Fully Automated Quality of Service (QoS) Aware Service Composition

by

Md. Mahfuzur Rahman

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfilment of the requirements of the degree of

MASTER OF SCIENCE

Department of Computer Science, University of Manitoba, Winnipeg

Thesis advisor

Dr. Peter Graham

Author

Md. Mahfuzur Rahman

Fully Automated Quality of Service (QoS)

Aware Service Composition

Abstract

In distributed computing, service composition provides the ability to combine existing services to produce new, value-added composite services that can be offered to end users. In general, when the type of the output data produced by one service matches the type of the input data needed by another service, then a composite service can be produced. Such Input/Output (IO) compatibility plays a major role in identifying potential composite services. A trivial example of such a composition might be combining the output of a scanner (a PDF file) with the input of a printer (also a PDF file) to produce a photocopy service. Ninja, eFlow, SpiderNet, TaskComputing, and CoSMoS are examples of service composition platforms [FS05; KKS06a], but they all lack support for several practical features including multiple protocol (e.g., UPnP, JINI) interoperability and handling of complex, multiple input services. Further, all existing systems depend on direct user involvement, which is unavailable in many pervasive-computing application environments. Earlier work by Pourreza and Graham [PG06a] looked at fully automating the composition process between services offered by the devices in a home area network. The work presented in this thesis focuses on two extensions to this earlier work: adding support for Quality of Service (QoS) to the composition process and extending the compositions outside of a single pervasive environment (e.g., home) to include offerings from third-party service providers (e.g. those provided via available Internet or 3G network access). I extend the existing

OWL-S -based composition system using properties to describe non-functional (i.e., QoS) constraints on services. I also introduce a means of describing services provided externally by service providers as software stubs installed automatically on a local gateway device. The ultimate goal is to offer only the most useful services to users thereby reducing their involvement in the composition process. I have built a prototype for the system to illustrate feasibility and to assess the overhead of supporting QoS in composition. I have also developed a regression model (based on collected user input regarding QoS preferences for services) that can be used to effectively rank compositions based on QoS for a variety of persistent environments. My results show that my approach is both feasible and effective.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my deepest gratitude to my advisor, Dr. Peter Graham, for his inspiration, guidance and support along the way. Thanks go to my committee, my parents, and all the people who have supported me. Finally, I would like to thank TRLabs for providing me their facility to pursue my research.

Chapter 1

Introduction

Pervasive computing is considered by many to be the next generation computing environment. In pervasive computing, devices with computing ability are everywhere and interact with one another and with users frequently, ideally, in intuitive ways. For this reason, users find pervasive environments to be very user-friendly environments that appear to understand and take care of user preferences. Pervasive devices are typically interconnected with each other in a mostly ad-hoc way. Each pervasive device must be able to advertise its available services and to recognize the services provided by other devices. Device mobility is also very common in pervasive computing environments. Thus, pervasive environments must be able to adjust to both topology and device availability changes. While meeting perceived user needs, the big-picture goal of my research is to explore how computing devices in pervasive environments can be made easier to use and more useful to their users.

Pervasive computing builds on five research areas: mobile computing, wireless networks, embedded computing, context awareness using sensor technology, and human computer interaction (HCI). Rapid developments in these research fields has recently made it possible to turn theoretical pervasive computing research into practical systems. A simple

example scenario of a pervasive environment could be the following: "When Bob enters his livingroom to watch a movie on television, the lights and TV in the room automatically turn on and the room temperature controls adjust to Bob's preferences. During the movie, Bob wants to cook something in the kitchen. When he goes to the kitchen, the pervasive computing system senses Bob's change in location and plays the movie audio on speakers on the kitchen wall. The pervasive environment resumes playing the movie on the television when Bob re-enters his livingroom." Pervasive computing systems treat cameras, set-top-boxes, sensors and the like as pervasive devices and use HomePNA [All], X-10 [xEEd], Bluetooth [Grob], IP [dmo], and other protocols to provide networking among those devices.

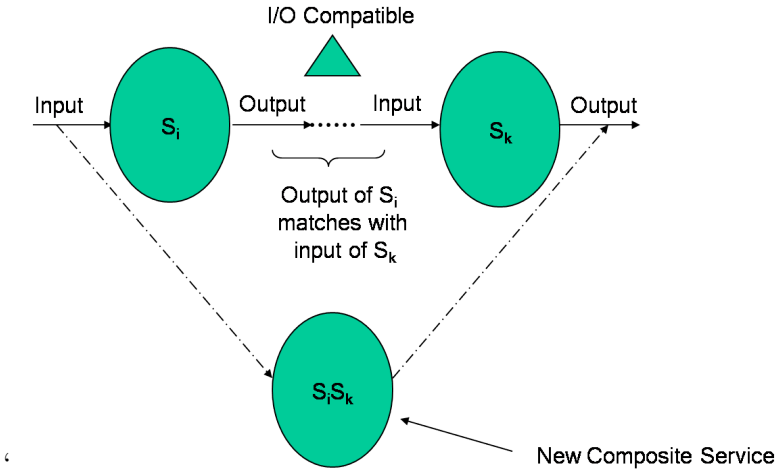

Figure 1.1: Service Composition

In pervasive environments, a service is a piece of software or hardware that does something useful. Service composition provides the ability to create new compound services using available ones. Each service has at least one input and output type. When the output type of one service matches the input type of another service, then a new composite service can be introduced. In Figure 1.1, the output type of a service, S_i, matches the input type of another service, S_k, as a result, a new composite service, S_iS_k, can be created. The input

type of S_iS_k is the same as the input type of S_i and the output type of S_iS_k is the same as the output type of S_k. Naturally, this composability generalizes to sequence of more than two component services. Using composition, it is possible to automatically create a number of useful services that can be made available to users in a given pervasive environment. Service composition techniques can also be used for such purposes as automatically configuring devices, integrating devices together, and upgrading device firmware. In this way, service composition provides a very simple, but attractive, as well as inexpensive, way to increase the ease of use and capabilities of a pervasive environment.

1.1 Motivation and Expected Contributions

To make service composition attractive to a broad audience, it is highly desirable to keep users free from any significant technical burden. With this in mind, earlier work has looked at designing a fully-automated service-composition platform. This has been explored for relatively simple applications and applied to managing devices within a home area network by Pourreza and Graham [PG06a]. Each participating service in a composition is referred to as a component service and described in such a way (using semantics) that they can be glued together on-the-fly by a composition manager. In this approach, all services must have semantic descriptions associated with them. These semantic descriptions include, at least, information about the input and output type of the services but may also include other information about, for example, the use of the services. The composition manager can use this semantic information to check the Input/Output (IO) type compatibility among available services. eFlow [CIJ+00], Ninja [ea01], SpiderNet [GNY04], Task Computing [MPL03; KKS06a], and CoSMoS [FS05] are different semantics-based service composition platforms. All these platforms need a user's requested composite service

as well as direct user involvement to complete the composition process. Moreover, none of them support multiple protocols or accept multiple inputs. Pourreza's fully-automated service-composition platform [PG06a] has successfully solved these problems. His framework provides two major improvements. First, it composes services on the fly without user involvement by introducing third party Service Enablers (SE) into the composition process. When a device enters a pervasive computing environment, the device automatically not only announces its own available services, but also uses the SEs to discover others (including composites) that are available. Second, Pourreza's framework optimizes the composition-matching process using repository-based matching and semantic caching. There is an SE-based workflow repository in Pourezza's framework used to store already-discovered composite services. The service composition matching algorithm tries to match newly-discovered services with components of existing composite services to recognize new composite services. Pourreza's use of the Open Services Gateway Initiative (OSGi) makes his prototype able to support multiple protocols and also helps to overcome both protocol, as well as hardware, heterogeneity.

Pourreza's service-composition framework is extended in this thesis. I make two major improvements to his framework. First, his composition process does not consider non-functional or QoS aspects (such as response time, cost, etc.) associated with services . Non-functional properties do not affect the correctness of composition, but do affect the quality of composition with respect to a users's QoS preferences. Non-functional properties can sometimes, but not always, be captured by the type specification. For example, the sampling rate for an MP3 file could be a separate QoS property or captured by specific subtypes of an MP3 supertype. What constitutes a non-functional property is commonly domain specific. In Pourreza's framework, whether or not two services may be composed is determined only by the compatibility of the data the first produces and the second consumes. This means

that, for example, he cannot control the rate of consumption or limit the combined cost of composed services. I introduce QoS constraints (to satisfy user preferences) into the framework to obtain more useful service compositions. User preferences are simply the QoS choices of the users which are used to help enhance the effectiveness and usability of service composition. With QoS constraints, the composition process can consider not only functional aspects but also non-functional aspects such as network bandwidth and cost to select the best possible service from available similar services. By associating QoS characteristics with each service and considering QoS in composition, it is possible to respect the QoS preferences of the users. With QoS aware composition, it is possible to offer the best preferred or suitable services to users in a pervasive environment.

Pourreza's composition framework also does not support linkage between in-home services and those provided outside the home (e.g., an IP TV service from an Internet Service Provider (ISP)). It is important to introduce a means to make compositions between locally-provided services and those provided by an external service provider. Integration of locally-provided services with external services increases the domain of services from which to choose the best suitable services. I also introduce the ability to discover external services and use them in compositions to Pourreza's framework.

1.2 Use of Example Scenarios

To help make discussions of service composition concrete later in this thesis, I use various real life scenarios. For example, when a User A starts a movie at home, the service composition manager might discover ISP_MovieSource and ISP_QualityChannel services offered by a service provider and also the Home_DisplayService offered by multiple in-home devices (e.g., TVinLivingRoom, PDA, Laptop_1024x768, or some other available display

device). Associated with each service (local or external) will be certain QoS parameters. Thus the composition manager might find the following QoS information associated with the discovered services:

Home_DisplayService: (resolution, refresh rate)

ISP_QualityChannel: (data rate)

ISP_MovieSource: (cost, availability)

The composition manager would then compose services using the available services to show the movie. Some possible composite services for this example might be:

- PlayMovieOnTV = ISP_MovieSource. ISP_QualityChannel . TVinLivingRoom
- PlayMovieOnPDA = ISP_MovieSource . ISP_QualityChannel . PDA
- PlayMovieOnLaptop = ISP_MovieSource . ISP_QualityChannel . Laptop_1024x768

To support QoS, the composition manager must select the most suitable composite services considering both the compatibility of the QoS characteristics of the component services and the user's QoS preferences. Each user will have different importance (i.e., expressed, possibly, as weights on various QoS parameters) for each QoS parameter. The composition manager will have to consider those QoS parameter weights in offering services to users. The composition manager will also have to consider the user's as well as computing device's location (availability in the environment) as context to the composition process.

1.3 Thesis Organization

The rest of the thesis is organized as follows. I discuss the necessary background and work related to my research in Chapter 2. I provide a problem description in Chapter 3

as well as a solution strategy; Chapter 4 describes my prototype architecture and the implementation details of my fully automated QoS aware service composition platform. Chpater 5 gives my experimental design and evaluation methods and presents the results of my research. Finally, Chapter 6 concludes the thesis and suggests some directions for possible future work.

Chapter 2

Background and Related Work

2.1 Pervasive Computing and Service Orientation

Some user environments are now populated with computing capabilities/sevices that are even embedded transparently to users in thin devices (see Figure 2.1) [CMS]. Pervasive computing is a paradigm to integrate these computing capabilities through underlying communication/network and other technologies. Pervasive computing thus can be considered as a distributed computing environment where there are different computing devices which have different operating environment, different level of mobility and heterogeneous network connectivity but which are able to automatically work together to provide useful services to users .

The primary objective of pervasive computing is to provide further use of consumer equipment (e.g. Mobile, Television etc.) and to serve users in the most effective way. In pervasive computing, service composition is one way to create new and useful services using existing services. This can increase the use and value of consumer devices. Since composed services can be made available to devices through the composition framework, devices with limited computational power can also use the computational capacity of higher-

Figure 2.1: Pervasive Devices [CMS]

end computational devices in the environment, if required to extend their capabilities. In this way, service composition provides an inexpensive way to enhance the usability of the available resources in a pervasive environment for users. Service composition also makes it possible to provide a great number of additional services in a pervasive environment which may be useful to the users. Unconstrained application of service composition may result in very many new services, some of limited usefulness. Therefore it is not practical or desirable to offer all available abstract or composite services to the users. Abstract services are basic services which can not be decomposed and composite service are composed of two or more abstract and/or other composite services. It is better to offer a select, reduced number of only the most useful services to users. This will require ranking to be done based on user preferences and/or present context information.

I will follow the Serviced Oriented Architecture (SOA) [PD03] actor style for describing QoS aware service compositions in pervasive environments (refer to Figure 2.2). SOA is the set of rules or design principles followed in service oriented development. SOA

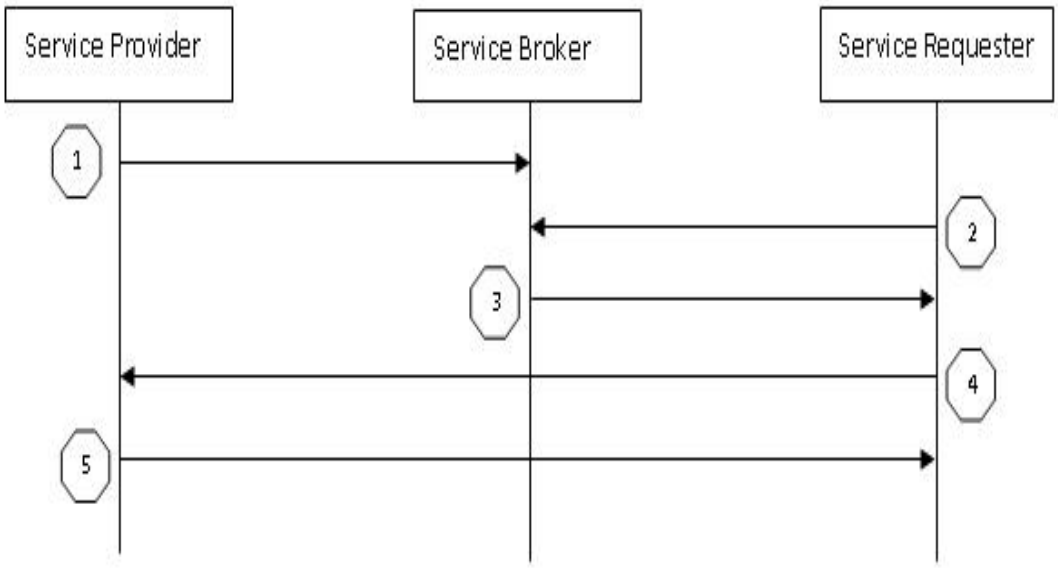

Figure 2.2: Actor's interaction in SOA [PD03]

considers services as basic computational entities or functional units which have well defined interfaces. In SOA, those entities that offer services are known as service providers and those that consume/use specific services are known as service users/requesters. There are also intermediary entities known as service brokers or service aggregators. Service aggregators maintain information regarding available services from service providers to satisfy lookup requests from service requesters. Service brokers also maintain information about how to access the offered services. The key steps in SOA (Figure 2.2) are as follows:

- Step 1: The service providers advertise their available services with one or more service brokers
- Step 2: The service requesters come to know about all the available services from different service providers through requests to service broker(s)
- Step 3: The service brokers maintain registries of the available services from different service sources (or service providers) and provide the service requesters the required information to access services
- Step 4: The service requesters can then place requests to service provider(s) directly using the information obtained from service broker(s)
- Step 5: The service providers serves the requests from service requesters, and responding accordingly

2.2 Service Orientation and Composition in Pervasive Environments

In a pervasive environment, there may be many different types of devices with varying capabilities. Many of the devices are embedded and devices are generally connected with each other using either wired or wireless connectivity. Kalasapur et al. [KKS05] categorizes the devices of pervasive environments using the classification shown in Table 2.1 which assigns a level (from L0 to L3) to each pervasive device based on its capabilities. Level 0 devices are resource poor and need to communicate with more resourceful devices to provide their services. Though Level 1 devices do not have significant resources, they themselves can host software to advertise services in a local environment. Due to resource limitations, they can not act as proxies for other devices. Level 2 devices possess sufficient

memory and computational power to also act as a proxy for other devices. Level 3 devices possess rich resources but they are not mobile as level 2 devices may be. The level assigned to devices can help in service discovery and in preparing a "service directory" in which services may be looked-up within a distributed pervasive computing environment.

Level	Features	Examples
0	No native personalization support	Sensors, legacy printers
1	Cannot be a proxy, possibly mobile	Cell-phone, mote sensors, smart printer
2	Can act as a proxy, possibly mobile, resource rich	PDA, laptop
3	Can act as a proxy, not mobile, resource rich	Server, PC, clusters

Table 2.1: Device classification chart [KKS05]

The Event Heap Framework [JF02] is an architecture supporting interaction among applications running in a pervasive environment. This framework focuses on the collaboration between applications when the applications address multiple devices simultaneously and coordinates such device usage.

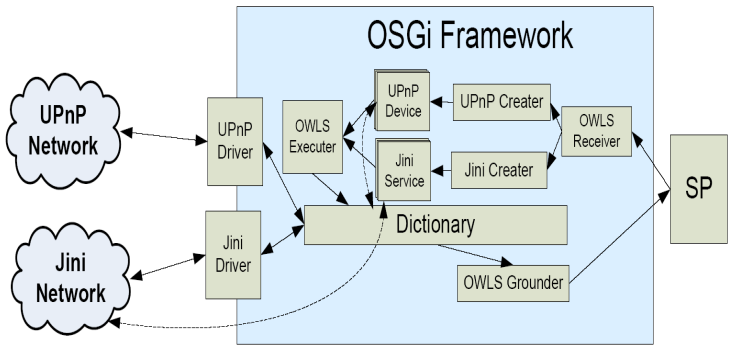

Figure 2.3: OSGi platform [PG06b]

There may also be devices using different protocols present in a single pervasive environment. To perform service composition with such devices, hardware heterogeneity

needs to be handled. Bottaro et al. [BBEL07] propose the use of the OSGi platform [Osc] to provide a common framework for the different types of protocols. OSGi has an API for the installation, activation, deactivation, update and removal of "deployment units" for the services of different protocols (e.g. Jini [Deva], UPnP [Devb]), but first, all services need to register with OSGi. Telecommunication Internet Gateways, TV connected set-top-boxes and utility service gateways already exist in many home environments. Bottaro et al. [BBEL07] propose an architecture to host their service composition framework on any of these gateways. Pourreza et al. [PG06b] also use the OSGi platform to deal with hardware heterogeneity in pervasive environment(refer to Figure 2.3).

Nakazawa et al. [NYT04] propose their Galaxy system as a framework to address the problems of device heterogeneity and the hidden service problem (when an application can not find the required services even though the services are present in the environment). In this framework, they suggest describing the capabilities of each service in an associated XML file. The capability description includes interfaces, operations and dependencies of the service. An application can easily transform such an XML document to other required formats using XSLT and can also search for a needed service.

Once all the available services are known, service composition can be used to produce more complex services. Each service has associated input and output type information. Syntactic matching is matching based only on the names (e.g. "print service", "scan service", etc.) of the services or types. Syntactic matching is possible when all the service providers agree on the names in advance. Otherwise, semantic matching can be used. Semantic matching not only uses the name of the services but also considers concepts in matching. A concept depends on the relationship (subclass, superclass, equivalent etc.) among services or their types.

Service composition using available services may be either static or dynamic. In

static composition, a "composition manager" knows before hand about the location and availability of the devices and their corresponding services. Accordingly, there exists an execution plan which includes devices that will participate in the composition. This approach is very common in web services applications. In dynamic composition, the composition manager dynamically selects the devices for composition on demand. Due to the mobile nature of many pervasive computing devices, dynamic composition serves applications best in many pervasive environments.

There may, of course, be more than one running pervasive application at a time. Thompson et al. [eab] define each composed service as a session and discuss the network component requirements to achieve various collaboration possibilities among different sessions in a pervasive environment. They introduce a proxy device architecture that does all the required work for the collaboration among the sessions in the environment.Gu et al. [GNY04] describe a very similar approach with Spidernet. Spidernet proposes a service composition framework for peer to peer networks [1]. Spidernet prepares a graph of the devices that are participating in the execution of an application and also keeps a backup graph of alternate devices. Spidernet uses this graph for interoperation between applications. When there are any changes in the environment affecting the composite service described by the primary graph, Spidernet can use the backup graph for the application.

Buford et al. [BKP06] propose a set of rules known as composition trust bindings (CTB) for use in service composition. During service composition, one device needs to access the resources of other devices so, there need to be some policies established for resource security and efficiency. The devices participating in a service composition should have an agreement among themselves about the use of their resources. A service invoking device sends its CTB to the service providing node(s) and the service providing node(s)

[1] Peer to peer networks are collections of devices where each device is considered as an equal peer. Such peer to peer networks are completely decentralized and self-organizing.

follow the constraint rules during execution.

Pervasive computing can provide the user with control over the devices and services used. But since pervasive computing mostly focuses on the personal and domestic aspects of life and typically most users are non-technical people, automation of service discovery, selection and composition is preferable. To get everything automated, context-awareness plays a vital role. In context aware systems, the composition manager considers "context information" during composition. Context depends on the current physical and environmental situations of the user as well as the devices involved in a composition. Context information helps the pervasive computing system to "understand" the user's environment: location, the current temperature, time, etc. Context allows composition according to a user's desires. Context awareness make pervasive environments user-centric and automatic [Lok].

Since many of the devices in pervasive environments are mobile, adaptation plays a major role in pervasive computing. During the execution of an application if any participating device becomes unavailable, then this problem needs to be solved by dynamic adaptation (to automatically provide an alternate composition to continue/complete the execution). To handle device mobility, Becker et al. [BHSR04] introduce three types of "adaptation" to their architecture : user specific manual adaptation, application specific adaptation and automatic adaptation. Manual adaptation involves user interaction which is not convenient. In this case, the users need to specify the alternative device(s) as well as service(s) to support the adaptation. In application specific adaptation, the service developers need to consider all possible cases, a-priori, and specify possibly complex routines to provide adaptation. Automatic adaptation reduces the burden on both the users and developers. In this case, the service developers only need to specify the input and output parameters of the services and the users need to provide their preferences before being able to use automatically created alternate services. OSGi also has an ongoing effort to address

service adaptation through a dynamic registry [Osc].

An operating system provides an efficient way to manage and use the resources of a computing system. Operating systems govern and control all the system components and provide tools and techniques for their use by end-users. Developing operating systems for pervasive computing systems is a challenge because of the constraints of many pervasive devices. Limited work has been done in this area. The Gaia OS [RHR+01] considers a pervasive environment as a distributed computing environment and provides an appropriate operating system for such an environment. Gaia OS assumes all the devices in the environment to be available hardware resources within the distributed system. Gaia keeps track of the services available from each of the devices and provides an efficient way to use those services for the execution of applications. The devices, on which Gaia can function, makes an "active space" (refer to Figure 2.4). Gaia OS integrates the management of hardware, software, networking, security of the devices in Active space. Gaia OS also hosts distributed applications. Any application running on Gaia OS needs to register first and then Gaia oversees all the necessary collaboration for the execution of the application. Applications are normally component based and run on multiple devices. Gaia performs the component creation, distribution and integration tasks for the application. To keep information about the devices in the active space as well as to present the available services, Gaia uses various components in its kernel.

Menon [Men03] proposes a file system protocol for pervasive environments. The protocol supports both wired and wireless connectivity. Three phases: negotiation phase, authentication phase and data transfer phase are used for the file system operations. In pervasive file systems, as files may remain in different devices, device mobility and data security are two major aspects to consider. Menon's protocol uses the mentioned three phases to address security problem. Research can be carried out to find out an efficient

replication algorithm to solve the device mobility problem for pervasive file system protocol.

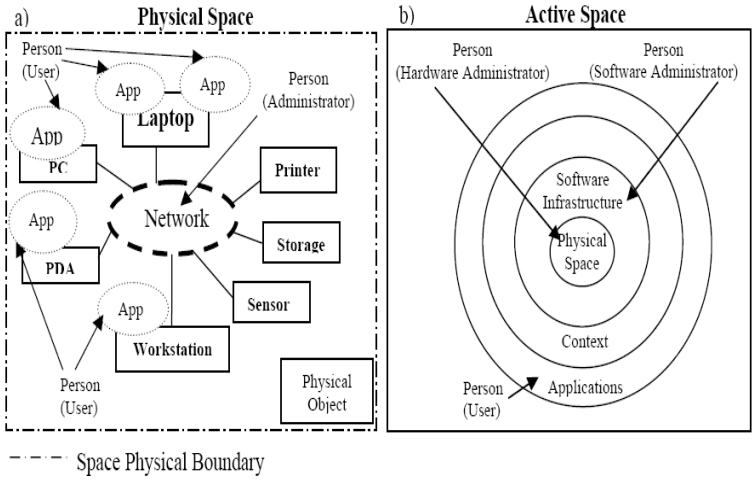

Figure 2.4: Physical Space and Active Space [RHR+01]

2.3 Service Description Languages

Service description plays a very important role in service discovery as well as service composition. To do composition, the services need to be described in a common way so that the composition task is feasible. Thompson et al. [TM05] introduce the 'Pervasive Service Description Language (PSDL)' to describe services. PSDL is based on XML and specifies all the required information regarding a service. Thompson et al. also introduce the 'Pervasive Service Query Language (PSQL)' to discover services that satisfy user requirements from a list of known services. Fujii et al. [FS04a] designed a 'Component Service Description Framework (CSDF)' to describe the semantic information associated with services. CSDF uses the Resource Description Framework (RDF) [Bri] Schema. The OWL-S (Ontology Web Language for Services) language [eaa] describes all the semantic information associated with services and is appropriate for an architecture that performs semantic matching for service composition. All of these description languages share some common elements and many

build on their predecessors but they have somewhat different goals.

Researchers have targeted four primary automation tasks for semantic description languages: automatic discovery, automatic service invocation, automatic service composition and interoperation, as well as automatic service execution monitoring. Semantic markup languages provide a framework to help achieve these goals [DFR04].

The Resource Description Framework (RDF) [HSB] is a platform that opened the door to semantic description. RDF primarily supports meta data about resources. A resource can be a service, QoS properties, context information or even objects that do not exist in the pervasive environment. Anything that can be described by an RDF expression is considered as a resource. RDF uses properties to define specific pieces of information (metadata) about a resource. Resources are named using Universal Resource Identifiers (URIs) and a property value can be another resource (again, identified by a URI).

Further, an RDF Schema (RDFS) provides a mechanism to describe concepts (about resources) as instances of classes. RDFS also considers subclass-superclass relationships and can define the properties of a resource as sub properties of other properties. Thus, RDFS enriches the vocabulary of a given domain by describing relationships between existing definitions.

OWL [Her] stands for Web Ontology Language. An earlier release of this language was known as DAML+OIL. DAML stands for the DARPA (Defense Advanced Research Projects Agency) Agent Markup Language and OIL stands for Ontology Interchange Language. OWL is richer in features than RDF and possesses a larger vocabulary and syntax than RDF. OWL provides a basis for performing automatic reasoning to infer information that doesn't explicitly exist. In OWL, it is possible to do "subsumption matching", "equivalence matching", and "consistency matching" among resources in the domain. Subsumption matching denotes a concept (i.e., a resource) that can be more general than

another concept (to create subclass-superclass relationship). Equivalence matching denotes two concepts that are exactly the same (to create synonyms). Consistency matching ensures that the matchings of the concepts (in terms of relationship among the resources) are accurate and consistent.

OWL-S is the OWL-based web *service* language. Previous releases of OWL-S were known as DAML-S. OWL-S provides a detailed representation of an ontology [2] with services. OWL-S supplies service providers with a core set of markup-language constructs for describing the properties and capabilities of their services in an unambiguous and computer-interpretable form. OWL-S markup of services facilitates the automation of service composition tasks, including automated service discovery, execution, composition, and interoperation. OWL-S offers control constructs (sequence, split, split+join, any order, choice, if-then-else, repeat-while) to create descriptions of potentially complex composite services from existing services [Con].

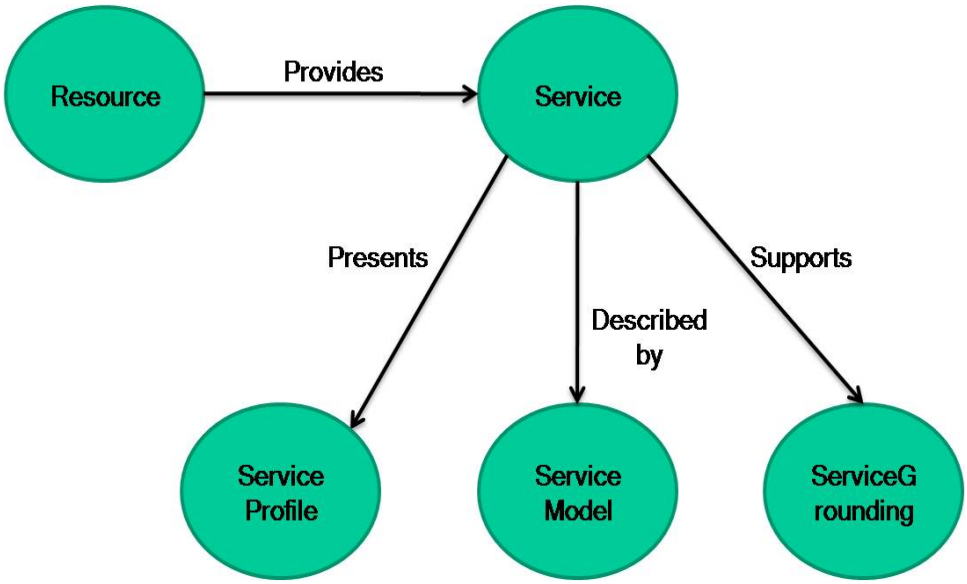

Figure 2.5: OWL-S Ontology [Con]

[2] A data model to describe a set of concepts and their relationships

OWL-S contains a rich service ontology and can describe a service in terms of service profiles, service process models and service grounding. A profile tells what the service does, a process model tells how the service works, and a grounding tells how to access the service (refer to Figure 2.5) [Con]. In OWL-S, a service profile includes the service name, contacts, and an abstract description of the service. This information is used in composition. The process model describes the services and tells us what happens when the service is carried out. The process model also specifies whether the service is a composite service or a simple service. The grounding specifies all the mechanisms (e.g. WSDL/SOAP) to allow access to the service's functionalities.

The Amigo Research Group [Groa] suggests that OWL-S does not provide all the required features to support specifying QoS characteristics for services. They have extended the OWL-S vocabulary to support additional, service-related QoS properties [KKR$^+$07], for different domains (e.g., personal computing, mobile communications, consumer electronics, and home automation). Such service-related QoS properties will also be used in my fully automated QoS-aware service composition platform.

2.4 Service Discovery

In pervasive computing environments, devices provide one or more services. Discovering services within a pervasive environment and from external service providers is another issue in service composition. Each device needs to know about the services provided by others. There exist two common approaches to service discovery. A fixed (non-mobile) device (i.e. central coordinator) can collect all the information about the services in an environment and about services offered from associated service providers and then notify other devices of the services available. Alternatively, each device could individually can be

responsible for collecting the available service information for themselves.

In the first approach, a composition manager can perform the necessary service composition using the collected information and the central coordinator can inform all the devices about the available services in the environment. New devices are also informed about all pervasive services when they enter the environment. Using this approach, each device is only responsible for the advertisement of its own services. When a user wants to run a composite service (application) using the available services, the centralized coordinator takes responsibility for the execution of the service by distributing the work among the devices providing the service(s) participating in the composition. In this case, the devices complete the responsibilities assigned by the coordinator. A problem with this approach is the single point of failure. Figure 2.6 shows a service-oriented architecture (SOA) with centralized-coordinator-based service composition [KKS06b].

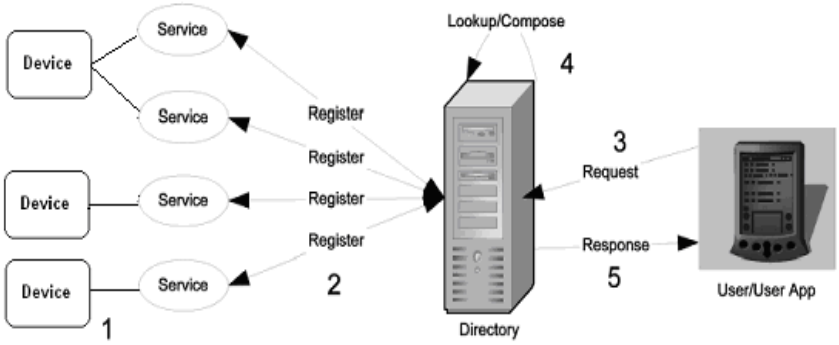

Figure 2.6: Centralized Coordinator-based Architecture; adapted from [KKS06b]

In the second approach, there exists no central coordinator to handle discovery and composition. Rather, all the devices have to individually collect service information from other devices in the environment. In this approach, when a user wants to execute a

composite service, the user device either itself acts as a coordinator or it dynamically selects another device to act as a coordinator for the composition. Basu et al. [TB05] describe the coordination of composition by a user device. Due to the resource limitations of many of the devices in a pervasive environment, this is not always feasible.

Pourreza uses a gateway device as the composition engine as well as the central coordinator for service composition. This provides a managed and reliable platform for composition, though computational limitations is still an issue. I will extend Pourreza's approach by including the feature of service discovery provided by the service providers.

2.5 Service Registry and Matching Techniques for Service Composition

Specific technique(s) for deciding how services can be composed are also needed. The services in a pervasive environment have input and output type information associated with them. An available service composition manager can use this information to do composition. There are different ways to do the composition. All require some kind of matching to recognize potential composite services.

In template-based matching, as described by Casati et al. [CIJ+00], the composition takes place following a user-provided template. A template contains the sequence of "abstract" services required and the rules for composition. The user creates a template or the composition manager can obtain it from a repository of existing templates. This type of matching requires a user-specific service goal (and corresponding template) and it is not usable when any of the component services are absent.

In interface-based matching [ea01], the composition manager does not depend on a template, rather it discovers the sequence of appropriate services based only on the

provided input and output type information. Interface-based composition is more adaptive than template-based composition. Interface-based matching is not usable when services have unknown input or output types.

In logic-based composition [WPS+03], the composition manager uses first order logic for composition. In this approach, each service needs to have extra associated information (i.e., preconditions and postconditions). Logic-based matching does not explicitly understand the meanings of data, it only follows the defined rules. This type of matching can only perform exact matching of input or output types. Exact matching allows substitution of an IO type for another type only if specific rules exist to declare their equivalency. Logic-based matching is complex and is not easily extensible.

2.5.1 Semantic Matching

Semantic-based matching allows in-exact ("partial") matches in addition to exact matches of IO types. Such partial matches allow inferencing about close and relative substitutes for any type even when there is no specific rule to specify the equivalency of those types. For example, if there is no exact matching for a particular type- "ColorPDF" then "PDF" (a superclass type of "ColorPDF" type) can suffice in that case. In the case of semantic matching, the matching engine, in some sense, understands the meanings of data and can perform composition even when the output type of one service does not exactly match the input type of the next. This understanding is achieved using an ontology that describes domain knowledge. In this case, each service needs to include its own semantic information. Semantic information may take different forms, often it is based on relatively simple knowledge about how types are related (e.g. Mokhtar et al. [MPG+08]).

Fujii et al. [FS04b] propose a semantic matching architecture, Semantic Graph-based Service Composition (SeGSeC), for service composition. SeGSeC uses its Component

Service Model with Semantics (CoSMoS) [FS04a] to describe the services in an environment semantically. Their Component Runtime Environment (CRE) helps SeGSeC to discover the services required for composition. Using the available semantic information about services, the composition manager prepares a workflow from a user's request and forwards the workflow to the composition coordinator for execution.

Lee et al. [LFBW04] introduce a "personal router", which is responsible for service selection in pervasive service composition. In their system, cost and quality (e.g., speed) parameters are associated with each service. The user can specify cost and quality for the services they want. A user's personal router performs negotiation and reasoning in selecting the services needed considering cost and quality constraints. It also has provisions for the user to specify either better or cheaper selections. But, in this system, the compositions are not done semantically.

COCOA [MGI06] provides two algorithms for dynamic composition of services. The COCOA Service Discovery (COCOA-SD) algorithm is responsible for interacting with the user. When the user enters a pervasive environment, COCOA-SD collects information about all the available services, determines the potential composed services, and shows them to the user. When the user selects any composite service to execute, COCOA-SD finds out which component services can perform the task. There may be more than one choice, but COCOA-SD selects only those choices that exactly match the inputs, outputs, preconditions, and effects of the user task. Then COCOA-SD finalizes the selection of the useful services that can participate in the composition. The COCOA Conversation Integration (COCOA-CI) algorithm oversees of communication between the services participating in the composition. COCOA-CI uses finite state automata for managing communication between, and integration of, the selected services.

Mingkhwan et al. [MFAM04] propose an architecture where a Service Integration

Controller (SIC) is responsible for the composition of services. The SIC implements semantic interoperability and signature matching for the composition of services. SIC uses an ontology with IOPE (Input, Output, Pre-conditions, Effects) information for all available services. In creating its ontology, 'subclass', 'superclass' and 'equivalent' relationships are used to provide meaning to the matching process. SIC handles all service requests through its matching process.

I will use semantic-based matching for service composition. Moreover, QoS parameters will also be supported as semantic characteristics in the composition process.

2.6 User Interaction and Automated Composition

In pervasive environments, the user needs to decide what to do, but should not have to consider how to do it. Moreover, users normally do not even know about the location and functions of devices. They should just place their requests and the pervasive computing system should provide them with the needed solutions. Research is going on to lessen user involvement in composition and to develop attractive user interaction techniques. In SeGSeC [FS04b], users can use natural language to place their requests. In Ninja [ea01], the user only needs to specify the input and output information of the requested composite service to the system. In e-Flow [CIJ+00], the user needs to prepare a request template, which is actually a service structure flowchart. In all these cases, the service composition starts after the user places the request. The execution of an application is efficient and fast when there is less user interaction with the system. ARIS [BB04] provides a graphical interface showing the available devices in a given environment allowing the user to select services for interest. ARIS runs on the Gaia OS [RHC+02] and helps the user to locate and interact with devices easily. Preuveneers et al. [PB05] suggest that pervasive services need

to support user personalization since users may have their own choices for service selection, resource selection, and choices of service behaviour.

In the architecture of Pourreza et al. [PG06a], the system itself tries to find all the potential using the available services and then presents those composite services to the user. This framework automatically composes services on the fly when new devices (and their services) are discovered. When any device enters a pervasive computing environment, the device automatically attempts to discover atomic or composed services that are available with the help of third party service enablers [PG06a]. The users of devices need only select their preferred service(s) at time of use. In this approach, the user may need to choose from a long list of similar services, so a ranking system is used to offer only the most useful/desired services to the user.

2.7 Context Information and QoS parameters

To make a pervasive environment smart, context information plays a vital role. Context is any information regarding the physical situations of users (e.g., user's location, orientation, distance from a particular device etc.) or environmental situations (e.g., temperature, humidity, time of day). Context information can be used to better help a pervasive computing system understand a user's present requirements. Thus, the computing system can offer better services to the user. Context information can be obtained through different sensor devices employed in the environment. I will also consider such context information in my system to improve the quality of compositions in terms of how well they meet a user's current QoS preferences [TBM$^+$].

Chapter 3

Problem Description and Solution Strategy

QoS is important to provide useful composite services generated from the services provided by underlying devices. Provision of QoS helps ensure that the system (and composite services therein) are performing at a user's desired level. QoS awareness in composition allows the selection of the most suitable services for a particular user in a given environment. For example, in Figure 3.1 there are three services (each one either atomic or composite): Service A, Service B, and Service C, all having the same functionality. To determine the most suitable service among these three, the QoS information associated with each can be used. Assessment of the QoS of composite services avoids the undesirable situation where functionally correct but inadequate composite services are provided to end users.

It is essential to be able to associate non-functional (i.e., QoS-related) properties with each service to enable QoS-aware service composition. It must then be possible to incorporate the provided QoS properties into the composition process and to be able to calculate the aggregate QoS properties for the resulting composite services. Finally, it must

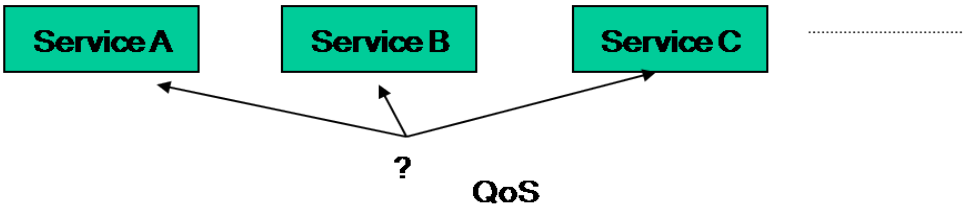

Figure 3.1: QoS for selecting best services [DD04]

be possible to select the most desirable composition (in terms of QoS properties) from among a set of otherwise equivalent compositions. This must also be done with reference to a set of desired QoS preferences for a given user (e.g., User A may have a QoS preference that states "low cost is more important than high quality, while user B may have the opposite preference").

To produce more useful composite services, it is also important to add support for compositions involving services from outside a persistent computing environment. For this, it is essential to develop "service broker" which can collect all available services from outside the persistent environment. Service brokers must consider the ontology aspects of persistent computing environment (before offering any service to it) to avoid conceptual ambiguity.

In the rest of the thesis, I will explain my strategy to design and to implement QoS aware service composition prototype. I will also explain how I asses any limitation to the strategy through experiments.

3.1 Solution Strategy

Pourreza's prototype [PG06a] uses the Web Ontology Language for Services (OWL-S) system together with some custom optimizations to provide fast and fully automated service composition. Additionally, this prototype uses OSGi to support both protocol as

well as hardware heterogeneity. The input and output properties of OWL-S specifications are used to implement IO type matching. Pourreza's system, however, does not provide support for QoS aware matching on for composing services from local devices with those from external sources.

OWL-S does not directly provide support for expressing QoS properties or for matching subject to them. I propose to include XML based QoS descriptions in OWL-S specifications in the same way as is done in Amigo [Groa]. The XML based QoS descriptions will be associated with service description in such a way that the service registry and composition engine can easily extract the required QoS information from the description. When a new composite service is found, its QoS properties will also be saved using XML tags in its OWL-S based description. I plan to use QoS properties such as those from the Amigo ontology [Groa] to add support for QoS properties in my prototype.

Each service may have many possible QoS properties associated with it. The following are some possible QoS properties :

Response Time (T): The time interval between service invocation and completion,

Availability (A): The probability that the service is available at some period of time,

Reliability (R): The probability that a request is correctly serviced within the expected time, and

Service Cost (C): The price that has to be paid to use the service

Many other QoS properties, of course, may also apply to specific services. It is important to mention here that different QoS properties may not have identical QoS characteristics. Some QoS properties may contain continuous values whereas some may only contain discrete values. High values of some QoS properties (e.g. ResponseTime) may be considered as good

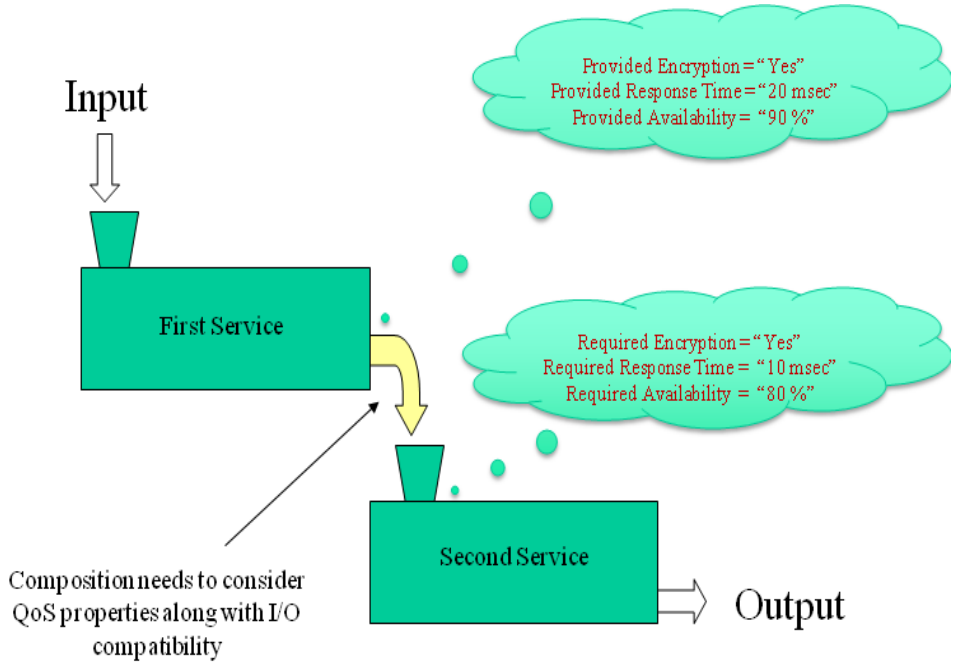

Figure 3.2: QoS aware composition

whereas high values may be bad for some other QoS properties (e.g. Cost). Some QoS properties (e.g. Reliability) may contain ordinal values whereas some (e.g. Security) may contain nominal values. These characteristics are needed to be considered in computing the QoS values of the QoS parameters of newly created composite services.

For a given service, there are two aspects for each QoS property. One is the specific QoS that the service offers and the other is the QoS the service requires. For example, for each service, there will be a required response time and a provided response time reflecting the two dimensions of the single "response time" QoS property. I will consider the compatibility of the provided-QoS values of the QoS parameters of first service and the required QoS values of the QoS parameters of second service along with IO compatibility. If the services are IO compatible (i.e. output type of first service matches the input type of second service) and the provided QoS values of all the QoS parameters of the first service

(e.g. S_i) exceeds or equal to the QoS values of corresponding QoS parameters of second service (e.g. S_k), then I produce the composite services (e.g. S_iS_k) (refer to Figure 3.2). In this figure, the second service requires encrypted data with at most 10 msec of response time and 80% of availability along with its input type and the first service is capable to satisfy second service providing all those QoS with its output type. The composite service, S_iS_k will have the QoS values for its QoS parameters computed from participating services (i.e. S_i and S_k). Each abstract or composite service possess QoS values for different QoS parameters which are considered to offer services to the users.

I plan to develop service broker which can act as an intermediary between service providers and persistent computing environment. Before offering any service from service providers to persistent computing environment, the service broker should prepare the required OWL-S description of the offering service and should also include QoS characteristics. In constructing the description, the service broker needs to consider IO types and QoS properties of the service in such a way that those are supported by the ontology of persistent computing environment and the presentation should support service ranking or filtering.

Chapter 4

Prototype Architecture and Implementation

I extended Pourreza's framework to support QoS-aware service composition. Figure 4.1 shows the high level structure of my QoS-aware service-composition prototype. This prototype performs composition, making the following assumptions and using the following steps.

- Each service carries its own semantic description, which provides enough QoS information in the form of <attribute, operator, value> triples (e.g., <bandwidth, equals, 100>) and input/output type information for the compatibility matching required to do QoS-aware service composition.

- The gateway device in each pervasive environment (refer to Figure 4.1) maintains a "directory" that includes a Service Information Module (SIM) and a Context Information Module (CIM). The SIM collects, maintains, and updates information about the currently-available devices and their services in the pervasive environment and also the services available in the environment from external service providers. The

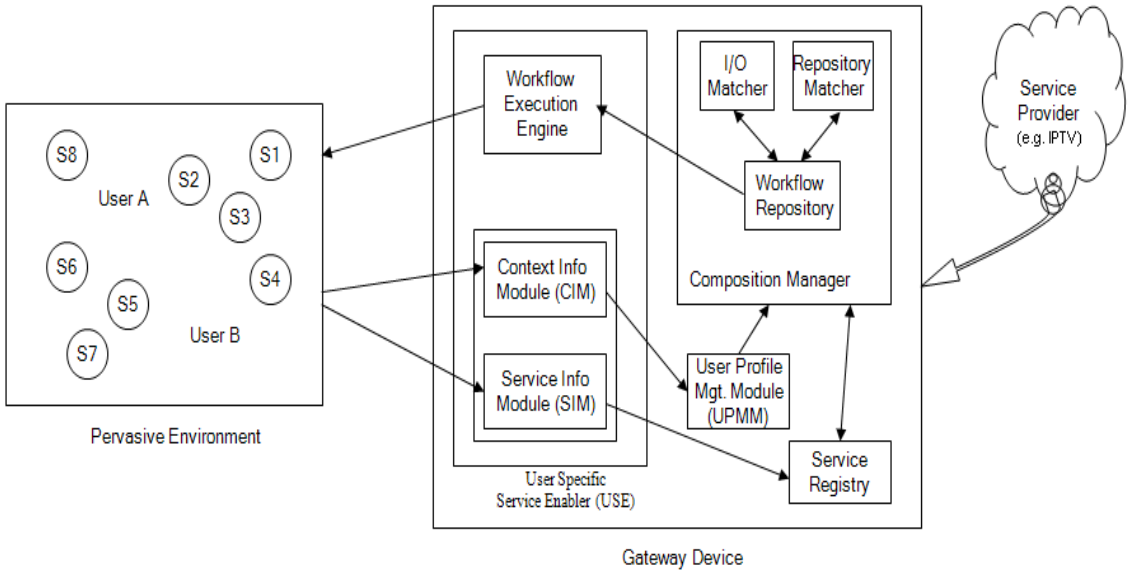

Figure 4.1: High Level Structure of the Prototype

CIM collects, maintains, and updates context information about the environment especially about the users in the environment (including user QoS preferences that may depend on current environmental conditions).

- The SIM detects and responds to any change in the environment regarding services and updates the service registry. The CIM updates the User Profile Management Module (UPMM) about any changes made in user preferences or context.

- The service registry maintains a listing of all available services in the environment (local or from known service providers). This registry also includes QoS information for each of the services.

- The User Profile Management Module (UPMM) contains all users' QoS preferences represented in the same way as for service QoS parameters. If there is any change in the

context of the environment, the CIM updates the UPMM as described. The UPMM must also be capable of aggregating multi-user QoS preferences. In multi-user QoS aggregation, the QoS preferences of different users are maintained in different profiles, but they are aggregated to determine the overall QoS preferences for multiple users in an environment. This is necessary since individuals QoS preferences may vary (e.g. Alice may like bright lighting in a room while Bob prefers it to be dim).

- The IO matcher, the repository matcher and the workflow repository form the composition manager [1]. The composition manager obtains the QoS information about each service from the service registry and collects the most up-to-date QoS preferences of users from the user profile management module. The composition manager selects the best services by considering a user's preferences, context information, and service QoS properties. User preferences are represented using relative weights for available QoS properties (e.g., weight(cost) = 0.3 and weight(availability) = 0.7, etc.) and each user may have different preferences/weights for different QoS parameters. These weights are used to combine more constraints in the case of multiple QoS parameters.

- In Pourreza's IO matcher, compound services are created incrementally by combining services that are IO type compatible. My IO matcher also considers QoS properties and produces QoS compatible (in terms of provided and required QoS levels) composite services with aggregate/combined QoS properties based on the component services. These aggregate QoS parameters are then later compared to user's QoS preferences to select the most useful composite services.

[1] In Pourreza's system, these were done outside the pervasive environment. I have integrated these functionalities into the gateway device in my prototype but that could equally well be implemented by a composition manager outside the pervasive environment to allow sharing of discovered compositions between environments.

- In the workflow repository, all the created composite services (with QoS parameters in my implementation) are stored to serve the repository matcher, which reduces computation overhead by avoiding unnecessary expensive re-execution of IO matching.
- In the repository matcher, previously discovered and stored composite services are searched to find composite services satisfying a user's QoS preferences.
- The workflow execution engine running on the gateway device oversees the execution of QoS-aware composite services.

Some specific challenges that I addressed in implementing my proof-of-concept prototype include:

- How to represent QoS properties using OWL-S and how to overcome any inherent limitations therein (e.g., handling QoS properties taking on discrete vs continuous values).
- How to combine QoS properties during composition (e.g., which properties should assume minimum versus maximum versus average values and how to handle multiple inputs).
- How to represent user QoS preferences (e.g., what limits should be placed on the complexity of specification, how types or media-specific specifications should be represented).
- How to combine multiple users' QoS preferences (e.g., how to handle conflicting preferences?)
- How to compare a composite service's QoS properties to user (single or multiple) QoS preferences to select the best (or a set of acceptable) available service(s) (e.g., mapping different representations).

How I met these challenges is described later in this chapter.

Service providers normally use gateway devices to offer services in a home area network or other pervasive environment (e.g. airport terminal, meeting facility). I also provide support to integrate services from external service providers with the services available in a pervasive environment (refer to Figure 4.1). I represent the services offered to the environment by service providers using software stubs for each provided service that are registered automatically on the gateway device and which act as proxies in the service composition process. Naturally, I also associate QoS parameters with these stubs. It is assumed that service providers provide the necessary QoS parameters such as the service response time and cost for their offered services. Such externally-provided services are then treated identically to local ones for the purpose of composition.

4.1 Implementation Details

In this section, I explain the different aspects of my implementation. This explanation includes describing the assumed domain ontology of type information, how I describe services and my QoS aware composition process. My implementation was done through different modules and most of the modules were developed using Visual Studio .Net. I chose Visual Studio .Net because it is an integrated development environment from Microsoft which has extensive support for the development of service oriented application and graphical user interface based applications.

4.1.1 Domain Ontology

I used the portege software tool [IP] [CP] to develop different ontologies descriptions and saved them as separate own files. For a particular ontology, concepts in a domain are related using subclass and superclass relationships. These concepts of the ontologies are

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://localhost/home/Media-ont.owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://localhost/home/Media-ont.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="AudioOnSpeaker">
    <rdfs:subClassOf> <owl:Class rdf:ID="Audio"/> </rdfs:subClassOf>
  </owl:Class>   <owl:Class rdf:ID="Video"/>   <owl:Class rdf:ID="Text"/>
  <owl:Class rdf:ID="AnimationOnDisplayDevice">
      <rdfs:subClassOf>  <owl:Class rdf:ID="Image"/>  </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="DigitalImageFile">  <rdfs:subClassOf rdf:resource="#Image"/>  </owl:Class>
  <owl:Class rdf:ID="ImageOnDisplayDevice">    <rdfs:subClassOf rdf:resource="#Image"/>  </owl:Class>
  <owl:Class rdf:ID="TextOnDisplayDevice">    <rdfs:subClassOf rdf:resource="#Text"/>  </owl:Class>
  <owl:Class rdf:ID="DigitalAudioFile">    <rdfs:subClassOf rdf:resource="#Audio"/>  </owl:Class>
  <owl:Class rdf:ID="DigitalTextFile">    <rdfs:subClassOf rdf:resource="#Text"/>  </owl:Class>
  <owl:Class rdf:ID="TextOnPaper">    <rdfs:subClassOf rdf:resource="#Text"/>  </owl:Class>
  <owl:Class rdf:ID="VideoOnDisplayDevice">    <rdfs:subClassOf rdf:resource="#Video"/>  </owl:Class>
  <owl:Class rdf:ID="ImageOnPaper">    <rdfs:subClassOf rdf:resource="#Image"/>  </owl:Class>
  <owl:Class rdf:ID="AudioStream">    <rdfs:subClassOf rdf:resource="#Audio"/>  </owl:Class>
  <owl:Class rdf:ID="TestStream">    <rdfs:subClassOf rdf:resource="#Text"/>  </owl:Class>
  <owl:Class rdf:ID="VideoStream">    <rdfs:subClassOf rdf:resource="#Video"/>  </owl:Class>
</rdf:RDF>
```

Figure 4.2: Ontology for different Media Types

used to describe both the IO types and QoS characteristics of different services. I designed four different test cases to study user preferences as well as user-oriented service-ranking. For each test case, I developed a number of ontologies to describe a range of abstract services.

As an example, the skeleton of a media ontology is shown in Figure 4.2 which was used to describe the IO types of services in a particular test case. In this simple ontology, different media types like Audio, Text, Video etc. are represented as classes. ImageOnDisplayDevice, ImageOnPaper are two subtypes of Image and the description (in Figure 4.2) includes the subclass-superclass relationship for Image as well as for other media-types. Naturally other ontological entries would need to be created for different persistent scenarios not considered in my test cases and devices but the principles involved are the

same as in my test cases.

```
<?xml version="1.0" encoding="UTF-8"?><rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://localhost/home/PrintService.owl" xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xmlns:j.0="http://localhost/OWLSExtensions.owl#" xmlns:j.1="http://localhost/Amigo/Amigo.owl#"
xmlns:j.2="http://localhost/Amigo/Amigo.owl#"
                    .....                                    ##Other Imports
<service: Service rdf:ID="PrintService">
    <service: supports> .....
<process:Input rdf:ID="input1"> <rdfs:label>input1</rdfs:label> <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost/home/Media-ont.owl#DigitalText
File</process:parameterType> </process:Input> .....
<process:Output rdf:ID="output1">  <rdfs:label>output1</rdfs:label> <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://localhost/home/Media-ont.owl#TextOnPap
er</process:parameterType> </process:Output>.....
     </service:supports>
    <service:presents> <profile:Profile rdf:ID="PrintServiceProfile"/> </service:presents>
    <service:describedBy> <process:AtomicProcess rdf:about="#PrintServiceProcess"/> </service:describedBy>
    <rdfs:label>PrintService</rdfs:label>
    <j.1:hasQoSProfile><j.1:QoSProfile rdf:ID="PrintServiceQoSProfile">
        <j.1:hasReliability><j.1:Reliability rdf:ID="PrintService_QoSR_Reliability"><j.1:QC_Qualify
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Equals</j.1:QC_Qualify><j.1:QC_Value
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">90</j.1:QC_Value><j.1:QC_Metric
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">percentage</j.1:QC_Metric></j.1:Reliability>
    </j.1:hasReliability>
</service:Service>

  <profile:Profile rdf:about="#PrintServiceProfile">
    <profile:serviceName>PrintService</profile:serviceName>
    <profile:hasInput rdf:resource="#input1"/> <profile:hasOutput rdf:resource="#output1"/>
    <service:presentedBy rdf:resource="#PrintService"/>
  </profile:Profile>
  <process:AtomicProcess rdf:about="#PrintServiceProcess"> <process:hasInput rdf:resource="#input1"/>
    <process:hasOutput rdf:resource="#output1"/><service:describes rdf:resource="#PrintService"/>
  </process:AtomicProcess>
</rdf:RDF>
```

Figure 4.3: Service Description for PrintService

4.1.2 Service Description

Services are described using OWL-S. It was because OWL-S provides the necessary "properties" support to describe QoS properties and for compatibility with Pourreza's

existing code that I chose to use OWL-S. For each service, there is an individual description file. This description includes information regarding service profile, process model and grounding. The profile includes QoS information for the services. Figure 4.3 shows selected portions of a PrintService description encoded in OWL-S. In this PrintService description, DigitalTextFile is shown as an InputType and TextOnPaper as an OutputType. Reliability (in respect to paper jam here) is the only QoS property and its value is 90%. The QoS properties are specified in this way extending service profile in OWL-S and also extending the QoS ontology obtained from Amigo [Groa]. For new composite services, a new description file is created along with IO and QoS information.

4.1.3 Composition Manager

My composition manager was developed in Java. I chose Java because it is platform independent and provides extensive coding supports for computation limited devices (thus allowing the use of simple devices as a composition manager). The composition manager is responsible for performing IO and QoS matching. For this, I have used a hash table design, like Pourreza, to achieve lookup efficiency. I also developed a web-based interface to control the composition manager i.e. starting, stopping, restarting, including/excluding ontologies etc. (Figure 4.4). The web interface was developed using Visual Studio .Net with Windows Communication Foundation (WCF) because WCF supports asynchronous and secured message transfer.

The composition manager produces new composite service through the matching of IO types and QoS characteristics of the available services. In my prototype, QoS matching takes place only when IO matching was successful beforehand. That means, if two services are not IO compatible then the composition manager will not do for QoS checking thereby saving time. The algorithm for QoS aware service composition is shown in Algorithm 1.

Algorithm 1: QoS aware Service Composition Algorithm

1: I_i=input of Service S_i and O_i=output of Service S_i
2: $Q^R{}_{i,j}$= Required QoS values for QoS parameter j of service S_i
3: $Q^P{}_{i,j}$= Provided QoS values for QoS parameter j of service S_i
4: **for** each S_k of already registered services do
5: I_k=input of Service S_k and O_k=output of Service S_k
6: $Q^R{}_{k,j}$= Required QoS values for QoS parameter j of service S_k
7: $Q^P{}_{k,j}$= Provided QoS values for QoS parameter j of service S_k
8: **if** O_i matches I_k **then**
9: **for** each q_i of $Q^P{}_{i,j}$ **do**
10: **for each** q_k of $I^R{}_{k,j}$ **do**
11: **if** parameterName(q_i) equals parameterName(q_k) **then**
12: **if** valueof(q_i) satisfies valueof(q_k) **then** set *success* to 1
13: **end if**
14: **else** set *success* to 0 and **go to** step 18
15: **end if**
16: **end for**
17: **end for**
18: **if** *success* equals 1 **then** add composite service S_iS_k to service registry and **exit**
19: **end if**
20: **else if** O_k matches I_i **then**
21: **for each** q_k of $Q^P{}_{k,j}$ **do**
22: **for each** q_i of $I^R{}_{i,j}$ **do**
23: **if** parameterName(q_k) equals parameterName(q_i) **then**
24: **if** valueof(q_k) satisfies valueof(q_i) **then** set *success* to 1
25: **end if**
26: **else** set *success* to 0 and **exit**
27: **end if**
28: **end for**
29: **end for**
30: **end if**
31: **if** *success* equals 1 **then** add composite service S_kS_i to service registry and **exit**
32: **end if**
33:**end for**

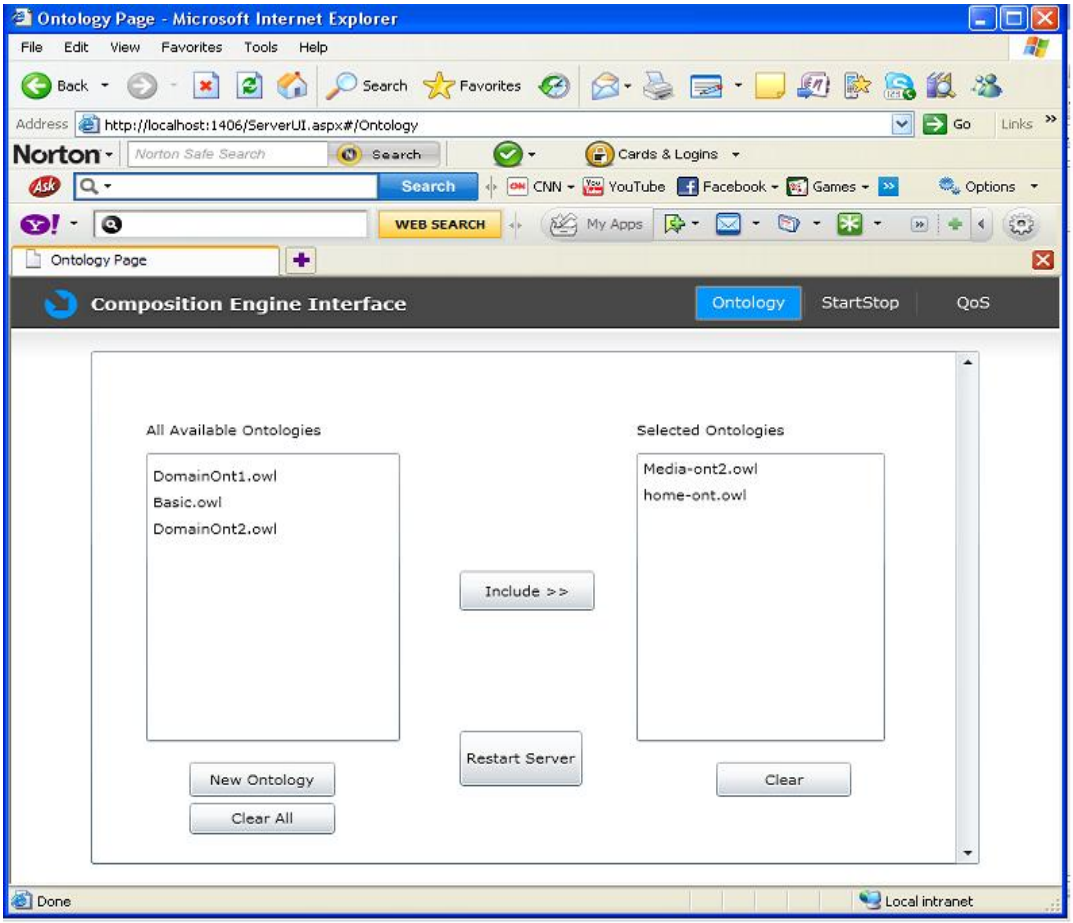

Figure 4.4: Selecting Ontologies for Composition Engine

In this algorithm, each arrival of a new service leads to compatibility checking with all the registered services. If the composition manager finds any IO compatibility, then it does QoS compatibility checking. When both IO and QoS matching are successful for a particular pair, then the composition manager registers the newly created service.

4.1.4 Service Registry

The service registry is an index of registered services that are available within a persistent environment from either local devices or external service providers. Every service

needs to be registered to participate in composition or to be offered to the users. If new

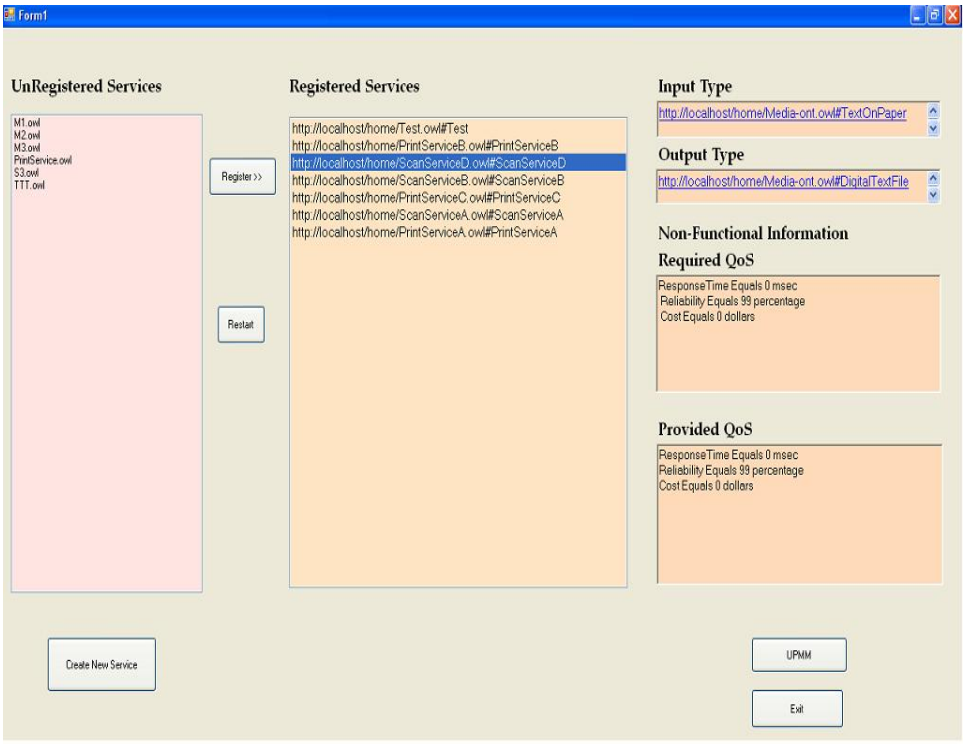

Figure 4.5: Service Registry

composite services are generated from the available services, these composite services are also registered in the service registry before being offered to any user. I implemented the registry, by extending Pourreza's registry to include QoS parameters. I also developed an interface tool to observe presently available services in the registry (refer to Figure 4.5). This interface tool allows the easy visualization of IO and QoS information for any registered service, atomic or composite.

There are two main types of sources from which services are available i.e. devices in a persistent environment and services from service providers. I developed my Service Information Module (SIM) using Visual Studio .Net so that it listens to a particular port for newly discovered services from both sources. In my implementation, a ServiceSender

module (developed using the compact framework of Visual Studio .Net) running on a PDA (playing the role of an arbitrary local device) is responsible for sending available service information to the SIM using Bluetooth connectivity. A Service Broker was developed (in Visual Studio .Net using Windows Communication Foundation (WCF)) as a component to collect all available services from outside the persistent environment and send the corresponding service registration requests to the SIM. When a service registration request (including an abstract service description) arrives, the SIM first checks whether the IO type and QoS parameters of the service are supported by the installed ontology in the composition manager or not. If the installed ontology supports the IO and QoS parameters of the service then the SIM sends the information to the composition manager to check if composition is possible. The composition manager performs IO matching and QoS matching as described earlier and registers all newly available services in the service registry. The SIM also collects, updates and maintains information about the currently-available services by keeping track of the devices in the environment. Device related updates (e.g. due to a mobile device leaving the environment) are important because of the need to de-register currently unavailable services.

4.1.5 User Profile Management Module (UPMM)

The responsibility of the UPMM is to offer only the most highly ranked services to a particular user. The services are ranked according to the user's QoS preferences. The CIM collects, maintains, and updates information regarding the presently available users in the environment and their QoS preferences to the UPMM. The UPMM uses a ranking function to find the best suited services for each particular user. Each user may have different QoS preferences for different QoS properties, so UPMM produces service specific particular profile for each user using the ranking function.

Ranking Function

Initially my QoS aware service composition prototype used to offer services to the users ranked using a simple weighting of some of the values of QoS properties. These weights were obtained from the users' preferences for QoS parameters. To improve the ranking function, I later created an online survey on user preferences on which to develop an enhanced ranking system. The survey described 4 real-life scenarios (refer to Appendix B). In each scenario, participants were asked to provide their QoS preferences and also to provide relative rankings for 5 functionally identical services with varying QoS characteristics. In general, the participants had to consider given QoS values (Q_1, Q_2, Q_3) for three QoS parameters associated with each service and corresponding self-given weights (W_1, W_2, W_3) for the QoS parameters generally (i.e. the user QoS preferences which represent the relative importance to an individual user of the three QoS values Q_1-Q_3 that might be used to decide the ranking of services).

For example, among the 4 real-life scenarios, the first one gathered information about how one deals with ordering an online movie. In this scenario the users were asked to assign importance to certain service characteristics (such as the cost of the movie) and to rate (considering their QoS preferences) some sample movie sources. The user's assigned importance for certain service characteristics was represented by weights (W_1, W_2, W_3) for QoS parameters.

From the collected data, I performed a linear regression analysis across all responses to develop a model of ranking for use in the UPMM. To perform the regression analysis, I used the R statistical software package [Wie]. I considered a total of seven different models. In the regression models shown in Table 4.1, $A_1 = Q_1 * W_1$, $A_2 = Q_2 * W_2$, $A_3 = Q_3 * W_3$. The check marks ($\surd$) indicate which factors were included in each model.

I also developed a series of ordinal regression models (refer to Table 4.2) that

	Q_1	Q_2	Q_3	W_1	W_2	W_3	A_1	A_2	A_3	W_1*W_2*W_3	A_1*A_2*A_3
Model A	√	√	√	√	√	√					
Model B	√	√	√	√	√	√	√	√	√		
Model C	√	√	√	√	√	√	√	√	√	√	
Model D	√	√	√	√	√	√	√	√	√	√	√
Model E							√	√	√		
Model F							√	√	√		√
Model G							√	√	√	√	

Table 4.1: Tested Linear Regression Models

	Q_1	Q_2	Q_3	W_1	W_2	W_3	A_1	A_2	A_3	W_1*W_2*W_3	A_1*A_2*A_3
Model H	√	√	√	√	√	√					
Model I	√	√	√	√	√	√	√	√	√		
Model J	√	√	√	√	√	√	√	√	√	√	
Model K	√	√	√	√	√	√	√	√	√	√	√
Model L							√	√	√		
Model M							√	√	√		√
Model N							√	√	√	√	

Table 4.2: Tested Ordinal Regression Models

parallel the non-ordinal models. Ordinal regression considers one or more parameters (participating in the regression) as ordinal data. Ordinal data normally considers the importance of order or magnitude (along with data values in a data series). In my ordinal regression models, I considered the human responses as ranked/ordinal data because it applied relative importance on human responses.

Chapter 5

Evaluation

Accuracy in composition corresponds to the perceived quality of the composition. If the accuracy in selecting the best component services is high, then the composition will be considered valuable by the end user. Speed of composition is also important. If composition is fast, then more QoS preferences can be imposed on the composition and, naturally, more compositions can be done in a given time. While speed of composition is unlikely to directly affect a single user in a given pervasive environment it is a factor, when compositions for many users in different environments need to be done concurrently by one composition provider offering composition services for many pervasive environments.

Accuracy of selection of composite services is assessed because it determines the usefulness of the system. Accuracy in composite service selection helps us to answer the following questions:

- Do we really select services that meet user needs?
- How effective is our strategy for combining user preferences vs. the quality of match for an individual user?

Speed of composition will determine whether or not the system is practical to use.

We need to assess speed of composition:

- with and without QoS support,
- for different numbers of QoS parameters,
- for different types of QoS parameters,
- for scalability with: more/fewer services available, and
- for many services from providers.

5.1 Overview of Assessment

I have initially implemented the gateway device components (refer to Figure 4.1) using a PC. In my testbed, this PC acts as a gateway device to compose local services in the persistent environment and also to compose the local services with services provided by service providers outside the pervasive environment. I used a number of emulated UPnP devices [TR-] (running on both normal PCs and Pocket PCs) with several sample Jini and UPnP services [For], which act as the available local services. I also considered IPTV service, Video-on-Demand service, Telephony Service, and a Video-Conference Service as examples of services from external service providers. When my composition system receives services from an external service provider, the system consumes network resources (bandwidth) and also incurs computational overhead to do the necessary composition to discover new compositions involving the newly introduced services. These costs must be considered in any performance assessment.

To assess my QoS aware composition approach, I have performed composition with QoS and without QoS constraints and in each case, I have varied the total number of available services from which compositions may be created. This results in changes to

the number of matches considered in doing composition. I have thus determined how much time is required to do composition under different scenarios. Similarly, I have also done experiments to assess how closely the composite services generated support different user preferences by changing the values of user QoS preferences.

To measure the accuracy of QoS-aware composition, I tried to find out how the offered services match with the user preferences. As described, I created a survey to collect user responses about QoS for different real-life scenarios and I compared the survey-results to the results produced by my composition system to assess my original ranking technique.

To measure the composition speed, I first considered cases in which there were no QoS parameters associated with any service and measured the time to compose the services. I then varied the total number of services and measured the required times. I then associated QoS parameters with each component service and measured the relative speed of service composition (comparing composition with and without QoS parameters). For the evaluation of the relative speed of composition and number of produced composite services, I ran different experiments with varying numbers of services available to compose. I also developed a general mathematical model to approximate the number of composite services for a given number of abstract services and number of QoS parameters. This number helps to predict the expected usefulness, computational efficiency and speed of QoS aware service composition over compositions without QoS.

5.2 Experiments

As a basis for the experiments, I created different ontologies (e.g. Media Ontology, QoS Ontology etc.) using the Pellet ontology editing tool [IP] [CP]. Using the created ontologies, I chose uniform randomly the IO types of each simulated service. QoS

characteristics were similarly selected according to the described ontologies.

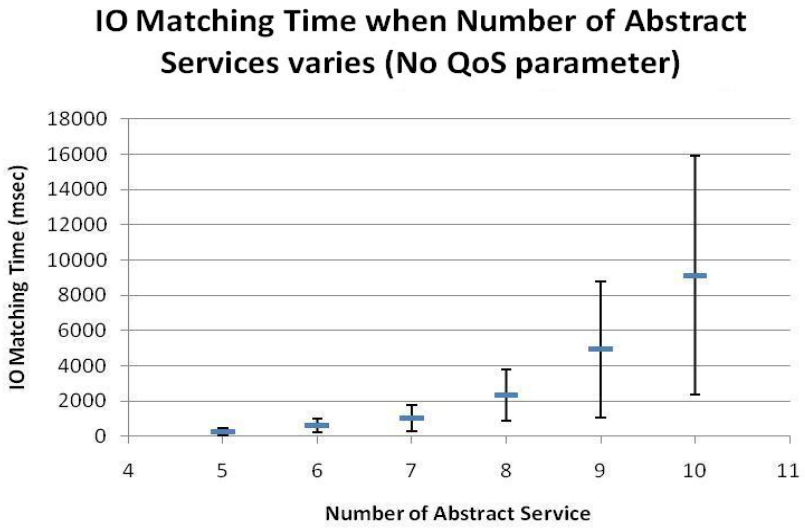

Figure 5.1: IO Matching Time with No QoS parameters

5.2.1 Experiment-1 (Matching Time)

The goal of this experiment was to find out how much the matching time varies when the number of abstract services changes. I ran the experiment each time with a fixed number of abstract services ranging from 5 to 10. I did 50 runs for each case and calculated the average of the results. In my prototype, the total matching time is the summation of IO matching time and QoS matching time. In the results, both types of matching times are presented individually. The error bars shown in the following graphs always consider $+/-1$ SE (Standard Deviation Error).

IO Matching Time

Figure 5.1 shows how the IO matching time varies with the number of abstract services when there were no QoS parameters associated with abstract services. From this

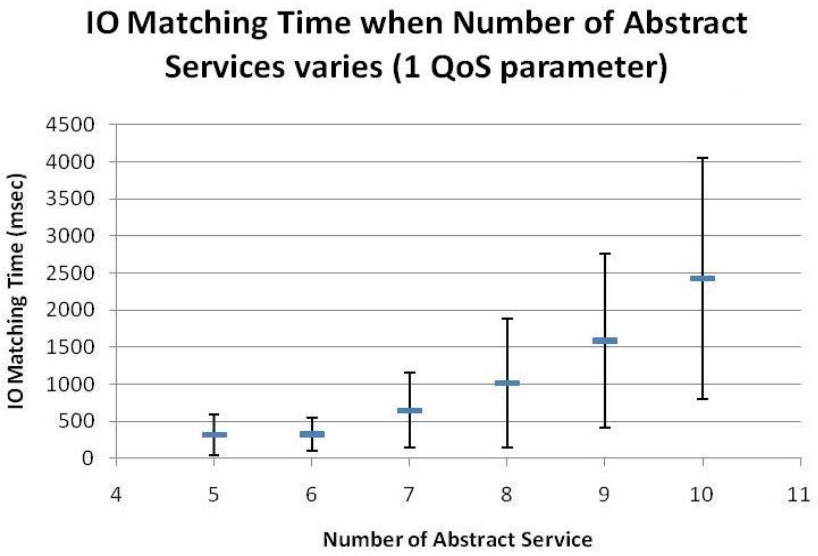

Figure 5.2: IO Matching Time with One QoS parameters

figure, we see that the IO matching time increases when the number of abstract services increases. This is because, with the increased number of abstract services, each service needed to do IO compatibility checking with more services (abstract and/or composite). Figure 5.2 shows how the IO matching time changes with the number of abstract services when there was one QoS parameter associated with each service. The reason for getting wide variances in the case of a larger number of abstract services is that the abstract services were selected uniform randomly and for some of the cases the composition system produced a larger number of composed services (for further IO consideration) and for some of the cases it produced less. With more experimental runs, the variance can be reduced. Moreover, using the prediction model for the expected number of composite service (proposed later in this chapter), the variation issue can be better understood. It is noteworthy that IO matching time had a very sharp decrease when I associated a QoS parameter with each service (refer to Figures 5.1, 5.2). The reason is that the composition engine had to

do fewer IO compatibility checks when there were one QoS parameter compared to no QoS parameters associated with each service because QoS incompatibility eliminated many potential composite services from further consideration.

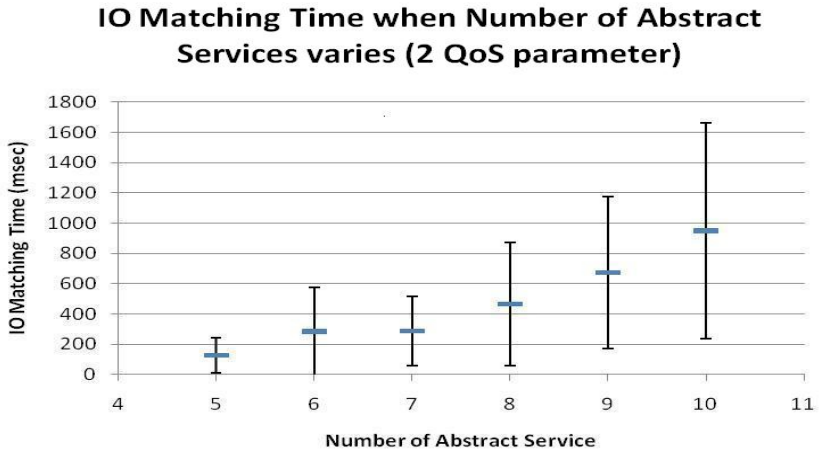

Figure 5.3: IO Matching Time with Two QoS parameters

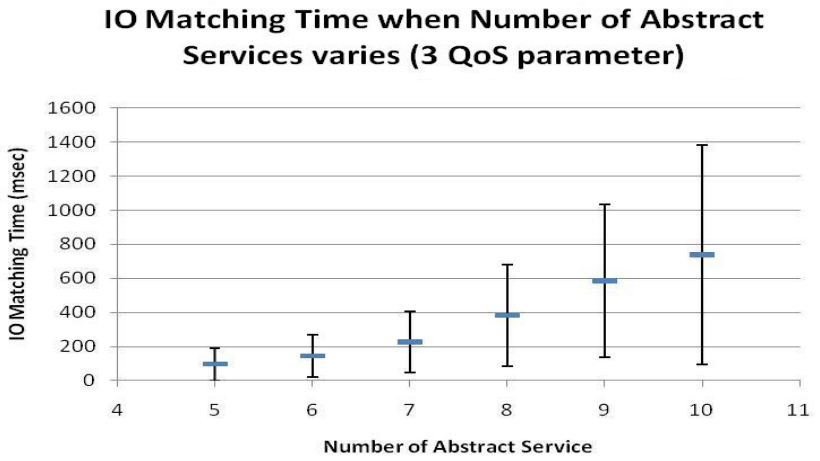

Figure 5.4: IO Matching Time with Three QoS parameters

In general, I found there was an increase in IO matching time against number of abstract services (refer to Figures 5.3, 5.4 and 5.5 where there are different numbers of

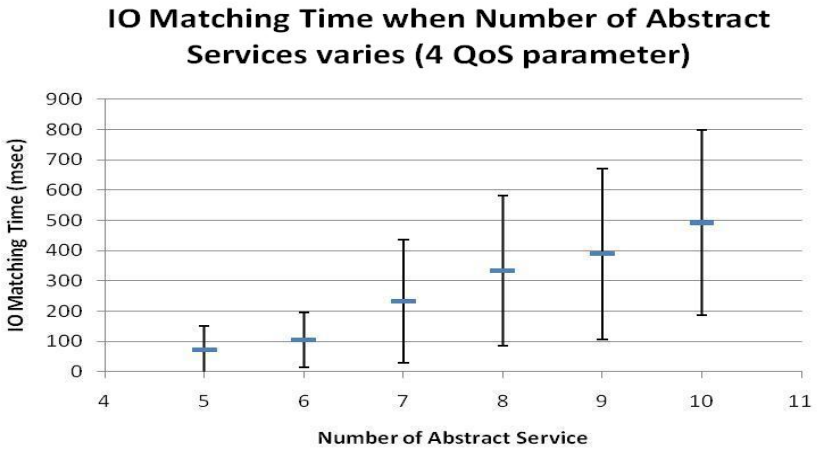

Figure 5.5: IO Matching Time with Four QoS parameters

associated QoS parameters).

QoS Matching Time

Figure 5.6 shows how the QoS matching time varies with the number of abstract services when there was a single QoS parameter associated with each abstract service. From this figure, we see that the QoS matching time appears to increase linearly as the number of abstract services increases. This is because, with the increased number of abstract services, each service needed to do QoS compatibility checking with more services (abstract and/or composite). Figure 5.7 shows the QoS matching time when there were two QoS parameters associated with each service. It is noteworthy that QoS matching time increased slightly with an increase in the number of QoS parameters from 1 to 2. (refer to Figures 5.6 and 5.7). The reason is that the composition engine had to consider more QoS parameters in each QoS compatibility checking.

In general, I found an increase (appears linearly) in QoS matching time with increasing numbers of abstract services (refer to Figures 5.8 and 5.9). This linear relationship

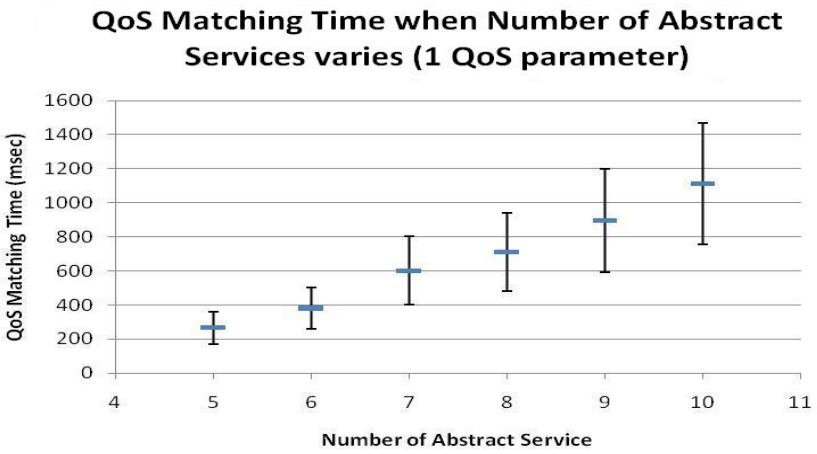

Figure 5.6: QoS Matching Time with One QoS parameters

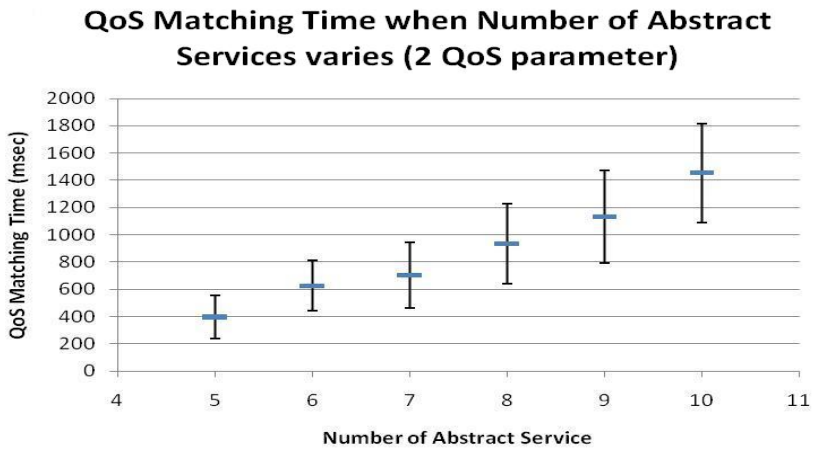

Figure 5.7: QoS Matching Time with Two QoS parameters

of number of abstract services with both IO and QoS matching time suggests that it should be quite feasible to do QoS aware service composition. Since individual compositions are independent of one another, pools of composition processors could be easily be used.

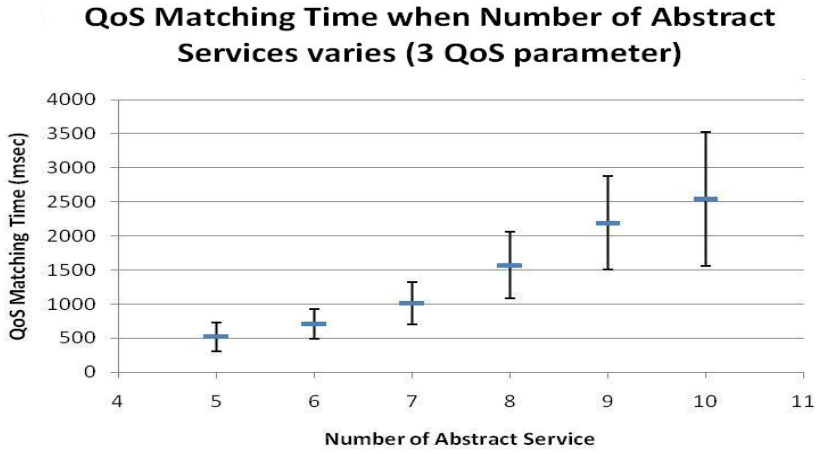

Figure 5.8: QoS Matching Time with Three QoS parameters

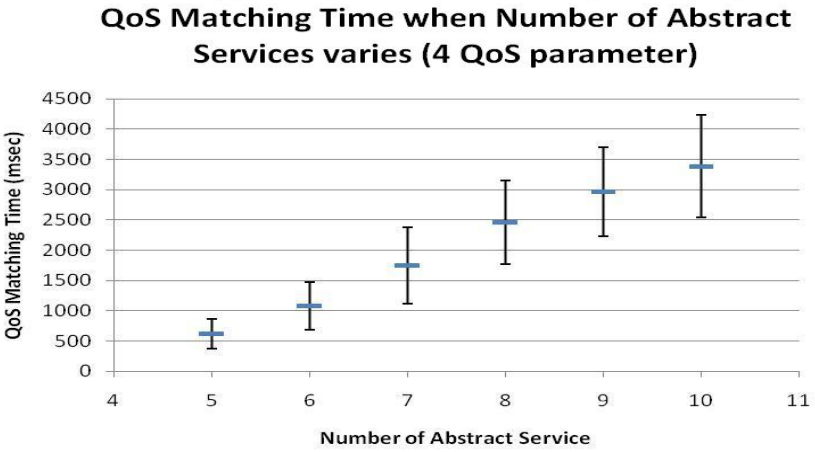

Figure 5.9: QoS Matching Time with Four QoS parameters

5.2.2 Experiment-2 (Number of Composite Services)

The goal of this experiment was to find out the total number of new composite services produced when the number of abstract services changes. I ran the experiment each time with a fixed number of abstract services ranging from 5 to 10. I did 50 runs for each case and calculated the average of the results. The error bars shown in the following

graphs always consider +/-1 SE(Standard Deviation Error) Figure 5.10 shows how the total

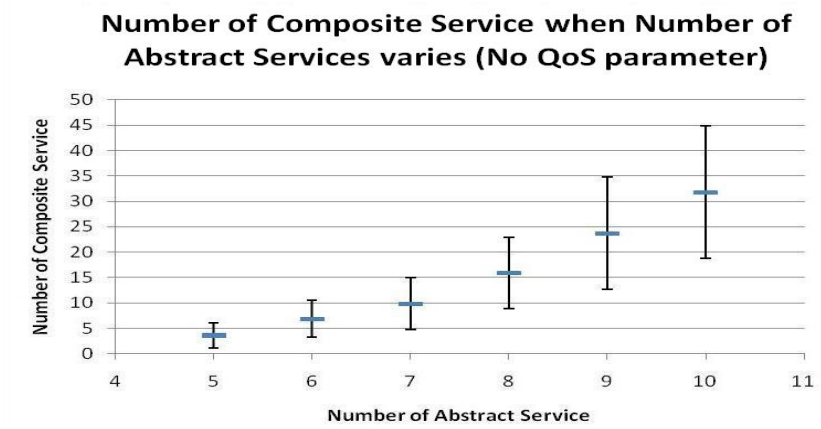

Figure 5.10: Number of new Composite services with No QoS parameters

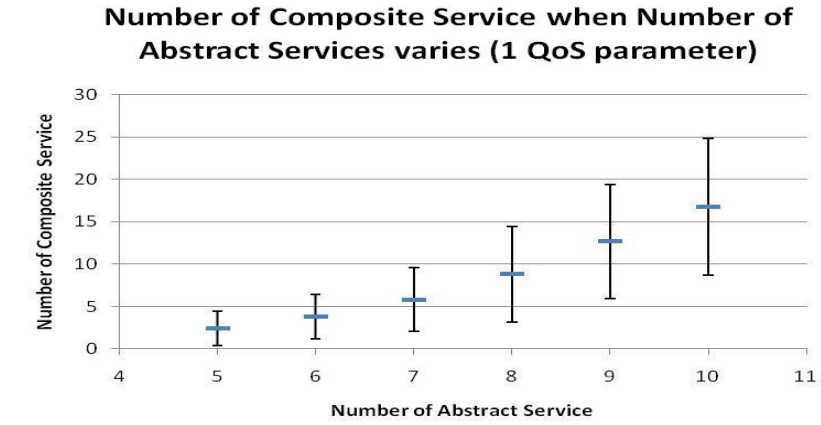

Figure 5.11: Number of new Composite services with One QoS parameters

number of new composite services produced varies with the number of abstract services when there were no QoS parameters associated with abstract services. From this figure, we see that the number of produced composite services increases when the number of abstract service increases. This is because, with the increased number of abstract services, each service performed both IO and QoS compatibility checking with more services (abstract

and/or composite). Figure 5.11 shows how the number of produced composite services

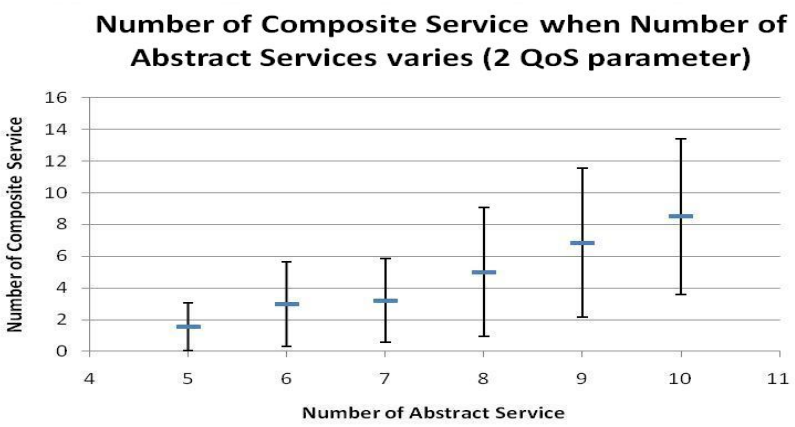

Figure 5.12: Number of new Composite services with Two QoS parameters

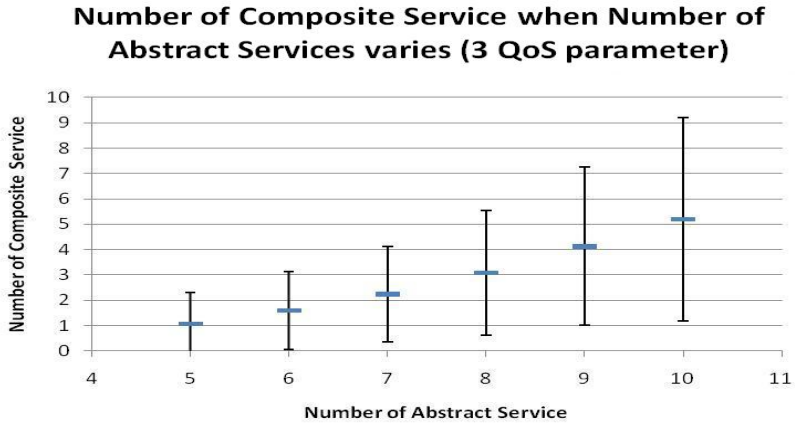

Figure 5.13: Number of new Composite services with Three QoS parameters

changes with the number of abstract services when there was only one QoS parameter associated with each service. It is noteworthy that the number of produced composite services slightly decreased when I associated a QoS parameter with each service (refer to Figures 5.10, 5.11). The reason is that the increased number of QoS parameters applied more constraints on producing composite services. In general, I found there was an increase

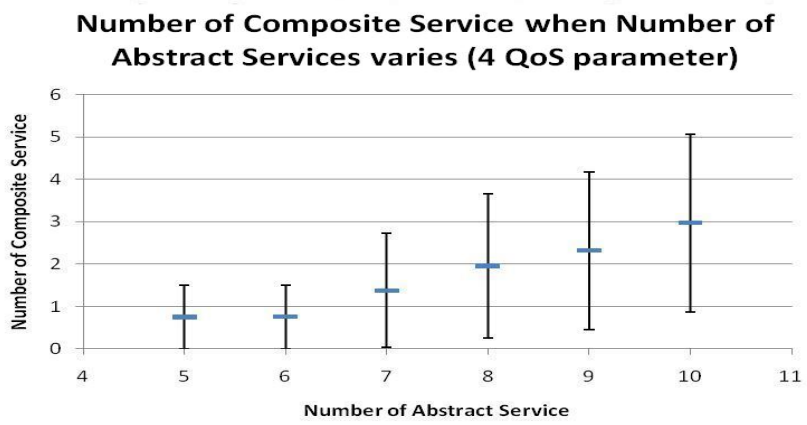

Figure 5.14: Number of new Composite services with Four QoS parameters

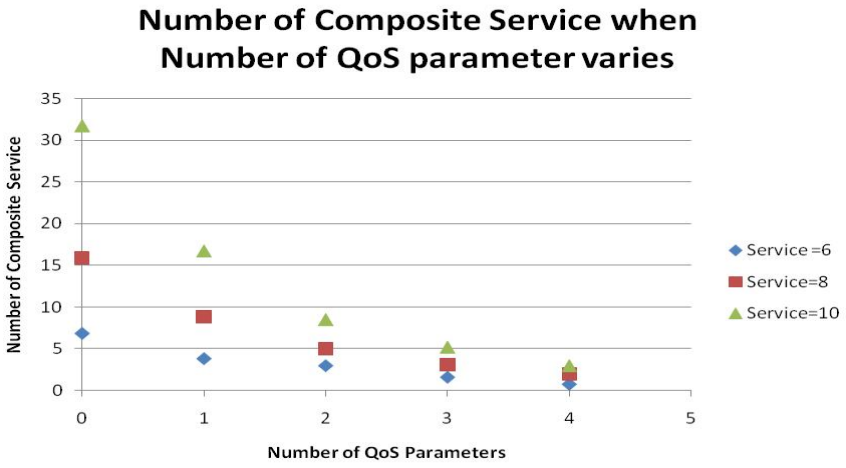

Figure 5.15: Number of Composite Service with different number of associated QoS parameters

in the number of produced composite services against the number of abstract services as the associated number of QoS parameters increased (refer to Figures 5.12, 5.13 and 5.14). This relationship of the number of composite service with the number of abstract services again suggests that it should be feasible to do QoS aware service composition.

5.2.3 Experiment-3 (Number of Composite Services for varying numbers of associated QoS parameters)

In this experiment setup, I found out how the total number of composite services varies when the number of QoS parameters changes. In this case, I fixed the total number of abstract services at 6 and in later cases at 8 and 10. It is noteworthy that the total number of composite services decreases with an increase of the number of QoS parameters associated with each abstract service (refer to Figure 5.15). The reason, again, is that the increased number of QoS parameters applied more constraints on producing composite services. This suggests that the QoS aware service composition can effectively reduce the total number of composite services, since it only allows production of useful composite services.

5.3 A Model for the Expected Number of Composed Services

In QoS aware service composition, the number of compositions varies with the number of abstract services. The developed model for the expected number of composite service can provide a rough estimate for how many composite services the composition manager can produce for a given number of abstract services and also for a given number of QoS parameters. Thus, this model gives a rough explanation of how the number of compositions varies with the number of abstract services and QoS parameters in the case of uniform distributions for QoS values.

Assume we have selected two services which have the same number (i.e K) of total QoS parameters. The functional and QoS properties of those services can be defined as,

First Service: S_1=(I_1, O_1, $q^R_{1,1}$, $q^R_{1,2}$, $q^R_{1,3}$, . . . , $q^R_{1,K}$, $q^P_{1,1}$, $q^P_{1,2}$, $q^P_{1,3}$, . . . , $q^P_{1,K}$)

and

Second Service: S_2=(I_2, O_2, $q^R_{2,1}$, $q^R_{2,2}$, $q^R_{2,3}$, . . . , $q^R_{2,K}$, $q^P_{2,1}$, $q^P_{2,2}$, $q^P_{2,3}$, . . . , $q^P_{2,K}$)

where,

I_i = Input type of service i

O_i = Output type of service i

$q^R_{i,j}$ = Required QoS values for for j-th QoS parameter of service i

$q^P_{i,j}$ = Provided QoS values for for j-th QoS parameter of service i

$q^P_{1,j}$ = Provided QoS value for j-th QoS parameter of selected First service

$q^R_{2,j}$ = Required QoS value for j-th QoS parameter of selected Second service

Q = Number of possible values for j-th QoS parameter

In QoS aware service composition, QoS compatibility for a particular QoS parameter only becomes successful when the Provided-QoS value of the j-th QoS parameter of the selected, First, service is greater than or equal to the Required-QoS value of the j-th QoS parameter of the selected, Second, service which is denoted as case 4 in the following text. To find out the probability of successful QoS checking, I have introduced 3 additional cases (cases 1, 2 and 3) which are defined as follows.

case1 = Provided-QoS value of j-th QoS parameter of selected First service is equal to the Required-QoS value of the j-th QoS parameter of selected Second service

case2 = Provided-QoS value of j-th QoS parameter of selected First service is different from the Required-QoS value of the j-th QoS parameter of selected Second service

case3 = Provided-QoS value of j-th QoS parameter of selected First service is greater than the Required-QoS value of the j-th QoS parameter of selected Second service

case4 = Provided-QoS value of j-th QoS parameter of selected First service is greater than or equal to the Required-QoS value of the j-th QoS parameter of selected Second service

P(case1) = P(Provided-QoS value of j-th QoS parameter of selected First service is equal to the Required-QoS value of the j-th QoS parameter of selected Second service) =

$P(q^{P}_{1,j} = q^{R}_{2,j}) = \frac{1}{Q}$

P(case2) = P(Provided-QoS value of j-th QoS parameter of selected First service is different from the Required-QoS value of the j-th QoS parameter of selected Second service) = $P(q^{P}_{1,j} \neq q^{R}_{2,j})$ = 1- $\frac{1}{Q}$

P(case3) = P(Provided-QoS value of j-th QoS parameter of selected First service is greater than the Required-QoS value of the j-th QoS parameter of selected Second service) = $P(q^{P}_{1,j} > q^{R}_{2,j} \mid \text{case2}) = \frac{(1-\frac{1}{Q})}{2}$

P(case4) = P(Provided-QoS value of j-th QoS parameter of selected First service is greater than or equal to the Required-QoS value of the j-th QoS parameter of selected Second service) = $P(q^{P}_{1,j} \geq q^{R}_{2,j})$

Assume that p (i.e. provided QoS values of selected First Service) and q (i.e. required QoS values of selected Second Service) are chosen from the same discrete uniform distribution with Q possibilities. Then $P(p \geq q) = \sum_{j=1}^{Q} P(q^{P}_{1,j} \geq q^{R}_{2,j}) = \sum_{j=1}^{Q} P(q = j)P(p \geq j) = \sum_{j=1}^{Q} (\frac{1}{Q})(\frac{j}{Q}) = \frac{Q+1}{2Q}$.

A service composition only takes place when the output type of the first service matches the input type of the second service ($O_1 = I_2$) along with successful QoS matching. If there are a total of T Input/Output types then:

P(Output type of selected First service matches the Input type of Second service and Provided-QoS values of First service is greater than or equal to corresponding Required-QoS values of selected Second service) = $P(O_1 = I_2 \text{ and } p \geq q) = \frac{1}{T} * \frac{(Q+1)}{2Q}$

If A is the total number of abstract services and P(A,T,Q) denotes the probability of service composition, then if we choose two abstract services uniformly at random then, we get a binomial distribution for service composition of the form, B(A(A-1), $\frac{(Q+1)}{2QT}$).

Knowing the number of compositions with this model would be used for capacity planning of composition manager (i.e. required number of CPUs, disks, RAMs etc.). This

model can also estimate the maximum number of QoS constraints which can be applied in service composition to produce expected number of composite services. In the next section, I have verified the correctness of this mathematical model with the help of experimental data.

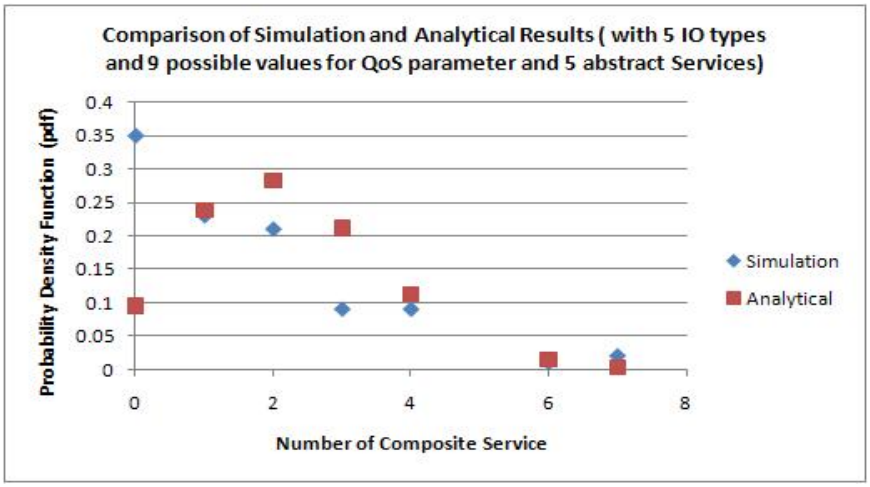

Figure 5.16: Mathematical Model vs Experimental Data-1

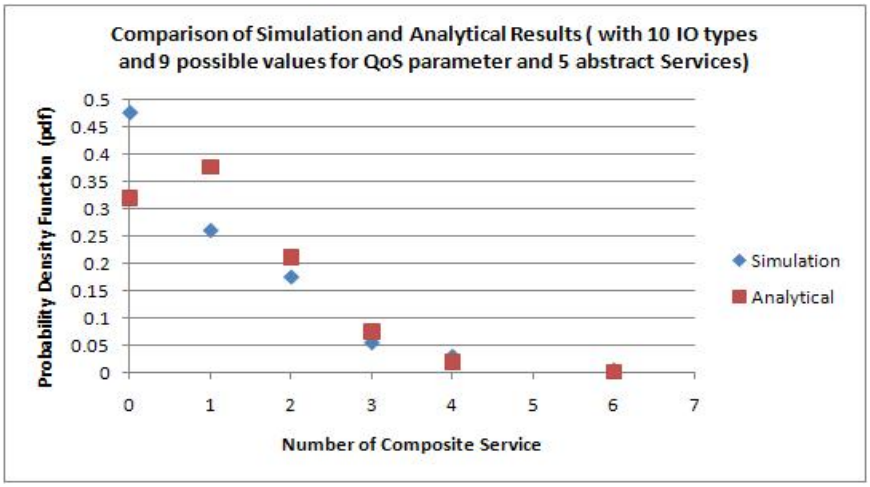

Figure 5.17: Mathematical Model vs Experimental Data-2

5.3.1 Model Data vs Experimental Data

The model provides only a rough approximation. In the experiment, QoS values and I/O types are drawn from a uniform distribution as in the model. However, the abstract services are created at the outset and re-used, in the sense that they are compared with each other abstract service. The sampling of QoS and I/O parameters therefore follows a complex pattern, whereas, in the model, they are simply sampled from a uniform distribution and therefore comprise a stream of independent and identically distributed random variables. In the experiment, the independence assumption is not satisfied. However, the model provides a good rough approximation which explains much of the behavior.

I compared my experimental data and the data obtained from my model for the expected number of composite services. The approximation appears to get better as the number of abstract services increases. In the first case (refer to Figure 5.16), there were a total of 5 IO types (to assign to the services) and in the second case (refer to Figure 5.17) there were total of 10 IO types. Other parameters were kept identical. In the first case, when the number of IO types was less (compared to the second case), the composition engine produced a higher number of composite services (in the experiment). As a result, the approximation was more accurate for higher numbers of composite services in the first case. In the second case, the approximation of lower number of composite services also became closer to experimental results since the composition engine produced fewer composite services than in first case. These graphs strongly suggest that the proposed model represents my composition prototype quite well.

5.3.2 Scalability Analysis

From the experimental results, I calculated that, over all experiments when there were no QoS parameters associated with services, each composite service had an average

service composition time (IO matching time plus QoS matching time) of 100 milliseconds. This service time increased to 166 milliseconds when there were 2 QoS parameters associated with each service and to 250 milliseconds when there were 4 QoS parameters associated with each service. Since there was no queuing delay in my simulation, I considered the arrival rate and the service rate to be identical. Thus, the arrival rate (λ) for different numbers of associated QoS parameters was:

- $\lambda^{(QoS=0)}$ = 10 services/second,
- $\lambda^{(QoS=2)}$ = 6 services/second, and
- $\lambda^{(QoS=4)}$ = 4 services/second

To determine scalability of my prototype in real life environment, I considered home as a persistent environment where there are less frequent arrival of new device with new service(s). Assume that, every 7 days there arrives a new service in the home environment i.e. the average arrival rate of a new service in home environment($\lambda_{(home)}$) = $\frac{1}{(7*24*3600)}$=1.65x10^{-6} services/second.
So, with a single composition engine,

- Maximum number of homes supported when there is no associated QoS,
  Max(n)$^{(QoS=0)}$ =$\frac{\lambda^{(QoS=0)})}{\lambda_{(home)}}$= 6.0x$10^6$ homes
- Maximum number of homes supported when there are 2 associated QoS,
  Max(n)$^{(QoS=2)}$ =$\frac{\lambda^{(QoS=2)}}{\lambda_{(home)}}$= 3.6x$10^6$ homes
- Maximum number of homes supported when there are 4 associated QoS,
  Max(n)$^{(QoS=4)}$ = $\frac{\lambda^{(QoS=4)}}{\lambda_{(home)}}$= 2.4x$10^6$ homes

So, my QoS aware service composition approach appears to be capable of supporting a reasonable number of concurrent requests and should be highly scalable in real life environments.

5.4 Regression Models

To find out an effective ranking function, I created an online survey and collected survey responses. With these survey responses, I developed seven different regression models, as described in chapter 4. I used the statistical package R [Wie] to find out the predicted values for each of these models and to compare those models. In R, the lm() function was used for linear regression models and the polr() function was used for ordinal regression models. The models along with the R functions are shown in Table 5.1 where Y is assumed as user responses.

There were 4 real-life scenario in the survey and in each scenario there were 5 functionally identical services with varying QoS characteristics and users were asked to rate the quality of the services based on their QoS preferences. For regression, I calculated normalized QoS values (i.e. q_1, q_2, q_3) from the associated QoS values (i.e. Q_1, Q_2, Q_3) of the offered services. For the QoS normalization in a particular scenario, I used the following function:

$$q_i = \frac{Q_i - Minimum\ of\ the\ values\ of\ QoS\ parameter_i}{Range\ of\ the\ values\ of\ QoS\ parameter_i (max-min)}$$

where, Q_i was the QoS value of the i-th QoS parameter before normalization and q_i was the QoS values of the i-th QoS parameter after normalization. Normalization was done in such a way that normalized QoS values always remained between 0 and 1 (where 0 was considered as the best QoS value and 1 was the worst) thus, I could use survey results of all 4 scenarios altogether for regression. Again, W_1, W_2, W_3 were the users' QoS preferences

or "weights" for corresponding QoS parameters and Y was the ranking of the users for services. Finally, I defined, the weighted QoS values (normalized), $A_1 = q_1 * W_1$, $A_2 = q_2 * W_2$, $A_3 = q_3 * W_3$.

Models	R functions for Regression
Model-A	lm(Y∼q_1+q_2+q_3+W_1+W_2+W_3)
Model-B	lm(Y∼A_1+A_2+A_3+q_1+q_2+q_3+W_1+W_2+W_3)
Model-C	lm(Y∼A_1+A_2+A_3+q_1+q_2+q_3+W_1+W_2+W_3+W_1*W_2*W_3)
Model-D	lm(Y∼A_1+A_2+A_3+q_1+q_2+q_3+W_1+W_2+W_3+W_1*W_2*W_3+A_1*A_2*A_3)
Model-E	lm(Y∼A_1+A_2+A_3)
Model-F	lm(Y∼A_1+A_2+A_3+A_1*A_2*A_3)
Model-G	lm(Y∼A_1+A_2+A_3+W_1*W_2*W_3)
Model-H	polr(as.ordered(Y)∼q_1+q_2+q_3+W_1+W_2+W_3)
Model-I	polr(as.ordered(Y)∼A_1+A_2+A_3+q_1+q_2+q_3+W_1+W_2+W_3)
Model-J	polr(as.ordered(Y)∼A_1+A_2+A_3+q_1+q_2+q_3+W_1+W_2+W_3+W_1*W_2*W_3)
Model-K	polr(as.ordered(Y)∼A_1+A_2+A_3+q_1+q_2+q_3+W_1+W_2+W_3+W_1*W_2*W_3+A_1*A_2*A_3)
Model-L	polr(as.ordered(Y)∼A_1+A_2+A_3)
Model-M	polr(as.ordered(Y)∼A_1+A_2+A_3+A_1*A_2*A_3)
Model-N	polr(as.ordered(Y)∼A_1+A_2+A_3+W_1*W_2*W_3)

Table 5.1: Regression Models

To compare the regression models, I used the probability function, P(|Z-R|≤ x). P(|Z-R|≤ x) is the probability of a prediction deviating from the human response in the range [1,5] (corresponding to survey question responses) to within x, where Z, in [1,5], is the predicted response and R, in [1,5], is the actual response.

Among the models, Model-I was found to perform the best (within 1 deviation) with the probability of prediction deviation (from human response) is 0.82 within 1(refer to Table 5.3. Though Model-K has a prediction deviation probability of .81 (slightly less than Model-I) within 1, Model-I has the highest probability of exact prediction which is

P(|Z-R|≤ x)	Model-A	Model-B	Model-C	Model-D	Model-E	Model-F	Model-G
x=1	0.59	0.63	0.63	0.63	0.53	0.53	0.54
x=2	0.92	0.93	0.92	0.92	0.92	0.92	0.91
x=3	0.99	0.99	0.99	0.99	0.99	0.99	0.99
x=4	1.00	1.00	1.00	1.00	1.00	1.00	1.00
x=5	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 5.2: Analysis Results on Linear Regression Models

P(|Z-R|≤ x)	Model-H	Model-I	Model-J	Model-K	Model-L	Model-M	Model-N
x=0	0.36	0.37	0.37	0.38	0.27	0.31	0.35
x=1	0.80	0.82	0.81	0.81	0.73	0.73	0.76
x=2	0.94	0.94	0.94	0.94	0.93	0.92	0.93
x=3	0.99	0.99	0.99	0.99	0.99	0.98	0.98
x=4	1.00	1.00	1.00	1.00	1.00	1.00	1.00
x=5	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 5.3: Analysis Results on Ordinal Regression Model

.38. Model-H is almost as good and it is significant that removing the A_is from Model-I has only minimal effect, especially since those were the only terms in my original model (i.e. before survey). My results suggest that I could improve my ranking function significantly with the regression model-I. Thus, the final model proposed for use by the UPMM is,

$$p(\text{rank}_i)=\alpha_i+\sum_{j=1}^{k}\beta_j^q q_j +\sum_{j=1}^{k}\beta_j^W W_j+\beta_1^A A_1 +\beta_1^A A_1+...+\beta_k^A A_k$$

Here, i is the index of the ranking, j is the index of the QoS parameter, k is the total number of QoS parameter, α_i is the intercept for a particular ranking. β_j^q, β_j^W and β_j^A are the coefficients. The values of these coefficients for my survey results are shown in Table 5.4. The Model-I looks like best we can predict is within 1, essentially giving us 3 divisions instead of 5.

This ranking model can be used to find the rank of an offered service (among other

Coefficients	Values
β_1^A	0.03
β_2^A	1.49
β_3^A	-0.8
β_1^q	-0.42
β_2^q	-6.3
β_3^q	1.49
β_1^W	.009
β_2^W	-0.8
β_3^W	0.46

Table 5.4: Regression Coefficients of Model-I

available services) in a particular scenario. This model also tells us which factors are more sensitive in QoS aware service composition. I successfully used this model to predict the ranking for services in my QoS aware service composition prototype.

Summary

From the results shown above, it can be said that this QoS aware composition is an efficient strategy. Though this model introduces QoS matching overhead along with IO type matching overhead during composition, the total matching time remains acceptable and appears to be highly scalable using modest computing equipment. Finally with the help of the regression models I developed, QoS aware service composition should be able to offer the most useful services to the users.

Chapter 6

Conclusion and Future Work

Service composition is an important aspect in pervasive computing environments. QoS-aware service composition provides a way of obtaining the most useful compositions in pervasive environments. Further, this supports full automation of the composition process can make persistent computing accessible to a wider range of users and can enhance the use of existing resources. Implementing ontologies for QoS aware service composition makes the composition process easier to automate and thus more useful.

6.1 Contribution

In this thesis, I have made the following contributions:

- I have developed a fully automated QoS aware service composition prototype which can support external services along with local services in any persistent environment.
- My prototype can consider any number of QoS parameters and match them to user QoS preferences to produce useful services.

- My prototype can offer only the best services to users using an efficient ranking function. I developed this ranking function using a regression model which satisfies users' QoS preferences. This ranking function automates the process of offering suitable services to end users and performs better than the original ranking scheme proposed.

- I have developed a mathematical model to predict the total number of composite services that my QoS aware service composition prototype can produce for a given number of abstract services and for a given number of QoS parameters.

Using my prototype, users can more easily exploit the best suitable services for them with less involvement in the composition process. The use of fully automated QoS aware service composition should be beneficial to any pervasive computing environment.

6.2 Future Work

My contribution in this thesis can be extended in following ways:

- In this work, I didn't consider the actual execution of services. So, the grounding information associated with each service can be used easily for real execution of the services.

- During development it was assumed that the service broker would only register those services whose IO and QoS types are supported by the ontology in the persistent environment. If ontology integration support can be developed, then the service broker could register services with any IO and QoS types and integrate the needed ontology components into the persistent environment.

- It might be useful to explore more complex regression models.

- It would be valuable to formally assess the ranking model produced for different QoS scenarios.

Appendix A

Acronyms and Definition

Abstract Service	Abstract services are basic services which can not be decomposed, i.e. “atomic”.
Composite Service	Composite service are composed of two or more abstract and/or other composite services.
Component Service	The participating services in a composite services are called component service.

Appendix B

Survey Questions

1. While watching an online movie, how important are the following characteristics to you?

	Very Important	Important	Average	Doesn't Matter Much	Not At All
Cost	○	○	○	○	○
Video/Image Quality	○	○	○	○	○
Number of advertisements included	○	○	○	○	○

When you want to watch a movie online, it may be available from multiple sources (e.g. Source A, Source B, Source C, etc.). Each movie source offers the same movie but with different service characteristics (i.e. Total Advertisement time, Video/Image Quality, Cost as shown in the table below).

Movie Source	Total Advertisement Time (in minutes)	Video/Image Quality *	Cost (in dollars)
Source A	3	Excellent	7
Source B	5	Fair	4
Source C	4	Bad	1
Source D	1	Poor	3
Source E	2	Good	8

*LEGEND	
TERM	DESCRIPTION
Excellent	No perceptible flaws
Good	Perceptible but not annoying flaws
Fair	Slightly annoying flaws
Poor	Annoying flaws
Bad	Very Annoying flaws

2. Please give your impression of each movie source in the table above considering your service preferences for watching movies from the first question.

	Excellent	Good	Average	Poor	Very Poor
Source A	○	○	○	○	○
Source B	○	○	○	○	○
Source C	○	○	○	○	○
Source D	○	○	○	○	○
Source E	○	○	○	○	○

Figure B.1: Movie Source Selection Scenario for User Preference Survey

1. When selecting a printing service, how important are the following characteristics to you?

	Very Important	Important	Average	Doesn't Matter Much	Not at All
Quality of the printed material	○	○	○	○	○
Time to print	○	○	○	○	○
Printing cost	○	○	○	○	○

When you want to print, you can select different print services (e.g. PrintService A, PrintService B, PrintService C, etc.). Each print service offers the same capabilities but with different service characteristics (i.e Printing Cost, Quality of Printed Material, and Time to Print as shown in the table below).

PrintService	Total Time to Print a large document (in minutes)	Print Quality*	Printing Cost per page (in cents)
PrintService A	3	Excellent	15
PrintService B	5	Fair	10
PrintService C	4	Bad	1
PrintService D	1	Poor	5
PrintService E	2	Good	20

*LEGEND	
TERM	DESCRIPTION
Excellent	No perceptible flaws
Good	Perceptible but not annoying flaws
Fair	Slightly annoying flaws
Poor	Annoying flaws
Bad	Very Annoying flaws

2. Please give your impression of each print service in the table above considering your service preferences for printing from the first question.

	Excellent	Good	Average	Poor	Very Poor
PrintService A	○	○	○	○	○
PrintService B	○	○	○	○	○
PrintService C	○	○	○	○	○
PrintService D	○	○	○	○	○
PrintService E	○	○	○	○	○

Figure B.2: Print Service Selection Scenario for User Preference Survey

1. When selecting an Internet-based telephone service (e.g. Skype), how important are the following service characteristics to you?

	Very Important	Important	Average	Doesn't Matter Much	Not at All
Audio Quality	○	○	○	○	○
Monthly Cost	○	○	○	○	○
Monthly Free Talk-time	○	○	○	○	○

When you choose a Telephone service, it may be available from different companies (e.g. Company A, Company B, Company C, etc.). Each Telephone company offers the same Telephone service but with different service characteristics (i.e. Monthly Free Talk-time, Service Quality, Monthly Cost as shown in the table below).

Companies	Monthly Free Talk Time (in minutes)	Service Quality*	Monthly Cost (in dollars)
Company A	120	Good	10
Company B	200	Bad	6
Company C	150	Fair	2
Company D	100	Poor	5
Company E	80	Excellent	8

*LEGEND	
TERM	DESCRIPTION
Excellent	No perceptible flaws
Good	Perceptible but not annoying flaws
Fair	Slightly annoying flaws
Poor	Annoying flaws
Bad	Very Annoying flaws

2. Please give your impression of each Telephone company in the table above considering your service preferences for selecting telephone services from the first question.

	Excellent	Good	Average	Poor	Very Poor
Company A	○	○	○	○	○
Company B	○	○	○	○	○
Company C	○	○	○	○	○
Company D	○	○	○	○	○
Company E	○	○	○	○	○

Figure B.3: Telephone Service Company Selection Scenario for User Preference Survey

1. While doing a video conference (i.e online meeting between two or more people), how important are the following characteristics to you?

	Very Important	Important	Average	Doesn't Matter Much	Not at All
Maximum number of allowed participants	○	○	○	○	○
Hourly Cost	○	○	○	○	○
Video and Audio Quality	○	○	○	○	○

When you want to do a video conference, you may be able to choose from multiple online providers (e.g. Provider A, Provider B, Provider C, etc.). Each provider offers video conferencing but with different service characteristics (i.e. Cost per Hour, Audio and Video Quality, Maximum allowed number of participants as shown in the table below).

Providers	Maximum allowed participants	Video & Audio Quality*	Cost per hour (in dollars)
Provider A	9	Excellent	5
Provider B	12	Fair	6
Provider C	2	Bad	2
Provider D	6	Poor	3
Provider E	5	Good	4

*LEGEND	
TERM	DESCRIPTION
Excellent	No perceptible flaws
Good	Perceptible but not annoying flaws
Fair	Slightly annoying flaws
Poor	Annoying flaws
Bad	Very Annoying flaws

2. Please give your impression of each provider in the table above considering your service preferences for video conferencing from the first question.

	Excellent	Good	Average	Poor	Very Poor
Source A	○	○	○	○	○
Source B	○	○	○	○	○
Source C	○	○	○	○	○
Source D	○	○	○	○	○
Source E	○	○	○	○	○

Figure B.4: Video Conference Source Selection Scenario for User Preference Survey

Bibliography

[All] HomePNA Alliance. HomePNA Home Networking. http://www.homepna.org. Accessed on August 10, 2009.

[BB04] Jacob T. Biehl and Brian P. Bailey. ARIS: An interface for application relocation in an interactive space. In *GI: Proceedings of Graphics Interface*, pages 107–116. Canadian Human-Computer Communications Society, 2004.

[BBEL07] Andrę Bottaro, Johann Bourcier, Clement Escoffier, and Philippe Lalanda. Context-aware service composition in a home control gateway. In *International Conference on Pervasive Services*, pages 223–231. IEEE, 2007.

[BHSR04] Christian Becker, Marcus Handte, Gregor Schiele, and Kurt Rothermel. PCOM: A component system for pervasive computing. In *Proceedings of the Second Annual Conference on Pervasive Computing and Communications (PerCom)*, pages 67–76. IEEE, 2004.

[BKP06] John Buford, Rakesh Kumar, and Greg Perkins. Composition trust bindings in pervasive computing service composition. In *PERCOMW: Proceedings of the 4th annual International Conference on Pervasive Computing and Communications Workshops*, pages 261–266. IEEE Computer Society, 2006.

[Bri] Dan Brickley. RDF vocabulary description language: RDF Schema. URL: www.w3.org/TR/rdf-schema/.

[CIJ+00] F. Casati, S. Ilnicki, L.J. Jin, V. Krishnamoorthy, and M.C. Shan. Adaptive and dynamic service composition in e-Flow. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 13–31, 2000.

[CMS] CMSimple. Project nightingale. http://praxis.cs.usyd.edu.au/ peteris/?Projects/Virtual+Personal+Server. Accessed on January 10, 2010.

[Con] W3C Consortium. OWL-S: Semantic markup for web services. http://www.w3.org/Submission/OWL-S/. Accessed on August 10, 2009.

[CP] Clark and Parsia. Pellet: Owl 2 reasoner for java. http://clarkparsia.com/pellet. Accessed on August 15, 2009.

[DD04] Julian Day and Ralph Deters. Selecting the best web service. In *Proceedings of the Grad Symposium, CS Dept, University of Saskachewan*, 2004.

[Deva] Jini Developers. Community resource for Jini technology. http://www.jini.org.

[Devb] UPnP Developers. UPnP forums. http://www.upnp.org.

[DFR04] N J Davies, D Fensel, and M Richardson. The future of web services. *BT Technology Journal*, 22:118–130, 2004.

[dmo] dmoz. Open directory project. http://www.dmoz.org/Computers/Internet/Protocols/IP/. Accessed on May 8, 2010.

[eaa] David Martin et al. OWL-S semantic markup for web languages. http://www.w3.org/Submission/OWL-S/.

[eab] Thompson et al. Ad-hoc networking support for pervasive collaboration. http://ubicomp.org/ubicomp2004/adjunct/posters/thompson.pdf.

[ea01] Steven D. Gribble et al. The Ninja architecture for robust internet-scale systems and services. *Computer Networks*, 35(4):473–497, 2001.

[For] Jini Forum. Jini architecture overview. http://www.jini.org/wiki/. Accessed on August 15, 2009.

[FS04a] Keita Fujii and Tatsuya Suda. Component service model with semantics (CoS-MoS): A new component model for dynamic service composition. In *SAINT-W: Proceedings of the International Symposium on Applications and the Internet-Workshops (SAINT Workshops)*, pages 348–354. IEEE, 2004.

[FS04b] Keita Fujii and Tatsuya Suda. Dynamic service composition using semantic information. In *ICSOC: Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 39–48. ACM, 2004.

[FS05] K. Fujii and T. Suda. Semantic-based dynamic service composition. *IEEE Journal on Selected Areas in Communications*, 23(12):2361–2372, 2005.

[GNY04] Xiaohui Gu, Klara Nahrstedt, and Bin Yu. Spidernet: An integrated peer-to-peer service composition framework. In *Proceedings of the 13th International Symposium on High Performance Distributed Computing*, pages 110–119. IEEE, 2004.

[Groa] Amigo Research Group. Amigo project. http://www.hitech-projects.com/euprojects/amigo/. Accessed on August 15, 2009.

[Grob] Bluetooth Special Interest Group. Bluetooth program, initiatives and wireless

technology development. http://www.bluetooth.org. Accessed on August 15, 2009.

[Her] Ivan Herman. Web Ontology Language (OWL). www.w3.org/tr/wsdl20-rdf. Accessed on August 15, 2009.

[HSB] Ivan Herman, Ralph Swick, and Dan Brickley. Resource description framework. http://www.w3.org/RDF/. Accessed on August 15, 2009.

[IP] Maryland Information and Network Dynamics Lab Semantic Web Agents Project. Pellet. http://www.mindswap.org/2003/pellet/. Accessed on August 15, 2009.

[JF02] Brad Johanson and Armando Fox. The Event Heap: A coordination infrastructure for interactive workspaces. In *Proceedings Fourth Workshop on Mobile Computing Systems and Applications*, pages 82–93. IEEE, 2002.

[KKR+07] Jarmo Kalaoja, Julia Kantorovitch, Ioanna Roussaki, Dimitrios Tsesmetzis, and Ioannis Papaioannou. *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, chapter Ontology Modelling for Ambient Intelligent Home Environments, pages 15–16. Springer, 2007.

[KKS05] Swaroop Kalasapur, Mohan Kumar, and Behrooz Shirazi. Seamless service composition (SeSCo) in pervasive environments. In *MSC: Proceedings of the First International Workshop on Multimedia Service Composition*, pages 11–20. ACM, 2005.

[KKS06a] Swaroop Kalasapur, Mohan Kumar, and Behrooz Shirazi. Evaluating service oriented architecture (SOA) in pervasive computing. In *Proceedings of the*

Fourth Annual IEEE International Conference on Pervasive Computing and Communication, pages 276–285, 2006.

[KKS06b] Swaroop Kalasapur, Mohan Kumar, and Behrooz Shirazi. Evaluating service oriented architectures (SOA) in pervasive computing. In *PERCOM: Proceedings of the Fourth Annual International Conference on Pervasive Computing and Communications*, pages 276–285. IEEE Computer Society, 2006.

[LFBW04] George Lee, Peyman Faratin, Steven Bauer, and John Wroclawski. A user-guided cognitive agent for network service selection in pervasive computing environments. In *Proceedings of the Second Annual Conference on Pervasive Computing and Communications (PerCom)*, pages 219–228. IEEE, 2004.

[Lok] Seng Loke. *Context-aware pervasive systems: Architecture for a new breed of applications*. Auerbach Publications.

[Men03] Prasad K. P. Menon. Ubiquitous file system protocol. Master's thesis, University of Florida, 2003.

[MFAM04] A. Mingkhwan, P. Fergus, O. Abuelmaatti, and M. Merabti. Implicit functionality: Dynamic services composition for home networked appliances. In *IEEE International Conference on Communications*, pages 43–47, 2004.

[MGI06] S. B. Mokhtar, N. Georgantas, and V. Issarny. COCOA: Conversation-based service composition for pervasive computing environments. In *ACS/IEEE International Conference on Pervasive Services*, pages 29–38, 2006.

[MPG+08] Sonia Ben Mokhtar, Davy Preuveneers, Nikolaos Geogantas, Valerie Issarny, and Yolande Berbers. EASY: Efficient semanAntic Service discoverY in perva-

sive computing environments. *Journal of System and Software*, 81(5):785–808, 2008.

[MPL03] R. Masuoka, B. Parsia, and Y. Labrou. Task Computing - the semantic Web meets pervasive computing. In *Proceedings of the 2nd International Semantic Web Conference (ISWC)*, pages 866–881, 2003.

[NYT04] Jin Nakazawa, Junichi Yura, and Hideyuki Tokuda. Galaxy: A service shaping approach for addressing the hidden service problem. In *Proceedings of the Second Workshop on Software Technologies for the Future Embedded and Ubiquitous Systems*, pages 35–39. IEEE Computer Society, 2004.

[Osc] Oscar. An OSGi framework implementation. http://oscar.objectweb.org/. Accessed on August 15, 2009.

[PB05] Davy Preuveneers and Yolande Berbers. Semantic and syntactic modeling of component-based services for context-aware pervasive systems using OWL-S. In *Proceedings of the 1st International Workshop on Managing Context Information in Mobile and Pervasive Environments (MCMP)*, pages 30–39, 2005.

[PD03] M.P. Papazoglou and D.G. *Service-oriented Computing*, chapter Special section in Communications of the ACM. ACM Press, 2003.

[PG06a] H. Pourreza and P. Graham. On the fly service composition for local interaction environemnts. In *Proceedings of the Fourth IEEE International Conference on Pervasive Computing and Communications Workshops, PerWare*, pages 393–398. IEEE, 2006.

[PG06b] Hossein Pourreza and Peter Graham. On the fly service composition for local interaction environments. In *PERCOMW: Proceedings of the 4th Annual Inter-*

national Conference on Pervasive Computing and Communications Workshops, pages 393–398. IEEE, 2006.

[RHC+02] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mobile Computing and Communications Review*, 6(4):65–67, 2002.

[RHR+01] Manuel Roman, Christopher K. Hess, Anand Ranganathan, Pradeep Madhavarapu, Bhaskar Borthakur, Prashant Viswanathan, Renato Cerqueira, Roy H. Campbell, and M. D Mickunas. Gaia OS: An infrastructure for active spaces. Technical report, University of Illinois, USA, 2001.

[TB05] D.C. Thomas and Little Prithwish Basu. A novel approach for execution of distributed tasks on mobile adhoc networks. *Journal on Selected Areas in Communications*, 23(12):2361–2372, 2005.

[TBM+] Graham Thomson, Sebastein Bianco, Sonia Ben Mokhtar, Nikolaos Geogantas, and Valerie Issarny. *Constructing Ambient Intelligence*, chapter Amigo Aware Services, pages 385–390.

[TM05] M.S. Thompson and S.F. Midkiff. Service description for pervasive service discovery. In *Proceedings of the 25th International Conference on Distributed Computing Systems Workshops*, pages 273–279. IEEE, 2005.

[TR-] DSL Forum TR-069. CPE WAN management protocol. http://www.broadband-forum.org/technical/download/TR-069Amendment1.pdf. Accessed on August 15, 2009.

[Wie] WU Wien. The r project for statistical computing. http://www.r-project.org/. Accessed on November 8, 2009.

[WPS+03] Dan Wu, Bijan Parsia, Evren Sirin, James Hendler, and Dana Nau. *Automating DAML-S web services composition using SHOP2*, chapter Semantic Web Services, pages 195–210. Springer Berlin, 2003.

[xEEd] x10 Europe Equipments developer. Wireless solutions for 230 volts. http://www.x10europe.com. Accessed on August 15, 2009.