingly relied on the resources to perform some tasks. However, the introductory of the service layer caused significant concerns to be taken into consideration during the implantation phase. The main problem for any service oriented architecture is that how to determine the right granularity of a service. If a service is too coarse-grained, consumers may receive more data than they actually need. If a service is too fine-grained, consumers will have to make several requests to the service to get all the data they need. Currently, our proposed architecture does not impose any restriction on the service design methodology. It is entirely the provider's responsibility to develop the right services for their potential consumers.
The proposed architecture does not specify the interactions between software services and their associated devices. As DBE Execution Environment is written in Java, which was not designed to directly communicate with low level devices, limited device supports have been provided only by the Java itself. To interact with physical devices, service providers have to utilise vendor specific proprietary solutions such as DLL on Win32 system which decreases the system interoperability.

Software entities in a traditional middleware solution such as CORBA are relatively tightly coupled. Both ends must run ORB and share the same interface, with a skeleton on the server side and the corresponding stub on the client side that handles the method invocation. Communication between ORBs is carried by IIOP which is based on the standard TCP/IP protocol. Given its dynamic nature, the static model above will cause considerable problems in reaching pervasive task supports. First, despite a few projects such as the Wireless CORBA specification [70] and the omni ORB [81], CORBA is less supported on mobile devices due to its complexity. Second, RPC-like method invocations make it overly restrictive, inflexible and inefficient for the communication requirements of pervasive systems [97]. Third, IIOP has no standard port, make it harder to traverse firewalls and fourthly, connection oriented protocols such as IIOP are not well suited to a dynamic environment where unreliable connections and changing network addresses occur.

Compared to CORBA, the proposed architecture enables software entities to be much more loosely coupled. Instead of knowing each other in advance, a service consumer can discover an appropriate provider at runtime and subsequently engage it by sending/receiving messages. Communication between both ends is based on the SOAP message protocol and a standard transport protocol such as HTTP or SMTP. Followed by an all-or-nothing approach, mobile devices willing to speak CORBA will have to support the ORB libraries [1] whereas the proposed architecture requires nothing but the ability to send and receive messages (normally written in an XML-like language). In contrast to strongly typed IDL validation, message validation happens at runtime and is only checked against its well-defined message structure. It is the responsibility of the application to understand the message payload. This flexible approach is well suited to the dynamic Pervasive environment where new services and applications are being developed everyday. Although the CORBA Firewall Traversal specification allows CORBA
requests to be routed through them, firewalls are usually configured to allow standard transport protocol traffic, thus enabling messages to cross firewall boundaries more easily in our proposed architecture.

Vinoski argued in [87] that CORBA as a success story will not be replaced in the near future. And the typical way to integrate existing systems like CORBA with newer approaches is to deploy gateways, bridges, or centralized intermediaries. The proposed architecture provides limited support for this scenario in terms of independency between service and the actual implementation. Much of the work is left to the developers implementing these approaches.

Although the demo application was built to finding the nearest printer for mobile users, the proposed architecture does not directly answer the question of whether the dynamic nature of human activity is supported. The mystery is clearly on the application side where the ability to understand context is crucial. For example, given the time and location of the current situation, the nearest printer does not always mean the geographically closest printer. However, our architecture does enable the developers to build software in such way by abstracting devices and context into consumable software services while we believe understanding human context an application concerns.

As the number of services grown, the demands of finding the right service become a prioritised issue. In particular, the service-oriented architecture by itself does not offer any guarantees on the semantic meaning of the service. In our proposed architecture, semantic interoperability depends on the common agreement that both consumer and provider shared. Technically, ontologies are imposed to constrain the way a service is described. However, the lack of a standard ontology annotation framework has limited the usability of our solution. Although OWL has been widely accepted as the standard web ontology language, there are no common agreements on any concepts that are needed to express the meaning of a particular service or product. Given the diversities of today's technology marketplace, there is still a long way to go to achieve such agreements.

Semantic service discovery is an open issue that has been researched by many institutes around the world. The tree-based discovery algorithm we proposed has a number of limitations. First, it is incapable of finding a service that is technically irrelevant but is still useful in the current context by other means. Second, it is potentially ineffective as
the number of services and ontologies grown. In a typical Pervasive Computing environment where thousands of services may reside, the construction of an ontology map is a resource consuming task. Each semantic query would require at least 3 individual sub queries on the map. The complexity of such a query depends on the number of nodes (concepts) that are directly or indirectly connected to the original node. The current implementation relies on the DBE knowledge base which will propagate the query to another FADA peer. It is unknown at the stage of the applicability in a real-world deployment.

Service composition is another issue that has been addressed by many researchers. The "disappearing devices" scenario in a Pervasive Computing environment implies the collaborations between multiple devices can be realised through a service chain. Although our proposed architecture has not described such a facility, the DBE architecture does plan to support service composition using BPEL. However, this support is in its infancy and yet to be fully working as an independent component. At the moment, service collaborations must be programmatically included as part of the service implementation, giving unnecessary complexity to the service development.

The quality of service is neither measured nor handled by our proposed architecture. In any real environment, quality measured by individual service itself is more likely to be inaccurate than that measured by the execution environment. A fourth component in the DBE architecture called the Evolutionary Network (EvE) is under experimental investigation to produce such measurements. Over time, service invocations and chaining are recorded by the EvE, which provides the infrastructure for automatic service-chain recombination and optimisation. The EvE will give priority to those popular services (individual) on performing recombination in the environment and forcing the death of those services that do not fit in the environment. On the whole, the EvE ensures the quality of service by which only the strongest service is allowed to survive and reproduce itself in the environment. Despite the fact that the EvE has not been implemented by the DBE project, metrics used to measure the quality of the service will greatly affect the optimisation results. For example, services who have received a high volume of invocations over a unit time do not necessarily mean a better quality is provided.
6.4 Summary

In this chapter, two simple services and a context aware printing application were presented to demonstrate the capability of our proposed architecture. The context service provides context information such as the date and temperature from the surrounding physical world as well as the location of a given object. A number of the Printing Service instances have been deployed to the execution environment with each representing a physical printer. The application will calculate the minimal distance between available services and itself, and thus send the document to the "nearest" printer. The locations of the printer and the application were obtained from the context service.

In the second half of the chapter, we presented a brief evaluation of our proposed architecture and implementation. Finally we conclude that by abstracting devices and context resources into discoverable services, our architecture enables developers to build software that carries out tasks on behalf of a user. However, more substantial and empirical experiments should be conducted to prove the effectiveness and performance of the implementation. We also believe that understanding human behaviour is an application concern, if we are to provide task support that fully meets the expectations of human users.
Chapter 7

7 Conclusion & Future Work

7.1 Conclusion

This report studied two distinct areas of interest: Pervasive Computing and Service Oriented Architecture. It makes two contributions to the research community. First, we proposed a system architecture for Pervasive Computing. The architecture is inspired by the principle of loose-coupling that was introduced by the Service Oriented Architecture. Second, we attempted to integrate ontologies throughout the system architecture, trying to make the software more responsive to the semantics of service requests, thus realising Weise's "technology disappearing" promises.

At first, the report reviewed some important characteristics of the Pervasive Computing system. Some architectural principles were identified and are observed in designing a software system for Pervasive Computing later in chapter 4. After that, a brief review on principles and techniques for software architecture was conducted. We also identified the potential problems of applying a component based design to the Pervasive Computing system. In particular, a component based system uses connectors to address connectivity between components. Although the principle architecture styles proposed by previous work still apply, heterogeneous devices in a Pervasive Environment imply the impracticability of connector-based interactions. The review then focused on an analysis of the service oriented architecture. As a de facto standard to realise an SOA, Web Services were also discussed. After that, ontology modelling principles were generally discussed followed by a feasibility discussion of combing SOA and ontology with Pervasive Computing.

Five related works were then studied. We argued that each project attempted to build a Pervasive Computing system from different perspectives. However, none of them explicitly used service as the fundamental element in constructing a Pervasive Computing system like the one we proposed in chapter 4.
The purpose of achieving loose-coupling is to directly address the problems of heterogeneities which are faced by any entity in an arbitrary Pervasive Computing environment. Other research work has explored a number of alternative solutions such as virtual machine [74], distributed object [22] or CORBA-based [50] approaches which are limited to be deployed in a fairly rich environment. In chapter 4, we proposed a service oriented software architecture for the inter-environment situation where the devices and services to be used are highly unpredictable. The architecture is flexible enough to be naturally converted to an inter-person scenario. A layered structure and a concrete system architecture for building a service oriented Pervasive application were proposed, components within the architecture were discussed with a strong emphasise on interoperability. We observed the requirements for accessing various contexts in an arbitrary environment and proposed a context provisioning mechanism using context services. Events in our architecture are provided through event providers that share many characterise with the ordinary services. The result is an homogeneous service oriented architecture with integrated context and event support. At the heart of the architecture, we proposed an ontology-based service matching algorithm by considering multiple service properties. The design philosophy behind is to provide a corrective action recommendation to the applications and their users.

The system implementation was discussed in chapter 5. It is composed by three coherent components: the Core Service, the Execution Environment and the Service Factory. The Core Service mainly contains the Knowledge Base service and Semantic Registry Service. The KBServe is used to manage various models under the context of DBE, particularly the Ontology Definition Model, the Business Model and the Semantic Service Model. The SRServe is used to store the Service Manifest and perform semantic service discovery. The Execution Environment is a framework where services live. It provides an exclusive execution container for each service instance. The environment exports proxy frameworks and an abstract protocol adapter to the client who needs to interact with the service deployed. Inside the environment is a FADA peer to peer network that can be interconnected to form a global Pervasive Computing Environment. Service consumption can be initiated from any connected node.

To demonstrate the implementation, we have developed two example services and a context aware printing application which prints the document to the nearest printer. The
location of printers and applications is obtained from the context service we developed. And each printer is represented by an instance of the Printing Service. A brief evaluation is then given in chapter 6.

## 7.2 Future Work

The proposed architecture does not solve all problems raised by the Pervasive Computing system. First, our architecture does not provide automated adaptation. One of the main challenges in designing software architecture for Pervasive Computing is to clarify the human's role in system modelling. We believe that a certain level of user distraction is necessary to solve the uncertainties which can not be taken away. The service matching algorithm and ontology based system modelling is one way to relax the judgement we made. However, the practical values are still unknown. Therefore, more solid experimental effort is needed to push the proposed architecture forward.

Context awareness in our architecture is addressed at the context provisioning level where context services are built to abstract the heterogeneous context resources. We rely on application specific context interpretations and adaptations to provide adequate task support. However, the effect of this dependence is unknown in a large scale development. We aim to develop a number of small applications which explicitly use context to adapt their behaviours in the future. To do so, we will need to implement the basic framework and necessary ontologies using the DBE Service Factory.

The service matching algorithm we proposed is yet to be solid, especially when the knowledge map becomes bigger and bigger; finding the hierarchy is a time consuming process. The situation is even worse when we consider multi-properties matching as a standard for the algorithm. One way to cope with this is to set a threshold for the depth of each searching process. However, an inadequate threshold will affect the quality and quantity of the results and reduce the system fidelity. Furthermore, judgement on who should specify the threshold needs further human role clarification. The after effect is unknown until we have conducted more empirical experiments.

The event pool is a mediator between the subscribers and event sources. Although it can be implemented in a service to service basis, it is not available in the current DBE
framework. Consequently, we will implement the event pool in the future. Furthermore, notifications are generated when certain subscriber specified rules are satisfied. For complex scenarios, logic and arithmetic operators such as AND, OR and >, = may not be enough to express the requirements. Our proposed architecture does not constrain the use of other rule-based language than logic or arithmetic operator, and further experiments are needed to compare the correctness and efficacy for different types of rules.

Since DBE was originally design for small and medium business enterprise, the use of service has been limited to intra enterprise level collaborations rather than cross devices collaborations in the Pervasive Computing environment. Particularly, the business model is redundant from a Pervasive Computing perspective. To overcome the problem, we plan to simplify the DBE architecture in the future.

One of the difficulties in evaluating software system for Pervasive Computing is the lack of standard benchmarks. There has been some work [4] [88] to define a standard framework for evaluation and assessment of the software system for Pervasive Computing. We plan to incorporate such evaluation framework in the near future.
Bibliography


Conference on Pervasive Computing and Communications (PERCOM'04), 2004.


[37] He Hao, "What is Service-Oriented Architecture?".


