UNIVERSITY OF SOUTHAMPTON

A System Architecture to Provide Enhanced Grid Access for Mobile Devices

by

Tao Guan

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Engineering, Science and Mathematics

School of Electronics and Computer Science

July 2008

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Tao Guan

More integration and computing power are required to support the proliferation of mobile devices. While new-generation mobile devices improve their absolute capabilities, there is no doubt that creating complicated applications remains a challenging objective because mobile devices are typically resource-constrained, relative to their static counterparts. One possible solution is that mobile devices use Grid services to enable users to access various distributed resources automatically on demand. The combination of mobile and Grid computing has the potential to establish a new field where high performance Grid facilities are accessed through resource-limited mobile devices. A great number of challenges must be solved to realize the vision of building the bridge between Grid services and mobile devices.

This thesis presents the author's research work that aims to enable users to accomplish complex tasks through their handheld devices by utilizing distributed resources in service-oriented Grid environments. A context-aware framework has been built with Semantic Web technologies to support intelligent interaction between mobile users and Grid services and a semantic matching framework has been implemented to facilitate effective Grid service discovery. Based on the above, a system architecture has been developed to provide enhanced Grid access for mobile devices. The middleware in the system architecture hides the diversity of heterogeneous mobile devices, enables the required Grid services to be discovered in a flexible way, and provides a reliable task execution mechanism. The performance of the system architecture has been evaluated using both comparison and simulation.

Although there is no doubt that there are still a number of challenges that must be overcome before the long-term vision of bringing Grid services into the world of ambient intelligence comes into existence, the system architecture designed, implemented, and evaluated in this thesis represents an important step to realize a variety of new computing models (in particular Grid computing) in the mobile computing world.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my sincerest thanks to my supervisor Ed Zaluska and David De Roure for their inspiration, guidance and support throughout the period of this PhD, and for the research environment that they have facilitated.

I would also like to thank my colleagues in the IAM group, for their stimulating discussion and invaluable support I received during the first period of this research.

Finally, I wish to thank my friends and family for all their enthusiastic encouragement and support throughout my life without which I could not complete this thesis.

Chapter 1

Introduction

1.1 New Computing Models

In the past several years, a number of new computing models have been proposed. A major objective of Grid computing [1] [2] is to coordinate resource sharing in dynamic, multi-institutional virtual organizations [3], enabling heterogeneous resources to be aggregated and exploited to accomplish new functionalities and capabilities. As a service-oriented approach to Grid computing is increasingly adopted, many systems able to discover Grid resources on-the-fly and access them dynamically become possible. The concept of Grid infrastructure has already been applied in areas such as high-energy physics [4], bio-medicine [5], aerospace and earth sciences [6], health-care [7], learning [8] and is continuing to evolve and expand. For many applications, the Grid has become the most effective computing platform for solving complex scientific problems and providing high-performance distributed applications.

Mobile computing [9], on the other hand, is an extension of traditional distributed and desktop computing, seamlessly integrating various computing systems into our daily lives, to provide information and services "anywhere, anytime". With ever decreasing costs and increasing functionalities in small-sized chips, mobile handheld devices (e.g. smart phones and Personal Digital Assistants) enrich our daily lives and play vital roles in personal and business productivity. Today, mobile computing is an active and evolving research field, which has already achieved mobile networking, mobile information access, support for adaptive applications, system-level energy saving techniques, location sensitivity etc. [10].

However, there is another trend noticeable in the modern computing technology behind the proliferation of consumer electronic devices: still more integration and computation power are required for increasingly complex applications. While new-generation mobile devices improve their absolute capabilities [11], there is no doubt that creating

complicated applications on them remains a challenging objective because such devices are typically resource-constrained relative to their static counterparts (e.g. desktops, workstations), with limited processing, memory, storage, energy, and network resources.

1.2 Integration between Mobile Devices and Grid Services

A possible solution to the shortfall in required processing power is that mobile devices make use of Grid services to enable users to access distributed computational resources automatically on demand. Various Grid services can enhance the capabilities of ubiquitous mobile devices so that complicated tasks can be completed through user handheld devices. For example, the traveler's smart phone with a built-in camera can produce a large volume of data which will need to be processed if any special tasks are required. Large volumes of data demand significant computational power (e.g. image processing, location recognition) which can be best supplied by Grid services. The traveler can use his smart phone to discover local available Grid services and submit a complex request. The Grid will assist the traveler to achieve the complex task through distributed resources available anywhere in the world and return the results to his smart phone. In addition to providing new application opportunities for ubiquitous users, offloading complex tasks from resource-limited devices to the Grid has the potential to save energy, storage space, memory capability, and processing cycles, hence possibly further reducing the size, weight, and cost of mobile and pervasive devices.

The concept of Grid service on mobile devices can benefit from another significant movement in computing, the move toward machine-processable explicit knowledge as exemplified by the Semantic Web [12]. Semantic Web technologies are already being used within mobile computing, for such tasks as representing context information which requires the description of suitable ontologies. An integration between Semantic Web technologies and Grid computing is also recognized, and several "Semantic Grid" [13] projects have demonstrated a high degree of easy-to-use and seamless automation to facilitate flexible collaborations and computations on a global scale [14]. Semantic Web technologies have the potential to provide a very considerable degree of automatic processing, interoperation and integration, which is a central requirement of the necessary system infrastructure to allow mobile devices to use Grid services effectively.

Mobile devices form the intersection between the physical world and the digital world [15]. In this view, the digital world of the Grid meets the physical world through a variety of sensors, instruments and interfaces. As a result of several case studies from various "e-Science" projects [16] , we can conclude that mobile devices need the Grid for computation and integration, while the Grid needs mobile devices to interface with the physical world. The Semantic Web provides the necessary automation and interoperability required to build an ambient intelligence infrastructure. Actually, it

is the interaction of the Semantic Grid and the physical world (interacting through pervasive and mobile devices) that will enable us to realize this new concept of "Ambient Intelligence", the notion of intelligence in the surrounding environment supporting the activities and interactions of users [17].

1.3 Challenges of the Integration

The combination of these two computing models has the potential to realize a very significant development in the adoption of high performance Grid accessed through mobile devices. At first glance, this combination does not seem either efficient or appropriate. Clients that need to interact with Grid resources before they can accomplish a task will be required to install and utilize Grid client end libraries. At present, the existing Grid client libraries are relatively resource-intensive when compared with the limitations of mobile devices. Moreover, most of the current Grid applications have been developed with the assumption that the end-systems possess sufficient resources for the task at hand and the communication between clients and resource providers will be reliable.

In a mobile computing environment, users are able to enter the range covered by services and access ubiquitous resources with their mobile devices conveniently and smoothly. It will be the responsibility of the computing environment to detect user presence or absence and configure services automatically based on various context information. If Grid services are included in the mobile computing environment, they are also required to be made context aware. For example, Grid services will need to be customized and provided in different ways under different application cases. One of the key features of a mobile Grid environment [18] is that it is highly dynamic: mobile users and Grid services can be integrated on-the-fly, and Grid services are required to be located, acquired, composed, and coordinated depending on various context information such as the mobile device capability and the user location. Self-management, self-configuration, and self-adaptation must also be taken into consideration for the mobile Grid environment.

Considering the assumptions of conventional Grid computing and the highly dynamic requirement of the mobile computing environment, it is quite evident that a great number of challenges must be solved to realize the vision of building the bridge between Grid services and mobile devices. The detailed challenges are discussed in [17], and the following are some key points:

- A framework is required to bring mobile devices into the service-oriented Grid environment in a flexible, open and interoperable way.
- A task execution mechanism is required to be built so that complex tasks can be performed by invoking appropriate Grid services.

- Grid and mobile computing have similar challenges in terms of service description, discovery and composition at the appropriate level of abstraction. The description mechanisms have not been standardized, and the techniques for service discovery and composition are not mature at this time.
- Because of the dynamic nature of mobile computing environment, context awareness is important for achieving the goal of providing Grid services that are appropriate for mobile users at the right time, in the right place, at the right device, in the right format.

1.4 Thesis Contributions

This thesis presents the author's research work that aims to enable users to accomplish complex tasks through their resource-limited handheld devices by utilizing distributed resources in service-oriented Grid environments. A system architecture is designed and implemented to support task offloading to relatively resource-rich machines. The middleware in the system architecture hides the diversity of heterogeneous mobile devices, enables the required Grid services to be discovered, and provides a reliable task execution mechanism to assist users to interact with Grid services. Finally, a number of Petri Nets models are built and evaluated in order to estimate the system performance.

The core contributions of this research are outlined as below:

- Building a context-aware framework with Semantic Web technologies and related supporting tools.

  The seamless interaction between mobile users and Grid services demands that the computer systems have the ability of understanding the context information of the computing environment. By defining a shared context model and integrating various context information into a public knowledge base, functionalities such as explicit context representation, context querying and context reasoning can be provided. The context-aware framework is the underlying component of the system architecture, based on which an intelligent interaction mechanism is supported.

- Designing a semantic service matching middleware for Grid service discovery.

  Existing service discovery mechanisms do not support flexible matching between service advertisements and requests, and users can only locate services on the basis of the syntactical equivalence of keywords or strings. To solve this problem, a semantic approach for service description and service discovery is presented. The extended OWL-S language is used to describe a variety of service attributes explicitly, and the logic reasoning mechanism is used to check the concept similarity. Compared to existing service discovery protocols, the semantic service discovery

mechanism supports flexible matching between service attributes and service requirements based on user-expected matching level, and offers ranking information to select the most appropriate service. The semantic service matching middleware is also the underlying component of the system architecture, providing an interface for users or other middleware to find required Grid services.

- Developing a system architecture to provide enhanced Grid access for mobile devices.

  Because existing Grid client middleware tools are not suitable for resource-limited mobile devices, a system architecture is developed in which mobile users offload their tasks to Grid gateways. Mobile devices with different hardware and software equipments can be integrated into the system architecture, and users do not need to know the prior knowledge of services deployed in the Grid environment. The mobile deputy middleware on the Grid gateway is responsible for executing submitted tasks and interacting with the required distributed resources on behalf of mobile devices. Characteristics such as offline processing and seamless interaction are supported by the system architecture. Offloading the tasks involving the Grid service invocation improves the system performance compared to invoking services directly from mobile devices because the overhead processing on mobile devices and the necessary message transfer between mobile clients and service providers are eliminated. The experimental results based on the test application scenarios clearly demonstrate the benefit of using the system architecture to execute complex tasks for mobile users.

- Modeling the interaction with Stochastic Petri Nets and evaluating the system performance.

  The simulation method is adopted in order to evaluate the system performance. A number of interaction models have been built with non-Markovian Stochastic Petri Nets, which are analyzed using the WebSPN tool. The simulation results confirm that offloading tasks required to access Grid resources to Grid gateways improves the system performance compared with invoking Grid services directly from mobile devices.

1.5 Thesis Structure

The remainder of this thesis is organized as follows:

The chapter that follows introduces briefly the history and the current development of both Grid computing and pervasive computing. Semantic Web technologies, connecting the gap between the human-understandable data and the machine-understandable, are also introduced because they are considered important for the development of both pervasive and Grid computing.

Chapter 3 describes two kinds of scenarios: an information-access scenario and a work-assistant scenario. Both of these scenarios have the potential demands that users need to utilize Grid services through their handheld devices to perform complicated tasks. Having identified the dependent relationship between Grid services and mobile devices, a number of common requirements of implementing the vision of integrating mobile devices into Grid environments have been discussed.

The intelligent interaction between mobile users and Grid services is required because of the dynamic nature of users, devices, and Grid services. Both service consumers and service providers need to share their knowledge so that various context information of the system can be acquired to support automatic behaviors. In order to address this issue, a context-aware framework is discussed in chapter 4, which represents information in ways that are suitable for machine processing, context query and reasoning.

An important challenge of enhancing mobile device capabilities using Grid services is that mobile devices need to locate, select and invoke the appropriate Grid services in an automatic and flexible way. In chapter 5, a semantic approach for Grid service description and discovery is discussed. Both service descriptions and requests are expressed in Semantic Web languages so that the service matching engine is able to discover similarities by using logic reasoning techniques.

Based on the context-aware framework and the semantic approach for Grid service discovery which are discussed in the above chapters, a system architecture which integrates mobile devices into the service-oriented Grid environment is presented in chapter 6. The overview, the internal components, and the detailed implementation of system architecture are discussed. In order to test the system functionalities, three sample applications are presented. The experimental results demonstrate the benefits of the system architecture developed.

To evaluate and analyze the performance of the system architecture, chapter 7 models the sequence of operations between mobile devices, the Grid gateway, and Grid services using non-Markovian Stochastic Petri Nets. The aims are to compare the response time for different Grid clients, optimize the interaction strategy, and analyze the comprehensive system performance.

Chapter 8 concludes by reviewing the current research work and discussing the future work necessary to realize fully "Ambient Intelligence".

1.6 Publications

During the research, the following papers have been published:

- T. Guan, E. Zaluska, and D. DeRoure. A Grid Service Infrastructure for Mobile Devices. In Proceedings of 1st Semantic Knowledge and Grid conference, Beijing, China, 2005

- T. Guan, E. Zaluska, and D. DeRoure. Extending Pervasive Devices with the Semantic Grid: A Service Infrastructure Approach. In Proceedings of The Sixth IEEE International Conference on Computer and Information Technology, Korea, 2006

- T. Guan, E. Zaluska, and D. DeRoure. An Autonomic Service Discovery Mechanism to Support Pervasive Device Accessing Semantic Grid. In Proceedings of Fourth International Conference on Autonomic Computing (ICAC'07), Jacksonville, US, 2007

- T. Guan, E. Zaluska, and D. DeRoure. A Semantic Service Matching Middleware for Mobile Devices Discovering Grid Services. In Proceedings of Third International Conference on Grid and Pervasive Computing (GPC'08), published by the Springer LNCS, Kunming, China, 2008

Chapter 2

Pervasive Computing, Grid Computing and the Semantic Web

2.1 Pervasive Computing

Fundamentally, Pervasive Computing (sometimes called Ubiquitous Computing) is one possible potential direction toward our future computing lifestyle, in which computer systems seamlessly integrate into our everyday lives providing services and information at any time and any place. It is envisaged that ultimately different kinds of computer will become such a natural part of our environments that people will not even be aware of their existence. The conception of Pervasive Computing is credited to Mark Weiser, the chief technology officer for Xerox's Palo Alto Research Centre. In 1991, Mark said: "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it". He believed "only when things disappear in this way are we freed to use them without thinking and so to focus beyond them on new goals" [9].

2.1.1 Evolution of Pervasive Computing

When Personal Computers first brought computing closer to ordinary people, the first step was taken toward making computers widely available, although the full potential of information technology was still not widely appreciated. With the appearance of networking, personal computing developed a distributed computing aspect. Distributed computing marked the next step toward pervasive computing by introducing seamless access to remote information resources and communication with fault tolerance, high availability and security [19].

The integration of cellular technology with the Web and wireless LANs in the late 1990s led to the emergence of mobile computing. Both the size and price of mobile hardware devices are falling continuously, providing the new opportunity of building a distributed system with mobile clients. Mobile computing is an important approach to information access and it prepares the way for pervasive computing - "anytime, anywhere". At present, mobile computing is still an active and evolving research field [20].

Pervasive computing is a superset of mobile computing [10]. The mobile computing goal of "anytime, anywhere" connectivity is extended to "all the time, everywhere" by integrating pervasiveness support technologies such as interoperability, scalability, smartness, and invisibility (Figure 2.1).

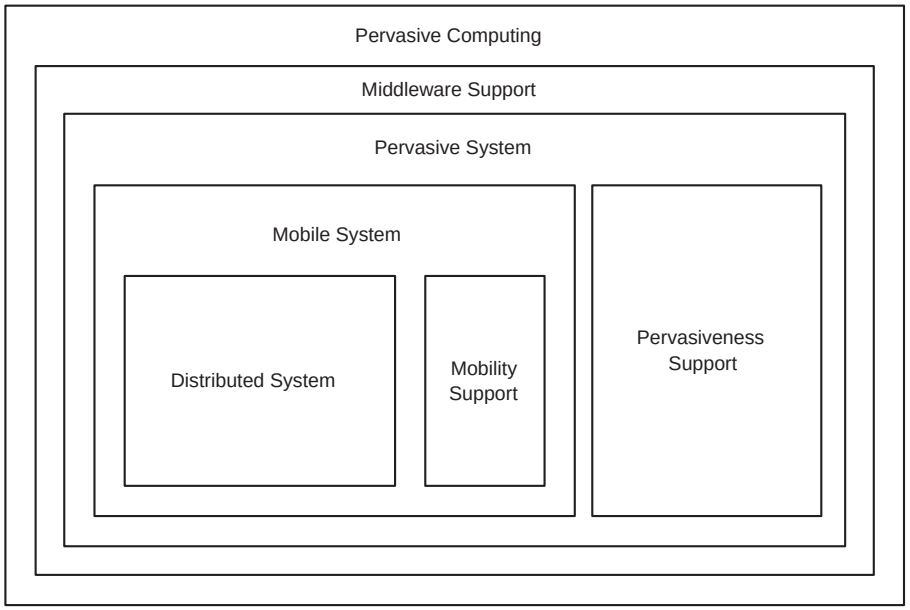

FIGURE 2.1: Relationship between Distributed, Mobile and Pervasive Computing (from [10])

2.1.2 Pervasive Computing Model

In order to build a pervasive computing environment, four broad areas are essential: device, networking, middleware, and application (Figure 2.2).

Traditional input or output devices, wireless mobile devices, and smart devices are the different device types which will be contained in an intelligent environment. The device could be an information source, or a processing centre, or even an agent responding to user actions. The way we consider devices is important. They are not only a repository of custom software managed by the user, but an interface of accessing an application or data space in addition [21].

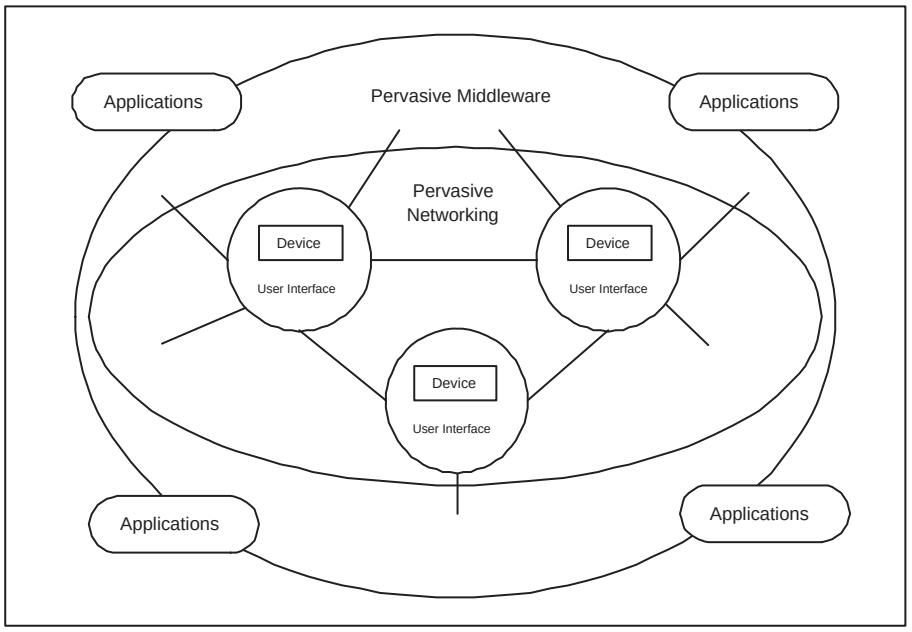

FIGURE 2.2: Pervasive Computing Framework.

With the proliferation of pervasive devices, strong and robust pervasive networking will need to be embedded in our entire environment. Extending the current backbone infrastructure and technologies is not a viable way to meet such an anticipated demand. Global networks (e.g. the Internet) will need to improve existing applications and integrate these pervasive computing devices into existing social systems completely [22].

The middleware is required to connect pervasive computing system kernels (such as various hardware devices, low-level software in general) with the end applications executing on different pervasive computing devices. It will provide transparent, autonomous, and continued services to solve problems due to the issues of mobility and heterogeneity. The final key element of the pervasive computing system is the application. Pervasive computing is more environment-centric than distributed and mobile computing, which means the application in a specific environment will determine the device, middleware and networking issues in that system. The computing environment is the information-enhanced physical surroundings of the user rather than a virtual space that exists to store and run software. The application is not a piece of software written to exploit the capabilities of a device, but a means by which a user performs a task [21].

2.1.3 Cyber Foraging

Many difficult design and implementation problems are required to be solved for the practical realization of the pervasive computing system, such as user intent, adaptation strategy, high-level energy management, client thickness, context awareness, balancing proactivity and transparency, privacy and so on. A number of publications (e.g. [19]

[10]) have presented them in detail. In this thesis, we will discuss "Cyber Foraging" in depth.

The ubiquity of modern pervasive devices provides an important step in the realization of the pervasive computing vision. In the past several years, small computing devices, such as cell phones, PDAs, sensors, and other embedded devices have become ubiquitous and affected everyday life. More importantly, with the improvements in the field of hardware and software (such as processing performance, battery efficiency, service coverage, location sensors, and wireless communication), new opportunities have been created for the services on these kinds of devices. However, it is a real challenge to create complex applications capable of executing on small, light, and mobile computers. On one hand, the small size and low weight necessary for pervasive devices restricts their processing power, memory capability, and battery length. On the other hand, the constantly-growing expectations of the users may require the execution of various applications whose resource demands are far beyond that which a pervasive handheld device can provide. Reconciling this dilemma is far from straightforward.

Cyber Foraging, which has been described as "living off the land", is an effective approach to dealing with this problem. The idea is to augment dynamically the computing resources of a wireless pervasive computer by exploiting the wired hardware infrastructure [19]. Desktops and various computer servers are becoming inexpensive and more plentiful. In the future, we can expect public areas such as shopping malls, airports, and coffee rooms to be equipped with high-performance computer servers, providing a generic infrastructure which supports various computing environments for extremely resource-intensive applications. At the same time, network connectivity is expected to become ubiquitous even for small devices due to the widespread deployment of wireless accessing points. The pervasive devices can then offload their tasks to computer servers around the environment via high-bandwidth networks, allowing execution of the most resource-intensive application components on the servers. The computer servers make their rich resources available to the user devices to realize the functions on their behalf. Figure 2.3 shows a simple cyber foraging scenario where local tasks have been moved to a surrogate machine.

Cyber foraging is therefore all about the issue of remote execution through ubiquitous networks which allow resource-constrained client devices to support applications and services that can not be executed on the client devices themselves. It combines the mobility of client devices and the high processing power of the surrogates. Of course, in order to realize such a Cyber foraging system, several research challenges need to be overcome. The following are some of the key challenges [19]:

- How do mobile clients know the presence of surrogates and discover a required one? A mechanism to allow surrogates to advertise their availability is required such that clients can locate surrogates with appropriate available resources.

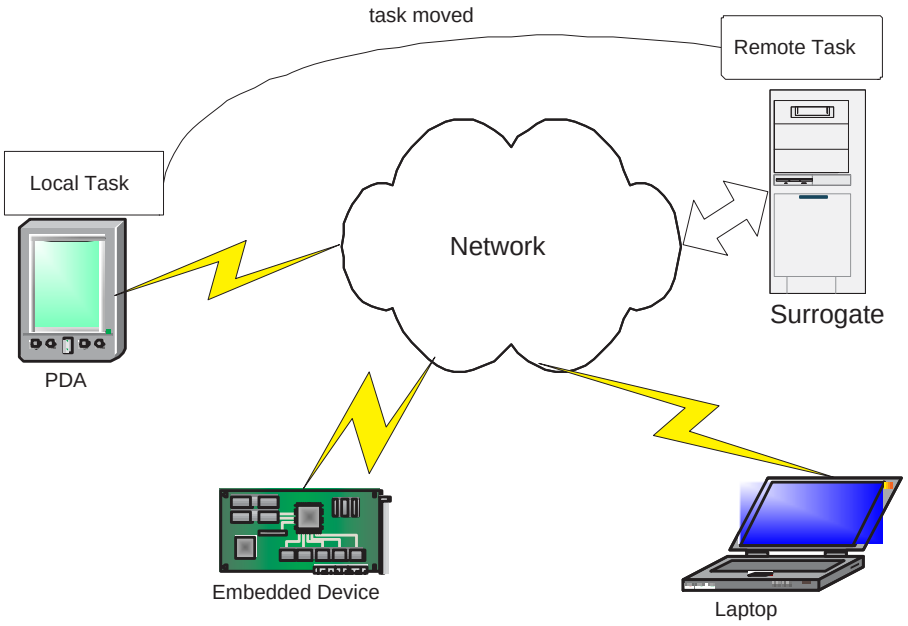

FIGURE 2.3: A Cyber Foraging Scenario (from [27])

- How do mobile devices "understand" the state of the surrogate? A mechanism whereby a potential surrogate can make some of its resources available should be developed.
- How does the surrogate enable a task from mobile clients to be achieved? The surrogate can acquire the executing components of the application directly from the client or from the service providers inside the smart environment.
- How is an appropriate level of trust established between surrogates and mobile devices? Security and trust mechanisms should be developed to ensure that it is the a valid client that is allowed to use the surrogate and that the surrogate is authorized to provide trustful services to the client.
- How does a surrogate deal with a fault during task execution? If an application is running on the surrogate but suddenly the surrogate crashes, what can the client do so that the application can continue to run? Some sort of fault tolerance is required to enable failures to be accommodated.

2.1.4 Existing "Cyber Foraging" System

To follow and support Satyanarayanan's vision, J.Flinn built "Spectra" [23] [24], a remote execution environment that allows a mobile device to use the processing power of a nearby surrogate computer. "Spectra" examined how a cyber foraging system could locate the best server and application partitioning to use given resources. Balan and others extended "Spectra" to support application partitioning, showing that automated

dynamic re-partitioning applications for remote execution can be reconciled with the need to exploit application-specific knowledge. The project presented in [25] demonstrated that an application relevant to remote execution can be captured in a compact declarative form. Further research concluded that Cyber Foraging is helpful for improving the performance of the system. Data staging is able to provide improved latency for file access by staging data at the remote surrogates even if the surrogates are not trusted. Chroma, a remote execution system, is able to use extra resources in the environment to improve application performance [26]. Both data staging and "Chroma" are instances of the "Cyber Foraging" application, the opportunistic use of surrogates to augment the capabilities of pervasive mobile computers.

There appears to be a recent trend that new research work in the field of cyber forging is orthogonal to the past projects. The new work focus is on building a lightweight secure cyber forging infrastructure for resource-constrained client devices. These infrastructures do not require clients and surrogates to share the same file system or require the client to run a relatively heavyweight middleware system, but rely instead on the surrogate being connected to the Internet to locate and download client application code automatically. Their ultimate goal is to realize a common strategy which entails running most of the application on the surrogate and restricting the client to input-output [27].

Another trend of the cyber foraging system architecture is to support stateful and latency-sensitive applications. Slingshot [28] [29] is such a new infrastructure. It replicates remote application state on surrogate computers co-located with a wireless access point. The first-class replica of each application executes on a remote server owned by the handheld user, which offers a "safe haven" for application state in the event of surrogate failure; the second-class replica is deployed on the nearby surrogate to improve application response time. Compared to the former systems, the major capability added by Slingshot is the ability to execute stateful services on surrogate computers due to its replication approach rather than migrating application from one surrogate to another. What is more, in order to reduce the time to instantiate new services at constant wireless accessing points, portable storage is used to store snapshots of service state along with logs that are available for deterministic replay.

In a task offloading system, the terminal has two basic options: local execution and remote execution. Choosing which applications (services) to use on local devices and which to use on the remote device (web/Grid) is essentially an optimization problem, which can be seen as a real-time scheduling problem. To decide optimally on the execution policy, [30] introduced a Markovian framework. They studied the associated energy vs. delay trade-offs, and investigated the performance improvements acquired in various test cases compared to the conventional paradigms of the exclusively local/remote execution. Unlike previous research work which did not take into account the time-varying wireless channel and the load at the server, or took binary decisions for local or remote execution, their focus lies in the optimization aspect of the problem and in investigating

the performance gains in wireless computing that can be obtained by joint processor and transmission power control. There is another approach to problem optimization. The Community-Driven Adaptation (CDA) project [31] grouped users and applications into communities based on common characteristics, and assumed that users and applications of the same community have similar adaptation requirements. CDA supports the content adaptation by observing how members of a community change adapted content to make it more useful to themselves.

2.2 Grid Computing

2.2.1 A Brief History of Grid Computing

Grid computing has its roots in the field of high-performance parallel computing, which has been successful on massively parallel processor (MPP) systems. MPP systems have utilized multiple CPUs within a single system to produce high performance and increased throughput. However, such systems inevitably become prohibitively expensive for large CPU configurations.

The ancestor of the Grid is Metacomputing [32] [33], which linked geographically diverse supercomputing resources via a high-speed network. One of the first infrastructures in the area of Metacomputing was named Information Wide Area Year (I-WAY), which strongly influenced the subsequent Grid computing activities. One of the researchers who led the project I-WAY was Ian Foster who along with Carl Kesselman published a paper [34] that clearly links the Globus Toolkit, which is currently used in many Grid projects, to Metacomputing.

The term"Grid" was born at a workshop entitled "Building a Computational Grid" in 1997 at Argonne National Laboratory. The workshop was followed by the publication of the book "The Grid: Blueprint for a New Computing Infrastructure" [1] in 1998. In the text, Ian Foster and Carl Kesselman wrote a definition: A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

2.2.2 Definition of the Grid

As Grid computing evolved, the emphasis has shifted from the early view to the notion of Virtual Organizations (VOs). Hence, in the paper, "The Anatomy of the Grid" [3], the authors refined the definition, concluding that Grid computing is concerned with coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. "*The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required*

by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization [3]." Here, the key concept is the ability to negotiate resource-sharing arrangements among a set of participating parties (providers and consumers) and then to use the resulting resource pool for some purpose.

According to the check list concluded by Ian Foster, the minimum properties of a Grid system are the following [35]:

- A Grid coordinates resources that are not subject to centralized control (e.g. resources owned by different companies or under the control of different administrative units) and at the same time addresses the issues of security, policy, payment, membership, and so forth that arise in these settings.
- A Grid uses standard, open, general-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery and resource access.
- A Grid delivers nontrivial service qualities, i.e. it is able to meet complex user demands.

Middleware is required to integrate various distributed and heterogeneous computational resources into a large, virtual computer that can be used to solve a single complex problem at a given time. By submitting the request through the Grid middleware, Grid applications can be completely decoupled from the physical components.

2.2.3 Grid Computing Infrastructure and Standards

A Grid application usually consists of several different component, such as VO management service (to manage what nodes and users are part of each Virtual Organization), resource discovery and management service (to discover resources that suit application needs and manage resources), job management service (to submit tasks to the Grid) and so on. Moreover, all of these services are interacting constantly. With so many services and so many interactions between them, there exists a clear potential for chaos. It is very difficult to get all the different software components produced by different organizations to work together.

The solution is standardization: define a common interface for each type of service. As suggested by the previous section, it is very important that standard protocols are

agreed for accessing Grid resources. The de-facto standard in the Grid computing area is the Globus Toolkit which is being developed by the Globus Alliance [36].

The Globus Alliance is a community of organizations and individuals developing the fundamental technologies behind the "Grid", to allow people to share computing power, databases, instruments, and other on-line tools securely across corporate, institutional, and geographic boundaries without sacrificing local autonomy.

The Globus Toolkit includes software services and libraries for distributed security, resource management, monitoring and discovery, and data management. These software components can be used to develop Grid applications.

- GRAM: The Globus Resource Allocation Manager maps requests expressed in a Resource Specification Language (RSL) into commands that local computers can understand.
- GSI: The Grid Security Infrastructure provides authentication services.
- MDS: The Monitoring and Discovery Service combines data discovery mechanisms with the Lightweight Directory Access Protocol (LDAP).
- GRIS: The Grid Resource Information Service provides information about resources, e.g. configuration, capabilities and status.
- GIIS: The Grid Index Information Service coordinates arbitrary GRIS services.
- GASS: The Global Access to Secondary Storage implements a variety of data access strategies, enabling programs running at remote locations to read and write local data.
- GridFTP: The GridFTP provides a high-performance, secure and robust data transfer mechanism.

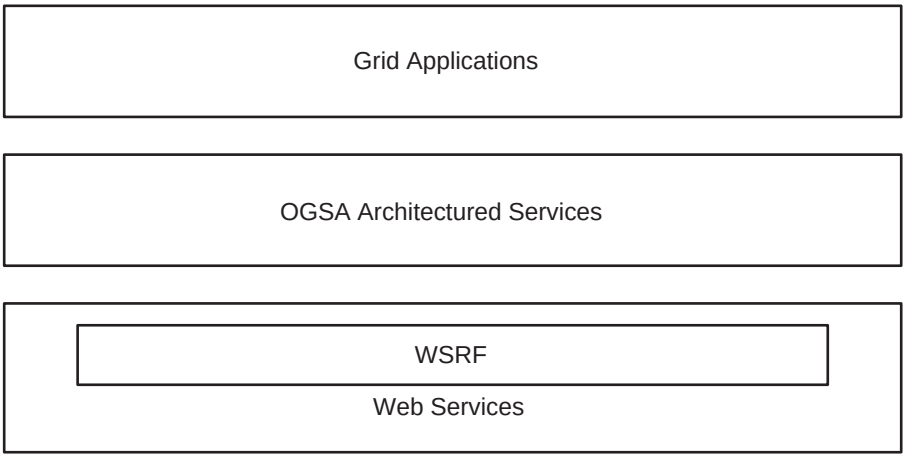

FIGURE 2.4: Layed Diagram of OGSA Architecture.

The latest version of Globus Toolkit is GT4 [36], which includes components for building systems that follow the Open Grid Service Architecture (OGSA) [37] framework defined by the Global Grid Forum (GGF). OGSA is a standard-based definition of a Service Oriented Architecture (SOA) for the Grid. Figure 2.4 shows the layed diagram of the OGSA architecture.

2.2.3.1 Web Service

Web services are a specific realization of a Service Oriented Architecture [38] in which various services interact with each other by exchanging messages in SOAP (Simple Object Access Protocol) [39] format while the contracts for the message exchanges that implement those interactions are described in WSDL (Web Service Definition Language) [40]. Figure 2.5 shows a typical web service invocation. Compared to other distributed computing technologies (e.g. COBRA, RMI, EJB), web services have several advantages:

- Web services are platform-independent and language-independent, since they use standard XML languages.
- Most web services use HTTP for transmitting messages (e.g. service request, service response). This is a major advantage if building an Internet-scale application, since most proxies and firewalls will not filter HTTP traffic.
- Web services are more appropriate for loosely-coupled systems, where the client might have no prior knowledge of the web service until it actually invokes it. This is better suited to meet the demands of an Internet-wide application.

OGSA needed to select a widely-used distributed middleware to build its own structure. In other words, when defining an interface that has a method, there has to be a common and standard way to invoke that method. This base for the OGSA architecture could be any distributed middleware (e.g. COBRA, RMI, RPC [41]). Considering the advantages that web services have compared to other technologies, OGSA selected web services as the underlying technology to be used.

2.2.3.2 WSRF

One of OGSA's most important requirements is the underlying middleware has to be stateful. However, although web services can be stateless or stateful in theory, they are usually stateless and there is no standard way of making them stateful. Hence, another layer is required between web services and OGSA architecture services.

Web Services Resources Framework (WSRF) [42], developed by OASIS, defines a generic and open framework for managing state in distributed systems based on web services.

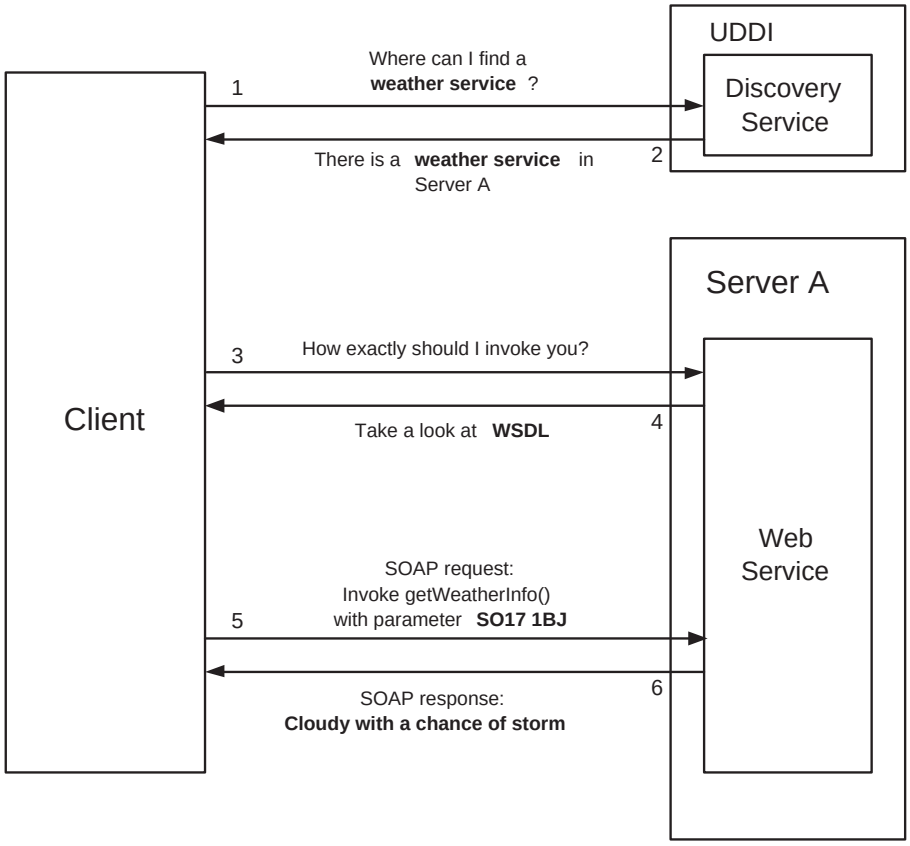

FIGURE 2.5: A Typical Web Service Invocation.

It provides the stateful services that OGSA needs. A concise way of expressing the relationship between OGSA and WSRF is that OGSA is the architecture, while WSRF is the infrastructure upon which that architecture is built on.

The four WSRF specifications being standardized define how to represent, access, manage, and group WS-Resources:

- WS-ResourceProperties [43]: A resource is composed of zero and more resource properties. WS-ResourceProperties specifies how resource properties are defined and accessed.
- WS-ResourceLifetime [44]: Resources can be created and destroyed at any time. The WS-ResourceLifetime supplies basic mechanisms to manage the lifecycle of resources.
- WS-ServiceGroup [45]: Managing groups of web services or groups of WS-Resource is usually a non-trivial operation. The WS-ServiceGroupspecifies how to group services and WS-Resources together.
- WS-BaseFaults [46]: This specification provides a standard way of reporting faults when something goes wrong during a WS-Service invocation.

Notification is not part of WSRF, but WSRF specifications reference notification in a generic manner. A WSRF implementation typically also implements at least some functionality defined in the WS-Notification specifications [47]. WS-Notification defines topic-based publish/subscribe mechanisms. A web service can be configured as a notification producer and certain clients configured to be notification consumers (or subscribers). If a change occurs in the web service or WS-Resource, that change is notified to all the subscribers

Figure 2.6 sets out the relationship between typical Grid and Web Service definitions.

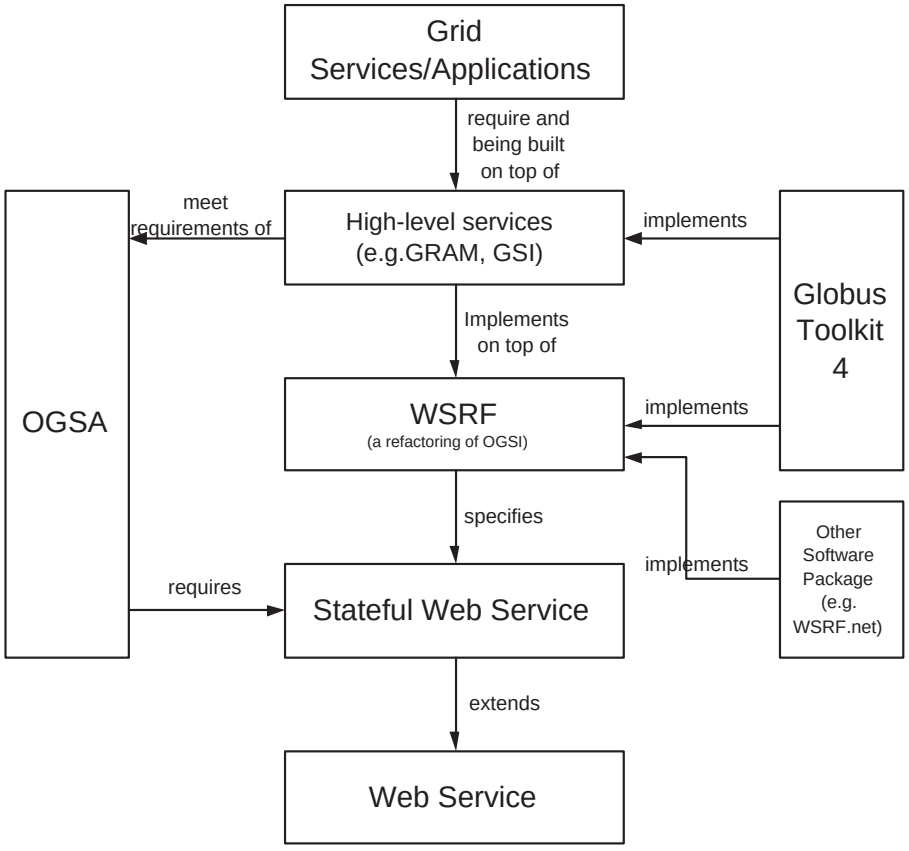

FIGURE 2.6: Relationship between GT4, OGSA, WSRF, and Web Services.

2.3 The Semantic Web

The Semantic Web is an initiative of the World Wide Web Consortium with the goal of extending the current Web to facilitate Web automation, universally accessible content, and the "Web of Trust". "*Facilities to put machine-understandable data on the Web are quickly becoming a high priority for many organizations, individuals and communities. The Web can reach its full potential only if it becomes a place where data can be shared and processed by automated tools as well as by people. For the Web to scale, tomorrow's*

programs must be able to share and process data even when these programs have been designed totally independently [12]."

Essentially, the Semantic Web, a web of data, is a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. The key difference from the traditional web, which concentrates on the interchange of documents, is that the semantic web exhibits multiple formats for integration and combination of data drawn from diverse sources. The semantic web also provides a set of languages for recording how the data relates to real world objects.

Semantic Web technologies attempt to connect the gap between human understanding data and the machine understanding data [12]. Enabling a computer to understand the information the human being knows is a historical problem studied in many areas, from past artificial intelligence to the current Internet computing and pervasive computing research.

In the past, most work on semantic technologies concentrated on the Semantic Web, and the semantic technologies developed have been used to implement the Semantic Web. For example, the management of resources in the Semantic Web is impossible without the use of ontologies, which can be considered as the high-level metadata of the data and knowledge on the web. However, the demand to state the meaning of data explicitly also exists in many other research areas. If different systems or different components in one system require to share data or transfer advanced "meaningful" messages, the relevant basic semantics need to be defined and assumed in advance.

2.3.1 Semantic Web Languages

A language which is able to express and describe various web resources and their relationships sufficiently is necessary for realizing the vision of the Semantic Web. At present, there are a number of languages designed and applied in the field of Semantic Web.

- RDF (Resource Description Framework)

  The Semantic Web technologies are based on the Resource Description Framework (RDF) [48]. It provides a standard way to represent metadata, the data about documents, images, objects, people, devices, resources, or everything around the world. In the World Wide Web Consortium (W3C) recommendation, RDF is given to the following definition: "*The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared*

resource. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved on the Web [48]." Figure 2.7 shows a rdf Graph describing Tao Guan.

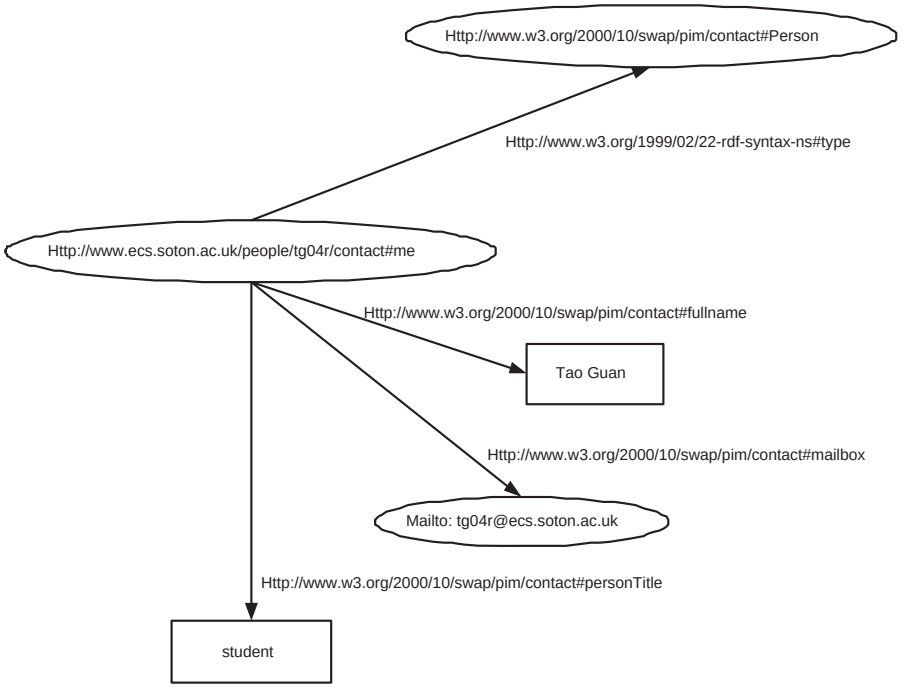

FIGURE 2.7: A Rdf Graph Describing Tao Guan.

Because RDF provides a way of representing metadata about everything around the world rather than the content on the web only, it can also be used to describe various entities in the field of Pervasive and Grid computing, for example, Grid resources, Grid services, pervasive devices, environment context, user profile and so on. In addition to describing various entities, RDF also allows us to build a model about the relationship between the entities it describes [49].

- RDF-Scheme

  RDF provides a way to express simple statements about resources, using named properties and values. However, RDF user communities also need the ability to define the vocabularies (terms) they intend to use in those statements, specifically, to indicate that they are describing specific kinds or classes of resources, and will use specific properties in describing those resources. For example, people interested in describing bibliographic resources would like to describe classes such as ex2:Book or ex2:MagazineArticle, and use properties such as ex2:author, ex2:title, and ex2:subject to describe them. Other applications might need to describe classes such as ex3:Person and ex3:Company, and properties such as ex3:age, ex3:jobTitle, ex3:stockSymbol, and ex3:numberOfEmployees. RDF itself provides no means for defining such application-specific classes and properties. Instead,

such classes and properties are described as an RDF vocabulary, using extensions to RDF provided by the RDF Vocabulary Description Language: RDF Schema [50].

RDF Schema does not provide a vocabulary of application-specific classes. Instead, it provides the facilities needed to describe such classes and properties, and to indicate which classes and properties are expected to be used together. In other words, RDF Schema provides a type system for RDF. The RDF Schema type system is similar in some respects to the type systems of object-oriented programming languages such as Java. For example, RDF Schema allows resources to be defined as instances of one or more classes. In addition, it allows classes to be organized in a hierarchical fashion; for example a class ex:Dog might be defined as a subclass of ex:Mammal which is a subclass of ex:Animal, meaning that any resource which is in class ex:Dog is also implicitly in class ex:Animal as well. However, RDF classes and properties are in some respects very different from programming language types. RDF class and property descriptions do not create a restriction into which information must be forced, but instead provide additional information about the RDF resources they describe.

- DAML+OIL (DARPA Agent Markup Language + Ontology Inference Layer)

  An ontology is a controlled vocabulary that describes objects and the relations between them in a formal way. "Ontology" is a term from philosophy, but today it is often used to refer to the specification of a conceptualization of a specific domain in the field of computer science, especially supporting knowledge sharing, context reasoning, and interoperability [51]. However, RDF can only describe the entities and the relationship between entities, and is not sufficiently powerful to expound the meaning of metadata. A specialized "ontology language" is needed to specify such concepts. DAML+OIL is one of these kinds of ontology languages.

  DAML+OIL is a semantic markup language for Web resources. It builds on earlier W3C standards such as RDF and RDF Schema, and extends these languages with richer modeling primitives, such as those commonly found in frame-based languages. DAML+OIL (March 2001) extends DAML+OIL (December 2000) with values from XML Schema datatypes [52]. DAML+OIL was built from the original DAML ontology language DAML-ONT (October 2000) in an effort to combine many of the language components of OIL, with a clean and well-defined semantics [53].

- OWL (Web Ontology Language)

  The OWL language is a Semantic Web language for use by computer applications that need to process the content of information instead of simply presenting

information to humans [54]. OWL is part of the Semantic Web initiatives sponsored by the World Wide Web Consortium (W3C). The OWL language can help to formalize a domain by defining classes and properties of those classes, define individuals and assert properties about them, and reason about these classes and individuals [55]. It builds on the DAML+OIL language and both are layered on top of the standard RDF triple data model (i.e., subject, predicate, and object). Both OWL and DAML+OIL enable the creation of ontologies for any domain and the instantiation of these ontologies in the description of specific Web sites. They are also amenable to efficient reasoning procedures and thus reasoning applications can be built to automatically determine the logical consequences of the ontological statements [56].

Using OWL, concepts and their relationships in any domain are represented in terms of classes and properties. Complex concepts can be defined from simple ones using structures (e.g. anonymous restriction) provided in the language. This definition mechanism is similar to that of Description logics (DL), which describes knowledge in terms of unary predicates or classes (concepts) and binary relations (roles) [57]. A DL knowledge base usually consists of TBox (Terminological Knowledge, including concepts definitions and relationship between concepts and roles) and ABox (Assertional Knowledge, including instances of concepts and relations between instances). Because of the definition similarity, a DL reasoner is often used to judge the semantic similarity between concepts composed with OWL. OWL has three sublanguages, OWL-lite, OWL-DL, and OWL-full, each of which gains in expressiveness but becomes harder for a description logics reasoner to process.

The Semantic Web enables users to access not only the content of the Web but also the services on the web. Discovering, invoking, composing, and monitoring web resources need to be performed with a high degree of automation. This means that powerful tools are required to process service descriptions so that services can be exposed to users. OWL-S is an ontology of services written with OWL that makes these functionalities possible [56]. OWL-S can also be considered as an extended OWL language for describing services, because it defines a set of standard vocabulary that can be used together with the other aspects of the OWL description language to create service descriptions.

2.3.2 Semantic Web Technologies in Pervasive Computing

In the vision of pervasive computing, computer systems seamlessly integrate into the user life, providing them with services and information in an “anywhere, anytime” fashion. In such an open and dynamic environment, various computing entities must be able to share knowledge and reason context. However, previous systems and architectures offer only weak support for knowledge sharing and context reasoning. Semantic web technologies

can be adopted in the pervasive computing area to provide the functionalities of sharing and reasoning knowledge.

2.3.2.1 Context and Ontology

Context is any information that can be used to characterize the status of various entities in an environment [58]. In a pervasive computing environment, context can be defined as the information that characterizes the identity and attributes of people, pervasive devices, network, and various services. This information includes the performance and status of computing devices, the available services, the function of various pervasive devices, the relationship between devices and services, and so on. It may also involve the surrounding environment of devices, such as networking capability, because these conditions are able to change the interaction between devices. A well-defined context model is important for implementing a context-aware system.

Ontology refers to the specification and conceptualization of a knowledge domain [59]. It is a group of controlled vocabularies that describe objects and the relations between them to express something meaningful within a specified interest domain. Historically, there are several context modeling approaches [60], such as key-value model, markup scheme models, graphical models, object-oriented models, and logic-based models. However, using ontology to model contexts for an ubiquitous computing environment offers several advantages:

- A shared ontology could support knowledge sharing, context reasoning and interoperability in the computer system.
- The ontology of pervasive devices will not only enable service consumers to find service providers conveniently according to the actual requirement of the user, but also be helpful for clients to monitor the executing status of various resource-intensive applications.
- The hierarchical structure enables ontology developers to reuse the existing different consensus ontologies and borrow the terms from those ontologies.

Previous pervasive computing systems lack knowledge about sharing and reasoning because they are not built on a foundation of common ontologies with explicit knowledge representation [61] [62], and the high-level knowledge representations require the establishment of a prior low-level implementation agreement between the programs that wish to share information. To address these issues, a shared ontology must be specifically designed for supporting knowledge sharing, context reasoning and interoperability in pervasive computing systems.

2.3.2.2 Existing Ontologies

The concept of Ontology is widespreadly used in many areas such as knowledge and content management, electronic commerce and the Semantic Web. Harnessing ontology can help to overcome some important problems in the development of pervasive computing environments, such as discovery and matchmaking in a pervasive computing environment, inter-operability between different entities of smart environments, and context awareness. [63] There are a number of consensus ontologies for different application area. The following describes the key features of some example ontologies:

- FOAF [64]: The vocabularies of this ontology are designed for expressing personal information and relationships, and it is useful as a building block to create information systems that support online communities [65]. Pervasive computing applications can use FOAF ontologies to express and reason about the contact profile of a person and social connections to other people in their close vicinity.
- DAML-Time and the Entry Sub-ontology of Time [66]: These ontologies define a comprehensive set of vocabularies for expressing temporal concepts and preparing common framework to any formalization of time. In pervasive computing applications, these ontologies can be exploited for sharing a common representation of time and reasoning about the temporal orders of different events.
- OpenCyc Spatial Ontologies [67] and Regional Connection Calculus [68]: The OpenCyc spatial ontologies allow the expression of symbolic representation of space. The ontology of Regional Connection Calculus is a useful building block for expressing spatial relations for qualitative spatial reasoning. They are typically used by pervasive computing applications to describe and reason about location and location context [69].
- COBRA-ONT [69] and MoGATU BDI Ontology [70]: The COBRA-ONT focuses on modeling contexts in smart meeting rooms, and the design of MoGATU BDI ontology focuses on modeling the belief, desire, and intention of human users and software agents.
- Rei Policy Ontology [71]: The Rei Policy Ontology is aimed at specifying and reasoning about security access control rules by defining a set of concepts (e.g. rights, obligations, dispensations). High-level rules for granting and revoking the access rights to and from different services can be specified by using this policy ontology in a pervasive computing environment [72].

An important advantages of the ontology approach to modeling the context is the knowledge reuse. The existing well-defined ontologies demonstrate various vocabularies that also meet the requirement of other applications. Hence, most of concepts and properties for a new application scenario can be extended and imported from the existing

ontologies, and only additional objects are required to be created. For example, simple relationships between people (e.g. friendOf) is defined in FOAF, which have to be extended to support complex properties for other context model such as *Semantic Space* [73].

2.3.2.3 Service Description

The pervasive computing environment has a potential requirement to integrate a great number of services, from a simple information query to extremely dynamic and powerful scientific computations. However, it is not straightforward for a user to know, acquire and finally invoke their appropriate services. Service discovery protocols simplify the interaction among users, devices, and services. In order to enable a services to be discovered by users, the characteristics (e.g. capabilities) that the service provides must be described in an explicit way.

Semantic description of services allow a more advanced discovery mechanism that is important given the flexible, open, and dynamic nature of pervasive systems. Ontology Web Language for Service (OWL-S) [56] is such a language which provides this function of the semantic service description, because it provides a semantic description layer for web services that is not supported by the more low-level service-syntax oriented Web Service Description Language (WSDL).

The types of knowledge about describing a service are the profile of the service, the functionalities of the service, and the interaction of the service. The structure of the OWL-S upper-level ontology is based on these three perspectives (Figure 2.8). The "Service Profile" provides the the high-level descriptive information of a service, such as the name, input/output of the service, and additional text description, for the purpose of advertising, constructing service requests, and matchmaking. The "Service Model" describes how the service works to enable service invocation, enactment, composition, and monitoring. The "Service Grounding" maps the constructs of the process model into the service detail, providing the information how to interact with the service.

An important point to notice from the service description model is that OWL-S supports a grounding to map the constructs of the process model onto WSDL. This is particularly interesting as it suggests that OWL-S can provide a semantic wrapper around the web service method invocations. Although this semantic wrapper would obviously lead to extra overheads in addition to those normally existing in traditional web services architecture, it offers the potential for system developers to be more focused on the abstraction level of semantics rather than trivial communication details. Current research work has demonstrated that OWL-S can help to enable fuller automation and dynamism in many aspects of Web service provision and use. In addition, OWL-S has

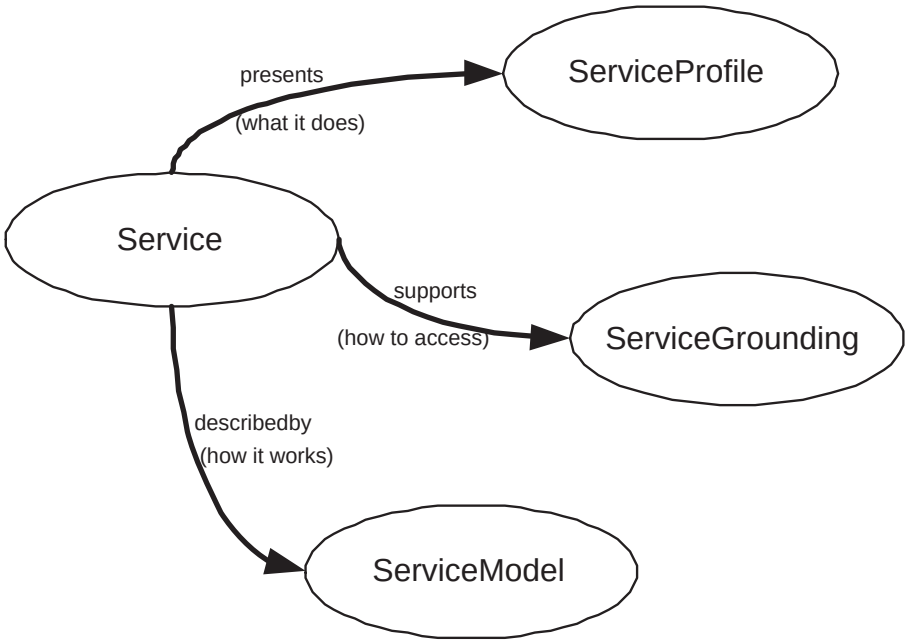

FIGURE 2.8: Top Level of OWL-S.

supported the construction of powerful tools and methodologies and promoted the use of semantically well-founded reasoning about services.

2.3.3 Semantic Grid

The notion of "Semantic Grid" was introduced by De Roure, Jennings and Shadbolt. In 2001, they advocated "the application of Semantic Web technologies both on and in the Grid" [74] in order to meet the requirement of maximum reuse of software, services, information and knowledge from the diverse set of UK e-Science applications. The vision of the semantic Grid is to achieve a high degree of easy-to-use and seamless automation to facilitate flexible collaborations and computations on a global scale by means of machine-processable knowledge both on and in the Grid [15].

The Semantic Grid is an extension of the current Grid in which information and services are given well-defined meaning through machine-processable descriptions. It is driven by e-Science projects and e-Science projects provide a valuable test platform for the semantic web technologies in Grid applications. The semantic Grid is still a developing field, and abstracted by experience with practical projects, a number of requirements are key for realizing the vision of the semantic Grid, including Resource description, discovery and use, process description and enactment, autonomic behavior, security and trust, annotation, information integration and so on. Web Services, Software Agents, Metadata, Semantic Web Services, Ontologies and Reasoning are five essential technologies to address these requirements [74].

2.4 Summary

This chapter has introduced briefly new computing models emerged in the past several years. Pervasive computing enables various computing systems to be embedded into our daily life, providing information and services "everywhere, all the time". Grid computing, original from high-performance parallel computing, has become the most effective model for processing very complex problems. Semantic web technologies, connecting the gap between the human-understanding data and the machine-understanding data, are beneficial for the current development in both Grid and pervasive computing fields.

"Cyber Foraging" is an approach to augmenting dynamically the capabilities of ubiquitous devices by exploiting the available computer servers. The existing "Cyber Foraging" systems show the performance promotion by offloading complex tasks to the resource-rich computing platforms. An interesting aspect of the "Cyber Foraging" system is that ubiquitous clients could access a range of nearby servers so that they are able to execute Grid-alike applications. Its idea is similar to the solution of enhancing the capabilities of pervasive devices by integrating them into a service-oriented Grid environment.

In the next chapter, we will analyse requirements of integrating mobile devices into the Grid environment and discuss the related research work.

Chapter 3

Scenarios and Requirements for Grid-enhanced Mobile Devices

3.1 Scenarios

Mobile devices form the intersection between the physical world and the digital world. They require a variety of services to accomplish various daily tasks. From a user viewpoint, two styles of scenario are identified: an information access scenario, and a work assistant scenario. Under these two scenarios, the capabilities of mobile devices are enhanced by the service-oriented Grid environment through accessing information providers and offloading resource-intensive work to more powerful devices and resources.

3.1.1 Information Access Scenario

In the information-access scenario, the mobile handheld device (e.g. smart phone, PDA) acts as a universal operating terminal for mobile users to access various available services [75] [76]. The users are able to consider their mobile handheld devices to be an information collection center to access and gather any desired information or knowledge [77]. The users can also install a lightweight steering client application on their small mobile handheld devices, providing an effective way of monitoring and controlling any potential devices or jobs in the Grid environment.

As a typical application, let us consider the following scenario:

A medical Grid was built in a hospital based on the existing Grid infrastructure in which the health conditions of the patients are available to the doctors in real time anywhere and anytime. Sensors in the hospital collect the vital health data of the patients and transmit the data to the medical Grid through a local wire or wireless network in real time. The medical Grid is responsible for classifying, storing, and analyzing the data

received from the sensors. On the other side, the doctors equipped with PDAs are able to check the data being collected from the patients in real time. The doctors can visualize the data from one or several patients at the same time and configure the monitoring software so that the doctors can be reminded when certain conditions are reached. The PDA is also the communication centre of the doctors, which can be used to share information and data between doctors, nurses and other staff in the hospital (Figure 3.1).

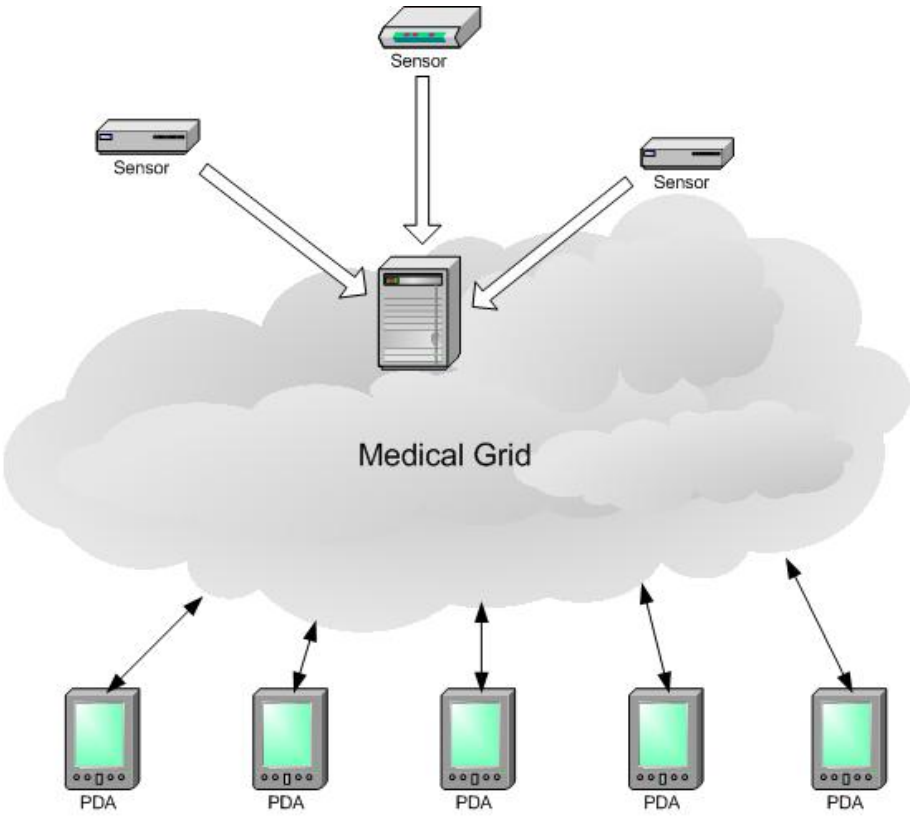

FIGURE 3.1: Real Time Data Monitoring Through Medical Grid.

Another kind of information-access scenario is the idea of implementing a Grid-enabled lightweight computational steering client (Figure 3.2), which was described in [78]. The important feature of the steering system is that it supports dynamic connection to the steering services. The user can connect and disconnect to their simulation when it executes, allowing the user to run several jobs simultaneously and switch between them. The RealityGrid steering system enables the simulation to be distributed over the Grid architecture [79]. However, the scientist has to connect and disconnect regularly with the simulations in order to inspect the developing results, which ties the scientist down to the desk.

A solution is to use a mobile device to steer a set of computation simulation jobs, which changes the scientist work habit that the Grid is originally accessed through high-performance desktop computers. However, this attractive new steering concept allows

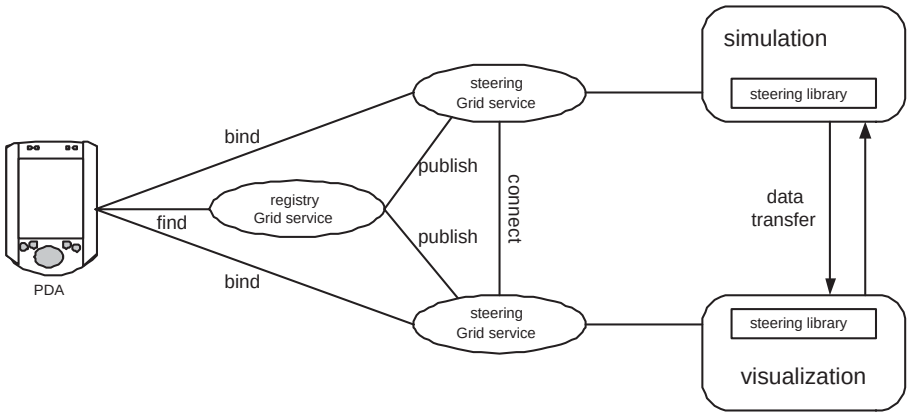

FIGURE 3.2: Grid-enabled Lightweight computational Steering Client (from [78])

scientists to connect to their simulation experiments with their mobile devices wherever there are wireless network resources, and gives them great freedom to leave their office or laboratory while still communicate with their work as well as providing an easy-to-use interface.

A central requirement of the information-access scenario is the connectivity of mobile devices with the Grid. When users arrive at a new place, they are able to locate and utilise services which are provided by various providers and deployed within the nearby Grid environment with their portable devices conveniently. In short, the information-access scenario represents the scene in which various sets of ubiquitous devices are integrated together with Grid technologies to create significant potential for building interesting applications.

3.1.2 Work Assistant Scenario

In additional to collecting information through accessing Grid services, users usually need to execute relatively complicated applications such as data-deluge programs to achieve specific tasks on their mobile device. However, due to resource limitations, most complex programs cannot be executed on a handheld device. In this situation, users have to offload the resource-demanding parts of the task to the Grid [26] and consider the Grid to be their executing environment to accomplish the task for them. This style of scenario is named the "*work assistant scenario*". The work assistant scenario describes the scene in which mobile users perform complicated tasks by using various distributed resources of the Grid.

Let us consider the following scenario (Figure 3.3): A fire has broken out suddenly in a multi-story building. The temperature sensors embedded at various locations inside the building are capable of producing streams of temperature data. When fire fighters arrive at the spot, they try to communicate with the sensor network of the building with their mobile handheld devices to obtain detailed information, such as the temperature of

different floors of the building, or the current temperature of a particular room possibly still with people inside. To get this response, a number of complex mathematical operations (e.g. solving three-dimensional partial differential equations) have to be addressed according to the data from the sensor devices as well as the static data describing the building architecture. It is not currently feasible that any devices inside the sensor network of the building including the mobile handheld device of the fire fighter can perform these series of computation. One approach would that the mobile handheld device of the fire fighter enables the data from the sensor network to be transferred to the Grid and then perform those series of complex mathematical operations by using Grid resources. Another possibility is that the occupants of building can take pictures of their current position with their camera-enabled mobile phones, and send the pictures to the nearby available Grid services which will perform the necessary computation and return the results back to the fire fighters.

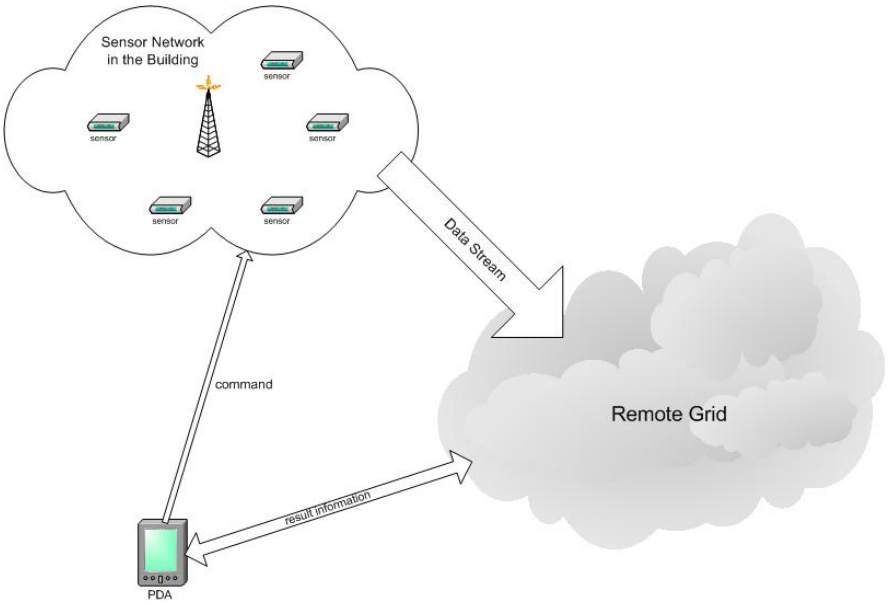

FIGURE 3.3: An Example of Work Offloading Scenario.

The work assistant scenario, in contrast to the information access scenario, is a resource-rich executing environment. Users usually need to perform their tasks with their handheld devices because no other devices are available, especially in an emergency situation. Mobile handheld devices transfer their tasks to the Grid rather than perform the tasks themselves. The work assistant scenario is similar to "Cyber Foraging", as discussed in chapter 2.1.3. The difference is that mobile users offload complex tasks to the local high-performance machines in the "Cyber Foraging" system, while the executing environment of the work assistant scenario is a variety of Grid services. To summarize, the work assistant scenario represents the environment in which various mobile devices submit their tasks to the resource-rich environment to enhance their capabilities and accomplish complicated tasks.

3.2 Analysis

The above two kinds of scenario differ in many aspects, such as the type and number of pervasive and mobile devices, the duration and interaction with the users and the computing environment, and the demand for services or resources from the Grid infrastructure. However, these scenarios also have much in common: pervasive and mobile devices need to interact with Grid environments to perform a variety of tasks. Furthermore, three interaction aspects between the Grid and the pervasive mobile device can be recognized based on above scenarios:

- Pervasive and mobile devices need the Grid for computation

  Pervasive devices form the intersection between the physical world and the digital world. With increasing numbers of devices embedded in our daily life, more computation and process power are required to support pervasive and mobile computing applications. The availability of Grid services makes it possible to achieve computational tasks that would not be possible using these devices themselves. For example, in the work assistant scenario, the sensor network inside the building can produce a great volume of data that can be transmitted to the fire fighters in an emergency. However, the original format of the data does not benefit the fire fighters in obtaining the detailed information of the building until it is processed and analyzed by the Grid. Besides this, many other interactive pervasive computing applications demand real-time processing and therefore require significant computational power, to be delivered on demand (e.g. language and speech translation, place recognition and so on). In conclusion, the Grid has an important potential role in support of new pervasive and mobile computing applications, especially as new devices generate and store larger quantities of data.

- Pervasive and mobile devices need the Grid for integration

  Pervasive applications normally deploy various pervasive devices in the environment separately. An interoperable infrastructure is also required to integrate these individual devices together to perform some tasks. However, there are not any established common distributed system infrastructure standards to handle the interworking of multiple diverse sets of pervasive devices. In [80], the authors discuss the possibility of building pervasive computing applications by using Grid technologies to link sets of devices together. In the above information access scenario, sensors embedded in the hospital collect the data from the patients in real time, while the general purpose pervasive device (PDA) is used to visualize and monitor the data from the patients. Both sensor devices and monitor devices are integrated into a medical Grid, which is also responsible for classifying data from sensors, storing data into the internal database, and converting the data format. Steering simulation jobs with PDAs is another interesting example of integrating

mobile devices into the Grid. The computation steering system is deployed as a Grid application so that scientists are able to inspect the experimental results by connecting to their simulation with a typical Grid client. Traditionally, high-end desktop computers are the terminal platforms of scientists. However, as long as their mobile devices are integrated into the Grid, scientists are freed from the constraint of the desktop, which potentially could make their research work more productive.

- The Grid needs pervasive and mobile devices to interface with the physical world

  Doctors, scientists, and fire fighters need Grid support so that they can perform additional tasks with their carry-on mobile devices. From the aspect of the Grid, it is clear that the Grid also needs mobile devices to interface with the physical world. If no mobile devices were available, the medical Grid can not expose the data to the doctors, the scientists would be tied down to their desktop system to observe the experiment results, and the fire fighters would have no idea about information from the inside building even though they could have the raw data from the sensor network. Traditionally, human user interfaces to the Grid have been through the graphical user interfaces of applications and portals which are installed as the Grid client software. In the context of pervasive and mobile computing, interfaces become mobile devices in the user environment. These devices may be carried into the environment by users or they may actually be part of the environment, and they may in addition collect data for the Grid, as well as providing notifications and information displays for users.

3.3 Requirements

There is a computing model emerging from the scenarios of access by mobile devices to Grid services to perform various tasks. As explained in the previous section, a dependent relationship exists between a number of mobile devices and the resource-rich Grid environment.

However, when considering how the vision of integrating mobile devices into the Grid environment might be realized, a number of common requirements are observed:

- Middleware is required to bring mobile devices into the Grid in a flexible, open and interoperable way.
- A service description, discovery and composition mechanism is required to enable users to to locate required Grid services to achieve their tasks.

- Context-awareness is required and information should be shared between service providers and service consumers so that Grid services can be provided for mobile users in an appropriate way.
- A task execution management mechanism is required to support mobility.
- Autonomic behaviors such as self-configuration, self-management, self-optimization and self-healing are desired properties and should be supported in the new computing environment which combines both Grid and pervasive mobile systems.

The section that follows discusses these issues in more detail, together with the existing and related work necessary to meet the requirements of building a system architecture which integrates mobile devices with the service-oriented Grid environment.

3.3.1 Middleware

The Open Grid Service Architecture (OGSA) is a standard-based definition of a Service Oriented Architecture (SOA) for the Grid. Using a service-oriented architecture to implement Grid computing environment have several advantages [37]:

- Modularity: Grid services can be dynamically coupled at runtime in a SOA.
- Interoperability: A set of standard service interfaces can be defined in a SOA for both service providers and service consumers.
- Extensibility: It is relatively easy to deploy new services or remove existing services from the service environment in a SOA.

Figure 3.4 shows the current Grid standards and existing toolkits. The base infrastructure specification for OGSA is Open Grid Service Infrastructure (OGSI) or Web Service Resource Framework (WSRF). The Grid community is currently in a transition period between OGSI and WSRF, which is reflected in the ongoing development of Grid infrastructures and tools.

OGSI takes the statelessness issues into account by essentially extending Web services to accommodate grid computing resources that are both transient and stateful. It specifies mechanisms for creating, naming, managing, monitoring, grouping and sharing information among Grid services. WSRF is a refactoring of OGSI, which defines a generic and open framework for modeling and accessing stateful resources. However, both OGSI and current WSRF implementations are designed for building enterprise level virtual organizations, which are too heavyweight for the loosely coupled mobile applications necessary in the pervasive and mobile computing domain. It is possible that in the future WSRF

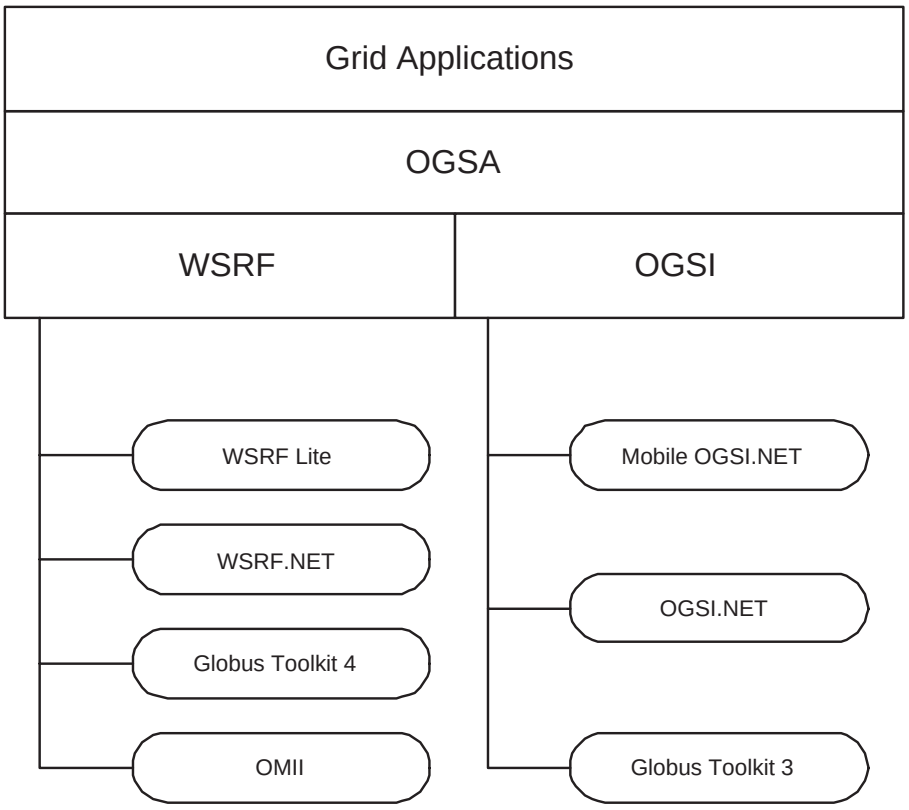

FIGURE 3.4: Grid Technologies and Existing Toolkits.

may improve support for Grid clients on mobile devices because it is fully web services based and has a simplified API [81].

Existing Grid infrastructure packages do not satisfy the requirement for integrating mobile devices into the service-oriented Grid environment. Also, most of the existing Grid applications are developed with the assumption that the client terminals possess sufficient resources for the task at hand and that the communication infrastructure is reliable. Hence, a service-oriented middleware is required to be designed to allow mobile handheld devices (e.g. smart phones, PDA units) to interact with Grid services, placing a minimal burden on the device itself.

The design of middleware for integrating mobile systems into the Grid environment is primarily affected by the following key issues: device capabilities, network connection, and dynamic characteristic.

- Device capabilities: With the continuous progress in the development of mobile devices, a wide variety of devices exist with improving absolute capabilities. However, compared to their static counterparts (e.g. desktops), mobile devices are still resource-limited (e.g. a slower CPU, a smaller storage space, and the limitation of the battery technology). Software heterogeneity is another issue that hinders the development, for example, the Palm and Win CE operation systems compete aggressively for PDA market share.

- Network connection: Typical Grid computing applications aim at the static machine which has "always-on" connection with a low error rate. However, in mobile computing systems, network connections usually have limited bandwidth, high error rate and frequent disconnections because of technical limitation and user mobility. Even if connectivity problem is solved, current wireless technology provide a lower bandwidth compared with a wired connection. Most wired networks around buildings are already at Gigabit-per-second speeds using inexpensive hardware. On the other hand, the most recent wireless technologies are limited to top speed of 108 Mbps.

- Dynamic characteristic: Mobile devices are able to roam freely, from one place to another, connecting at different locations with various networks. Hence, the middleware should support these dynamic characteristics in a scalable and robust fashion. Security is also an issue with mobile wireless devices because mobile devices are moving in different sensitive environments. Various protection mechanisms such as end-to-end protection are necessary to defend the wireless traffic.

3.3.2 Service Discovery and Composition

One of challenges required to be addressed to realize the vision of integrating mobile devices into the service oriented Grid environment is to provide a means for advertising the availability of resources to enable mobile devices to locate appropriate Grid services. Service discovery protocols simplify the interaction between users, devices and services.

Two typical categories of service discovery approaches are the client-service model and the client-service-directory model. In relatively simple interaction environments such as home environments, the client-service model may be used. Clients first inquire about the service availability and matching services make a return based on the queries of clients. After receiving responses from services, clients select and interact with services. In the clients-service-directory model, a client queries a service directory with a template of attributes about a service that they wish to discover. Based on the request description, the information of matching services (e.g. service address, service handle) is returned. The client then contacts services with these service information. The prerequisite of this service discovery model is that all of the service description information is advertised and stored in the location of the service directory.

During the past several years, various mature service discovery protocols have been established (e.g. DEAPspace [82], Intentional Naming System (INS) and INS/Twine [83] [84], Secure Service Discovery Service (SSDS) [85]). In pervasive computing field, there are a number of device-oriented service discovery mechanisms which have been widely used, such as the Bluetooth Service Discovery Protocol [86], , Jini [87], Salutation [88], Service Location Protocol (SLP) Version 2 [89], and Universal Plug and Play (UPnP)

[90]. However, none of these service discovery mechanisms provide the semantic level at the interaction behavior and they only support key-word comparisons or string-based searches. For example, in the Bluetooth Service Discovery Protocol, a 24-bit Class or Device field and class groupings are defined for describing service information, and a 128-bit UUID (Universally Unique Identifiers) is used for matching service requests against existing devices; in Salutation, a service template is defined to describe services and a directory service matches service requests against service descriptions precisely according to the same attribute values. Furthermore, these protocols were designed for various different discovery ranges and some of them have certain assumptions, such as the requirement of a Java virtual machine for Salutation, Jini and SLP, which may make such approaches impractical in a number of application scenarios.

Traditional service discovery approaches do not consider the user information, which means any services may be discovered and located by any users. This does not satisfy the privacy requirement for pervasive and mobile computing applications. Hence, built-in security features are gradually becoming a fundamental demand for the service discovery model. One of the possible approaches is to make clients and services authenticate with the directories for service lookups and announcements respectively. Various security features could be added to this architecture including authorization, data and service privacy, and information integrity. This solution is suitable for enterprise environments, where services are willing to expose their service information to central directories [85].

A service discovery model, named "Splendor", supports nomadic users in public environments [91]. Splendor offers mutual authentication among components, simplifies service authorization, provides communication confidentiality and message integrity, and supports non-repudiation. User privacy, data privacy, and user location privacy are achieved. Also, location-awareness is integrated into this service discovery protocol to support location dependent services better and reduce the requirements of the underlying network infrastructure. Splendor provides a client-service-proxy-directory model explicitly, where the proxy is used to achieve privacy, authentication and load-sharing. Based on "Splendor", authors built a user-centric model, named "Prudent Exposure", keeping both service and user information privately and securely. Trust between users and service providers is established based on code words exchanged in an efficient form. The "Prudent Exposure" model provides an effective way for authorized users to discover services simply, while hiding services from unauthorized uses [92] [93].

Frequently, a task cannot be performed by using a simple service only, but the combination of more than one service can provide all the desired functionalities. Service composition is the process of selecting, combining, and executing services to achieve the user objective. Service composition can be performed manually by users themselves or by programs automatically. The foundation of automatic service composition is a well-defined service description mechanism. As long as the information of the service is encoded explicitly in an unambiguous and computer-understandable way, software is

then able to compose, select and execute a collection of services to achieve the required user task.

In the field of web services, a set of standards named web service orchestration are defined to arrange and manage different services to achieve a desired goal. Web service orchestration transforms service discovery and coordination into long-running transactions and business processes [94], and provides an open, standards-based approach for connecting web services together to create higher-level business processes. Here, "business process" means a high-level description of the process and an abstraction of any particular implementation of the process. Web service orchestration can be achieved by using specialist languages to describe executing workflows. These languages include:

- Web Service Choreography Interface (WSCI) [95]: a specification from BEA Systems, Intalio, SAP AG, and Sun Microsystems which was issued in May 2002. WSCI defines an XML-based interface description language that describes the messages between web services that participate in a collaborative exchange.
- Business Process Execution Language (BPEL) [96]: created by IBM, BEA and Microsoft in August 2002 and supported by two complementary specifications (WS-transactions and WS-Coordination). A BPEL process is an XML document typically generated with graphical design tools by business analysts rather than programmers and the process is executed by an execution engine.
- Business Process Management Language (BPML) [97]: a specification from the BPMI.org (Business Process Management Initiative Organisation). It aims at providing a comprehensive means of specifying the process of an enterprise.

Many existing research projects have been focusing on building the workflow executing engine to implement the web service composition and invocation. For example, in [98], the workflow engine in terms of the BPEL language supports the execution of any processes from the service providers. The user submit the request of the service invocation to the workflow engine and the workflow engine is responsible for communicating with service providers via SOAP messaging. In the Access to Knowledge through the Grid in a Mobile World (Akogrimo) project [99], the system architecture contains a workflow layer, in which the most appropriate workflow will be selected for the user semantic requests. In both of these projects, the workflow was regarded as the available executing unit and the underlying advertised services are totally transparent for users.

3.3.3 Context Awareness

Context awareness is an important requirement for achieving the goal of providing Grid services that are appropriate for mobile users at the right time, in the right format, at

the right device. "Context" refers to the information that can be used to characterize the identity and attributes of entities in the environment and is required for both pervasive computing and Grid computing applications. When both of these systems are combined together, the context information about the interaction between various computing entities is more critical.

A key requirement for realizing a context-aware system is to provide computer systems with the capability to understand their situational conditions. As discussed in the chapter two, semantic web languages are believed to be suitable for the purpose of representing the contextual information and a shared ontology is usually required to be designed and recorded in a public database for supporting knowledge sharing, context reasoning and information interoperability.

Generally speaking, there are two main approaches of creating context awareness systems [100]: self-supported context awareness, and infrastructure-supported context awareness. In the self-supported context awareness approach, context receiving, reasoning, and responding are designed within a device. One of the typical examples is SenSay projects [101] that proposes to create self-supported context-awareness mobile phones. The SenSay device uses internal sensors to detect phone situations (e.g. on a table, in the hand) and also uses a number of wearable sensors (e.g. microphones, accelerometers) to determine the user location. Based on the internal state of the phone which is determined by the situation of both users and phones, SenSay is able to decide ringer or vibration levels and which action should be taken in response to an incoming call.

In the infrastructure-supported context awareness approach, a system infrastructure receives the contextual information from sensors and reasons with them if required. The system infrastructure then gives feedback to the related applications (Figure 3.5). The context information is shared between the infrastructure and other entities in the computing environment.

The infrastructure-supported approach provides more complex context awareness behaviors than the self-supported context awareness approach. The benefits of applying standard, widely-used and publicly accessible technologies to build a context awareness system involve:

- The infrastructure is decoupled from the specific applications, new entities such as linked services, sensors, and processing components can be added and updated while the whole system is still running.
- The infrastructure allows different devices and applications to use various computation resources in the environment. Both devices and applications are hardware, platform and language independence.

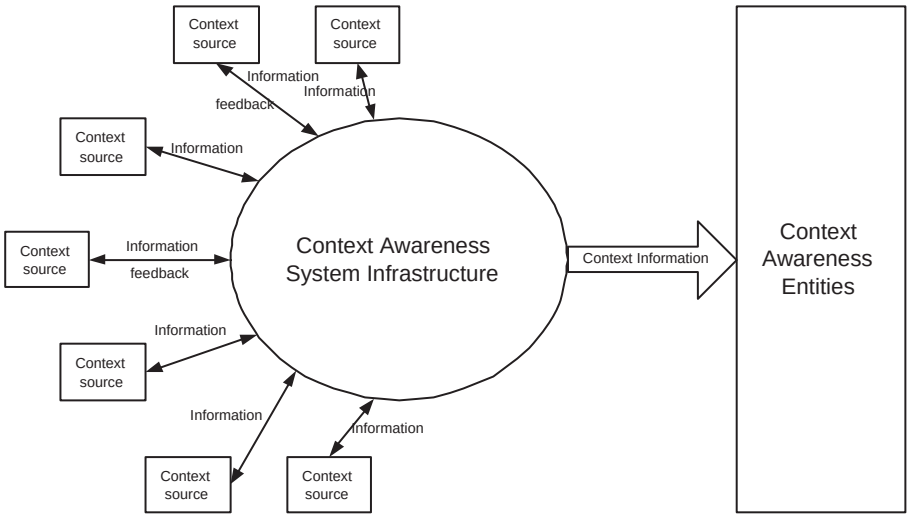

FIGURE 3.5: The relationship between context awareness infrastructure and other entities.

- Various sensors can feed the context information to the infrastructure, which is then used to produce the respond action for different applications. In this condition, a sensor is able to produce the maximum effect.

A representative context awareness infrastructure is a context broker architecture (CoBrA) [102], which was built for smart spaces by utilizing ontology approach. The ContextBroker infrastructure is based on a central agent responsible for gathering, processing and sharing context information with other elements of the system infrastructure.

The ContextBroker infrastructure using Semantic Web languages (RDF and OWL) to define context ontologies is implemented based on them. Defined context ontologies assist the context broker to share knowledge with other agents. In the Context Broker Architecture, a resource-rich agent is provided to manage and maintain a shared model of context for all devices, services and agents in an associated space. The context reasoning in CoBrA offers context brokers the ability to infer new context knowledge that cannot be directly acquired from the physical sensors. The use of policies in CoBrA allows users to control their contextual information, specifying the range of information that can be shared by the systems and choosing recipients to receive notifications when the context changes.

The ContextBroker system attempts to address the bottleneck issue by replacing a single central context broker with several federal brokers in a large-scale case, where each broker is responsible for a part of overall space. However, there is still a single broker agent which is responsible for all context information in a certain area. More important, trust must be assumed between different broker agents to allow all relevant context information to be shared and accessed.

A context-aware system architecture consists of a number of components which are responsible for implementing various required functions. Different context-aware systems have diverse internal structures and the components are named in terms of the developer preference. However, through the analysis and comparison, it is found that components that provide similar functionalities exist in context-aware system infrastructures (e.g. CoBra, MyCampus [103]) which rely heavily on using ontology techniques to define underlying concepts, although their naming may vary. Hence, choosing a mature context-aware system architecture to be the example is beneficial for developing the new system in the future.

Semantic space [73] is such a standard context awareness system infrastructure that adopts semantic web technologies to support context representation, querying and reasoning. Figure 3.6 shows its system infrastructure:

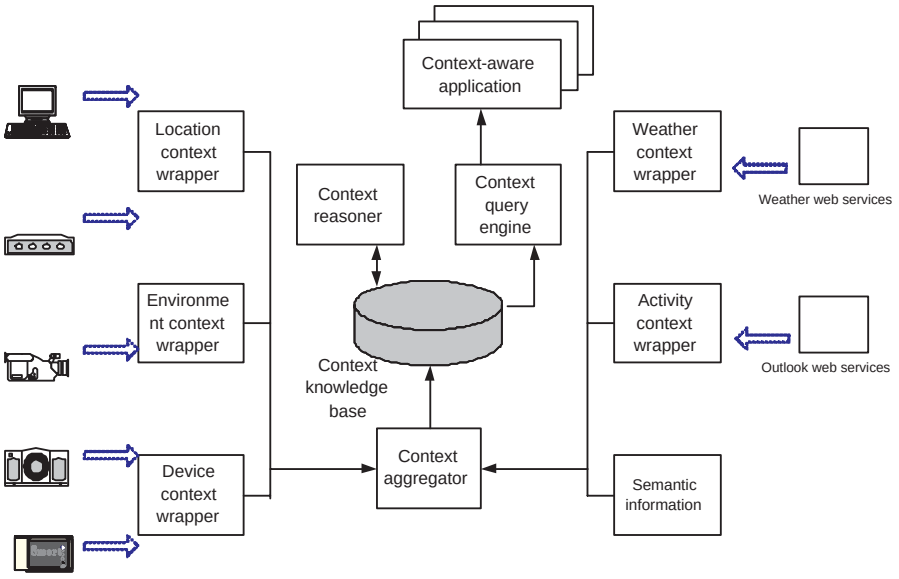

FIGURE 3.6: The Semantic Space System Infrastructure (from [73])

- Context wrapper: obtaining raw context information from both hardware sensors and software programs.
- Context aggregator: discovering context wrappers, registering context wrappers, and gathering information from context wrappers; monitoring context wrappers and managing the scope of contexts in the context knowledge base.
- Context knowledge base: storing the context knowledge.
- Context query engine: providing an interface for applications to extract desired contexts from the context knowledge base.
- Context reasoner: inferring abstract high-level contexts according to basic sensed contexts.

3.3.4 Mobility

Generally speaking, mobility can be considered to be a characteristic in three aspects: device, service, and person. Device mobility is about the location change of a device whilst still maintaining the active communication or reconnect to the network in the new location. Service mobility refers to the idea of a service moving between different devices, connected in different places of the network. Personal mobility indicates a person moving and using different devices to access a remote service.

Traditional Grid service technologies do not support the mobility of Grid services. Grid services are usually fixed on the Grid node machines and cannot be moved to other nodes even if these nodes have extra resources. By adding the ability of service migration, the original static Grid service is improved to become a mobile Grid service. In a mobile Grid service environment, Grid services are able to leave their host node and migrate to other Grid nodes which contain extra free resources. By using this kind of service mobility technology, both services and resources can be coordinated and the usages of the Grid resource can be maximized [104].

Although Grid service migration is a recently-developed research area, there are several research projects working on it. In [105], the authors aim to develop a middleware framework to support secure mobile Grid services. The system framework is built by combining the Java Agent Development Framework (JADE) [106] and a generic Grid system toolkit. The mobility vision of Grid services is realized by distributing the actual working tasks to JADE mobile agents. This mobile Grid service middleware is a plug-in component to the Globus Grid architecture.

The terms of "Wireless" and "Mobility" have many interacting features. However, they are not exactly the same thing. If a client uses mobile wireless devices to access Grid services in a fixed point, no mobility issues are required to addressed. On the contrary, a client may move the wired devices between different places. Every time the event such as network disconnection and reconnection occurs, mobility issues have to be considered.

Many research work attempts to find possible solutions for device mobility. For example, in [107], a scheduling algorithm for mobile Grid environments has been proposed. The algorithm takes into account the intermittent connectivity of mobile nodes, which determines whether it would result in the best job performance when including a mobile node in the computing. The algorithm is evaluated by both mathematical analysis and experimental implementation methods.

Mobile devices (users) are able to roam freely, from one place to another, connecting to different networks at different places. When a mobile device arrives at a new location, the application often receives a better processing performance by using the resources at the new place than by using the previous resources. One option for the application moving is migration: suspending the application on the previous resources, transmitting

the application state to the resources at the new place, and resuming the application. However, this approach of the application migration has an unavoidable drawback: the application is unavailable for users while it is migrating.

Slingshot [28] uses an alternative strategy that solves the problem of device moving while the application is still running. It creates a first-class replica for every application on a reliable server known to the mobile user and allows all the application state to be extracted and reconstructed from the information stored on this reliable server. When the second-class replica is moved from one place to another, the application program deployed on the first-class server is still available for mobile users. When the application moving is finished, the application resumes in the new resources after reconstructing the current executing state by querying the first-class replica on the reliable server.

In a pervasive Grid environment, mobile users may enter a physical site and leave without completing their tasks. The environment and the service middleware have to detect the absence of the user and inform other elements so that they can response correctly. For example, when a new mobile device enters a smart site, the inspection service can detect it and notify this event to the related device management service. The management service may create a new object for the incoming device and insert the new device into its monitoring list. If the device moves in the environment to a new location, the device management service will get the information from the detection service and may migrate its executing object to the new location. Finally, when the device disappears from the environment (either disconnecting intentionally or accidently), the device management will delete the relevant object with assigned resources.

When users or devices move from one location to another, corresponding system components should be notified to make right decisions for further actions. In both the pervasive and Grid computing fields, the event-oriented middleware infrastructure is utilized to enable system components to communicate with each other. For example, Globus Toolkit 4, implements parts of WS-BaseNotification and WS-Topics. The interaction model defined by a set of WS-Notification specifications is based on the publish and subscribe paradigm. Producers publish a class of events, and consumers subscribe to events in order to obtain the notification of events of that class. In [108], an asynchronous communication broker is implemented which is in charge of dispatching asynchronous events in the pervasive Grid environment.

3.3.5 Autonomic Behavior

In both Grid and pervasive computing systems, autonomic behaviors such as self-configuration, self-management, self-optimization and self-healing are desired properties [17]. For example, in the dynamically changing pervasive computing environment, the system should be configured automatically to satisfy the requirement of multiple users.

When more and more devices are added to the Grid environment, a self-optimization resource discovery and scheduling protocol may allow these mobile nodes to share computing resources in an efficient way.

It is well known that realizing autonomic features is difficult, and many researchers from various areas (e.g. computer systems, networking) have been concentrating on it. In [109], the authors believe one of the key aspects is the ability to understand the system behavior more deeply. A new approach to understanding the behavior of a distributed system and helping to debug performance problems is outlined in the paper.

The policy-based system draws much attention in the autonomic computing area because system administrators represent operation rules to guide the system behavior. It is more straightforward to define and modify the system behavior by using policies compared to the traditional method of direct program coding in the system. At present, a number of research projects are focusing on building policy-based systems. For example, AGILE [110], as both a policy expression language and a framework, facilitates the integration and composition of several autonomic computing techniques and technologies. The policy language semantics can bind various components together as required, to support run-time reconfiguration and dynamical composition of the self-management architecture.

Semantic web technologies, defining a way of presenting the machine-understandable knowledge, provides a means of realizing the autonomic vision. For example, by using OWL-S language to encode the information of a web service in an unambiguous machine-processable form, web service composition can be automatically completed by the program. Several research projects have been concentrating on this area [111] [112]. They may either resolve the service matching through the comparison between web service inputs, outputs, preconditions and effects (IOPE), or select services by checking the properties, capabilities or functionalities of web services.

Agent-based computing offers the potential to implement a range of autonomic behavior. An agent, by definition, (e.g. "An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives" [113]), is capable of autonomous actions. Agents are often implemented with various kinds of reasoning mechanism. It is believed that agent technologies will address the issue of autonomic behavior if they are utilized by both pervasive and Grid computing middlewares [114].

3.4 The State of the Art

Generally speaking, there are two possible role for mobile devices in a service-oriented Grid environment [107]. The first one is that mobile devices can be considered to

be physical interfaces to the Grid. In this role, mobile devices act as Grid service consumers only, in order to enable mobile clients to execute complex tasks through their portable devices, submit requests for various distributed resources in the Grid environment, monitor the tasks being executed on the Grid and collect service execution results from the Grid to complete their tasks. The other role is that mobile devices can act as resource nodes to participate in the Grid. In this kind of scenario, mobile devices are Grid service or resource providers rather than service consumers.

The existing Grid infrastructure and architecture do not take into account the design and implementation of integrating mobile systems because mobile devices have always been considered to be not well suited as Grid computing clients, interfaces, and resource nodes. However, integrating mobile devices into the Grid environment brings considerable attention and interest to current research people. Because mobile devices are considered as service consumers in our research work, the sections that follows will discuss the related research work which is specifically focused on Grid interface research work.

3.4.1 Grid Interface Research Work

The idea of enhancing capabilities of mobile devices by using Grid services can be traced to "Cyber Foraging". As discussed in chapter two, it is the concept of dynamically augmenting capabilities of a pervasive device by offloading complex tasks to local high-performance desktops or workstations. Although in the "Cyber Foraging" systems, the complex tasks are transferred to local computing resources, which restricts the integration and computation power enhancement of mobile devices, it is not difficult to extend the utilization of the local computing resources to remote distributed resources. In [27], authors believe that the widely-deployed"Cyber Foraging" infrastructures can allow mobile clients to access a group of servers on which Grid-like computation could be performed.

As improvement of hardware equipments and development of wireless network technologies, mobile devices are gradually integrated into Grid computing field. In [115], after discussing the challenges required to be addressed to integrate mobile wireless devices into the computational Grid as resource nodes, the authors present a proxy-based clustered architecture. In this architecture, each cluster is composed by a central proxy device named interlocutor and a number of mobile device units named minors. The interlocutor represents minor devices to the Grid and publishes itself as a normal node of the Grid. Although no testbeds of the interlocutor and minor devices are implemented, the authors believe their proxy-based clustered architecture addresses a number of important research challenges:

- The interlocutor acts as an agent of a number of mobile devices, which reduces N long-haul request negotiations to N short-haul request negotiations and one long-haul request.
- The interlocutor hides the unreliability of mobile devices, by publishing only one node on the Grid and reconsidering the resources of the node when any mobile device decides to leave.
- After the request is partitioned into different mobile devices, some devices may disconnected. The interlocutor hides this disconnection by caching results from other devices and waiting for the reconnection of those offline devices.
- By reducing the long-haul communication to the short-haul communication, mobile devices consume less power.
- On behalf of its mobile devices, the interlocutor simplify the interaction between nodes on the Grid, such as Grid service discovery.

The proxy-based approach is a popular means of integrating mobile devices into the Grid environment. There are many other research projects concentrating on deploying a proxy device between mobile clients and Grid services. In [81], the authors describe their attempts to write Grid clients for mobile devices and implement a web-based proxy using Globus Toolkit 3 to communicate with distributed Grid services. Their experiences are based on an implementation of a mobile Grid client for an existing web-based e-learning system. The proxy-based Grid client is regarded as the best option when taking into account resource and network limitations of mobile devices after the authors investigate the other two approaches - Java and .NET frameworks. However, the function of the proxy device is only responsible for transmitting the request from the mobile device to the Grid by using the lightweight Java Server Page (JSP) technology, with very little independent processing and executing management.

In [116] and [117], the Grid is modeled as a flexible, self-configuring dynamic network of independent, mobile, intelligent agents which use resources of other agents to solve a shared computational task. A problem-solving environment for wireless mobile devices is built using the computational Grid paradigm based on the distributed mobile agent architecture in cellular networks. Each mobile device in this Grid-based environment runs a client application which implements a lightweight communication protocol for information exchange with the Grid brokering service. Several aspects about building the architecture are discussed, including agent roles, mobility issues, network configuration, and so on. However, the Grid in their computing model is not the traditional static resource-rich system infrastructure, but a network infrastructure where the computation load can be redistributed to other wireless devices.

The ITRC program sponsored by the Korea Ministry of Information and Communications supports a series of projects to design middlewares that assist mobile handheld

devices to interact with Grid services. In [118] and [119], the authors present a three-tier system architecture, in which the "Gateway Surrogate Host" provides the execution environment and access to extensive resources for the surrogate of the mobile handheld device. The service access and task processing mechanism are wrapped in a "surrogate" module, which is hosted in the file system of the device or stored at a web server. The surrogate can communicate back with the middleware stack at the mobile device. By using this architecture, the authors believe a mobile handheld device can be enabled to access the generic Grid service and offload an intensive task processing to a resource rich system. A bare-bones implementation of the proposed architecture is implemented with the reference implementation of Jini Surrogate Architecture specification to measure the impact of the executing system and highlight benefits and shortcomings of the approach. In the test application, the surrogate has been coded for the mobile handheld device with the functionality of monitoring the remote server and the result display. Besides presenting the three-tier architecture, the authors also discuss an optimization mechanism for discovering and selecting the host machine and the resource on the host.

The middleware is exposed as a web service for the client applications when deploying a proxy device as the intermediary between mobile clients and Grid services, in [120]. The middleware service purposes to support the delegation of jobs to the Grid, the secure communication between clients and service, the offline processing and the adaptation to network connectivity. Although it seems that the middleware is not actually implemented, the authors have built a formal model to simulate system operation and demonstrate the potential performance by executing this interaction model.

SuMMIT (Submission, Monitoring and Management of Interactions of Tasks) is another proxy-based framework for coordinating applications execution in mobile Grid environment[121]. Mobile users are able to submit several tasks at one time for solving a complex problem and these tasks are accepted and dispatched to the Grid scheduler by the Workflow Manager component in the middleware. In order to prevent a failure state due to the device disconnection, the Agent component checks the connection of mobile devices in a determined time interval. Mobile users can also monitor the execution process through handheld devices. SuMMIT adopts workflow concept to manage submitted tasks because it was built mainly for the sequence analysis of DNA, RNA and proteins in bioinformatic area. In order to enable it to be practically used, a flexible Grid resource matching engine is required.

Except the proxy-based system architecture, there are other existing approaches to integrating mobile devices into the Grid environment. Microsoft's .NET framework is a development environment that enables users to develop service-oriented applications. The OGSI.NET implementation is a container that allows .NET applications to access Grid services. The cut-down version, mobile OGSI.NET, aims to bring the benefit of Grid computing to mobile users while solving problems such as resource limitations and intermittent network connectivity [122]. The authors believe mobile OGSI.NET is a

valid attempt to provide an OGSI implementation for mobile devices. However, perhaps because the OGSI specification has been replaced by WSRF, the mobile OGSI.NET tool that supports the investigation into potential interoperability issues with GT3 is no longer available. The authors suggest that in the future, the working experience with Mobile OGSI.NET will be applied to design and implement Mobile WSRF.NET.

As discussed in chapter 3.3.2 and 3.3.3, Grid service discovery and context awareness are two important requirements when designing a mobile Grid system. However, most of current research work concentrates on building a framework which can integrate mobile devices into the Grid environment and few of them demonstrate solutions for these two requirements. The issues of Grid service discovery and context-awareness are considered in our research work, and Semantic Web technologies are adopted to implement a flexible service matching engine and a context-aware interaction mechanism between mobile users and the computing environment.

3.4.2 Grid Resource Provider Research Work

The capabilities of mobile devices have been continuously improved over recent years and these devices are now in widespread use. It is possible that mobile devices can be aggregated into the Grid environment as resource providers for computational distribution effectively. This new potential has attracted many researchers to work on realizing a vision of executing Grid computations over networks of mobile devices.

Although the proxy-based architecture can hide the heterogeneity and uncertainty of mobile devices, allowing mobile devices to act as Grid resources nodes needs to address a greater number of problems compared with Grid interface research work. For example, job decomposition and distribution must be considered because a job which can be processed by a single static Grid node definitely cannot be accomplished by a single mobile node. Furthermore, a fault tolerance mechanism is required because the disconnection of a mobile device may cause the failure of task execution.

Job scheduling is the task of managing resources and application execution depending on the properties of resource allocators and resource owners, which is more important in mobile Grid environment. In [107], a scheduling algorithm which takes into account the intermittent connectivity of mobile nodes is proposed. The purpose of the algorithm is to acquire the best job execution performance when using appropriate mobile nodes. In [123], the authors present a generic mobile Grid infrastructure and introduce a simple node mobility tracking scheme and a optimal job allocation algorithm based on the infrastructure. The authors are confident with their algorithm, concluding that it shows much better system performance compared with other existing schemes.

In [124], a group-based resource selection algorithm that supports fault-tolerance in mobile Grid is proposed. Mobile devices are grouped and ranked based on their battery power, mobility and performance. After calculating rank, the job will be assigned to top-N groups concurrently. The algorithm will be applied in the real mobile Grid environment in the future.

The project discussed in [125] implements a proxy-based clustered architecture to deal with the mobility in the Grid. The proxy device groups all mobile devices located in the same subnet and presents them as one single virtual resource to the Grid. The service aggregation and indexing components of the proxy engine are implemented by using existing web service and Grid technologies. The authors model a heavily loaded data-centric Grid environment with the OPNET evaluation tool to ensure the whole Grid environment does not have a performance or network traffic bottleneck introduced due to the proxy layer architecture.

3.5 Summary

Mobile devices are becoming more and more prevalent around the world. However, users are expecting more integration and power to support new applications on mobile devices. The Grid computing infrastructure, designed to provide flexible, secure and coordinated resource sharing, has significant potential to enhance the capability of mobile devices. General speaking, mobile devices need the Grid for computation and integration, and the Grid needs mobile devices to interface with the physical world.

This chapter begins with two kinds of scenario: an information-access scenario and a work-assistant scenario. In both of these scenarios, mobile devices need to interact with the Grid environment to accomplish a variety of tasks. The capabilities of mobile devices are enhanced by utilizing the required Grid services and offloading resource-intensive work to more powerful Grid resources.

Having identified the dependent relationship between the Grid and intelligent mobile computing, several common requirements of realizing the vision of integrating mobile devices into the Grid environment have been discussed. These requirements are suitable middleware, service discovery and composition, context awareness, mobility, and autonomic behavior. Several existing research projects which focus on combining mobile devices with Grid services have also been discussed. Mobile devices can act as two kinds of possible role in the Grid environment: physical Grid interfaces, or Grid resource providers.

The next chapter presents a context-aware framework to provide enhanced Grid access for mobile devices. The mobile device is thus the Grid service consumer, utilizing various services to executing complicated tasks.

Chapter 4

Context-aware Framework

4.1 Introduction

In a mobile computing environment, users are able to access ubiquitous services smoothly with their portable devices. The computing environment detects the user existence or absence and configures service operations automatically based on the related context information. If Grid services are deployed in a mobile computing environment, applications based on Grid services are also required to be made context aware because of the dynamic nature of users and service terminals. In fact, one of the important requirements of realizing enhanced Grid access for mobile devices is that the interaction between mobile users and Grid services should be in the right way at the right time in the right place at the right device. For example, let us consider the following example:

Tao submits a data-processing task to the service-oriented Grid environment through his mobile phone. The system accepts the task and starts to execute the task. During the task execution, additional inputs are required from Tao. Before sending the data request to Tao, the system checks the current context information and finds Tao is in a meeting with his supervisor, an activity which cannot be interrupted. So the system postpones the remainder message until the meeting is over. Tao receives the message and inputs required parameters to continue the task execution. After the task execution is over, the system checks the context information again to see whether it is suitable to return the result now and which kind of format is appropriate. Tao is at a seminar room with his laptop. The system then sends a message to Tao, asking whether he wants to see the result on his laptop. If so, the result is upscaled for the display size of the laptop and transferred back to Tao.

Building such an intelligent interaction mechanism requires computer systems to have the ability of understanding the context information of the computing environment. Both Grid service consumers and service providers need to share their knowledge with

each other, acquiring various context information of the system, and support automatic behaviors. Applications need to be context-aware so that they can adapt to rapid changing conditions [126] [127].

Two essential issues are required to be considered for implementing a context-aware system:

- In a ubiquitous computing environment, a variety of raw context data is produced in heterogeneous formats from different data sources (e.g. embedded devices). However, the raw context data cannot be used in applications which have no prior knowledge about the context representation. An explicit representation of the context meanings therefore is required to enable raw data to be understood by independently-developed applications.
- Context is any information that characterizes the status of entities in the computing environment, and its utilization is essential for implementing a context-aware system. Requests such as a context query or context reasoning may be submitted by applications or other entities of the system, and the answers for these context requests have to be replied. For example, is a specific user still available in the system? What is the status of the task submitted three hours ago?

To address these issues, a context-aware framework is required to implement the intelligent interaction mechanism between mobile users and Grid services. The context-aware framework represents various information in ways that are adequate for machine understanding, processing and reasoning. As discussed in the previous chapter, Semantic Web technologies with their supporting tools which provide a very considerable degree of automatic processing, interoperation and integration can be used to build the context-aware framework. By defining a shared context model with standard Semantic Web languages, and storing the context model in a public knowledge base, the system architecture can be enhanced with information interoperability, knowledge sharing, context querying and reasoning [128].

4.2 Context Model

4.2.1 Context

Context is any information that can be used to characterize the status of various entities in the computing space [58]. In the system architecture of mobile devices accessing Grid services, context can be further defined as all information that characterizes the identity and attributes of physical and virtual entities (e.g. mobile users, computing devices, Grid services and submitted tasks). This information includes the personal information of mobile users, the performance and capability of various computing devices,

the description of Grid services, and the task that a user submits. Context may also involve the surrounding environment of entities, such as the location of the mobile users and Grid services, because these conditions are able to affect the interaction between entities. A well-defined underlying information model is required to be built in order to represent all of these context information explicitly.

4.2.2 Ontology Design

Ontology building is a relatively immature field without any established methodology. In [129], the authors summarize existing ontology design methodologies that have been employed by ontology working groups. Some methods are designed for use with particular applications and are not appropriate for ontology building [67] [130]. Methontology [131] is an application-independent approach, which is suitable for developing the ontology because our purpose is to establish a general context model for various applications in the mobile Grid environment. Furthermore, Methontology has been used by several different research groups for ontology development in diverse domains. In this thesis, we adopt the Methontology approach to design the ontology.

Methontology consists of four development stages to build an ontology:

1. Specification: determining the possible and potential uses of the ontology.
2. Conceptualization: identifying a conceptual model, including concepts and relationships between concepts.
3. Formalization and implementation: transforming the conceptual model into a formal model and representing it with a formal ontology language.
4. Maintenance: checking, improving and updating the constructed ontology.

Context awareness is a general concept for the dynamic interaction environment. The context model, as the foundation of the context-aware framework, can benefit any applications which utilize mobile devices as a user interface for offloading tasks to a resource-rich Grid environment. The ontology is designed to represent the context model in an explicit way so that applications are able to understand various information in the computing environment. In order to provide the universality and to enhance extensibility, the ontology is divided into two distinctive but related parts, the basic context ontology and the extended ontology. The basic ontology defines a set of fundamental concepts and general vocabularies for a general service-oriented mobile Grid computing environment, while the extended ontology, inherited from the basic ontology, provides additional concepts and vocabularies to support practical application scenarios in the computing environment.

The basic ontology consist of six classes intended to cover most of the basic elements envisaged in a general mobile Grid environment. These classes are "User", "Device", "Place", "Time", "Task" and "Status". Each class has a number of subclasses for representing elements more explicitly. The six up level classes, together with their subclasses and other extended classes, form the framework of the system context information, and provide the possibility of modeling other potential context in the application scenario.

Figure 4.1 shows the diagram of six up-level classes, several example extended classes and their relationship in the ontology structure.

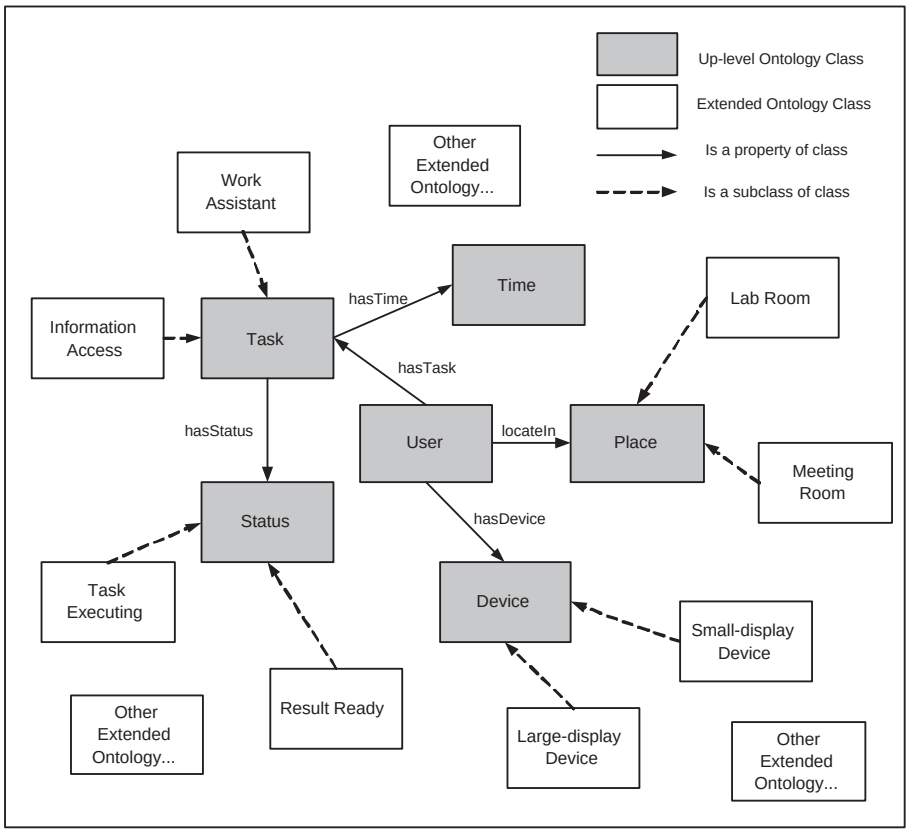

FIGURE 4.1: Ontology Architecture.

The class "User" describes the general features of a mobile user in the service-oriented Grid environment, which has a number of properties. The following are several important properties:

- hasDevice: Mobile users attempt to access the Grid environment through their handheld devices. The value of this property is the instance of class "Device".
- hasTask: Mobile users may submit their tasks to the Grid environment through their handheld devices. The value of this property is the instance of class "Task". It is assumed that at most one task can be submitted by a mobile user.
- locateIn: Users have the location information anytime, for example in the meeting room or in the office. Its value is the instance of class "Place".

- accessLevel: An accessing level is assigned to every user during the initialization process, which is used to determine whether a service can can be exposed to the user.

The class "Device" describes the general features of a device in the service-oriented Grid environment. It may be either a static computing resource in the Grid environment, or a mobile device used by users to access the Grid environment. Several important properties include:

- hasProfile: A device may be a service consumer or a service provider in the computing environment. Only one profile value can be associated with a device.
- hardwareDescription: The value of this property is an instance of the class "HardwareResource", which describes the details about the hardware of the device, including the display size, the memory capacity of a device and so on.
- softwareDescription: This property is used to describe the software resources of the device. Its value is an instance of class "SoftwareResource".
- connectStatus: This property is defined for the "Mobile Device", a subclass of the "Device" class. It indicates whether a mobile device connects to the Grid environment.

The class "Task" describes the general features of a task submitted from a mobile user. Its key properties includes:

- hasUser: This property indicates where the task comes from. Its value is an instance of the "User" class.
- hasTime: This property indicates the time when the task submission occurs. Its value is an instance of the "Time" class.
- hasStatus: The task has its execution status, which is recorded by using this property. Its value is given according to the value collection the "Status" class defines. Only one status description can be associated to a task at one time.

The extended ontology, as the complement of the classes defined in the basic ontology, is used to customize the context model for a particular application. Additional classes required for new applications can be obtained by inheritance from the top-level classes, which enables application developers to build context models conveniently for new computing environments. Furthermore, because of the shared terms and definitions of the top-level ontology, better interoperability is supported between different extended context ontologies.

4.2.3 Describing Contexts

In our service-oriented mobile Grid environment, users submit requests to invoke Grid services to perform their tasks. The ontology classes can be used to describe the relevant concepts and their possible relationships defining the interaction between mobile users and Grid services. The central unit of this interaction is the "User" class (from the ontology diagram), which has a direct connection with the "Device", "Place", and "Task" classes.

The "User" class defines the most general attributes about a mobile user in the Grid environment. A mechanism of binding properties between users and their mobile devices is adopted because devices are the physical interfaces to the virtual Grid environment. In this approach, the entry interface asks the users to type their personal information, based on which a "User" instance will be created. At the same time, related instances such as "Device", "Place", and "Task" may also be created, to indicate that this mobile user submits a task at a given place through a given mobile device. The "User" instance also includes basic personal information (e.g. the name, contact information) and the relationship with other user instances. The user attributes such as the location or the current task status can be acquired through querying with related ontology instances. The following shows a partial context description for the user "Tao Guan" based on defined ontology classes.

```
<User rdf:ID=‘‘TaoGuan’’>
  <name rdf:datatype=‘‘&xsd;string>TaoGuan</name>
  <gender rdf:resource=‘‘&usr;Male’’/>
  <birthdate rdf:datatype=‘‘&xsd;date’’>1979-05-23</birthdate>
  <homepage rdf:resource=‘‘http://www.ecs.soton.ac.uk/people/tg04r’’/>
  <email rdf:about=‘‘mailto:tg04r@ecs.soton.ac.uk’’/>
  <hasschoolcontact rdf:resource=‘‘#SchoolContact’’/>
  <supervisorOf rdf:resource=‘‘#EdZaluska’’/>
  <hasDevice rdf:resource=‘‘#NokiaN73’’/>
  <hasTask rdf:resource=‘‘#Task01’’/>
  < ......More properties......  >
</User>

<ContactProfile rdf:ID=‘‘SchoolContact’’>
  <address rdf:datatype=‘‘&xsd;string’’>School of ECS, University of
Southampton, Southampton, SO17 1BJ, UK</address>
  <phone rdf:datatype=‘‘$xsd;string’’>+44 (0)23 80598371</phone>
  <email rdf:resource=‘‘mailto:tg04r@ecs.soton.ac.uk’’/>
</ContactProfile>
```

In the system architecture, various computing devices are either service consumers or service providers. The "Device" class is used to characterize their attributes. Standard vocabularies are defined for describing the information and the profile of the device, including the device ID, the physical address and listening port of the Grid gateway, the current working status of the device, the available resources of the gateway, the registered Grid services in a information centre, the service discovery protocol, the authentication mechanism an so on. The following example shows an ontology description for a Grid gateway machine to be used in the demonstration:

```
<Device rdf:ID=‘‘DemoGateway’’>
  <hasProfile rdf:resource=‘‘#Serviceprovider’’/>
  <hardwareDescription rdf:resource=‘‘#Hardwareresource1’’/>
  <softwareDescription rdf:resource=‘‘#Softwaresource1’’/>
  < ......More properties......  >
</Device>

<DeviceProfile rdf:ID=‘‘Hardwareresource1’’>
  <type rdf:resource=‘‘#Desktop’’>
  <CPU rdf:datatype=‘‘&xsd;string’’>P4 3.0GHz</dev:CPU>
  <memory rdf:datatype=‘‘$xsd;string’’>Dual Channel 512MBx2</memory>
  <network rdf:datatype=‘‘$xsd;string’’>100MB Ethenet</network>
  <harddisk rdf:datatype=‘‘$xsd;string’’>Matrox 120GB</harddisk>
  <listeningport rdf:datatype=‘‘$xsd;string’’>1000</listeningport>
  < ......More properties......  >
</DeviceProfile>

<DeviceProfile rdf:ID=‘‘Softwareresource1’’>
  <OS rdf:datatype=‘‘&xsd;string’’>Ubuntu Breezy</OS>
  <Platform rdf:datatype=‘‘$xsd;string’’>Apache Tomcat</platform>
  <VirtualTech rdf:datatype=‘‘$xsd;string’’>Linux-Vserver</VirtualTech>
  <Gridservice rdf:resource=‘‘#AvailableGridService1/>
  < ......More properties......  >
</DeviceProfile>
```

The "Place" class is designed for expressing spatial relations, which can be used to describe the location properties of the task submission that occurs in the physical world. The "Time" class, designed for expressing temporal relations, which can be used to describe the temporal properties of the task submission related with the system architecture. When a mobile user submits a task for the system, the system records its location and time, which are stored as properties of the task instance and the user instance. The context reasoner may produce a new context information based on the time and location of the user and transmit the information to other computing entities in the system architecture.

The above four classes, “User”, “Device”, “Time” and “Place” are classes defining objects of the physical world. The “Task” and “Status” class, on the other hand, specifies the conceptual object in the digital world. In our mobile Grid system architecture, the ”Task” class is normally used to describe the occurrence of a mobile user request. For example, a traveler may request the task of resizing photos, uploading the processed photos to a given web location, and sending photos to his friends. A physicist may request to process a large volume of data obtained from various sensors. Every “Task” instance has a set of properties, including the requester of the task, the location and the time the task submits, and the “Status” of the task execution.

The ontology instances are often used together to describe a practical event that occurs in the mobile Grid environment. The following description shows “TaoGuan” submitting “LocalMapSearch” task at the “MeetingRoom” location through his “Nokia N73” device.

```
<owl:Thing rdf:ID=‘‘Task01’’ />

<owl:Thing rdf:about=‘‘#Task01’’ >
  <rdf:type rdf:resource=‘‘#Task’’ />
  <hasUser rdf:resource=‘‘#TaoGuan’’/>
  <taskName rdf:resource=‘‘#LocalMapSearch’’/>
  <time rdf:datatype=‘‘xsd;dateTime’’> 2006-08-01 16:45:03 </time>
</owl:Thing>

<owl:Thing rdf:ID=‘‘TaoGuan’’ />

<owl:Thing rdf:about=‘‘#TaoGuan’’>
  <rdf:type rdf:resource=‘‘#User’’ />
  <hasDevice rdf:resource=‘‘#Nokia N73’’/>
  <hasTask rdf:resource=‘‘#Task01’’/>
  <location rdf:resource=‘‘#Meetingroom’’/>
</Task01>
```

4.3 The Context Framework

Based on the context model constructed which represents various user activities and environment entities in an explicit way, a context-aware framework enables applications to retrieve required context information and acquire high-level contexts through reasoning using these basic contexts. A context-aware framework consists of context resources, a knowledge base, and applications, as shown in Figure 4.2.

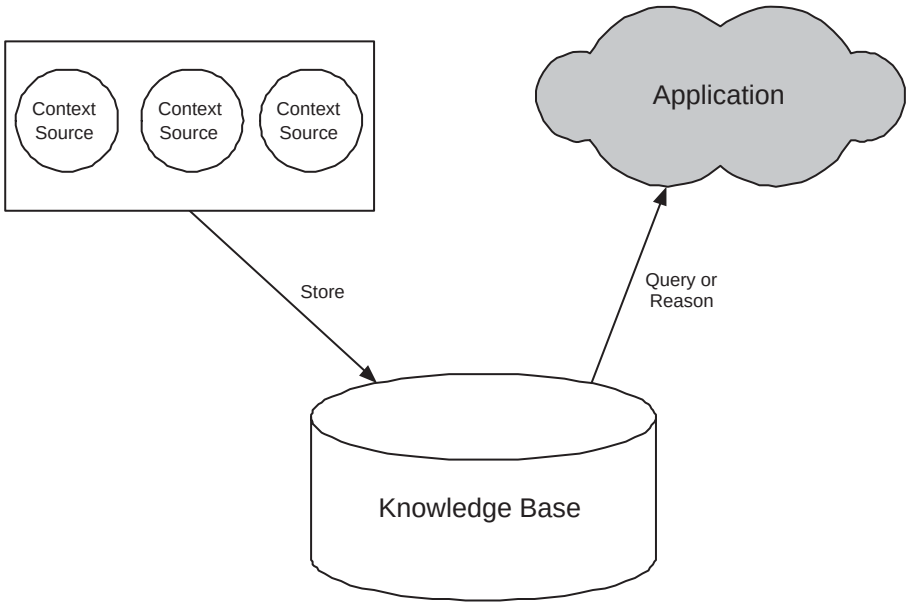

FIGURE 4.2: The Interaction between context resources, the knowledge base and context-aware applications.

4.3.1 Context Sources

Context sources process raw context data and convert them into the knowledge base. Generally speaking, context sources can be divided into two categories: hardware sensors and software programs. Hardware-based context sources comprise a variety of equipments to detect the conditions of physical entities in the computing environment. A typical example is an RFID sensor, which is able to add new context markups into the knowledge base when detecting that a new user comes into the environment. Software-based context sources are a set of programs which are deployed in the system and responsible for monitoring or extracting required information for the knowledge base.

In our context-aware framework to support intelligent interaction between mobile users and Grid services, three kinds of context sources are considered:

- User Location context: The "User" class of the context model has the "locateIn" property, the value of which indicates the place the user is located at. The location context sources require that either the environment is equipped with sensors that detect the presence of mobile users, or particular devices which mobile users can connect to with their handheld devices. There exist a number of mature positioning technologies which can be utilized to implement user location context sources. For example, a Wi-Fi locating device is able to locate Wi-Fi enabled mobile devices by periodically interacting with Wi-Fi access points. When a new device enters into the Wi-Fi covered area, an event is written into the log file. By comparing log files on different Wi-Fi access points, it is possible to know that users change location from one place to another [132]. The RFID technique is another approach

for locating RFID-tagged users in the computing environment [133]. The detailed discussion and implementation of the user location context source are beyond the scope of this thesis and it is assumed that the user location context source is already available for us.

- Device Connection context: The device connection context source is responsible for monitoring the connection status between mobile devices and the service-oriented Grid environment. If a device loses the connection with the system, the device connection context source will change the value of the connection status in the "Device" class. The implementation of the connection monitoring component is discussed in the Chapter 6.

- User Profile context: When a new user connects to the Grid environment, the "User" instance as well as its related "Device" and "Task" instances will be generated based on personal information, the handheld device, and the submitted tasks. In our system architecture, when a new user logs into the system, the JavaScript interface and processing programs enable required instances to be created and stored in the knowledge base.

4.3.2 The Knowledge Base

A place is required to store the ontology classes and the context description information produced by the computing environment. The knowledge base is a special kind of database which provides persistent context knowledge storage and management, supporting a means for the context information collection, organization and retrieval.

Knowledge base technologies can be categorized into two major types [134]:

- Machine-readable knowledge bases store knowledge in a computer-readable form, usually for the purpose of having automated deductive reasoning applied to them. They contain a set of data, often in the form of rules that describe the knowledge in a logically-consistent manner. Logical operators such as And (conjunction), Or (disjunction), material implication and negation may be used to build the data up from atomic knowledge. This allows classical deduction logic to be used to reason about the knowledge.

- Human-readable knowledge bases are designed to allow people to retrieve and use the knowledge they contain, primarily for training purposes. They are commonly used to capture explicit knowledge of an organization, including troubleshooting, articles, white papers, user manuals and others. The primary benefit of such a knowledge base is to provide a means to discover solutions to problems that have existing known solutions which can then be reapplied by others, less experienced in the specific problem area.

The knowledge base is the component which stores the ontology in the format of RDF triples. A RDF triple consists of a subject, a predicate and an object. The subject identifies what resource the triple is describing, the predicate defines the properties in the resource a value is given, and the object is the actual value. Triple stores are the key database in the semantic web world and designed to share a large volume of RDF triples with simple retrieval.

The knowledge base can use an ontology to specify its structure and its classification scheme. An ontology, together with a set of instances of ontology classes, constitutes a knowledge base.

The problem of storing and managing the data in the format of triples has been explored by the graph database, object database, PROLOG language [135] and, more recently, Semantic Web communities. Several toolkits for building a knowledge base which stores the ontology and the RDF data have been implemented. For example, Sesame [136], a generic architecture for storing and querying RDFS and RDF, is designed to use existing storage systems such as various relational database management systems (RDBMS) as the underlying persistent store. Sesame has already implemented the interface for existing database technologies (e.g. MySQL, Oracle, PostgreSQL), and been used for developing the triple store in many systems.

Jena [137] is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena also provides for persistent storage of the RDF data in relational databases. The same interfaces such as Model, Resource and Query, are used to access and manipulate persistently stored RDF. The application does not directly access the database nor does it need to be aware of the database schema.

4.3.2.1 Context Storage

In the Jena software environment, the storage of RDF triples and ontologies is implemented by integrating the RDBMS-specific codes into a Java API, which can be used to create and manipulate RDF graphs. Jena has object classes to represent graphs, resources, properties and literals. The interfaces representing resources, properties and literals are called Resource, Property and literal respectively. Several existing RDBMS, RDF stores and RDF files can be imported by using Jena APIs. The first step in any operation that involves Jena programming codes is to create an empty model using the ModelFactory class. After the model is created, RDF data can be added to the repository and users are able to query the knowledge repository to acquire any resources they require.

- Creating a RDF model. In Jena, all databases are multi-model and each model is, by default, stored in a separate table. Models may share database tables using the

StoreWithModel option. There are two mechanisms for creating persistent models, one using factory methods and another using constructors for the ModelRDB class. Creating or opening a model is a three-step process. Firstly, the driver class must be loaded and a connection established to the database (In Jena, the database type is specified as part of the database connection). Secondly, a model maker class is constructed. The model maker creates persistent instances of the Model class. Thirdly, the model maker class is invoked to create new models or to open existing models.

- Adding RDF to the Model. The Jena Java API offers several methods for adding data to a created Model. Data can be added by specifying the location of a file that contains RDF data, and statements can be added individually or in collections. The following code demonstrates creating a RDF persistent model using factory methods and reading the statements recorded in a RDF/XML file into a Model:

```
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.ontology.*;
import java.io.*;
import java.util.*;

// database URL
String M-DB-URL = ‘‘jdbc:mysql://localhost/test’’;
// User name
String M-DB-USER = ‘‘test’’;
// Password
String M-DB-PASSWD = ‘‘’’;
// Database engine name
String M-DB = ‘‘MySQL’’;
// JDBC driver
String M-DBDRIVER-CLASS = ‘‘com.mysql.jdbc.Driver’’;
// load the the driver class
Class.forName(M-DBDRIVER-CLASS);
// create a database connection
IDBConnection conn = new DBConnection(M-DB-URL, M-DB-USER,
M-DB-PASSWD,M-DB);
// create a model maker with the given connection parameters
ModelMaker maker = ModelFactory.createModelRDBMaker(conn);
// create a default model
Model defModel = maker.createDefaultModel();
...
// Open existing default model
```

```
Model defModel = maker.openModel();
InputStream in = FileManager.get().open(inputFileName);
//read the RDF file
defModel.read(in, ‘‘’’);
```

4.3.2.2 Context Query

In the Jena software environment, the repository API provides a number of methods to access and search information held in a Model. Given the URI of a resource, the resource object can be retrieved from a model using *Model.getResource(String uri)* method. This method is defined to return a Resource object if one exists in the model, or otherwise to create a new one.

If there are no known URIs, Jena provides several methods which deal with searching a model. Generally, the result of the model search is a set of RDF statements, which is very useful for extracting sub-graphs from the stored RDF data. The core Jena API supports only a limited query primitive. The more powerful query facility is SPARQL language [138] [139].

SPARQL is a query language and a protocol for accessing RDF designed by the W3C RDF Data Access Working Group, which can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL offers context querying in the created semantic model based on RDF triple patterns (<subject, predicate, object>). The following example shows a SPARQL query string to find the task that Tao submits in the knowledge base.

```
SELECT ?task
WHERE
{
<http://example.org/user/Tao>
  <http://example.org/property/hasTask> ?task
}
```

The Jena Java APIs provide methods of executing queries based on the SPARQL query string. A query is created from a string using the QueryFactory. The query and model or RDF dataset to be queried are then passed to QueryExecutionFactory to produce an instance of a query execution. Result are handled in a loop and finally the query execution is closed. The following shows a code fragment including the basic steps when executing a select query:

```
import com.hp.hpl.jena.query.* ;
Model model = ...  ;
```

```
String queryString = ...
Query query = QueryFactory.create(queryString);
//Execute the query and obtain results
QueryExecution qe = QueryExecutionFactory.create(query, model);
ResultSet results = qe.execSelect();
//Output query results
ResultSetFormatter.out(System.out, results, query);
//Important - free up resources used running the query
qe.close();
model.close();
```

4.3.2.3 Context Reasoning

Sometimes, external applications require high-level context information which can be acquired directly by querying the context model. The Jena inference subsystem provides a range of inference engines or reasoners, which are used to derive required high-level context information from basic RDF and ontology information as well as rules associated with reasoners.

For example, during the process of task execution, the system will check the user current status before it contacts the user for additional task input data. By querying the knowledge base, the information that the user is at his supervisor's office may be obtained. If a predefined application rule is added into the reasoning system, the result that the user cannot be interrupted can be obtained. The following shows the related customized rules, which indicate that the user1 is meeting his supervisor when the user1 and the user2 are in the office and the user1 is the student of the user2.

```
[rule:  (?user1 studentOf ?user2) (?user1 locateIn ?officeroom)
  (?user2 locateIn ?officeroom) -> (?user1 status MeetingSupervisor)]
```

A number of reasoners are provided in the Jena distribution, including RDF rule and OWL reasoners. For our context-aware framework, the generic rule reasoner is more useful because it allows rule-based inference based on application-specific rules. The following shows an example of the code fragment which implements the general rule reasoning:

```
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.ontology.*;

.. .. .. .. ..
Model model = ‘‘...’’;
```

```
String rules = ‘‘[r:  (?user1 studentOf ?user2) (?user1
locateIn ?officeroom) (?user2 locateIn ?officeroom) -> (?user1
status MeetingSupervisor)]’’;
Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rules));
InfModel inf = ModelFactory.createInfModel(reasoner, model);
Iterator list = inf.listStatements(user1, status, (RDFNode)null);
while (list.hasNext()) {
  System.out.println(‘‘ - ’’ + list.next());
}
```

4.3.3 The Context-aware Framework Architecture

The context-aware framework represents various context information in ways that are adequate for machine processing, querying and reasoning, which enables the intelligent interactions between mobile users and Grid services. Figure 4.3 shows the internal architecture, which is composed of context sources, the context monitor, the query and reasoning engine, and the knowledge base. Other components in the service-oriented Grid environment can add and query the context information stored in the knowledge base through exposed external interfaces.

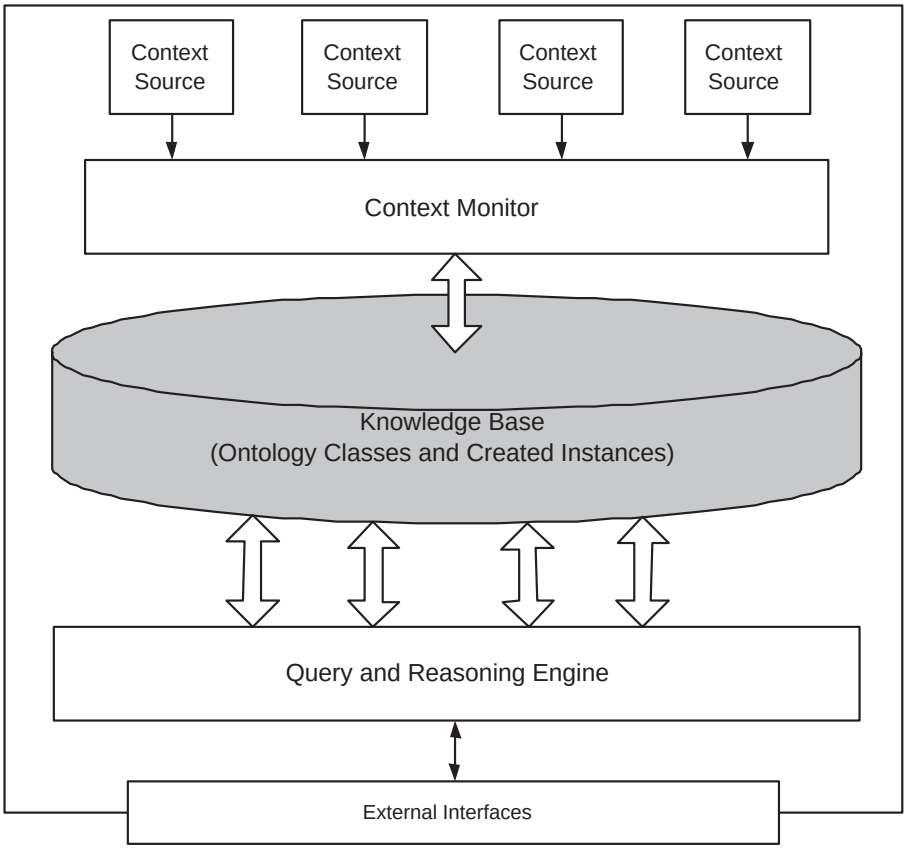

FIGURE 4.3: Internal Architecture of the Context-aware Framework.

4.4 Implementation

The ontology designed for the context model has been implemented using the Protege toolkit [140], an open-source editor and knowledge-based framework, and exported as an OWL file using the Protege OWL plug-in. Jena provides the function of syntax and consistency checking for defined ontologies. The ontology is maintained and updated in Protege (Figure 4.4), for example, adding a new class, removing an inconsistent restriction for a given property.

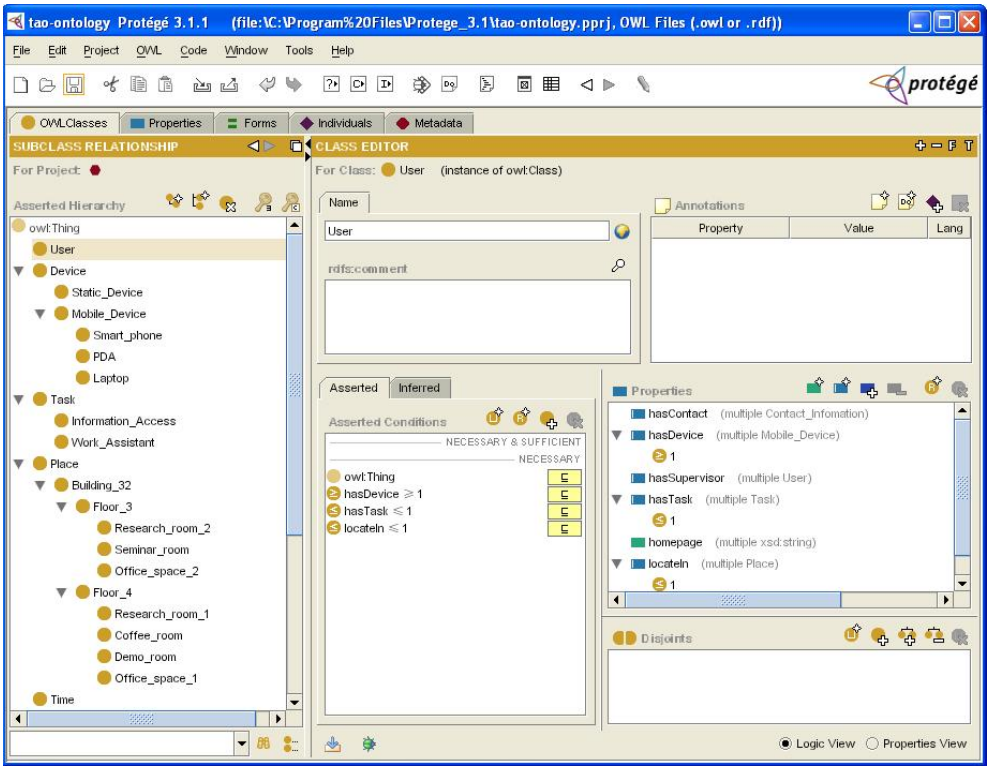

FIGURE 4.4: A Snapshot of the Ontology under Protege.

Based on the ontology created, the knowledge base, context query and context reasoning engine are implemented using the Jena Semantic Web toolkit. All of them have same the interfaces which are used to access and operate on persistently stored RDF data, hiding the underlying implementation details from high-level applications. In our framework, MySQL database is adopted to store the RDF triple data.

The context-aware framework is difficult to evaluate because users are more concerned about its functionalities rather than its performance. Our context-aware framework represents explicitly various contexts of the computing environment in which mobile devices are able to access Grid services, allows external applications to retrieve contexts using the query interface and provides high-level contexts through reasoning basic contexts. The context-aware framework exposes a number of programming interfaces which enable external components of the system architecture to add and obtain required

context information, supporting an seamless interaction between mobile users and the service-oriented Grid environment.

4.4.1 Application Experiments

Two experiments were carried out in order to test the functionalities of the context-aware framework and verify whether it satisfies the intended purpose described in the example scenario at the beginning of this chapter. The experiments also demonstrate that the context-aware framework can help to implement intelligent interaction between mobile users and Grid applications.

Experiment 1: determining the appropriate action according to information acquired from context reasoning.

In this experiment, we create new instances and inserted them into the knowledge base. A web interface is designed for users to input various information (e.g. personal information, the type of handheld devices, submitted tasks). In the experiment, we assume that user *Tao* submits a complex task through his handheld devices and additional data inputs are required during the task execution. Corresponding individual instances (e.g. User instance, Device instance, Task instance) are created by the background server based on the ontology class definition which can be retrieved from the knowledge base. We then used Jena Java query interfaces to check whether the property values are what were given from the web interface. A program is started, simulating that a data request is sent back to *Tao*. Before the action, the program checks the current status of *Tao*. Here, we added an application rule into the system framework: if *Tao* is in *Ed*'s office and *Ed* is the supervisor of *Tao* (*Ed* is another user instance which is created before the experiment), then *Tao* is in the "busy" status because he is meeting his supervisor. If Tao is busy, the program has to suspend for a while and check Tao's state after a given time. If not, the program will output a string, simulating the data request sends to *Tao*.

Experiment 2: transferring the appropriate result format according to the user handheld device.

In this experiment, we assume the task submitted in the experiment one has been completed and its result needs to be transferred back to *Tao*. Before the program transfers the result, it checks *Tao*'s current handheld device using SPARQL query strings:

```
    SELECT ?device
    WHERE
    {
    <http://example.org/user/Tao> <http://example.org/property/hasDevice>
?device
```

}

After obtaining the query result, the program assesses whether it is a large-size display device or a small-size display device (In the ontology class definition, we define the mobile phone as the small-size display device and the laptop as the large-size display device). Based on the assessment, the program will send the results back to *Tao* in the appropriate format.

Through the test, the intended purposes are satisfied. In the experiment one, we set *Ed's office* to the value of *Tao*'s location at the beginning. The simulation program has been suspended because *Tao* is always in the "busy" status. After changing the value of Tao's location to *research area*, the program breaks away from the waiting state and a string is displayed to simulate that *Tao* receives the data request. In the experiment two, the program output changes according to the value of *Tao*'s handheld device type which is modified periodically by another program.

In Chapter six, the mobile deputy middleware will be associated tightly with the context-aware framework, providing a reliable task execution mechanism to assist mobile users to interact with Grid services.

4.4.2 Performance

To evaluate the performance of the context-aware framework, we designed the following experiment. A testing program is written which implements a series of discrete steps:

- a semantic model loads a defined ontology file.
- a number of individual instances are created based on "User", "Device" and "Task" classes, simulating users submitting tasks through their mobile devices.
- corresponding instances are edited, including setting the correct value for instance properties.
- a context query operation with the SPARQL string is executed.
- a context reasoning operation based on custom rules is executed.

The testing program includes such a series of steps because these are typical operations for a context model and usually required by external components to implement the seamless interaction between mobile users and Grid applications. The testing program does not include the procedure of initializing a semantic model because it is a prerequisite of constructing a context-aware framework and is assumed to be completed during the system initialization.

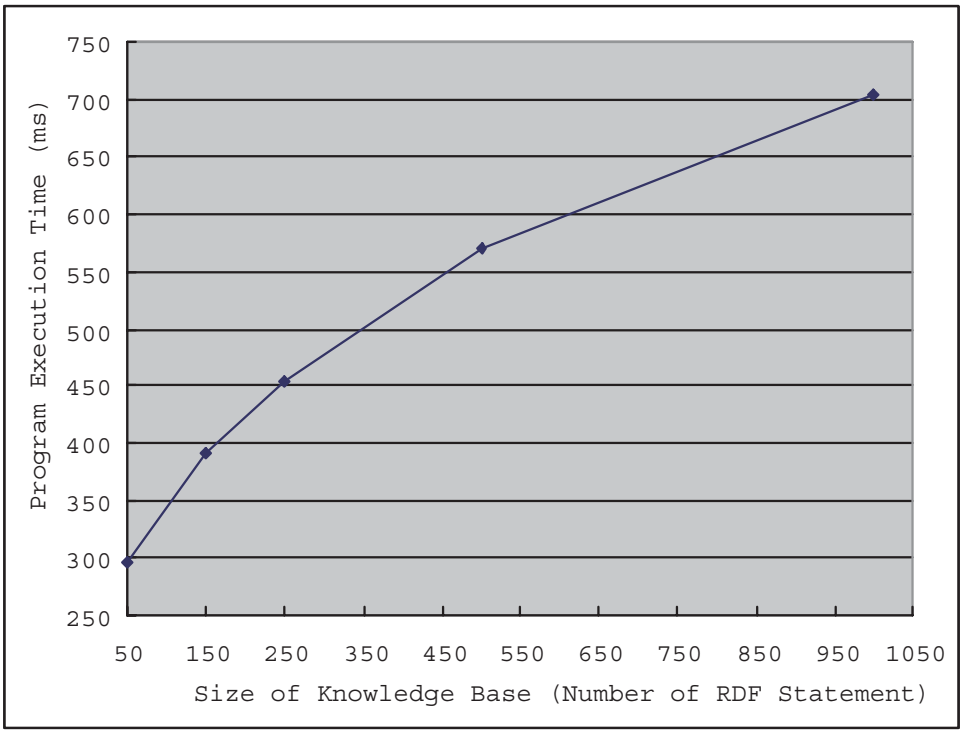

FIGURE 4.5: Program Execution Time vs. Size of Knowledge Base.

The testing program is executed on a desktop which has a 2.4GHz Pentium CPU and 1.0 Gbyte of RAM. We measured the program execution time under different sizes of the context knowledge base. Figure 4.5 shows the experimental results, which indicates that the program execution time increases with the number of RDF triples of the knowledge base (from 50 to 1000). However, the program execution time is loosely proportional to the size of the knowledge base, and the time of operating a context model is within an acceptable limit (below one second) when the context model does not have a large number of classes and the context query and reasoning statement is not too complex.

4.5 Summary

Because of the dynamic nature, the interaction between mobile users and Grid services should be implemented in an intelligent way, which demands that the underlying computer systems have the ability of understanding the context information of the computing environment. To achieve this, a variety of contextual information data should be represented in ways that are suitable for machine understanding, processing, querying and reasoning.

We believe that a context-aware framework is required in order to implement the envisaged intelligent interaction between mobile users and Grid services. In this chapter, a context-aware framework has been built with Semantic Web technologies and their

supporting tools. By defining a shared context model and integrating various context information into a public knowledge base, the context-aware framework provides the functionalities of explicit context representation, context query, and context reasoning.

The context-aware framework is the foundation for building a system architecture to realize enhanced Grid access for mobile devices. With its assistance, the mobile deputy middleware can provide a reliable task execution mechanism for mobile users (which will be discussed in the chapter six).

Chapter 5

Grid Service Description and Discovery

5.1 Introduction

The purpose of Grid computing is to coordinate resource sharing in dynamic and multi-institutional virtual organizations to enable heterogeneous resources to be joined up to accomplish new functionalities and capabilities. As a service-oriented approach to Grid computing is increasingly adopted, many systems which can discover Grid resources on-the-fly and access them on demand come into existence. In a service-oriented Grid environment, clients are usually concerned about three aspects when considering using Grid services to perform their tasks. The first is a method that describes what capabilities need to be developed so that services can be advertised to provide contributions for the task achievement. The second is building a Grid service discovery mechanism so that services can be located and selected. The third is the mechanism of Grid service invocation so that required information or resources can be acquired during the process of task execution. Essentially, Grid service description, discovery and execution are interdependent: Grid service description is a prerequisite for Grid service discovery; the mechanism of Grid service discovery determines how a Grid service should be described; the service execution process depends on the discovery of the Grid service.

Generally speaking, service discovery protocols simplify the interaction between service consumers and service providers. Various existing service discovery protocols have been introduced during the past few years. In the field of mobile computing, example description and discovery solutions are Bluetooth Service Discovery Protocol [86], Jini [87], and Universal Plug and Play (UPnP) [90]. In the field of Web Services, Universal Description Discovery and Integration (UDDI) [141] is a platform-independent and XML-based registry which enables businesses to publish service listings and discover each other. However, none of these service discovery mechanisms support flexible matching between

service advertisements and requests, and users can only locate services on the basis of the syntactical equivalence of keywords or strings which have been agreed beforehand.

Semantic Web technologies attempt to connect the gap between human-understandable data and machine-understandable data. Enabling a computer to understand human-oriented information is a historical problem in many areas, from artificial intelligence to Internet computing and pervasive computing. The original purpose of Semantic Web technologies was to concentrate on the development and implementation of the vision of Semantic Web. For example, management of resources in the Semantic Web currently relies on the use of ontologies, which can be considered as the high-level metadata of web data and knowledge. However, demands to state the meaning of data explicitly also exist in other research areas. If different systems or different components in one system share data or transfer "meaningful" messages, the relevant basic semantics are required to be defined and agreed in advance.

With the proliferation of Grid services, semantic specifications of Grid services are gradually becoming a necessary requirement of the automatic, flexible service provision and utilization necessary for Grid clients to perform various tasks. It is not straightforward for a request to locate required services in order to execute a task in a service-rich Grid environment. Semantics of Grid services abstract top-level concepts and relationships between concepts so that both the service discovery and the automatic conversion of interaction formats for the service execution can be realized. Furthermore, a semantic definition mechanism provides a comprehensive representation of a variety of Grid service aspects, building an essential foundation for possible automatic behaviors throughout the whole Grid service development lifecycle.

A semantic approach for the service discovery includes two essential research issues: service description and service matching. Service description provides a foundation for service matching because a service provider should represent all characteristics of a service in an explicit way so that it can be understood and processed by the service matching engine. The service matching engine is responsible for comparing the service request against the service description in order to assess whether a service satisfies the requirement of a request. As long as users describe their service requirements with terms from the same semantic model used to build the service descriptions, logical reasoning mechanisms in the service matching engine can discover any semantic similarity between the service description and the user requirement.

Semantic Web technologies give services rich semantic specifications to enable flexible automation of the service provision and use. A number of research projects have been undertaken to enhance service discovery mechanisms with Semantic Web technologies in a variety of domains. For example, in the field of Web Services, in order to solve limitations in traditional Web Service discovery techniques, a semantic layer was built between users and the Web Service WSDL description [142] [143] [144], which makes

it possible to generate a web service request using high-level abstract concepts rather than syntactic level terms. However, most of these systems perform service matching by using service function attributes (e.g. service inputs, outputs, preconditions and effects), which is not adequate for building a comprehensive service description. Their service matching engine also have limitations for concept comparison (e.g. a subsumption relationship has to exist between concepts). These restrictions must be considered in our service discovery mechanism.

In this chapter, we present an approach to flexible service discovery by comparing the semantic content of the service request and service advertisements in the mobile Grid environment. The traditional and improved semantic-enhanced Web Service discovery mechanisms are introduced first, because generally speaking, Grid services are stateful Web Services. A number of service attributes have been defined to represent service characteristics in the service description. The service matching engine compares the service request against the service description in two steps: strict requirements have to be matched precisely; while general requirements are checked based on the user-expected matching level. Finally, matching services as well as their matching degrees are returned as results for the service request.

5.2 Web Service Description and Discovery

Web services are a specific realization of a service oriented architecture in which various services registered in UDDI interact with each other by exchanging messages in SOAP format while the contracts for the message exchange that implement those interactions are described in WSDL. This description incorporates three web service architecture important specifications:

- SOAP: the Simple Object Access Protocol, which exchanges XML-based messages over computer networks, normally using HTTP/HTTPS. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework that more abstract layers can build on.

  Invoking a web service involves passing messages between the client and the server. SOAP specifies how requests should be formated to the server by the client, and how the server should format its responses. There are alternative web service invocation protocols (e.g. XML-RPC, COBRA), but SOAP is currently the most prevalent choice for web services.

- WSDL: the Web Services Description Language, which is an XML-formatted language used to describe capabilities of a web service as collections of communication endpoints capable of exchanging messages. WSDL is established as an important

building block in the evolving development of web service technologies and is standardized in the W3C's Web Services Description Working Group.

One of the most interesting features of web services is that they are self-describing: when a client locates a web service, the service can be asked to describe itself and tell the client what operations it provides and how to invoke it. WSDL is responsible for transmitting detailed information from the service provider to the service consumer.

- UDDI: Universal Description Discovery and Integration (UDDI), an open industry initiative sponsored by OASIS, is a platform-independent, XML-based registry which enables businesses to publish service listings and discover each other, and defines how the services or software applications interact over the Internet. The objective of UDDI is to create an Internet wide network of registries of web services.

  UDDI obtains the support from a number of prominent software and hardware companies that concentrate on the development and application of web services, and it has become the de facto standard web service repository. Although UDDI only provides a weak discovery mechanism, it provides a data structure to detail a set of characteristics of web services that can help the process of web service discovery and selection.

5.2.1 Traditional Web Service Description and Discovery

Figure 5.1 shows a detailed view of the interaction between the web service client, the web service registry, and the web service provider.

The first procedure of the interaction is that the web service advertises itself in a UDDI registry. The registration exposes the information about the service provider which allows the service to be discovered. When a client needs to access a web service, several key words about the desired web service should be prepared and submitted to the UDDI system in a service discovery request. The responsibility of the UDDI system is to find and select the services which seems to match key words closely to satisfy the service client. A reply is produced which indicates the candidate servers which may provide the services required to match the request. Finally, the client knows the location of the desired web service, and it will initiate a communication to ask the web service to describe itself. The web service answers a description document in the WSDL format which includes information about how to invoke it.

Web services describe themselves with WSDL language and adopt UDDI for the purpose of advertisement and discovery. Although they have been well established as the foundational standard specifications of web service technologies, they can only provide an service discovery mechanism based on the exact syntactic match:

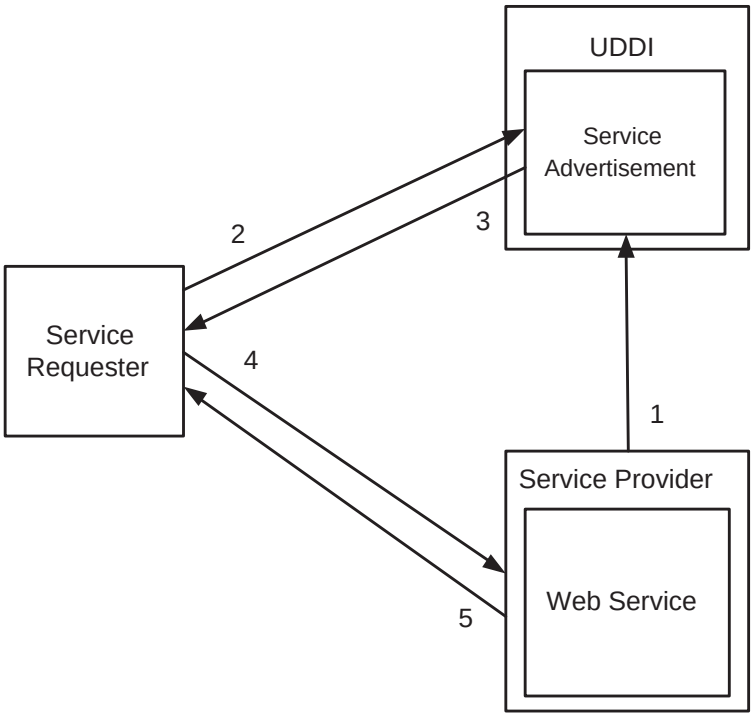

FIGURE 5.1: Description of Web Service Discovery and Interaction.

- WSDL uses XML to describe the syntax of input and output messages of a web service, as well as other details required for the invocation of the service. XML provides syntactic interoperability between service clients and service providers. However, it fails to enable semantic operation between the client and the provider. For example, two similar WSDL service descriptions may have quite different meaning, while two different WSDL descriptions may have similar meaning. Using XML to describe web services thus does not permit the vision that the matching process will understand services to be realized.

- Although UDDI allows businesses to publish service listings and discover each other, creating a general-purpose web service registry, it does not provide a discovery mechanism based on web service capabilities. The internal data structure of UDDI makes it possible to combine an unbounded set of attributes for the web service description. However, it only provides a keyword-based search engine for service providers, service entities and tModels (the internal data structure of the UDDI system) in the repository. The underlying reason of this limitation is that it does not have an explicit mechanism of representing web services. Hence, the problem of the lack of a service description mechanism has to be solved. Otherwise, it will hinder the implementation of discovering web services dynamically.

5.2.2 The Semantic Approach to Web Service Discovery

In order to solve the limitations in the Web Service standard specification, a number of research projects have been undertaken to extend the basic Web Service architecture and enhance the service discovery mechanism by adding the semantic interoperability on top of existing techniques.

In [145], the authors introduce a new concept of "Semantic Web Services", which associates the growing Web Service architecture with the semantic web technologies. They believe the semantic web and Web Services are synergistic: "the semantic web transforms the web into a repository of computer readable data, while Web Services provide the tools for the automatic use of the data". DAML-S, an ontology for describing semantic web services, is used to support the implementation of the effective service capability matching and interaction management between service consumers and service providers.

DAML-S ontology consists of three up level components: the DAML-S service profile, the DAML-S process model, and the DAML-S grounding. The service profile describes the capability (including both functional and non-functional attributes) of web services and specifies the intended purposes of the service. The process model provides a more detailed description of web services than the service profile, which enables the service client to extract the interaction protocol and decides how to interact with the service provider. The DAML-S grounding maps processes and instances in the service model into the WSDL operations and messages, integrating the abstract information in the service description with the detailed service implementation.

The most important contribution of DAML-S ontology for Web Service discovery is that it can be used to describe various aspects of service advertisements, especially for the description of the service capabilities. Based on service capability description using DAML-S ontology, a new system component, the matchmaker, is designed and implemented in [145]. The matchmaker uses the description logic reasoner to compare ontology based service descriptions. Figure 5.2 shows a detailed view of DAML-S web service interaction, which is similar to the traditional client-UDDI-provider web service interaction model. The difference is that UDDI is replaced by the DAML-S matchmaker and all components in the web service infrastructure can understand the meaning of the messages exchanged.

At first, the web service provider advertises its capabilities in the DAML-S Matchmaker. The registration of service capabilities enables the web service to be discovered by the client submitting a service request description. The matchmaker bases the automatic service discovery on reasoning and finding a match between the explicit service descriptions and the client requirement. The underlying interaction between the service client and the service provider is defined by the process model and grounding, which providers a set of standard interfaces to implement the automatic service invocation.

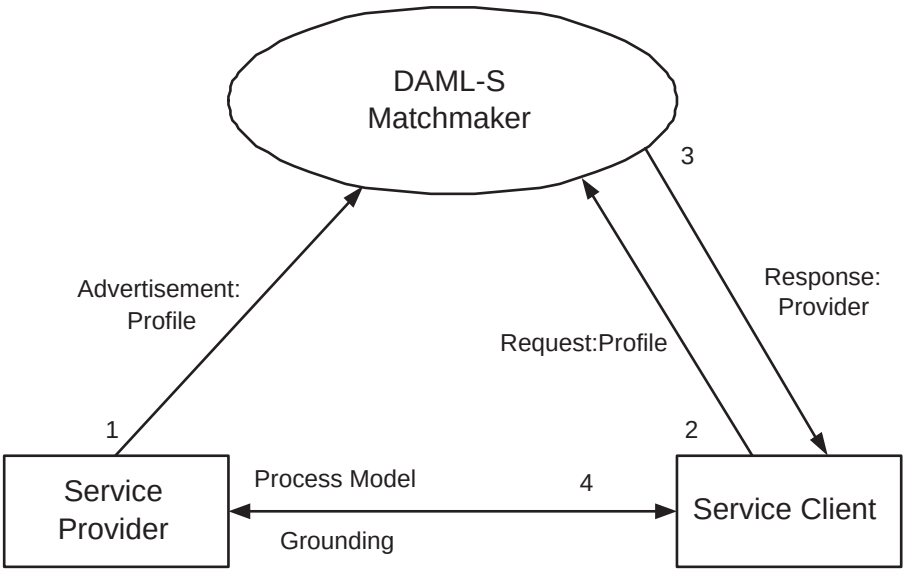

FIGURE 5.2: DAML-S Web Service Discovery and Interaction.

OWL-S, which is built from DAML-S, is an ontology language to create service descriptions together with other aspects of the OWL description language. It provides a semantic description layer for web services which is not supported by the more low-level service-syntax oriented Web Service Description Language (WSDL). Similar to its predecessor DAML-S, the structure of the OWL-S upper-level ontology is based on three perspectives:

- The "Service Profile" provides the high-level descriptive information of a service, including its name, contractor, capabilities (inputs, outputs, preconditions, effects), the classification of the service in a taxonomy and additional text description, for the purpose of advertising services, constructing requesters and matchmaking. The service profile specifies the intended purpose of the service, because not all functionalities of a service are described in the service profile. The service provider has the choice of advertising all or part of the functionality to the public, and the exposed information determines how the service will be discovered by the client.

- The "Process Model" describes how a service performs its function. It is a specification of the ways a client may interact with a service and includes comprehensive and detailed information about service inputs, outputs, preconditions and effects. The process model defines two interaction models between the client and the service: the atomic process and the composite process. An atomic process is an elementary "black box" of the service functionality, in effect a procedure that expects one request message and returns one message in response. A composite process is the combination of several atomic processes, which shows the possible

flow of control and data in a complex web service. The process model is important for automatic service invocation, enactment, composition and monitoring.

- The "Service Grounding" maps the atomic process in the service process model onto a detailed messaging protocol and specifies how a service invoked. OWL-S provides for different types of groundings to be used, however, the only type developed at present is the WSDL grounding, which makes it possible to abstract a web service to be a semantic web service using OWL-S. In the service grounding, a particular WSDL operation corresponds to an atomic process, and syntactic IO elements specified by WSDL are transformed to OWL-expressed inputs or outputs directly or with an XSLT script. In other words, the service grounding bridges the gap between the abstract semantic-based service description and the detailed web service implementation.

In [146], the authors attempt to give semantics to web services using the OWL-S approach. They adopt OWL-S ontology classes to establish the capability representation of the web service. Generally speaking, representing capabilities has two methods. The first approach, which is named explicit capability representations [145], is to extend the current ontology of functions where each class in the service description ontology corresponds to a group of homogeneous functionalities. The second approach, named implicit capability representation, is to concentrate on the function description based on the state transformation the service produces. Both approaches have advantages and disadvantages, and should be used according to the analysis the application requirement.

After describing web services with OWL-S, [146] discuss the possibility of implementing automatic service enactment, discovery and composition. The service enactment is the process in which a client program submits a request to the service provider in terms of an explicit service description and interprets the response. The OWL-S based service description appends a semantic layer between users and WSDL description, which makes it possible to combine the service request with the service process model. The operations and the semantic types of the input/output data are mapped to the WSDL description which is dealt with by the service grounding. As a result, understanding and interpreting the web service takes advantage of the OWL logic and OWL-S ontologies, and the actual service invocation is still based on the service WSDL description.

The prerequisite of the service discovery is that web services advertise their capabilities in a service registry. The service client will submit the query request to the registry for web services with particular capability requirements. Based on the semantic-based web service description, automatic service discovery can also be implemented.

The service registry plays an important role in service discovery. It is responsible for storing the advertisements of web services and finding a match between query requests and service advertisements. As we have discussed in the previous section, UDDI only

supports keyword-based search and does not meet the requirement of the capability-based service discovery. To solve this problem, [146] summarize the idea of mapping the OWL-S service profile into the UDDI web service representation. They believe OWL-S and UDDI can complement each other. UDDI is an industry standard providing a World Wide distributed service registry, while OWL-S defines the ontology structure for describing the service information required by service capability matching. The OWL-S/UDDI matchmaker is implemented by integrating OWL-S capability matching in the UDDI registry, which can be found in several projects concentrating on the capability-based web service discovery [142] [147] [148].

Generally speaking, Grid services are stateful Web Services, and most of the current service-oriented Grid middleware techniques are based on the Web Service standards. The semantic approach in the web service discovery thus provides a potential direction for implementing a semantic-enhanced Grid service discovery mechanism. However, these current solutions have their limitations (e.g. no ranking in the service matching, taking every individual requirements of a service request equally) and other issues are required to be considered when building a semantic framework for the service information centre middleware in the mobile Grid environment. In particular, most of the semantic approach for web service discovery performs service matching with service function attributes (e.g. inputs, outputs, preconditions, effects), designed for service discovery in the enterprise environment. Our system architecture, on the other hand, combines both Grid and mobile computing. Other service characteristics as well as service capabilities must be considered for the service discovery, for example, service types, service resources and service context information.

5.3 A Methodology for Semantic Service Discovery

A semantic knowledge management approach is adopted to build the service discovery mechanism. Figure 5.3 shows an integrated process of the semantic approach. Two key issues are required to be addressed: a metadata model for describing services and structuring related domain concepts, and a service matching engine for processing service knowledge. Ontologies are usually expressed in a logic-based language, so that detailed accurate, consistent, sound and meaningful distinctions can be made among the classes, properties, and relations. Automated reasoning, based on ontologies, can be achieved by supporting tools, which provide advanced functions to intelligent applications such as semantic search and retrieval [149]. When using Semantic Web technologies to enhance service discovery, ontology is adopted to represent both service advertisements and service requests with abstract and high-level concepts, and the logic reasoning mechanism is utilized to build the service matching engine. As long as users describe their service requirements with terms from the same ontology model used to build the service descriptions, logical reasoning mechanisms can find the semantic similarity between the service

descriptions and the user requirements, enabling the matching services to be discovered and returned to users.

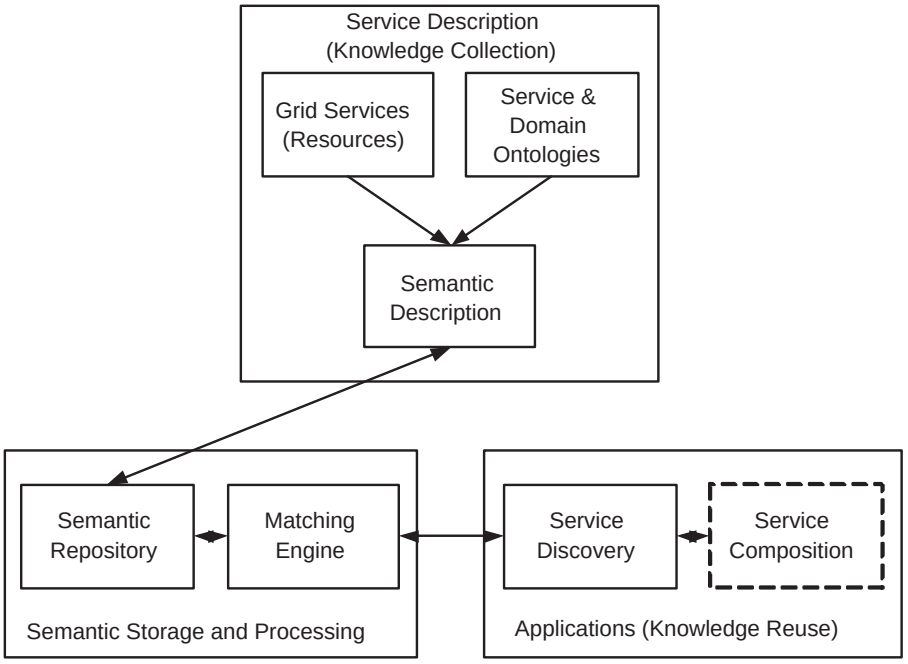

FIGURE 5.3: Semantic Knowledge Management Approach for Service Discovery

The service provider represents all characteristics of a service in the service description. Here, we define a term, *description collection*, which indicates all possible attributes to be described for a service. The attributes may be either a concept or a restriction for existent concepts. For each kind of individual service attribute, an ontology is usually designed to illustrate the attribute definition and its relationship with other service attributes. The service description can be completely separated from the detailed service implementation. After service providers deploy their services on the Grid computing platform, they need to advertise the service description information in a service registry. Different service descriptions can be made and published for one service, which enables a service to be reused for several purposes.

Similar to the service description, a service request often consists of a number of individual requirements, specifying the service attributes to be expected in a service. These requirements may include service outputs, inputs, function, location or any other possible attributes in terms of different service requests. For a specific service request, all of the requirements may be divided into two categories, a group of strict requirements and a set of general requirements. The strict requirement indicates that this kind of requirement is essential for the service request and has to be met precisely in the service matching, while the general requirement means this kind of requirement is not as important as the strict one and only a rough matching is necessary between the user requirement and the related service attribute.

The OWL language is a Semantic Web language used by computer applications that need to process the content of information instead of simply presenting information to humans. It is used to describe concepts and their relations in our service discovery mechanism because there are a number of tools which support editing, parsing, and reasoning. Furthermore, its extension, OWL-S, defines an ontology language for describing services, which provides a standard structure that can be used together with other aspects of the OWL description language to create service descriptions. The following is a partial OWL-S code of an example service description and a service request.

```
<service:presents>
  <profile:Profile rdf:ID=‘‘OnlineShopping’’>
    <profile:Name rdf:datatype=‘‘string’’>Shopping</profile:Name>
    <profile:hasInput rdf:resource=‘‘#ShoppingItems’’/>
    <profile:hasInput rdf:resource=‘‘#PersonalInfo’’/>
    <profile:hasOutput rdf:resource=‘‘#ReservationID’’/>
    <profile:hasType rdf:resource=‘‘#Selling’’/>
    <profile:hasResource rdf:resource=‘‘#ShoppingCart’’/>
    < ......More properties......  >
  </profile:Profile>
</service:presents>

<request:presents>
  <request:hasInput rdf:resource=‘‘#FoodItems’’/>
  <request:hasInput rdf:resource=‘‘#CreditCard’’//>
  <request:hasOutput rdf:resource=‘‘#Confirmation’’/>
  <request:hasType rdf:resource=‘‘#FoodSelling’’/>
  <request:hasResource rdf:resource=‘‘#VirtualShoppingCart’’/>
  < ......More properties......  >
</request:presents>
```

Although we assume that the service request attempts to describe expected requirements with terms from the same ontology model used to build the service description, it is impractical that every service request can acquire the exact desired service even though the required services have already been deployed and advertised because one service could have a number of description formats so that there may be the deviation in the process of the service matching. In fact, the responsibility of the service matching engine is to obtain all of the related services including those that differ from the request to some defined extend. These deviation matches should not be rejected but be classified using a predefined rule (e.g. matching degree), enabling service to be selected based on the information returned from the service discovery middleware. Our service information centre middleware takes a service request and available service description collections as inputs, and outputs a list of candidate services as well as their matching degrees.

The service matching engine is responsible for comparing the service request against each service description and judging whether a service should be put onto the list of candidate results. The assessment of the semantic similarity between concepts is a fundamental requirement for implementing the service matching engine. Most of the previous work adopt the subsumption reasoning to determine the semantic distance between concepts in the request and in the description. However, this is not sufficient for building an effective service matching engine. Consider the following example: a user tries to find a printing service in the meeting room while a printing service is deployed in the nearby office. When used subsumption reasoning only, the printing service deployed in the office is not be regarded as a candidate service returned for the user because in the ontology "Meeting Room" and "Office" are two disjoint concepts. However, the user may select the printing service in the office if there are no other services around the meeting room. This means that a more comprehensive approach to the semantic similarity judgment has to be adopted.

We use the method introduced in [150] to check the semantic similarity between two concepts. The attributes in a service description are categorized into three types. Type one includes conceptual attributes whose similarity can be judged using subsumption reasoning. Type two includes conceptual attributes whose similarity cannot be judged using the subsumption reasoning only. In this condition, the author assumes the knowledge of similarities between these concepts can be acquired by using available similarity measurement approaches such as [151] and [152]. Type three refers to numeric attributes only. The similarity between this type of concept can be judged either by using a percentage deviation from the requested value or a fuzzy membership function.

5.4 Attribute Definition for Grid Service Description

5.4.1 Service IOPEs

Inputs, outputs, preconditions and effects (IOPEs) are important functional attributes for both Web and Grid services. Inputs and preconditions define the constrains required for a service invocation, and outputs and effects indicate the results or the state transformation of a service execution. OWL-S, a standard Web Service description language, provides an all-side ontology structure definition for describing a service IOPE. As discussed in section 5.2.2 of this chapter, a number of semantic approaches have adopted a standard OWL-S ontology class structure to describe services and based the service discovery on the state transformation the service produces.

IOPEs are useful and effective characteristics for describing a service. However, in order to build a semantic service discovery mechanism for mobile clients discovering Grid services, other types of service attributes also have to be considered.

5.4.2 Service Resources

Service-oriented Grid computing architecture is an extension of current Web Service technologies. In the Grid computing architecture (Figure 5.4), applications are built on top of a set of common, standard and high-level services, which meet the definition of Open Grid Services Architecture (OGSA). One of important requirements of OGSA is that the underlying middleware should store information about the service state because Grid application users usually need this kind of information to be maintained from one invocation to another.

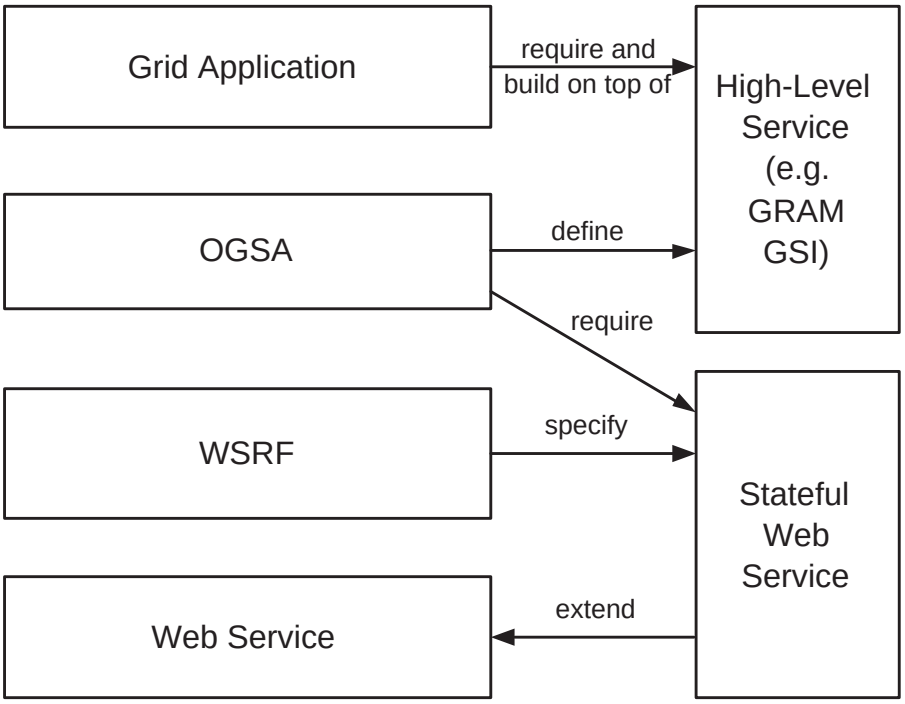

FIGURE 5.4: Grid Computing Architecture.

Web Service Resource Framework (WSRF) provides a mechanism of building the stateful services required by OGSA. It specifies a straightforward solution of recording the service state: keep the web service and its state information completely separate, and store all the state information in an entity named "resource". Each resource entity in a web service is assigned a unique key. When service clients want to invoke a service, they submit the request including both the URI of the service provider and the key of the required resource. A service may have several resource instances, which enables the state information to be kept for different purposes.

Service resources have different types and characteristics. They may be a integer value, keeping the value of mathematical operations; they may be a virtual shopping cart, recording the items chosen by shopper; they may represent a physical device (e.g. printer), logging its working status. WSRF specifications define several styles of interaction mechanisms, providing different ways of representing and accessing multiple

resource instances. We regard the service resource as an important functional attribute in the Grid service description because it is a key parameter for the Grid service invocation, .

5.4.3 Service Type

In a service-oriented mobile Grid environment, users execute their tasks by using a variety of Grid services through their mobile handheld devices. As discussed in Chapter 3, two main styles of application scenarios are identified from the user viewpoint: an information access scenario, and a work assistant scenario. In the information access scenario, the mobile device acts as a universal operating terminal to access various available Grid services, collecting required information knowledge for it user. A typical example is that a doctor is able to check the data being collected form patients in real time with his/her PDA by utilizing the medical Grid services deployed by the hospital.

In the work assistant scenario, users usually need to execute relatively complicated applications such as data-deluge programs to achieve specific tasks through their mobile devices. However, due to resource limitations, most complex programs cannot be executed on a handheld device. Users have to offload resource-demanding tasks to the Grid, and the Grid provides the underlying executing environment. A possible example is that a fire fighter may submit streams of temperature data about a multi-story building to the Grid. The Grid assists fire fighters to solve three-dimensional partial differential equations in order to obtain the detailed information such as the temperature of different floors of the building, or the temperature of a particular room. The work assistant scenario describes the scene in which mobile users achieve complicated tasks with the assistant of Grid services.

An ontology is defined on the basis of the analysis of two application scenarios. The ontology represents a hierarchy of possible application scenarios and contains a taxonomy of service types which are usable for mobile clients. Figure 5.5 shows a class diagram of the service type ontology. The top-level concept of the ontology is *Service*, which represents the most generic type. There are two direct subclasses of *Service*: the *InfoAccess* class represents the general service for the information access scenario; the *WorkAssistant* class represents the general service for the work assistant scenario. Every service published in the service registry has a type attribute, so that a requester may be able to express the type of service required directly rather than through the service IOPEs.

5.4.4 Service Context

The vision of integrating mobile devices into the Grid environment is to embed a variety of distributed resources into our everyday life seamlessly, enabling pervasive users to

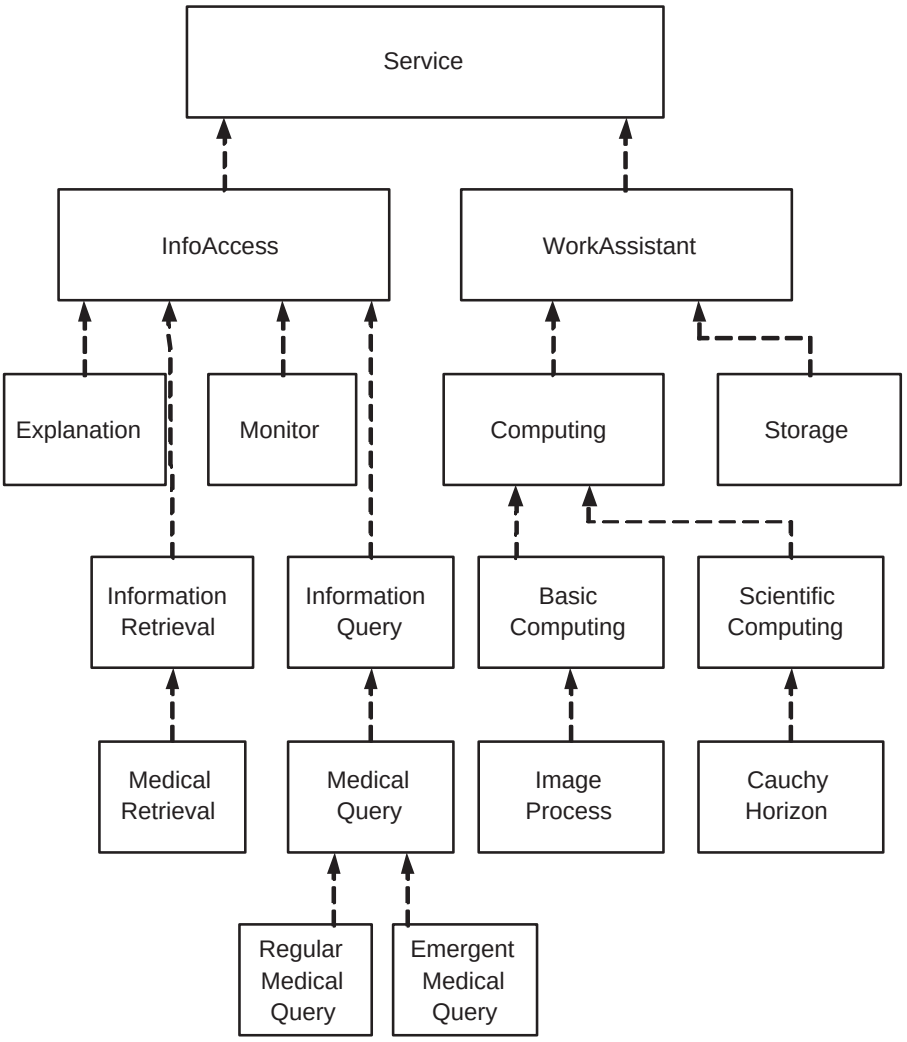

FIGURE 5.5: Service Type Ontology Diagram.

access Grid services anytime and anywhere. In such a dynamic mobile Grid environment, Grid services should be provided for suitable mobile users and mobile users should be able to find and invoke Grid services in the right way at the right time in the right place. To achieve this goal, both service consumers and service providers need to share their knowledge with each other. In the previous chapter, an ontology is designed to model the interaction between mobile users and Grid services. The top-level of the context ontology, together with their subclasses, forms a basic framework of the context information in the service-oriented mobile Grid environment.

The context information attributes of the service are required when describing a Grid service. At present, we consider two context attributes in the Grid service description: the first is the service location, which corresponds to the "Place" class of the interaction context model; the other is the service access range, based on which a service discovery restricting mechanism is implemented.

Mobile users access Grid services with their portable devices, which may expose their personal information. For example, if the service directory is so "open" that every mobile users can discover and obtain all deployed Grid services, a user location information may be exposed to other users as long as they can locate and try to invoke corresponding location-monitoring services. This means that protecting personal privacy is an important consideration when designing a service discovery mechanism. The user personal information decides their accessing level during the authorization process, which is shown and recorded in the "User" class. The service provider defines a service range attribute in the service description. When a new mobile user sends a request to search Grid services, the service matching engine will reason and determine whether a Grid service can be exposed to the user by comparing the service access range in the service description with the accessing level of this mobile user.

5.4.5 Service Details

As well as the service IOPEs, service resources, service types, and service context, there are many other particular service attributes which are important when considering the service discovery. For example, for a "Printer" service, its "PaperSize" attribute specifies the supporting paper size. When a user intends to print a large-size image (e.g. A1 size), the "PaperSize" has to be considered to be a high priority matching requirement, because if it is ignored, the user may locate a number of useless services which do not support large-size printing. These important attributes are classified into the "Service Detail" collection, and its definition totally depends on the detailed implementation of the individual service.

5.4.6 Service Description with Extended OWL-S

OWL-S is a language for describing services, which provides a standard vocabulary that can be used together with other aspects of the OWL description language to create service descriptions. The structure of the OWL-S upper-level ontology is based on the types of knowledge of service description: the "Service Profile" provides the high-level descriptive information of a service, such as the name, input/output of the service, and additional text description; the "Service Model" and "Service Grounding", provide sufficient information of how the service is utilized and how to interact with the service.

The "Service Profile" class provides a superclass of every type of high-level description of the service, and it is adequate for the service discovery to use the "Service Profile" class to describe a service. When deciding to use the OWL-S ontology structure to describe Grid services, it is obvious to notice that the "Service Profile" does not specify all Grid service attributes required in the mobile Grid computing environment. It must therefore be extended by adding new service characteristics discussed above.

It is reasonable to modify the original OWL-S classes or properties and define new concepts in the top level of the service ontology so that it can be more suitable for individual applications. In fact, the OWL-S group of W3C encourages other researchers to construct an alternative approach because the intent of the definition of OWL-S ontology structure is not to restrict an only approach to describing a service, but to provide a basic example for other potential application cases.

Figure 5.6 illustrates the extended service profile class, adding new six classes and related properties.

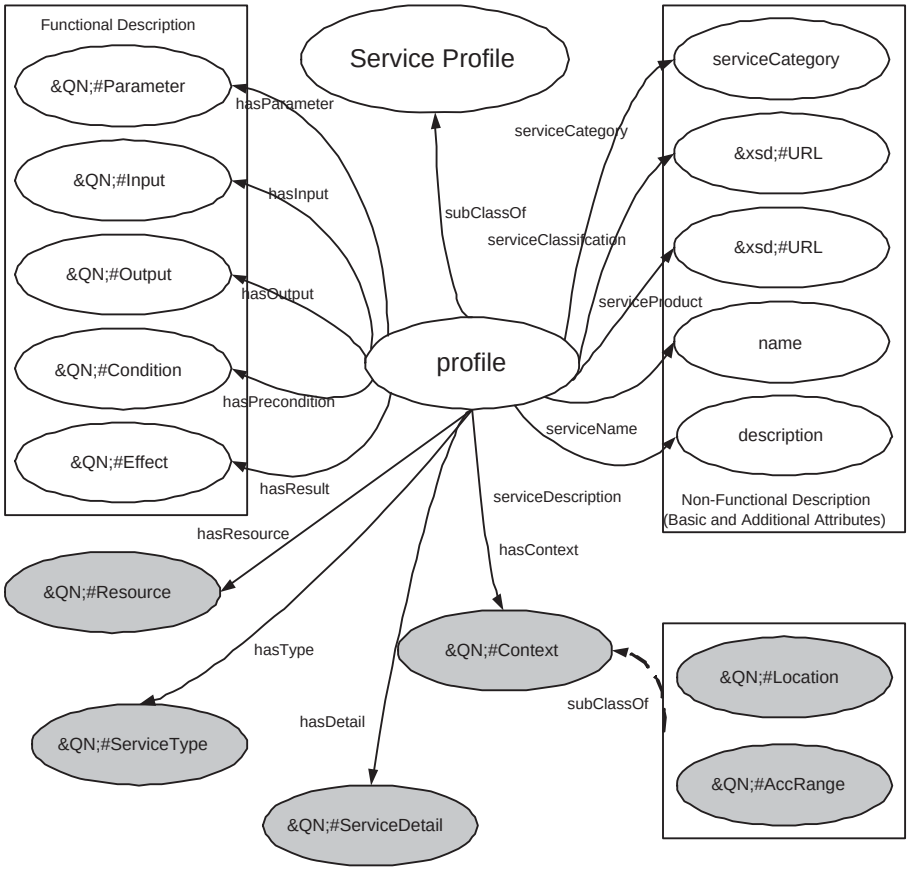

FIGURE 5.6: Extended Service Profile.

- Service Basic Information: These properties and related values provide human-readable information, including serviceName, textDescription, contactInformation and so on.

- Functionality Description: These are important attributes in the service profile, including hasParameter, hasInput, hasOutput, hasPrecondition, and hasResult properties. Inputs and outputs represent the information transformation of a service, while preconditions and effects indicate the state change of a service. The IOPEs published by the "Service Profile" is a subset of those published by the "Process Model".

- Additional Attributes: These properties and related values introduce the additional attributes for the service description, including serviceParameter and serviceCategory. The values of these are the instance of the class ServiceParameter and ServiceCategory.
- Extended Attributes: These are extended attributes for the service description, including serviceResource, serviceType, serviceDetail, and serviceContext. The value of these are instances defined in the corresponding OWL ontologies, which are different to those of the profile attributes.

Although a comprehensive OWL-S extension includes the extension for all types of knowledge of a service, detailed discussion of the "Service Model" and "Service Grounding" extension is beyond the scope of this thesis because the aim of our research is to design a service discovery mechanism, for which using the extended "Service Profile" to describe services is perfectly adequate. The work of describing a service with a combination of OWL and WSDL can be found in [153].

Grid services are described based on the extended service profile. For a Grid service, its description collection includes the functional attributes (e.g. inputs, outputs, preconditions, effects), the service type (an instance or subclass of *InfoAccess* class or *WorkAssistant* class), the service resource, the service detail, and the service context information (the location of the service provider and the service access range).

5.5 The Service Matching Algorithm

A service request is composed of a number of individual requirements, specifying various attributes to be expected in services. The service matching engine takes a service request and a group of service description collections as inputs, and is responsible for determining whether a Grid service is a matching service for this service request. The comparison between the service request and the service description collection consists of two steps. Initially, the service matching engine will check to judge whether each strict requirement can be matched precisely in the service description. If a service description does not contain the expected attributes, it will be dismissed and the service matching engine will compare the next service description to the service request. If a Grid service satisfies all of the strict requirements, the matching engine will then turn to estimate the general requirements.

As discussed in the Methodology section, the attributes are categorized into three types in order to check the semantic similarity. Type one and type two refer to conceptual attributes. Their similarity can be checked by using subsumption reasoning based on the taxonomic relation or other semantic similarity measurements. Four expected matching level for general requirements are defined:

- "Substitute" indicate that the user expects to find a concept in the service description which is equal to or is the direct superclass of the concept in the requirement.
- "Cover" indicates that a concept which subsumes the concept in the service request is expected to be found.
- "Fuzzy" means this requirement is of little importance for service matching. As long as a concept in the service description can be found which has the subsumption relationship (either superclass or subclass) with the concept in the requirement, it will be satisfied.
- "Close" indicates that the user expects to find a concept in the service description which has the same direct superclass in the defined concept (ontology) structure with the concept in the service requirement. For example, in the ontology structure shown in Figure 5.5, "Regular Medical Query" and "Emergent Medical Query" have a "Close" relation. This expected matching level is defined for the type 2 conceptual attribute, whose similarity cannot be assessed with the subsumption reasoning approach.

These expected matching levels can be set when the service request is submitted to the service matching engine. The service matching engine will check the similarity between each general requirement in the service request and the related service attribute in the service description. The actual matching level is determined by the semantic relationship in the predefined ontology structure. If all of the expected matching levels are satisfied, this service will be a reasonable candidate matching service for the service request.

The service matching engine may find a number of candidate services for a specific service request. Although the service discovery mechanism is not responsible for the service selection, matching degree information about each candidate service is required to be provided as a result for the service request. We use the term "MatchingScore" to show the matching degree of the candidate service. For a candidate service, its MatchingScore is calculated using the following equation:

$$MatchingScore = \sum_{i=1}^{n} Score_i/n$$

The "$Score_i$" indicates the matching degree of every individual general requirement in the service request against the related service attribute in the service description, which is obtained based on the concept types categorized for checking the semantic similarity. For type one, because the subsumption relation exists between these concepts, the score can be obtained based on the semantic distance $||C_r, C_a||$ between the individual requirement (C_r) and the related service attributes (C_a) in the ontology structure. The following

equations are used to calculate the individual score:

$$Score_i = \begin{cases} 1 & if\ C_a\ =\ C_r \\ \frac{1}{2} + \frac{1}{2*(||C_r, C_a||+1)} & if\ C_a\ is\ a\ superclass\ of\ C_r \\ \frac{1}{2*(||C_r, C_a||+1)} & if\ C_r\ is\ a\ superclass\ of\ C_a \end{cases}$$

For type two, the knowledge of similarities between concepts is assumed to be available, and the service matching engine will take the decision according to all of close degrees between user requirements and related attributes. The score is assumed to be acquired from an available similarity measurement approach (e.g. [151], [152]). For type three, both the attributes and the requirements are numeric. Their similarity score can be obtained using the percentage deviation from the requested value or a fuzzy membership function, depending on the detailed service implementation and the user requirement [154].

The matching score of each candidate service is calculated based on $Score_i$, and it will determine the ranking of candidate services. The higher the score is, the higher ranking the candidate service has.

5.6 Implementation

5.6.1 The Implementation of Service Description

The ontology classes for the service description are defined with the OWL language using the Protege toolkit [140]. Protege is an open-source technology editor and knowledge-based framework, which can export ontologies into a number of formats including OWL. Figure 5.7 shows the created service type ontology. In order to test the functionality of the service discovery mechanism, a set of ontologies (e.g. "eService" and a "eWine") are edited and maintained in Protege to define concepts and relationships for describing various service attributes (e.g. IOPEs, service resources).

The service description is built with the OWL-S Editor software, a tool for creating OWL-S services, which is developed as a plug-in for Protege environment. As introduced in the above section, every OWL-S service, which is the instance of the OWL-S class "Service", has at most one instance of the "Process Model" class through the property "describedBy", variable "Service Profile" and "Service Grounding" instances through

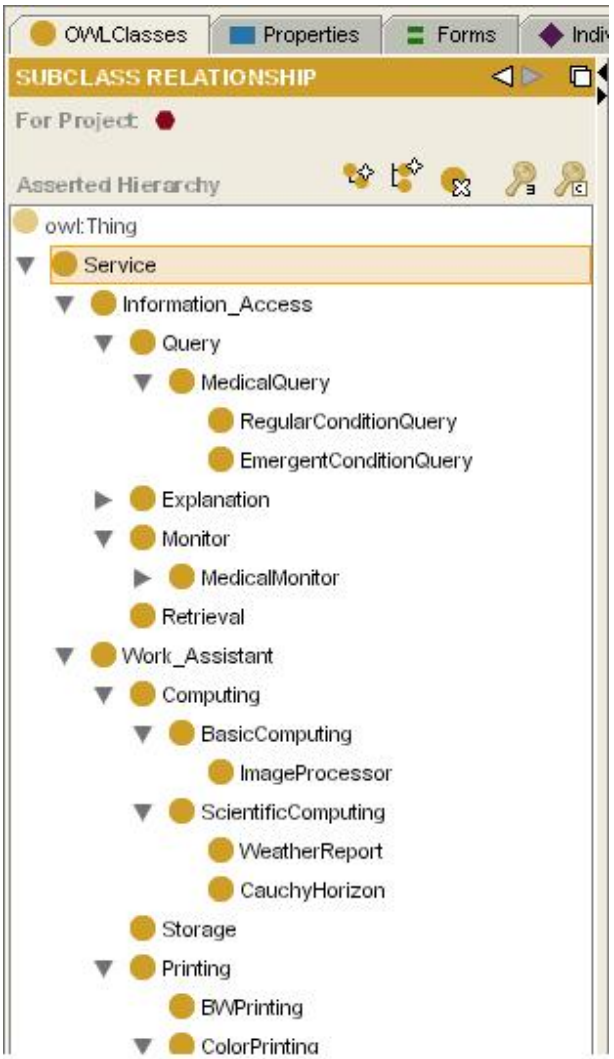

FIGURE 5.7: A Screenshot of the Service Type Ontology developed with Protege.

the "presentBy" and "supportBy" properties. There is no order restriction of how to build these four classes.

Taking the "Online Shopping" service as an example, building a comprehensive service description requires five steps:

- Creating atomic processes: The "Process Model" class defines two kinds of interaction modes. A complex service corresponds to a composite process, which is composed by a list of atomic processes. The "Online Shopping" service provides three sub-functionalities: Search Items, Select an Item, and Place the Order (Figure 5.8). Three sub-functionalities are modeled as three atomic processes, and the "Online Shopping" service will be described as a composite process. Before creating atomic processes, the type of inputs, outputs, resources and other used attributes should be defined in related ontologies.

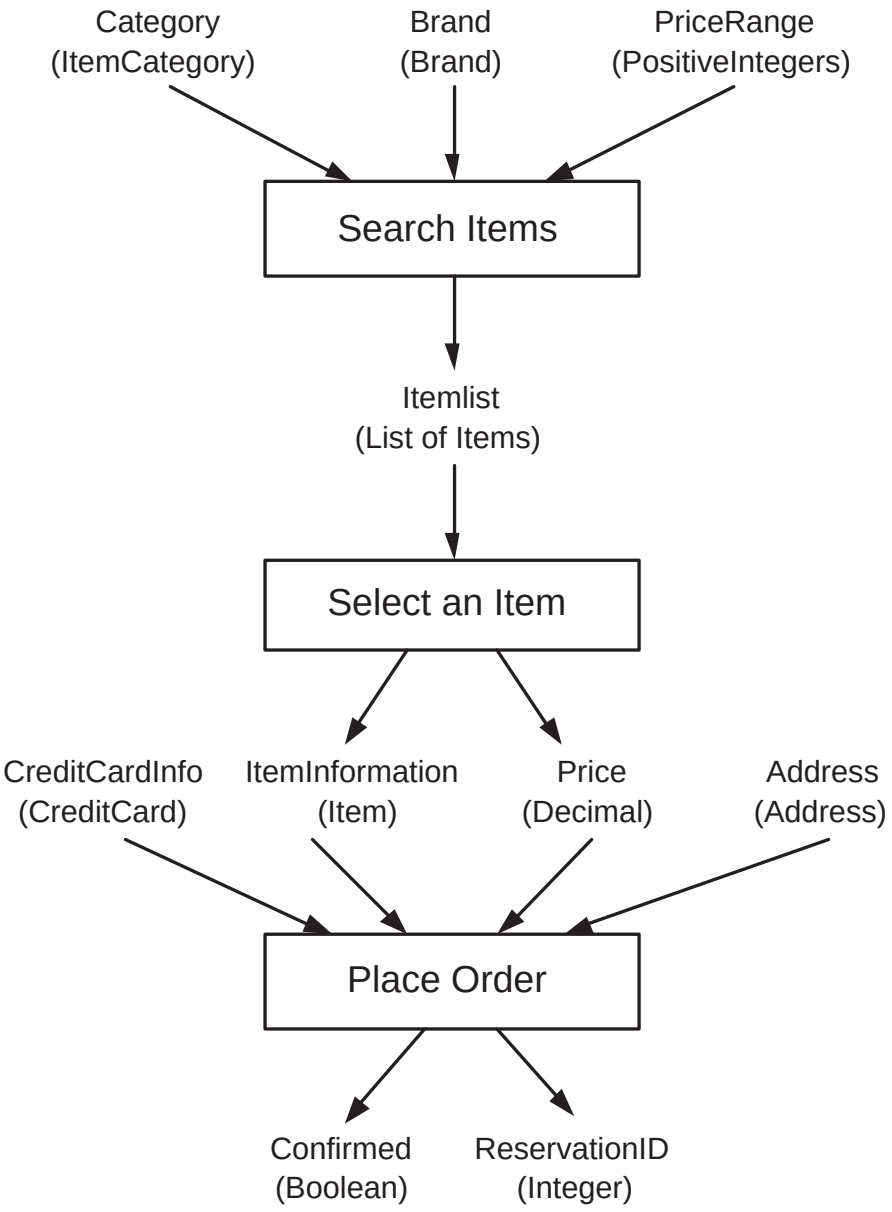

FIGURE 5.8: Three Atomic Processes of the OnlineShopping Service.

- Creating composite process: Once atomic process instances are created, they are required to be combined into a composite process which describes the "Online Shopping" service. The OWL-S editor provides a powerful visual tool as well as a number of control flow interfaces to simplify the creation of a composite process. The dataflow can be defined to connect the inputs parameters of an atomic process to output parameters of another atomic process.

- Creating grounding instance: For each atomic process, an instance of WsdlAtomicProcessGrounding is created to implement the map between abstract service processes and detailed WSDL operations. The creation of a WsdlAtomicProcessGrounding includes naming the instance, defining properties, mapping inputs and outputs

to message parts of a WSDL implementation, and preparing new references of WsdlOperation and wsdlPorttype for atomic process instances.

- Creating profile instance: Because the "Process Model" has already been created, required parameters can be selected conveniently according to properties of the service process. The parameters in the service profile are a subset of those in the service process model because it is up to service providers to select and expose functionalities of the service in an service advertisement.

- Creating service instance: A service instance usually has three properties: the value of *service:presentBy* is the instance of the service profile; the value of *service:describedBy* is the instance of the service process; and the instance of the service grounding is the value of *service:supportedBy*.

5.6.2 The Implementation of Service Matching Engine

The service matching engine takes the service request and a group of service description collections as inputs, and output a list of candidate matching services as well as their matching degrees. However, we have to consider two practical problems for the detailed implementation of the service matching engine:

- Users who need to locate required services may not know where the service description collections are, and they only submit their service requests to the service matching engine in most cases.

- It is inefficient to process all of service description collections for every service request. Especially, as the number of the service advertisements increases highly, the time of processing a service query will increase dramatically.

In order to solve these problems and avoid the bottleneck of the system performance, the service publishing component was integrated into the service matching engine. The domain concepts (service attributes) in a service description are extracted, and related ontology instances are created and stored in the ontology repository. When a service request is received, the service matching engine only needs to parse a request description, and check similarity between concepts in the service request and instances in the ontology repository. The service matching engine then collects the service information based on the matching concepts (expected service attributes). This pre-reasoning approach speeds up the time of a service query request because it saves the time of processing a number of service advertisements.

The Grid service information centre middleware (service matching engine) has been implemented in Java with the MySQL database, the Jena framework, the Racer system [155], the jUDDI toolkit and other related techniques. Jena provides a programming

environment for OWL ontologies which is used to parse OWL-S service descriptions and manage required ontologies. The Racer system is responsible for execute the necessary reasoning tasks during the service matching process. Grid service advertisement information is stored in the MySQL database based on the data structure which is defined in the JUDDI toolkit. The service information centre middleware is implemented as both a Java Web Service for use by other middleware in the system architecture and a web application using the AJAX design mode which can be accessed through a standard web interface.

Figure 5.9 shows the internal modules of the Grid service information centre.

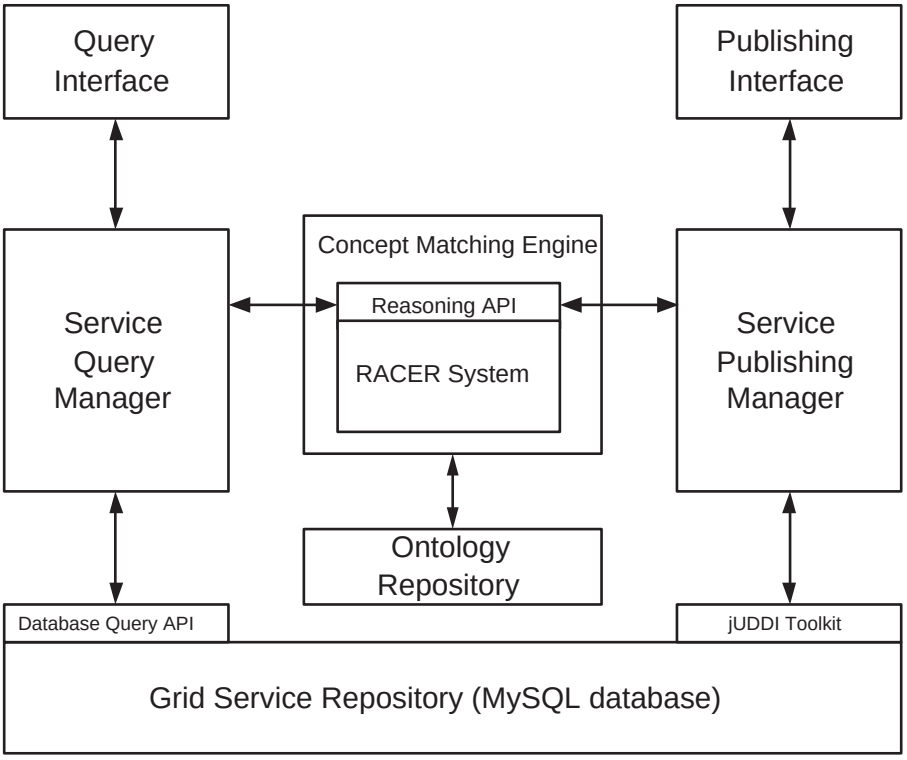

FIGURE 5.9: Internal Modules of Service Information Centre Middleware.

- Service Query Manager: The service query manager component is responsible for the communication with service requests through the service query port. It accepts the service query request and processes the request by interacting with both the concept matching engine and the Grid service repository. Matching services as well as their matching degrees will be returned to users by the service query manager through the service query port.

- Service Publishing Manger: The service publishing manager component is responsible for the communication with service providers through the service publishing port. It accepts the service advertisement requests, classifies the service description information by collaborating with the concept matching engine, and puts the services as well as their metadata into the Grid service repository.

- Concept Matching Engine: The concept matching engine is implemented by Racer system and its reasoning APIs. During the process of a service publication, the concept matching engine provides an interface for classifying the semantic service description information based on the concept hierarchy in defined ontologies and creating corresponding instances adopted by the service description. During the process of a service query, the concept matching engine takes a concepts from the service query request as the input and returns the similar concepts stored in the ontology repository based on the reasoning requirement (e.g. superclass concepts, subclass concepts, disjoint-class concepts).

- Grid Service Repository: The service repository component is a database which stores advertised Grid services as well as their metadata. Various service information (e.g. service URI, service method description) can be looked up with service attributes (e.g. service type, service resources) through database query interfaces.

- Ontology Repository: A database or a file which stores a variety of domain concepts used by the service description and the service request, including ontology classes, created instances and their relationships.

5.7 Testing Scenarios

The application of the service discovery mechanism is demonstrated using the following testing scenarios.

Scenario one assumes an application where a user wants to locate a service which can provide the weather information. There are two related services published on the service registry. Both the service request and the service description collections are expressed in the format of OWL-S structure together with required OWL ontologies.

```
<request:presents>
  <request:hasOutput rdf:resource=‘‘#WeatherReport’’/>
  <request:hasType rdf:resource=‘‘#InfoQuery’’/>
  < ......More properties......  >
</request:presents>

<service:presents>
  <profile:Profile rdf:ID=‘‘Service1’’>
    <profile:hasInput rdf:resource=‘‘#Date’’/>
    <profile:hasOutput rdf:resource=‘‘#WeatherReport’’/>
    <profile:hasType rdf:resource=‘‘#InfoAccess’’/>
    < ......More properties......  >
  </profile:Profile>
</service:presents>
```

```
<service:presents>
  <profile:Profile rdf:ID=‘‘Service2’’>
    <profile:hasInput rdf:resource=‘‘#DataStream’’/>
    <profile:hasOutput rdf:resource=‘‘#WeatherReport’’/>
    <profile:hasType rdf:resource=‘‘#ScientificComputing’’/>
    < ......More properties......  >
  </profile:Profile>
</service:presents>
```

The attributes that the service request is concerned about are the service output and the service type. Although two services have the same output, they provide different functions. In fact, service one is a typical information query service, taking the date as input and returning the weather condition of that day. Service two is a scientific computing service, taking a stream of data as input and generating a weather report after the complex computation. Based on the comparison between requirements and related service attributes, it is obvious that the service matching engine will discard the second service because “ScientificComputing” is a subclass of “WorkAssistant, which has no relationship with “InfoQuery” in the ontology structure shown in Figure 5.5. The first service is a candidate service for the request because the “InfoAccess” is a superclass of the “InfoQuery”.

This testing scenario also shows that considering the IOPEs is not adequate for the service discovery. If there is no “ServiceType” restriction, both service one and service two will be regarded as the candidate result because they provide the same service output. However, service one and service two are two different types of service, which can be distinguished by considering another service attribute.

Scenario two assumes an application in which a service about the emergent medical query is required. Four related services are published on the service registry. Both of them are described in the following:

```
<request:presents>
  <request:hasType rdf:resource=‘‘#EmergencyMedicalQuery’’/>
</request:presents>

<service:presents>
  <profile:Profile rdf:ID=‘‘Service1’’>
    <profile:hasPrecondition rdf:resource=‘‘#Symptom’’/>
    <profile:hasType rdf:resource=‘‘#MedicalQuery’’/>
  </profile:Profile>
</service:presents>

<service:presents>
  <profile:Profile rdf:ID=‘‘Service2’’>
```

```
      <profile:hasInput rdf:resource=‘‘#Information’’/>
      <profile:hasType rdf:resource=‘‘#MedicalRetrieval’’/>
    </profile:Profile>
  </service:presents>

  <service:presents>
    <profile:Profile rdf:ID=‘‘Service3’’>
      <profile:hasType rdf:resource=‘‘#InfoQuery’’/>
    </profile:Profile>
  </service:presents>

  <service:presents>
    <profile:Profile rdf:ID=‘‘Service4’’>
      <profile:hasType rdf:resource=‘‘#RegularMedicalQuery’’/>
    </profile:Profile>
  </service:presents>
```

The user also sets the expected matching level of the service type “Cover” with the request description, which indicates the corresponding concept in the service description should subsume the concept in the service request. Based on the similarity checking between individual requirements and available service descriptions, the service matching engine discards service two and service four because the “EmergencyMedicalQuery” has no subsumption relationship with both the “MedicalRetrieval” and the “RegularMedicalQuery” in the ontology definition (Figure 5.5). “RegularMedicalQuery” and “EmergencyMedicalQuery” have the same direct superclass “MedicalQuery”. If the expected matching level is set to “Close”, the service three will be regarded as a candidate for this service request.

Service one and service three are both candidate services after the concept comparison. Hence, their matching scores are required for the service request. According to the equations discussed above, because the semantic distance between “EmergencyMedicalQuery” and “MedicalQuery” is one, the matching score for service one is

$$0.5 + 1/2 * (1 + 1) = 0.75;$$

because the semantic distance between “EmergencyMedicalQuery” and “InfoQuery” is two, the matching score for service three is

$$0.5 + 1/2 * (1 + 2) = 0.667.$$

Service one, service three and their matching score are returned for the service request.

Scenario three assumes an application in which a user wants to submit a task of image processing through the mobile device. The service request and published services are expressed in the following:

```
<request:presents>
  <request:hasResource rdf:resource=‘‘#ImageProcessor’’/>
  <request:hasContext rdf:resource=‘‘#reqContext’’/>
</request:presents>

<request:Resource rdf:ID=‘‘ImageProcessor’’>
  <request:imgFormat rdf:resource=‘‘#JPG’’/>
  <request:currentState rdf:resource=‘‘#Idle’’/>
  <request:maxOutputSize rdf:resource=‘‘#A1’’/>
  <request:comRate rdf:datatype=‘‘&xsd;Integer’’>50</comRate>
</request:Resource>

<request:Context rdf:ID=‘‘reqContext’’>
  <request:location rdf:resource=‘‘#MeetingRoom’’/>
</request:Resource>

<service:presents>
  <profile:Profile rdf:ID=‘‘Service1’’>
    <profile:hasResource rdf:resource=‘‘#ImageProcessor1’’/>
    <profile:hasContext rdf:resource=‘‘#Context1’’/>
  </profile:Profile>
</service:presents>

<ser1:Resource rdf:ID=‘‘ImageProcessor1’’>
  <ser1:imgFormat rdf:resource=‘‘#JPG’’/>
  <ser1:currentState rdf:resource=‘‘#Idle’’/>
  <ser1:maxOutputSize rdf:resource=‘‘#A1’’/>
  <ser1:comRate rdf:datatype=‘‘&xsd;Integer’’>40</ser:comRate>
</ser1:Resource>

<ser1:Context rdf:ID=‘‘Context1’’>
  <ser1:location rdf:resource=‘‘#MeetingRoom’’/>
  <ser1:accessRange rdf:resource=‘‘#High’’/>
</ser1:Context>

<service:presents>
  <profile:Profile rdf:ID=‘‘Service2’’>
    <profile:hasResource rdf:resource=‘‘#ImageProcessor2’’/>
    <profile:hasContext rdf:resource=‘‘#Context2’’/>
  </profile:Profile>
</service:presents>

<ser2:Resource rdf:ID=‘‘ImageProcessor2’’>
  <ser2:imgFormat rdf:resource=‘‘#BMP’’/>
  <ser2:currentState rdf:resource=‘‘#Idle’’/>
```

```
  <ser2:maxOutputSize rdf:resource=‘‘#A2’’/>
  <ser2:comRate rdf:datatype=‘‘&xsd;Integer’’>70</ser:comRate>
</ser2:Resource>

<ser2:Context rdf:ID=‘‘Context2’’>
  <ser2:location rdf:resource=‘‘#Office’’/>
  <ser2:accessRange rdf:resource=‘‘#Normal’’/>
</ser2:Context>

<service:presents>
  <profile:Profile rdf:ID=‘‘Service3’’>
    <profile:hasResource rdf:resource=‘‘#ImageProcessor3’’/>
    <profile:hasContext rdf:resource=‘‘#Context3’’/>
  </profile:Profile>
</service:presents>

<ser3:Resource rdf:ID=‘‘ImageProcessor3’’>
  <ser3:imgFormat rdf:resource=‘‘#JPG’’/>
  <ser3:currentState rdf:resource=‘‘#Idle’’/>
  <ser3:maxOutputSize rdf:resource=‘‘#A1’’/>
  <ser3:comRate rdf:datatype=‘‘&xsd;Integer’’>40</ser:comRate>
</ser3:Resource>

<ser3:Context rdf:ID=‘‘Context3’’>
  <ser3:location rdf:resource=‘‘#DemoRoom’’/>
  <ser3:accessRange rdf:resource=‘‘#Normal’’/>
</ser3:Context>

<service:presents>
  <profile:Profile rdf:ID=‘‘Service4’’>
    <profile:hasResource rdf:resource=‘‘#ImageProcessor4’’/>
    <profile:hasContext rdf:resource=‘‘#Context4’’/>
  </profile:Profile>
</service:presents>

<ser4:Resource rdf:ID=‘‘ImageProcessor4’’>
  <ser4:imgFormat rdf:resource=‘‘#JPG’’/>
  <ser4:currentState rdf:resource=‘‘#Busy’’/>
  <ser4:maxOutputSize rdf:resource=‘‘#A1’’/>
  <ser4:comRate rdf:datatype=‘‘&xsd;Integer’’>50</ser:comRate>
</ser4:Resource>

<ser4:Context rdf:ID=‘‘Context4’’>
  <ser4:location rdf:resource=‘‘#MeetingRoom’’/>
```

```
    <ser4:accessRange rdf:resource=‘‘#Normal’’/>
  </ser4:Context>

  <service:presents>
    <profile:Profile rdf:ID=‘‘Service5’’>
      <profile:hasResource rdf:resource=‘‘#ImageProcessor5’’/>
      <profile:hasContext rdf:resource=‘‘#Context5’’/>
    </profile:Profile>
  </service:presents>

  <ser5:Resource rdf:ID=‘‘ImageProcessor5’’>
    <ser5:imgFormat rdf:resource=‘‘#JPG’’/>
    <ser5:currentState rdf:resource=‘‘#Idle’’/>
    <ser5:maxOutputSize rdf:resource=‘‘#A3’’/>
    <ser5:comRate rdf:datatype=‘‘&xsd;Integer’’>60</ser:comRate>
  </ser5:Resource>

  <ser5:Context rdf:ID=‘‘Context5’’>
    <ser5:location rdf:resource=‘‘#MeetingRoom’’/>
    <ser5:accessRange rdf:resource=‘‘#Normal’’/>
  </ser5:Context>
```

In this example, it is assumed that the user expects that the image can be processed immediately. Hence, the printer current state is a strict requirement and should be matched precisely. Service four is discarded at first because it cannot satisfy the strict requirement. The expected matching level for general requirements is set to “Close”. Through comparison, service one, two, three and five are all candidate services because “JPG” and “BMP” are two types of image format, “A1” “A2” and “A3” are three kinds of paper size, and “MeetingRoom” “Office” and “DemoRoom” are the locations in the same building (according to the related ontology definition). Their matching degrees must be acquired for the service request.

In the service description, the service attributes are type two or type three concepts and there is no subsumption relation between them. The “Format” attribute is a type two concept and we assume that the similarity value between “JPG” and “BMP” is 0.7. The “maxOutputSize” attribute is a type two concept and we assume that the similarity values between “A1”, “A2” and “A3” are 0.6 and 0.25. The “location” attribute is a type two concept and we assume that the similarity values between “MeetingRoom”, “Office” and “DemoRoom” are 0.5 and 0.5. The “compressRate” is a type three concept and we use the percentage deviation from the requested value to calculate the similarity value.

Based on the above assumptions, the matching score of service one, two, three and five can be calculated using the equation discussed in the above section. The score of the

service one is 0.95, the score of the service two is 0.6, the score of the service three is 0.825, and the score of the service five is 0.7625. However, before returning the candidate services and their matching score, the service matching engine will check the access range of each service. Here we assume the user only has a normal access range. Because the access range of the service one is "High", it cannot be exposed for a normal user. Hence, although the service one has the highest matching score, unfortunately, it is not allowed to be discovered by the user. As a result, the service two, three and five will be returned for the service request as well as their matching score.

5.8 Performance Evaluation

The semantic service matching middleware must have a reasonable service query time in order to be used practically. An integrated practical service query process can be divided into two procedures:

1. Analyzing the service request and obtaining expected service attributes by comparing every individual requirement with concepts in the ontology repository.
2. Based on the expected service attributes, collecting the candidate services from the service repository.

We believe four key parameters affect the response time when processing a service request:

- the number of individual requirements (n_{ir}) in a service request
- the size of ontology repository (n_{or}): indicated by the quantity of defined classes
- the number of matching services (n_{ms}) for a service request
- the size of service repository (n_{sr}): indicated by the quantity of advertised services in a repository

5.8.1 UDDI vs. Semantic Matching Middleware

In the first experiment, we compare our semantic service matching middleware with UDDI, the traditional web service registry. We use the system response time as the performance index and focus on calculating the time required to process a query. The time of publishing a Grid service is not considered because in the system architecture, mobile users are usually Grid service consumers rather than service providers.

The purpose of this experiment is to obtain the measured time of querying a Grid service. Both the advertisement information of real Grid services and a large number of

pseudo services are published in the service repository. Altogether, fifty services can be accessed by the semantic service discovery middleware (n_{sr}=50). We also set the number of individual requirement (n_{ir}), the number of matching service (n_{ms}), and the size of ontology repository (n_{or}) to be one, one, and sixty respectively. A UDDI web service registry was built and a number of web services are published onto it (same with n_{sr}). Table 5.1 shows the average time of querying a service on two different service discovery platforms. The time of querying a Grid service with semantic concepts is longer, because the additional computation efforts are required to determine the concept similarity in the logic reasoning system.

	UDDI	Semantic Matching Middleware
Time (ms)	37.4	52.1

TABLE 5.1: Time of Querying a Service

Although UDDI has a faster system querying performance than our semantic service discovery middleware, it has several shortcomings when used in practice for the service discovery. UDDI does not provide sufficient technical details of the service, does not support any inference based on the concepts, can only support the search based on the string comparison, and cannot identify a match between functionally equivalent services that are described by different key words. Our service discovery middleware overcomes these shortcomings by using the semantic service description and discovery mechanism. We believe it is worth obtaining a relatively-significant improvement in system function at the price of a small increase in the service discovery time.

5.8.2 Scalability

In the above experiment, we keep four key parameters (n_{ir}=1, n_{or}=60, n_{ms}=1, n_{sr}=50) constant and measure the service query time using the semantic matching middleware and UDDI. In this evaluation stage, we evaluate the scalability of our semantic service matching middleware in terms of these key parameters. The objective of the evaluation is to acquire the variation trend of the service query time as the number of individual requirement, the size of ontology repository, the number of matching services, and the size of service repository vary. The service query time is expected not to be tightly proportional to the increase of these parameters, and should be within an acceptable limit.

We designed two experiments to investigate the scalability of the semantic service matching middleware. The experiment platform is a desktop equipped with Intel Pentium 2.4 GHz processor and 1GB memory.

Experiment one: we keep the number of individual requirement at 1 (n_{ir}=1) and the size of the ontology repository at 60 (n_{or}=60). The service query time is measured when the size of service repository (n_{sr}) varies from 10 to 400 and the number of matching

services (n_{ms}) is assigned to be one, two, four and eight. This experiment has been repeated twenty times and the final value is the average of experiment results.

The values of service query time gained in each case are listed in table 5.2 and figure 5.10.

Service Repository	n_{ms}=1 (ms)	n_{ms}=2 (ms)	n_{ms}=4 (ms)	n_{ms}=8 (ms)
10	39.3	58.2	90.3	140.4
20	42.5	64.8	92.3	143.5
50	52.1	74.9	104.1	165.7
100	55.8	75.4	109.5	175.3
200	60.4	80.3	115.4	181.9
400	66.1	91.6	120.8	192.3

TABLE 5.2: Average Query Time When Increasing Size of Service Repository and Number of Matching Services

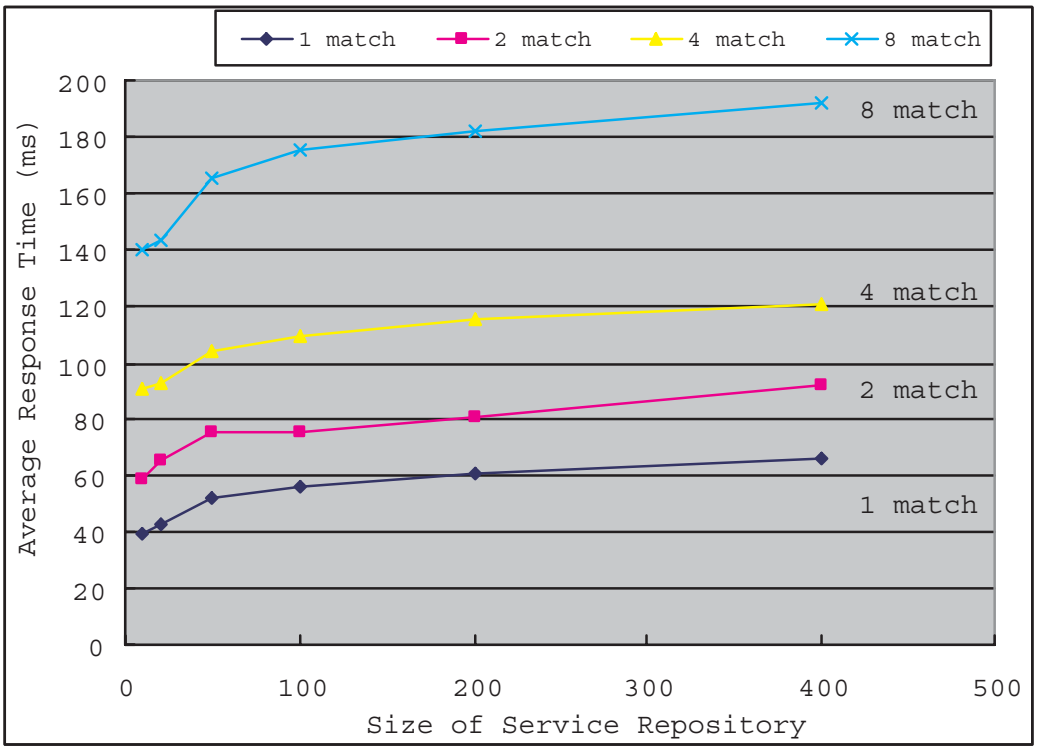

FIGURE 5.10: Average Service Query Time vs. Size of Service Repository and Number of Matching Services

From table 5.2 and figure 5.10, it can be observed that the service query time increases as the size of service repository and the number of matching services increase. However, the service query time is loosely proportional to these two parameters. Furthermore, the maximum value of the service query time in our experiment is within an acceptable limit (192ms). We believe the values assigned for both the number of matching service (from 1 to 8) and the size of service repository (from 10 to 400) are in a reasonable range, especially for the practical application scenario that mobile devices need to locate required

Grid services for the task execution from a service repository which has four hundred available services registered and at most eight candidate services are returned. Hence, it can be concluded that the service query time of the semantic matching middleware is satisfied for reasonable numbers of matching service and the size of service repository under the practical application scenario.

Experiment two: we keep the number of matching service at 4 (n_{ms}=1) and the size of service repository at 100 (n_{sr}=100). The service query time is measured when the number of individual requirement (n_{ir}) varies from 1 to 8 and under three kinds of ontology repository, which include 60, 150 and 250 classes respectively. In each case, the service query process has been executed for twenty times and the final value is the average of experiment results.

The values of service query time gained in each case are listed in table 5.3 and figure 5.11.

Individual Requirement	n_{or}=60 (ms)	n_{or}=150 (ms)	n_{or}=250 (ms)
1	109.5	126.2	147.6
2	138.2	169.6	219.5
3	162.4	202.4	276.9
4	204.5	252.2	336.4
5	231.1	290.9	390.1
6	260.2	335.1	450.7
7	290.1	370.8	520.3
8	328.3	414.9	584.8

TABLE 5.3: Average Query Time When Increasing Number of Individual Requirement Under Different Ontology

From table 5.3 and figure 5.11, it can be observed that the service query time increases almost linearly as the number of individual requirement increases from one to eight. This is because for each extra requirement in the service request, it takes time to analyze and compare the individual requirement with related concepts in the ontology repository.

The service query time also varies under different sizes of ontology repository. From figure 5.11, it can also be observed that the service query time increases as the concept number of the service repository. For example, when using the ontology one, which contains 60 classes, the service query time is 205ms (n_{ir}=4); when using the ontology three, which contains 250 classes, the service query time rises to 336ms (n_{ir}=4). This is because it takes more time to check the concept similarity in a larger size of ontology repository.

The maximum service query time in the experiment is an acceptable value (585ms). Considering in most cases the number of individual requirement in one service request does not exceed eight and the ontology repository does not contain the concept definition of more than 250 classes, it can be concluded the service query time is within

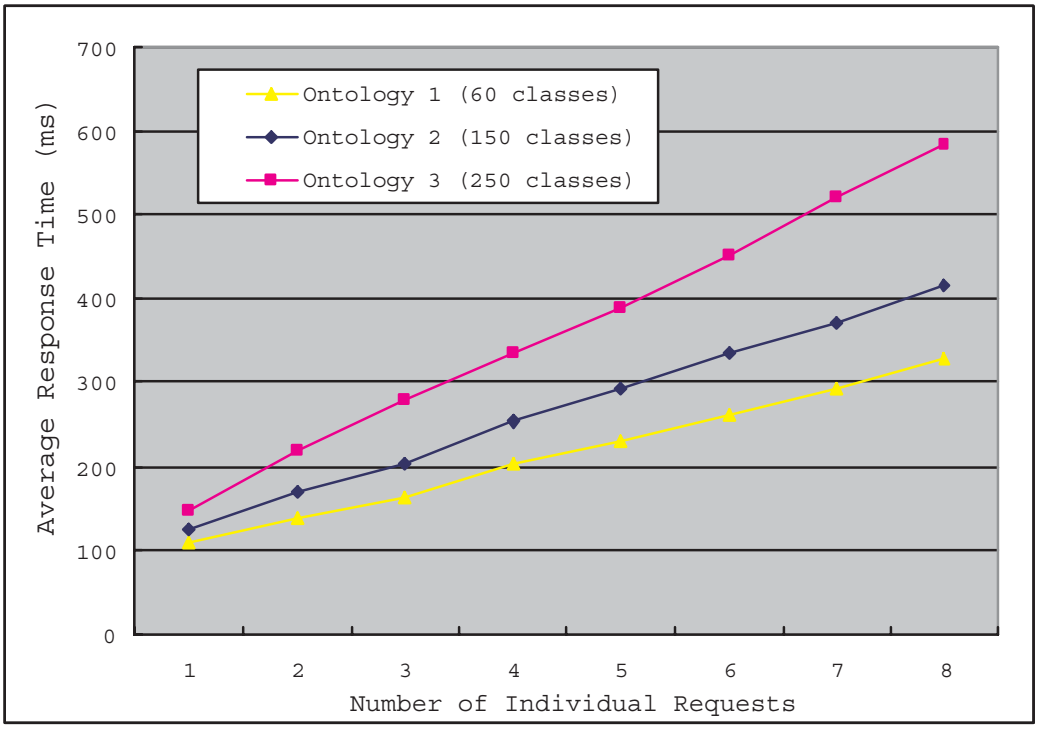

FIGURE 5.11: Average Service Query Time vs. Number of Individual Requirement and Size of Ontology Repository

an acceptable limit in terms of reasonable value of the ontology repository size and the individual requirement in a service request.

5.9 Summary

An important challenge of integrating mobile devices into the service-oriented Grid computing environment is that mobile devices need to locate, find, select and invoke appropriate Grid services. With the proliferation of various Grid services, semantic specifications are gradually becoming a necessary requirement for supporting a flexible and interoperable service discovery mechanism. Furthermore, a rich semantic definition provides a comprehensive representation of a variety of aspects of Grid services and builds an essential foundation for possible automatic behaviors (e.g. discovery, composition, invocation) throughout the whole Grid service lifecycle.

In this chapter, we have presented a semantic approach for both service description and service discovery to build service matching middleware in the system architecture. A number of service attributes as well as service IOPEs need to be considered to represent service characteristics in the service description. OWL-S is a service description language that constructs a conceptual framework for describing semantic services. Using the extended OWL-S ontology structure enables Grid services to be described more explicitly

and allows their detailed properties to be understood by semantic-based applications in a consistent and universal way. Compared to existing service discovery protocols, our semantic service discovery mechanism supports flexible matching between service attributes and service requirements on the basis of user-expected matching level, and offers ranking information for users to select the most appropriate service.

The service information centre middleware is built with the Jena Java programming APIs for OWL ontologies and the RACER ontology reasoning system, which provides query interfaces for users or other middlewares to locate required services. A service request is compared with a service description collection in two steps: strict requirements have to be matched precisely; general requirements are evaluated with the semantic similarity between concepts according to user-expected matching levels. The matching services as well as their matching degrees are returned as the result of a service request. The semantic service discovery middleware has been integrated into the system architecture and it interacts properly with other middleware by providing a programming interface. Its performance has been measured, and the results show that a significant improvement of the system function has been obtained with only a small increase in the service discovery time (compared to the traditional service discovery mechanism).

Chapter 6

System Architecture

The primary goal of this research is to build a system architecture to provide enhanced Grid access for mobile devices. Grid services enhance the capabilities of mobile devices to perform complex tasks which cannot be achieved on the devices themselves. Chapter four described a context-aware framework, which supports the intelligent interaction between mobile users and the computing environment. Chapter five presented a semantic approach for service description and service discovery, based on which a Grid service matching middleware is implemented. Both the context-aware framework and the semantic service matching middleware represent the foundation necessary to build a system architecture.

Although there are a number of mature and developing Grid client middleware tools, most of them make assumptions about the capabilities of their executing environment. These assumptions mean that it is very difficult to configure resource-constrained mobile devices to be capable of interacting with the Grid. In our system architecture, mobile devices utilize Grid services through "deputy" objects, which are created for every mobile user by middleware in the communication initialization stage. The deputy object is a program that exists in the service-oriented Grid environment and interacts with distributed resources on behalf of mobile devices. Furthermore, offloading the task involving Grid service invocations to a resource-rich and stable platform improves the potential system performance compared to executing tasks (invoking services) directly from mobile devices themselves (by eliminating the overhead processing on resource-limited devices and the necessary message transfer between mobile devices and Grid service providers).

In this chapter, we present our system architecture to support the task execution through resource-limited mobile devices. The middleware built upon the ontology-based context-aware framework and the semantic service matching approach hides the diversity of heterogeneous mobile devices, enables the required services to be discovered, and provides a reliable task execution mechanism to assist mobile users to interact with Grid services.

6.1 Design Principles

The definition of transparency (originally from the humanities) implies openness, communication and accountability. It is a metaphorical extension of the meaning used in the physical science: a "transparent" object is one that can be seen through [156]. In a computer system, transparency refers to hiding unimportant implementation details from system users. The middleware in the system architecture is required to provide a transparent access mechanism enabling mobile devices to utilize comprehensive Grid resources while shielding them from the complex system implementation details.

In the system design and implementation, transparency is achieved in two specific aspects:

- Device Adaptation

  Mobile devices have various hardware equipments and software environments. The laptop has a similar interface to the traditional desktop, while the interface of the smart phone is very different from that of the desktop, because the phone has smaller displays, no keyboards, compact software libraries and so on. The client software of the mobile Grid middleware has to be utilized on a wide range of mobile devices to provide the same functionalities. Furthermore, task results should be provided in different formats so that they can be displayed properly on a number of heterogeneous devices.

- Reliable Task Achievement

  Mobile devices are characterized by their limited resources and high mobility, which leads to an unreliable availability in the static Grid environment. This makes it very difficult to manage task execution from mobile devices. The middleware has to provide a reliable management mechanism to execute tasks submitted from mobile devices, including analyzing tasks, locating Grid services required, invoking Grid services, monitoring task execution and so on. Because of the intermittent connectivity, the mobile device may lose connection to the Grid environment. The middleware should support offline processing and automatic status recording so that mobile users do not need to know the detailed underlying complexity when the connection between devices and the Grid environment is re-established.

6.2 Architecture Overview

Service-oriented Grid environments share resources between distributed users, sites and virtual organizations. Resources can be clusters, workstations, or any other type including information provider, printers, data centre and so on. All of these resources are

encapsulated and exposed to users through Grid services. Generally speaking, a virtual organization provides various Grid services to other organizations which are able to consume services through static terminals. Grid services can also be made available to mobile devices as long as they are valid users in the service-oriented Grid environment.

In our system architecture, mobile devices access Grid services through their "deputy" objects deployed on a resource-rich static platform. As a program that interacts with distributed resources on behalf of mobile devices, the deputy object is responsible for analyzing tasks from mobile devices, preprocessing tasks for the service invocation, locating and invoking required Grid services, monitoring the process of the task execution, collecting service invocation results from the Grid environment, and returning final results to users.

Figure 6.1 shows a diagram of the overall system architecture. The system architecture is designed to overcome both the slow processing capability of mobile devices and the unreliable data transmission through the limited-bandwidth wireless network.

As illustrated in the Figure 6.1, the static remote distributed resources and mobile devices are interconnected by the Grid gateway. The Grid gateway is a small server available for nearby mobile devices within the covered range through the local wireless network, providing a relatively resource-rich, stable execution environment and network connectivity when compared to handheld devices. The mobile deputy middleware on the Grid gateway provides a reliable task execution mechanism, including task submission, task execution, execution monitoring and result recording, by creating a deputy object for every mobile device which interacts with the Grid environment on behalf of the mobile device. During task execution, the deputy object invokes required Grid services through Grid service interfaces.

Also note that two components exist in the service-oriented Grid environment. The service information centre is responsible for the implementation of the Grid service registration, discovery and management for the deputy object to locate and select required Grid services. The required service matching approach and detailed implementation have already been discussed in chapter five. The context information centre stores various context information about users, mobile devices and other entities in the computing environment, which can be acquired by the mobile deputy middleware through the context query and reasoning interfaces, to support the intelligent interaction between mobile users and the Grid computing environment. Its design and detailed implementation are based on the context-aware framework which is discussed in chapter four. In the following section ,we concentrate on the other elements of the overall system architecture.

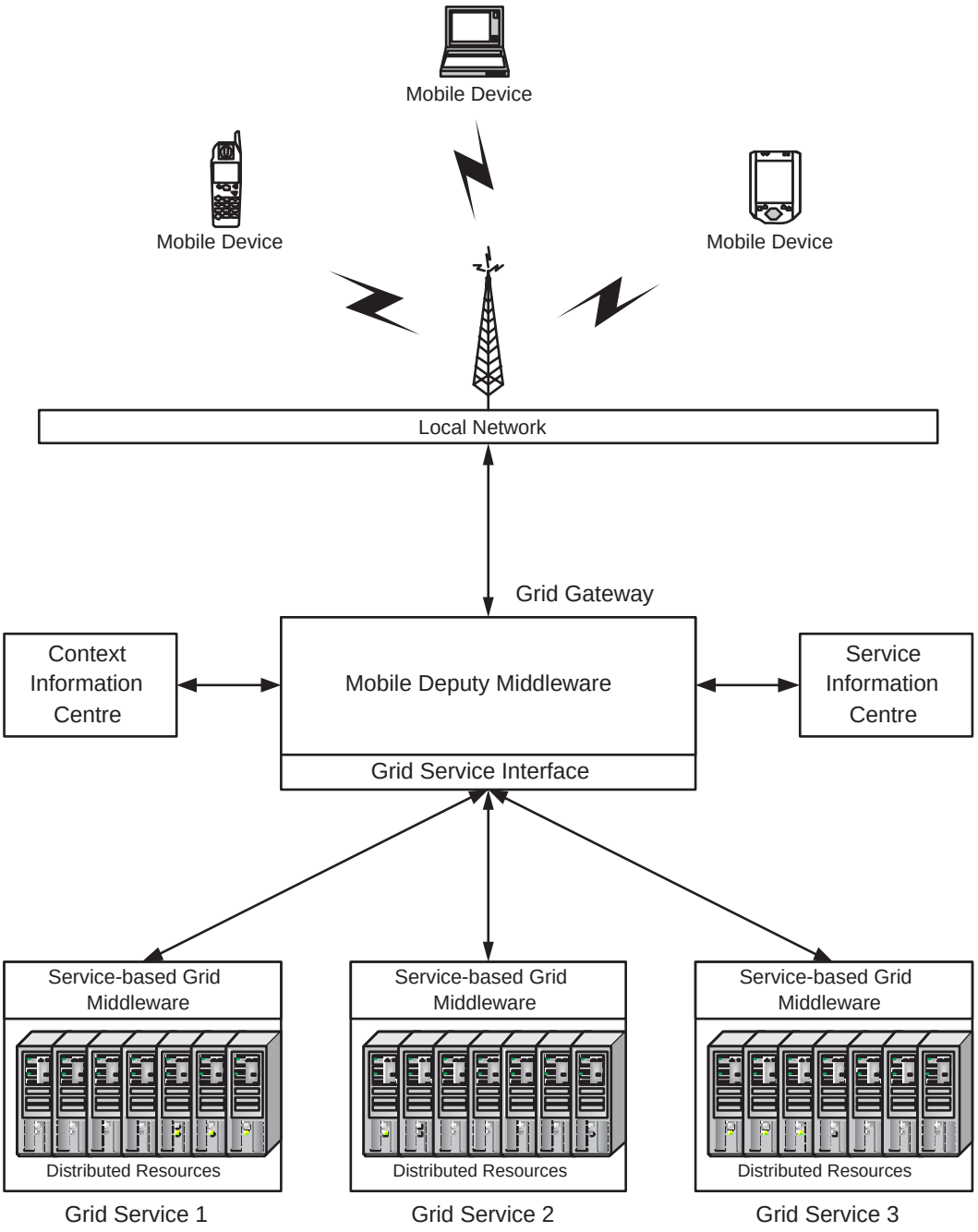

FIGURE 6.1: Overall System Architecture.

6.3 Details of the Overall System Architecture

6.3.1 Mobile Devices

As a convenient carry-on equipment, mobile devices enable various services around the environment to be available and accessible for pervasive users. Mobile devices have the capability of communicating with other devices through a variety wireless network protocols, including IEEE 802.11 families [157] and Bluetooth, depending on the resource

of different styles of devices (802.11 consumes more energy, which exhausts the battery in a cell phone quickly, but may well be suitable for laptop devices). Because current mobile devices do not run on uniform hardware platforms, and their software development environments vary widely, it is almost impossible to design a unified client software to be implemented and installed on all mobile devices.

The user personal profile and the device information may be stored on mobile devices in the format of ontology metadata, because they are demanded by the deputy middleware in the process of communication initialization. The user private data may also be required before requesting the invocation of specific and distinct Grid services. For example, the profile of a scientist needs to be sent to the Grid every time the scientist requests modification of the important lab parameters, to ensure that it is the right person who has obtained authorization. In some application scenarios, unfinished task status and intermediate results can also be stored on mobile devices rather than on the deputy middleware, if there is enough storage space, to avoid submitting the same request to the Grid environment and increase the speed of the task execution.

The client software on mobile devices is responsible for interacting with the Grid gateway, organizing the task request from users, and transferring the request to the mobile deputy middleware. When the task execution is completed, it is also responsible for displaying the results for users. The detailed implementation of a client software on mobile devices depends on the interface of the mobile deputy middleware and the capability of mobile devices. For example, if the mobile deputy middleware is written as a web application, users can submit their tasks through a web browser; if the mobile deputy middleware is implemented as a standard web service, a web service client program is required for users to submit their tasks. No matter what format the mobile deputy middleware is, the following functional modules need to be included in the mobile device software architecture (Figure 6.2).

- Wireless Network Module: this module is responsible for low-level communication with the deputy middleware and any other devices through various wireless network protocols. The function of this module is usually provided by Operating Systems or drivers installed on top of Operating Systems.
- Connection Management Module: this module discovers and selects the Grid gateway, and is responsible for initializing and terminating the connection between mobile devices and Grid gateways.
- Monitor Module: this module is responsible for keeping track of the status of the submitted task and ensuring that the middleware knows the mobile device is still connected to the Grid gateway, for example, sending a “keep-alive” message to the mobile deputy middleware periodically.

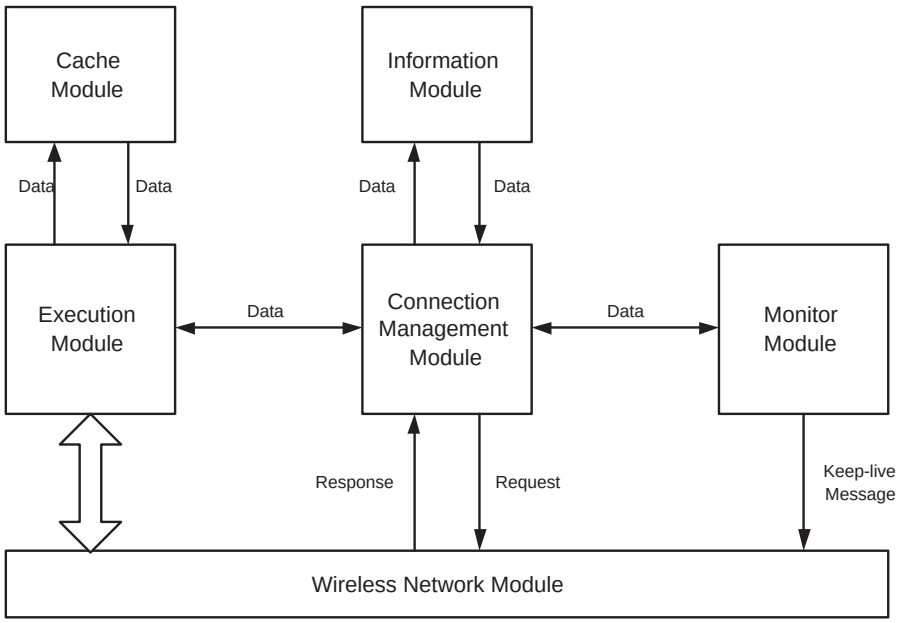

FIGURE 6.2: Modules in the Device Software Architecture.

- Execution Module: this module starts to work after the connection between the mobile device and the mobile deputy middleware is established by the connection management module. It is responsible for initializing a task and submitting the task to the mobile deputy middleware. During the task execution, the execution module collects the results from the Grid gateway, processes the results, and displays the results for users.
- Information Module: the information of the users and their mobile devices is stored in this module as a metadata file. During connection initialization, the connection management may collect and send the metadata to the mobile deputy middleware.
- Cache Module: this module is optional and is used only for appropriate tasks. It stores the temporal results or the status of the task execution to avoid submitting the same request.

6.3.2 Mobile Deputy Middleware

The deputy middleware offers a connection interface for mobile devices to submit their tasks. If the user request is authorized and the personal information has been stored in the system, local resources on the Grid gateway will be allocated to mobile users and a deputy object is created for analyzing and transforming the task into an executing entity. If the allocation of the local resources and creation of the deputy object are successful, a connection contract is established between the mobile device and the Grid gateway. The deputy middleware executes the user task by utilizing local resources and Grid services, while at the same time it is also responsible for communicating back with

mobile devices if additional task inputs are required. When the task is completed and the mobile device terminates the connection to the Grid gateway, local resources will be released and the deputy object will be destroyed.

The implementation of the mobile deputy middleware depends on various context which is stored in the context information centre. Figure 6.3 shows logical modules in the mobile deputy middleware and their connection with the context information centre.

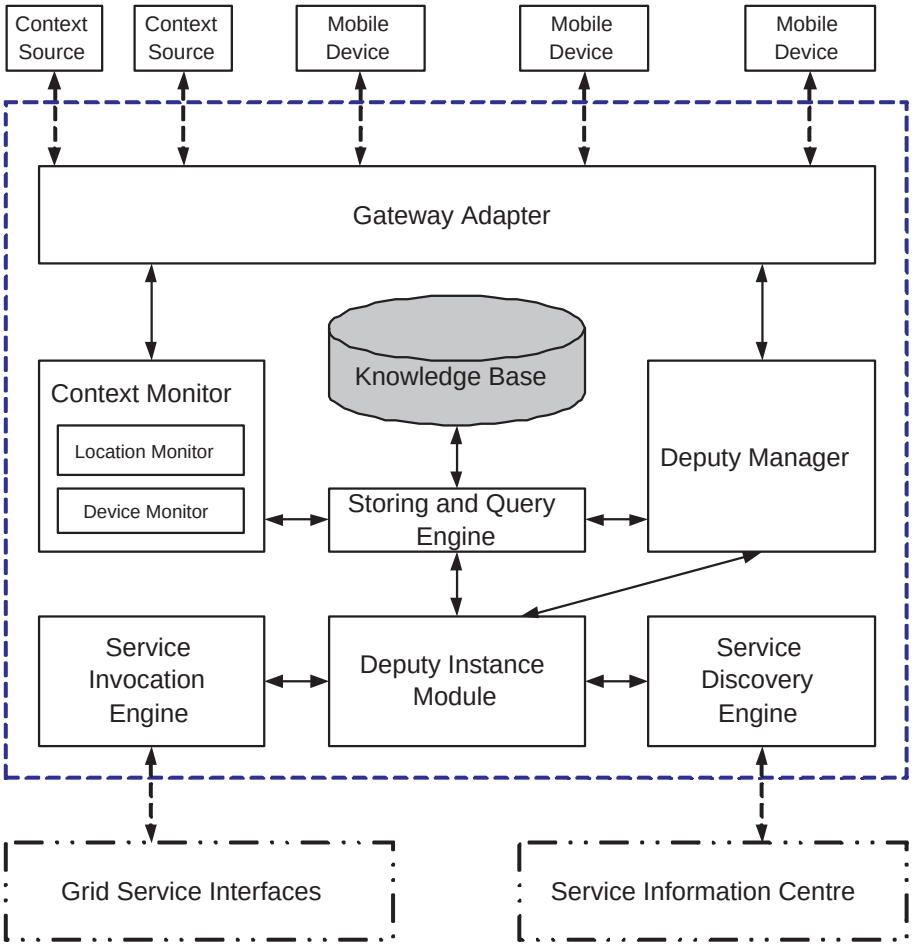

FIGURE 6.3: Logical Modules of Mobile Deputy Middleware and Context Information Centre.

- The Gateway Adapter is a connection interface for mobile devices. It enables the initial communication between a mobile device and the Grid gateway, and transfers the request to the Deputy Manager. The deputy middleware may operate as a web service or a web application. Mobile devices can communicate with the gateway adapter via the SOAP framework or the HTTP protocol for different interaction modes.

- The Deputy Manager is the central module of the mobile deputy middleware and is responsible for making decisions in a task execution process. The Deputy Manager authorizes the connection between mobile clients and Grid gateways, creates a

deputy object for every mobile client, and generates a "User" instance stored in the knowledge base according to the defined "User" class. The "User" instance records various information about the mobile client, including his/her device, accessing level, submitted tasks, task status and so on, which are used by context queries for further interaction. A thread in the Device Monitor is started by the Deputy Manager to monitor the connection status of mobile devices. When a task is completed or the mobile device loses the connection to the Grid gateway for a threshold time, the Deputy Manager will destroy the deputy object created for this mobile client.

- The Deputy Instance accepts tasks from the mobile client, analyzes tasks, locates required Grid services by interacting with the Service Discovery Engine, wraps tasks to be the service executing entity, and activates the Service Invocation Engine for the task execution. Intermediate results (e.g. the result of the service invocation) are stored in the deputy object, and when the task is achieved, the deputy object sends the final result back to the mobile device based on the properties of the user device (the "device" instance was created at the period of the task request) as long as the mobile device is still online. Otherwise, the result will be stored in the deputy object.

- The Context Monitor obtains the raw environment information from the Context Source and stores the context in the knowledge base. At present, two kinds of monitor are concerned. The device monitor is responsible for assessing whether the mobile device is still connected to the Grid gateway. If a mobile device is offline, the device Monitor updates the knowledge base and informs the Deputy Manager the disconnection. The location monitor is responsible for accepting context markup from external detection sensors and updating the user location information in the knowledge base.

- The Service Invocation Engine interacts with the Service-based Grid Middleware and is responsible for the service method invocation. The Service Discovery Engine interfaces externally with the Service Information Centre middleware in order to find services that are required during the process of the task execution.

- The Knowledge Base is used to store defined ontology classes and generated instances which update the context information of the computing environment. Its main function is to connect the mobile client and the service-oriented Grid environment seamlessly. Other modules of the middleware are able to obtain required context information by querying the knowledge base so that they can make intelligent decisions such as blocking the notification message to users when they are involved in an emergent condition and downscaling the results according to the mobile device profile.

The deputy object is a program that interacts with Grid services on behalf of mobile devices. It is built by the mobile deputy middleware when a mobile client connects to the Grid gateway. Not only does it help to analyze tasks, locate and select services, create the specific processing procedure based on the discoverable services, and activate the task execution, but it hides the long-haul and reliable communication requirement for general Grid clients. Figure 6.4 shows the interaction between a deputy object and other components of the service oriented mobile Grid environment.

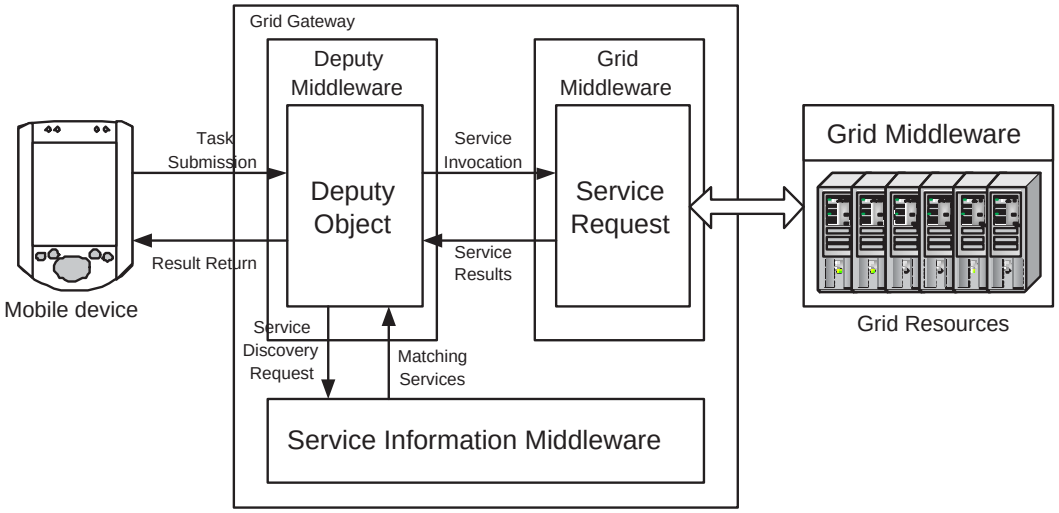

FIGURE 6.4: Deputy Object in the Grid Gateway.

The format of the task submitted from mobile devices to the mobile deputy middleware varies depending on different application scenarios. It may be a group of strings which involve the service name to be invoked (e.g. the "Hello Grid" test applications). For a very complex task which requires to invoke a number of Grid services, a task description document will be submitted by mobile users. The task description can be composed with RDF triples (e.g. the "Searching for Information" test application) or with workflow scripts [158], which can be parsed by the deputy object with corresponding tools (e.g. Jena Java library) before the Grid service invocation. For more general conditions, a large-size file which encapsulates the executing codes of the task, or a URI, which indicates the location where the task executing codes are located, may be transferred from mobile clients to their deputy objects.

The deputy object is located inside the mobile networking environment and can send the Grid service request using the service invocation interface. The task execution status is updated into the knowledge base by the deputy object. When the mobile device disconnects from the Grid gateways, the device monitor marks the related property value of the "Device" instance. The deputy object continues to execute the task. When additional interactions between mobile devices and Grid services are required (e.g. a new input data is required), the deputy object checks the current connection status of mobile devices by querying the knowledge base. If offline, the deputy object suspends the task

execution and records the stopping point. The task will resume after the connection is reestablished.

The support for intermittent connections provides great convenience in some application scenarios. Task execution is often time-consuming, especially for scientific problem-solving cases. The mobile client is able to disconnect from the Grid gateway intentionally after submitting tasks. Because the information of the mobile client is stored in the knowledge base, the deputy object continues to execute the task, monitor the execution status and record intermediate results. When the task execution is completed, a message may be sent to mobile users, informing them that the results are ready to be collected.

6.3.3 Service-based Grid Middleware

Grid middleware is software that mediates the interaction between various distributed resources and users. In a service-oriented Grid environment, distributed resources are wrapped and exposed in the format of Grid services. Generally speaking, a service-based Grid middleware is a toolkit which enables service providers to deploy their services, and the deployed services are managed and maintained in a standard way so that a uniform set of interfaces of the Grid service interaction are supported. In our system architecture, the service-based Grid middleware is responsible for providing a group of programming interfaces for deputy objects invoking required Grid services to perform user tasks.

There are many existing Grid middleware technologies which can be used in the system architecture to implement the communication between the Grid gateway and the service provider. Two particular examples are Globus Toolkit 4 (GT4) and Open Middleware Infrastructure Institute (OMII).

The Globus Toolkit is the de-facto standard in the Grid computing area which is developed by the Globus Alliance. Its latest version, GT4, includes components for building systems that follow the Open Grid Service Architecture (OGSA) and a complete implementation of the Web Services Resource Framework (WSRF) specification.

OMII is an open source, robust and secure Web Services platform for building Grid applications, supporting the most stable elements of the WSRF standard, a refactoring of the OGSI, which is totally based on Web Service technologies. Figure 6.5 shows a conceptual overview of the OMII server stack [159]. OMII integrates a set of software components, facilitating applications of accessing various Grid resources (e.g. OGSA-DAI allows data resources to be accessed via web services).

OMII implements a client/server infrastructure to make clients and service providers collaborate. Its server component enables secure authorized access to Grid resources and its client component provides the means to interface with an OMII Server to enable clients to consume deployed services. OMII client software provides both the command

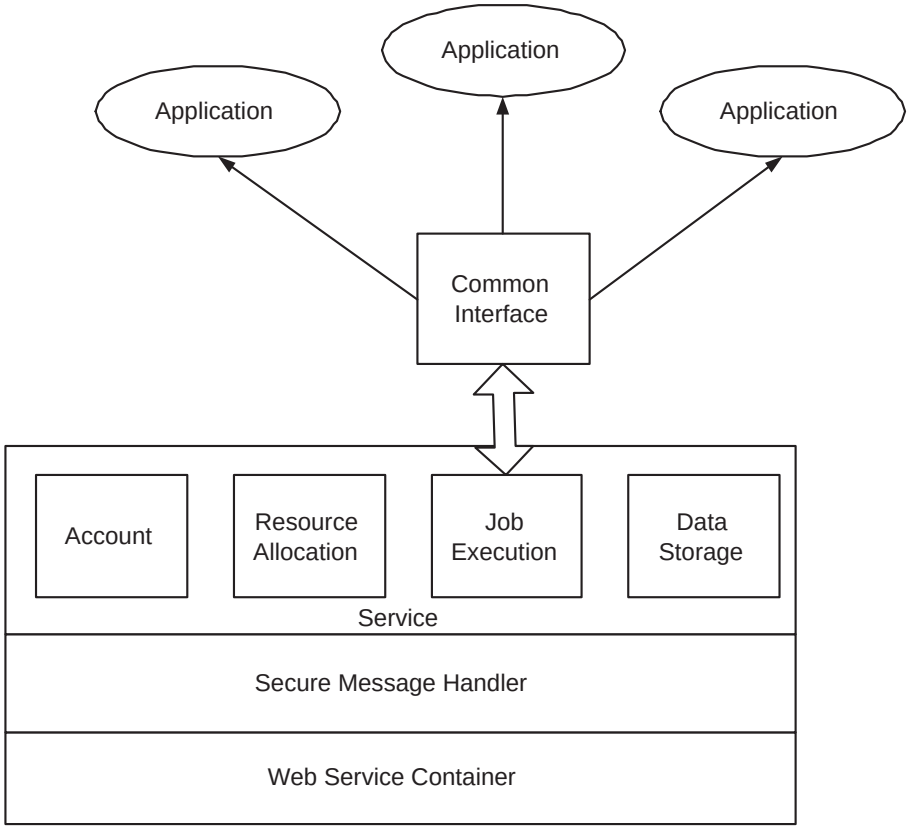

FIGURE 6.5: OMII server stack.

line interface and the Java interface to allow users to write their own programs to interact with Grid resources.

Because OMII is based on Web Service technologies, the communication between the client and the server uses the SOAP protocol, which is sent over HTTPs via the Axis Client. The OMII client supports both dynamical invocation (using DynamicInvoker class) and static service invocation (using Java client stubs).

6.4 Interaction Protocol

Figure 6.6 shows a general interaction protocol which takes place when mobile devices submit a task which involves the Grid service invocation. It is assumed that the required Grid services can be discovered and located through the service information centre middleware. There has been a trust relationship already established between the Grid gateway and Grid services, so that the deputy object is able to access Grid resources smoothly.

An integrated process of the mobile client accessing Grid services to accomplish the complex task, is described as following:

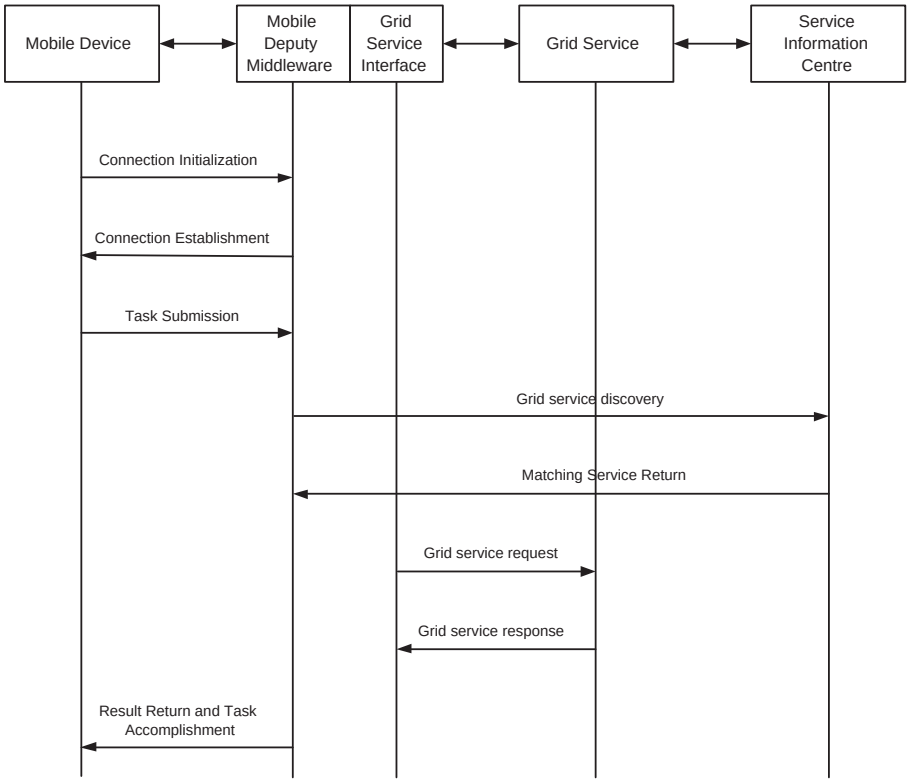

FIGURE 6.6: Interaction Protocol.

1. Connection Initialization: The mobile device connects to a Grid gateway, transferring the user information and submitting a request to execute a complex task.

2. Connection Establishment: If the mobile device is authorized to interact with the Grid gateway and there are adequate resources on the gateway available for the mobile device, a connection contract is established between the mobile device and the Grid gateway, and the mobile deputy middleware creates a deputy object for the mobile device.

3. Task Submission: The mobile device submits the task to be executed to the Grid gateway. The created deputy instance accepts the task.

4. Grid Service Discovery: The deputy object analyzes the task, decomposes the task, and interacts with the service information centre to find and locate the required Grid services for the task execution.

5. Matching Service Return: The information centre locates the existing Grid service(s) as well as the matching degree for the service discovery request.

6. Grid Service Request: The deputy object starts to process the task, requests the invocations of the Grid service through the Grid middleware interface, and waits for the results from the service execution. At this time, the mobile device may disconnect from the Grid environment. The deputy object stores the state of the

task execution, monitors the service execution, and suspends task execution if required.

7. Grid Service Response: The deputy object acquires the Grid service execution results. These results needs to be processed for preparing the final result which will be displayed on the mobile device.

8. Result Return and Task Accomplishment: The final result of the task is returned and displayed on the mobile device. The Grid gateway terminates the connection contract with the mobile device and releases resources and deputy objects allocated. If the mobile device cannot acquire the result due to being offline, the deputy object stores the result in a given time and waits for the connection re-established for result collection.

6.5 System Architecture Implementation

6.5.1 Interface of Context Information Centre

The context information centre is implemented based on the context-aware framework discussed in chapter four. However, in order to enable smooth interaction with the mobile deputy middleware (e.g. importing contexts into the knowledge base, handling context queries), we improved its external programming interfaces and expose them to the mobile deputy middleware. The following shows several of the main programming methods:

- CreateIndividual(OntClass class, String string): This method is used for create an instance of the class in the context model. When a new user submits a task for the Grid gateway, the deputy manager uses this method to create the corresponding user, device and task instances.

- ResourceObtain(URI uri): This method is used for getting a resource object in the context model. Given the URI of a resource, this method returns a resource object if one exists in the context model, or fails if there is not such a resource object.

- ModelUpdate(Resource res, Property pro, String value): This method is used to update a property value of a context model resource. When the condition changes (e.g. device disconnection, task status change, user presence or absence), this method will be invoked by other components to update the context information stored in the knowledge base.

- ModelQuery(String query): Given a query string, this method returns statements in the context model. It is used by the deputy object to check user information

(e.g. current location, carrying device). The user can invoke this method to check the task execution state.

- ModelReasoner(String rule): Given an application rule, this method returns the high-level context information which is acquired through reasoning process. The generic rule reasoner provided by the Jena tool is adopted in the context information centre.

6.5.2 Interface of Service Information Centre

The service information centre is implemented based on the service matching mechanism discussed in chapter five. A programming interface is provided so that the mobile deputy middleware can discover required Grid services for the task execution:

$$public\ \ Service\ \ serviceDiscovery\ (Profile\ req)$$

This function takes a service request as input and returns a service object, which encapsulates various service description information (especially the URI and service methods), supporting the service invocation in the next step. In chapter five, we noted that the service discovery result may include a set of candidate services as well as their matching degrees. However, in order to simplify the service selection procedure and enable automatic service invocation, only the service which has the highest matching score is returned by the "serviceDiscovery" function.

6.5.3 Mobile Deputy Middleware

The Grid gateway is the physical interface for mobile devices accessing the digital Grid environment, providing a relatively resource-rich and stable task execution platform for nearby mobile devices. In our system, a high-performance desktop functions as the Grid gateway, on which the mobile deputy middleware is deployed. We assume Grid service providers deploy their services using OMII Grid middleware so that Grid services can be invoked through OMII client interfaces. The server-side applications can be executed through either a command line or writing client programs by using the middleware Java API libraries.

The prototype implementation of the mobile deputy middleware is developed and tested in the J2EE environment, which is written as both a standard Web Service and a Java web application using the AJAX design mode. The Web Service enables the mobile deputy middleware to expose a set of programming interfaces so that mobile devices can communicate with the deputy middleware. The software on mobile devices, as a Web Service client, is able to invoke the methods provided by the mobile deputy middleware.

A number of methods are required to implement the prototype of the mobile deputy middleware and the key ones include:

- CreateDeputy(): The CreateDeputy() method is used by the client program on mobile devices. It is the first step for mobile devices to interact with the mobile deputy middleware. If this method is successful, the mobile deputy middleware creates a deputy object and establishes a connection contract between the mobile device and the Grid gateway.
- DeviceDisconnection(): Mobile users may disconnect to the Grid gateway after submitting a time-consuming task. This method is designed for the client program to inform the deputy middleware that the user will drop the connection. After successful invocation, the deputy object continues to execute the task and monitors the progress of the Grid service invocation independently. If additional inputs or decisions are required from mobile users, the deputy object suspends the task execution and records the stopping point on both the deputy object and the knowledge base.
- TaskSubmission(): The client program uses this method to submit the task. As discussed in the above section, the input parameters of this method vary depending on different application scenarios. If this method returns successfully, it indicates that task execution has started.

The mobile deputy middleware can also be implemented as a Java web application. This is particularly suitable for mobile devices which cannot install a web service client program due to the lack of the software library support (e.g. J2ME is not supported on some mobile device platforms). The prerequisite is that mobile devices are equipped with standard web browsers so that they can submit tasks and retrieve results through the web application interface. This is a reasonable assumption because multimode browsers are gradually becoming general equipment on new mobile devices. Figure 6.7 shows the detailed interaction between mobile devices, Grid gateways and Grid services, in which the mobile deputy middleware is developed as a web application.

We evaluate our system architecture in two stages: first, we built several sample application scenarios to test the functionalities of the system architecture before this design is used for the real Grid service interaction; second, we compared the task execution time when the mobile client accesses the service-oriented Grid environment with and without using the mobile deputy middleware in order to demonstrate the value of offloading the Grid service invocation from mobile devices to a resource-rich and stable execution platform.

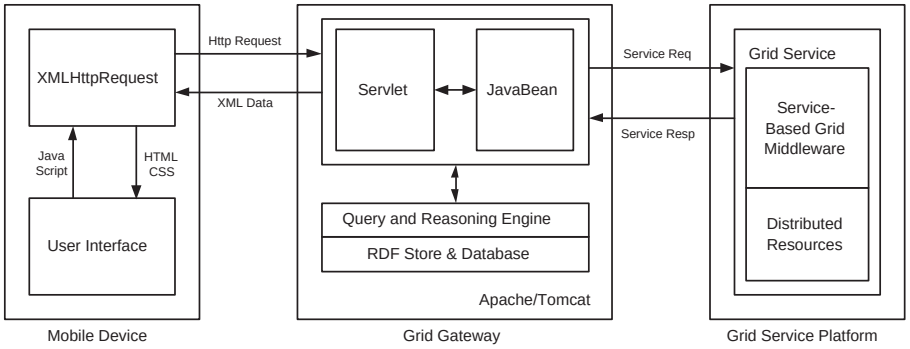

FIGURE 6.7: Interaction between mobile devices, gateways and Grid Services.

6.6 Test Applications

6.6.1 "Hello Grid"

The "Hello Grid" is a test application used to test the connectivity between mobile devices, Grid gateways, and Grid services. In this application scenario, mobile users are able to submit a request to invoke the "Hello Grid" service deployed on the Grid platform. The deputy object created for the mobile user performs this task through the method interface provided by the "Hello Grid" service. If successful, a welcome message will be produced by the deputy middleware and returned to the mobile user. Figure 6.8 shows the image of a PDA submitting the "Hello Grid" task and obtaining the welcome message after the task execution. Although this "Hello Grid" application represents only a straightforward test scenario and does not demonstrate the value of offloading complex tasks from mobile devices to the Grid environment, nevertheless, it proves that the approach of mobile devices accessing Grid services by using our designed system architecture is viable.

6.6.2 "Mobile Shopping"

"Mobile Shopping" is another application scenario, which is developed to test the functionality of the system architecture, and compare the system performance with and without using the mobile deputy middleware. In this application scenario, users are able to perform online shopping through their mobile devices in their spare time. An online shopping service is provided on the local Grid platform. The completed shopping process consists of four operation procedures:

- A user submits the shopping task to the deputy middleware on the Grid gateway through mobile devices before an important meeting starts.
- The deputy middleware accepts the shopping request and starts to locate the online shopping service using the query interface provided by the service information

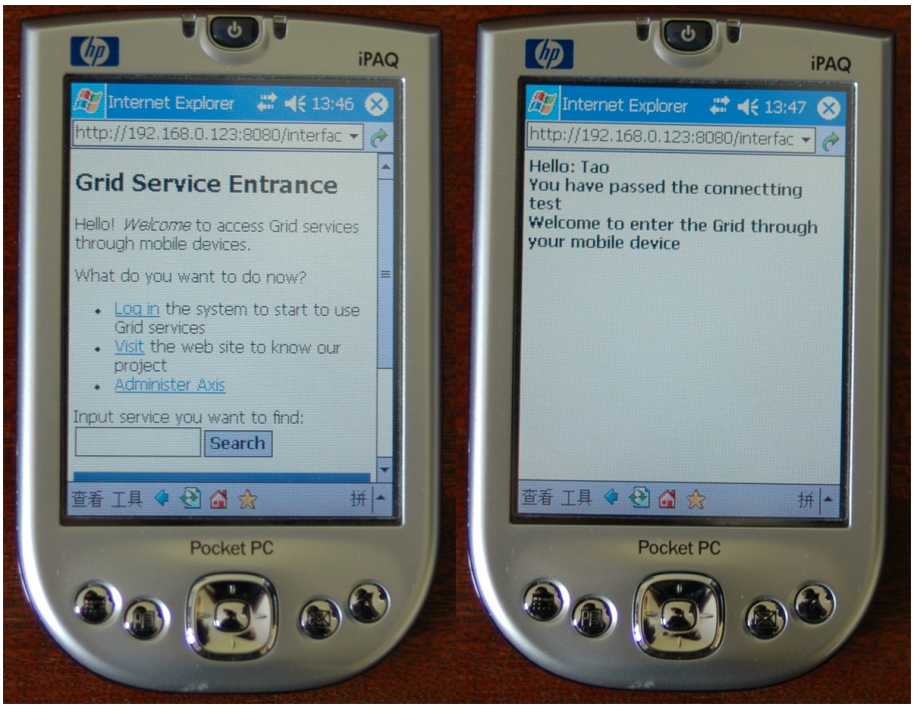

FIGURE 6.8: "Hello Grid" on PDA.

centre middleware. After the process of the application initialization and the service location, the deputy middleware checks whether the user is still in the meeting room by querying the context information stored in the knowledge base. If the user has left the meeting room, which suggests he can be interrupted (not in the meeting), a shopping interface is returned to the user mobile device to enable the user to view and select desired items and specify the quantity for each kinds of selected item.

- After choosing goods and sending them back to the Grid gateway, another interface is transferred to the handheld device, asking the user to input personal information, including his name, address, and credit card number.

- When the user confirms the transaction, the deputy instance submits the user order to the online shopping service by invoking "order()" method. The service then processes the request, checks the personal information, keeps items and returns a confirmation of order number. The deputy object will send the information back to the user mobile device when the user is not at work.

The online shopping service is implemented using OMII Grid middleware. The OMII software is based on both Web Service and Grid service standards. Services can be provided by Apache Axis hosted by Jakarta's Apache Tomcat web server, and utilized conveniently by the client through SOAP for communication. A resource object is declared for the online shopping service, the virtual user shopping cart which keeps the item information the user orders. The Web Services Resource Framework (WSRF)

specification defines a set of mechanisms for service clients to manage stateful resources through a standard set of interfaces. The OMII software component supports WSRF compliant Web Services and provides a number of base classes for building WSRF Web Services. Resources can be specified by either a process which holds and manages their state or a file which stores their state between calls to the service. We adopted the file-based WS-Resource mechanism to operate the user shopping cart of the online shopping service.

When the user attempts to build a communication channel with the online shopping service, a new shopping cart instance is created with a UUID generated as a reference to this shopping cart instance. The UUID is returned to the user deputy object, so that the user can operate the shopping cart (e.g. viewing, adding or removing items). The user may choose to save the virtual shopping cart before confirming the transaction (e.g. leaving the virtual shop because of a coming meeting and continuing to shop after meeting). Because the value and the status of the user shopping cart are stored in a file, the file locking technique is required to ensure that only one operation is performed on one resource instance at one time.

The online shopping service contains a number of methods which are required to implement the service function. It also includes several methods for clients to view items in the shopping cart, add desired items and specify quantity of each item, remove items from the shopping cart. The most common use method of operating the service resource is

$$public\ \ void\ \ addItem\ (String\ uid, Object\ item, int\ quantity)$$

The addItem method is responsible for putting the items and their quantity to the user shopping cart. The online shopping service needs to access the service resource within the addItem method. This is achieved by obtaining the file lock and loading the resource based on the *uid* parameter, adding the *item* and its *quantity* into the virtual shopping cart, and update the new value and status of the resource. The detailed interaction is implemented by several file-oriented functions (e.g. *load*(), *save*(), *obtainLock*(), *releaseLock*()) which are responsible for getting and setting values of the service resource, providing a transparent operating mechanism for service users.

To deploy the online shopping service on the OMII service container, a Web Service Deployment Descriptor (WSDD) file is required. The WSDD file defines several deployment parameters, including the service name, the class name, allowed method, and the deployment scope. Based on the WSDD file, and the jar file which is created by packing the service classes, the service can be deployed with the Axis AdminClient tool.

As a service consumer, the deputy instance needs to communicate with the online shopping service on behalf of the user. The client program implements several methods, which are responsible for initializing the communication, managing the service resource,

and requesting the processing of the order. The OMII client supports both the dynamical service invocation (using DynamicInvoker class) and the static service invocation (using Java client stubs). We implement the service invocation using WSDL2Java tool, because writing the client by generating stub classes is straightforward.

In application testing, we set the location information of the user manually through the programming interface of the context information centre because the user location sensors have not been deployed in the computing environment. After the user submits the shopping request, the user location is set to "meeting room", which assumes the user is attending a meeting. The deputy object queries the location information of the user after initializing the communication and locating online service. Because the value is meeting room, the deputy object postpones the response message. The location information of the user is checked periodically. After some time, we change the value of the user location into "coffee room", which assumes the user has free time now. The deputy object acquires the location change and sends the shopping interface back to the user mobile device.

Another dynamic condition is considered in the application testing. Before the user confirms the transaction, it is assumed that the mobile device consumes all of the power. In the testing, we turn the mobile device down suddenly. Some time later, we turn on the mobile device and make it connect to the Grid gateway again. Because the personal and task information is still recorded in the knowledge base ("User" and "Task" instance) and the created deputy object (the deputy is stored in a file and its ID is recorded in the "User" instance) is not cleaned, the user is able to continue to shop.

6.6.3 "Searching for Information"

In the "Mobile Shopping" application scenario, the service is deployed on the local Grid platform. To test the system performance of invoking remote Grid services, we have built a "Searching for Information" application scenario. In this application scenario, "Google" is assumed to be a Grid server which provides powerful functions of the information search. In fact, Google releases a set of searching APIs to enable the Google search to be put in the user program using the JavaScript language. Dynamic search boxes can be embedded in the server container, and searching results can be used in various innovative and programmatic ways (e.g. being displayed in the local web page for users).

The "Searching for Information" application scenario consists of two main operating procedures:

- The first step is to search the information based on key words from the mobile user. The Google AJAX search API provides a number of searching modes, including

the local search, the web search, the news search, the blog search, the video search, the book search and so on. The user is able to choose which searching modes no matter which ones they like.

- After getting the searching results from the Google service, the user looks through these results, select their interested lists, and store selected lists into the local database.

Due to the minimal requirement of the current Google AJAX API, mobile devices are restricted to be a laptop. As with the "Mobile Shopping" scenario, two kinds of mobile clients have been built in the "Searching for Information" application scenario, the "with deputy" client and "without deputy" client. The "with deputy" client submits the searching key words and the personal preference to the Grid gateway, which is responsible for executing the searching request on behalf of the mobile client and transferring the results to the local database. On the other hand, the "without deputy" client has to submit the searching request to the Google server directly and perform the job of posting the desired searching lists to the local database. Figure 6.9 shows the diagram of these two kinds of searching information clients.

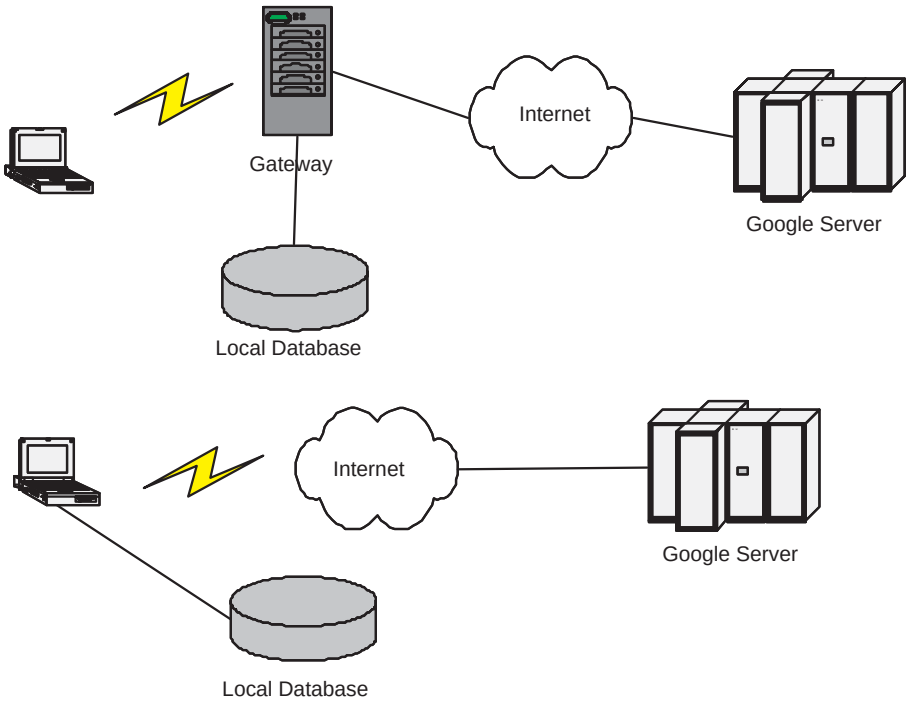

FIGURE 6.9: "With deputy" and "without deputy" client in the "Searching for Information" scenario.

Mobile devices are tested in above application scenarios through the system architecture we built. A user instance is created and stored in the system knowledge base, indicating the personal information, the device the user has, and the task the user submits. For the simple application scenario, such as the "Hello Grid", the user submits the service name

so that its deputy instance can invoke corresponding service directly. For complex applications (e.g. "Searching for Information"), the user submits a task description as well as input parameters, which is decomposed by its deputy instance. The deputy instance starts procedures (e.g. preprocessing tasks, locating services, invoking services) to work toward achieving the user task based on the requirement of the task decomposition.

The system architecture supports offline processing. During the test, we enable mobile devices to disconnect the Grid gateway intentionally. For the "Mobile Shopping" application scenario, as we discussed, the disconnection may occur during the process of user selecting items or the process of the transaction confirmation; with respect to the "Searching for Information" application scenario, the user may disconnect the Grid gateway after submitting requests. Because the client information including the instant task state is recorded in the knowledge base and the temporary results are kept in the deputy object, the task can be resumed automatically when the mobile client reconnects to the Grid gateway. In the "Searching for Information" application scenario, the results of the local search can be marked on the map, which supports two possible display formats (large map or small map). The deputy object bases the profile information of the user device by querying the context information centre to determine the suitable format returned for the mobile user (the large map is demonstrated on the "large-screen" device, while the small map is displayed on the "small-screen" device).

6.6.4 Experimental Results

As we discussed in the above section, two kinds of mobile clients are built for both the "Mobile Shopping" and the "Searching for Information" scenarios to demonstrate the value of mobile devices using the deputy middleware to invoke Grid services. The "with deputy" client refers to mobile devices invoking test services deployed through the mobile deputy middleware, while the "without deputy" client means mobile devices communicate with the service provider directly without using our system architecture. We analyze two application scenarios in terms of the system response time over the wireless network. For "with deputy" clients, the response time is the duration between mobile clients submitting the task and obtaining the task result from the Grid gateway. For "without deputy" clients, the response time is the duration of the task execution process. The "*CurrentTimeMillis*" function, provided by the standard Java API, and the "getTime()" function, provided by the JavaScript Data object, are used to measure the system response time. The unit of measure time is a millisecond.

Figure 6.10 shows the total response time for the "Mobile Shopping" scenario - specifically, it illustrates "with deputy" and "without deputy" client performance over the wireless network. The total average response time of the order service is approximately 8.2 seconds for the "with deputy" client, which is 24.1 percent shorter than the "without deputy" client (10.8 seconds).

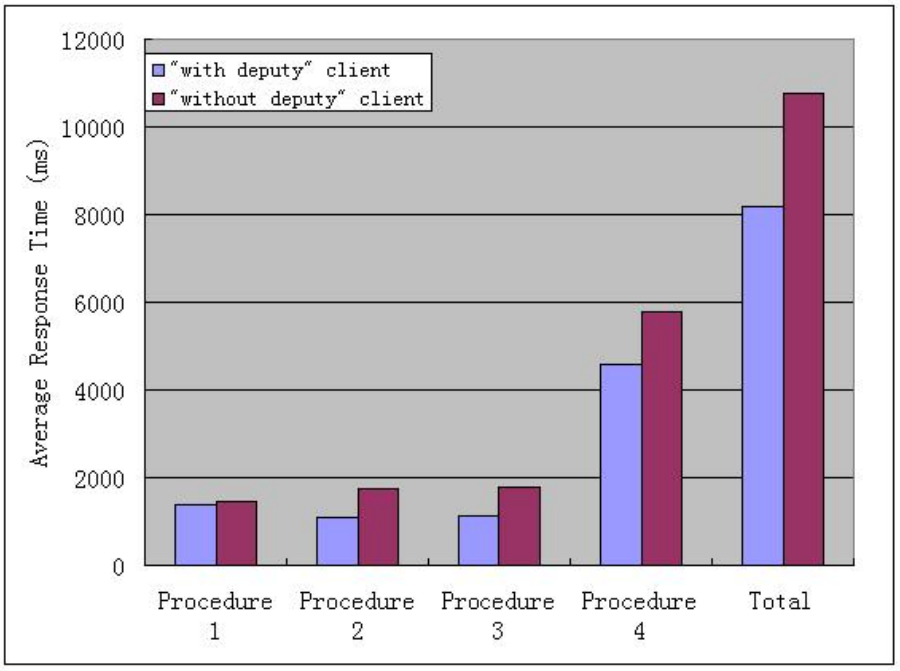

FIGURE 6.10: Average response time for the "Mobile Shopping" scenario over a wireless network.

The "with deputy" mobile client sends the request to the Grid gateway, which triggers the communication initialization between the mobile device, the Grid gateway, and the online shopping service. A mobile deputy object is created by the "Deputy Manager" module. After the connection is established, the deputy instance collects and sends required parameters for the online shopping service invocation. At the end of the service invocation, an order confirmation is returned to the deputy instance, which is then transferred back to the mobile client, indicating the task execution is over. The "without deputy" mobile client, on the other hand, creates a connection to the online shopping service and activates the service procedures directly. Any communication overheads between the service client and the service provider are required to be processed on the mobile device, thus taking additional time in every service procedure. However, in the service procedure one, the response time of two types of client is almost the same, 1.37s vs. 1.47s. This is because the mobile deputy creation and the initial interaction with the online shopping service provider are required for the "with deputy" client during the first procedure execution. The fourth procedure execution composes the main part of the total response time for both kinds of client (approximately 50 - 60 percent of the total time). This is because the procedure four is the main step in the task execution (e.g. checking the personal information, reserving items, and producing the confirmation).

For the "Searching for Information" application scenario, we still concentrate on measuring the system response time when using both the "with deputy" client and the "without deputy" client. Figure 6.11 shows the average response time of the execution

twenty times with each type of client over the wireless network. Table 6.1 gives the average response time and the standard deviation.

	"with deputy" client	**"without deputy" client**
Average response time (s)	2.31	5.68
Standard deviation (s)	0.09	0.20

TABLE 6.1: Average response time (seconds) and standard deviation for the information search scenario.

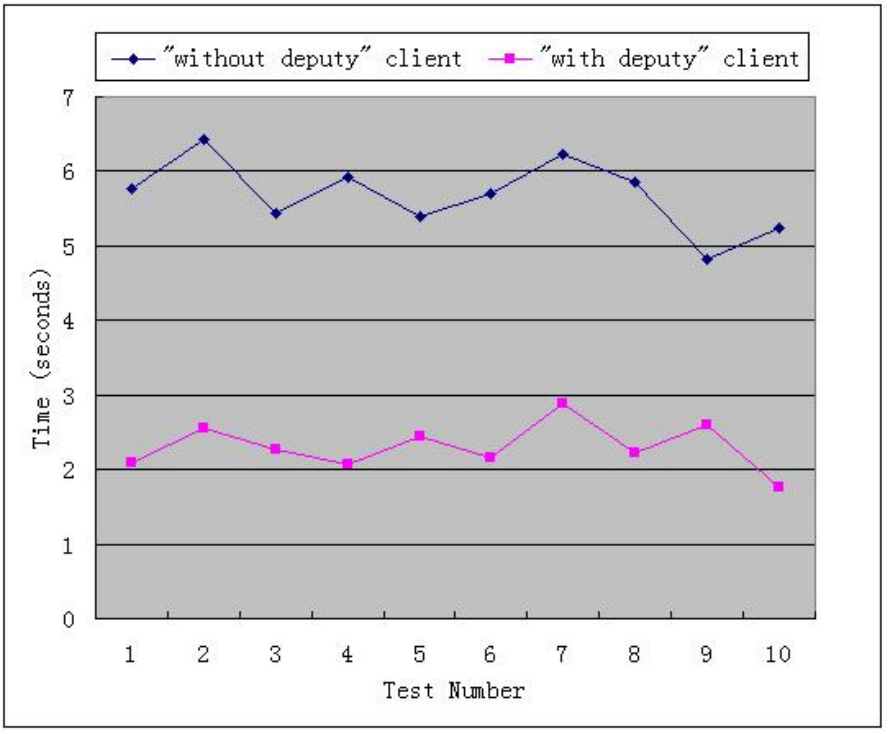

FIGURE 6.11: Response time of the first 10 executions over the wireless network.

From the Figure 6.11 and Table 6.1, we can see that the response time of the "with deputy" client is close to each other, between 1.76 seconds to 2.88 seconds, with an average of 2.31 second. The response time of the "without deputy" client falls between 4.82 seconds and 6.42 seconds, with an average of 5.68 seconds. The networking congestion is possible during the experiments because of the real working environment. The low values of the standard deviation indicate the consistency of the testing results.

Similar to the "Mobile Shopping" application scenario, the response time of the "with deputy" client in the "Searching for Information" application scenario is lower than that of the "without deputy" client (almost 60 percent). In this experiment, the network protocol between mobile devices and Grid gateways or the Google server is the typical wireless network, the maximum bandwidth of which is 11Mbps. We believe as the available bandwidth decreases, for example the connection protocol is changed to GPRS, the difference of the average response time between two types of client will increase, because

transmitting data between the mobile device and the Grid environment will consume much more time. Furthermore, in the experiment the mobile device is a laptop due to the minimal requirement of the Google AJAX API, which has hardware equipments including 1.5GHz CPU, 768 MB DDR SDRAM and 40 GB hard disk. This laptop cannot be considered as an exact example of current mobile devices because its capability is similar to that of a desktop. We believe if the Google AJAX API is extended to the current mobile device platform (e.g. smart phones, PDAs), and a mobile phone is adopted in our experiment, it will take the "without deputy" client much more time to acquire searching results from the Google server and store them into the local database. However, even in this optimistic connecting and processing circumstance, the performance of the "with deputy" client is far better than that of the direct communication system because mobile clients offload the resource-intensive operations in the task execution to the Grid gateway.

The system response time of the "Searching for Information" application scenario is made up of two parts, the time of searching the information based on the key words, and the time of storing the desired lists into the local database. For both kinds of the client, the measurement does not consider the time of the result selection because it is uncertain and depends on the user preference. In the experiment, we assume this period of time is same for both kinds of client and does not affect the comparison of the system response time.

The "Hello Grid" test application shows the possibility of mobile devices invoking Grid services through the mobile deputy middleware deployed on the Grid gateway. In the "Mobile Shopping" and "Searching for Information" application scenarios, services are invoked by two different types of mobile clients. The experimental results clearly indicate that using the mobile deputy middleware can lead to significant improvements in the system response time for both testing application scenarios. Because of the restriction of the experiment, the energy consumption of mobile devices is not measured. However, we believe using the deputy middleware can reduce the energy consumption because mobile devices are in the idle condition most of time during the task execution, offloading the processing and transmitting work to the Grid gateway. In the experiments, the capability of the laptop is better than that of current common mobile devices, and the 802.11 protocol is a high-bandwidth wireless network in the mobile computing environment. This means that the "device-gateway-Grid" architecture will provide greater potential to improving the system performance for the practical applications.

6.7 Summary

One of the important challenges of realizing the vision of building a bridge between the mobile and Grid computing field is to implement a system architecture which enables

mobile devices to access Grid services in an open, flexible and interoperable way. In our system architecture, mobile devices connect to Grid services with the Grid gateway, a physical interface which provides a relatively resource-rich and stable task execution platform for nearby mobile devices. The mobile deputy middleware (deployed on the Grid gateway), accepts and executes the task on behalf of mobile devices by invoking required Grid services. During the process of the task execution, the mobile deputy middleware needs to interact with two important components in the service-oriented Grid environment, the context information centre and the service information centre, which are built based on the context-aware framework and the Grid service matching mechanism discussed in chapter four and chapter five. The context information centre provides the context query interface to enable the deputy object to make appropriate decisions for further actions during the task execution and the service information centre supports a flexible service discovery mechanism so that the required Grid services can be located for the task execution.

Three sample application scenarios have been built for testing the functionality of the system architecture. "Hello Grid" is a simple application to check the connectivity between mobile devices and Grid services through the mobile deputy middleware. The "Mobile Shopping" scenario is developed to compare the system performance of mobile devices invoking local Grid services with and without the assist of the middleware. The "Searching for Information" scenario is built to evaluate the system performance of accessing remote Grid services. Throughout the experiments, the "Hello Grid" application shows the possibility of mobile devices invoking Grid services via their deputy objects on the Grid gateway. The results from both the "Mobile Shopping" and "Searching for Information" scenarios clearly indicate that using the mobile deputy middleware can result in significant improvements in the system response time because mobile devices offload the computing-intensive and transmitting-intensive operations of applications to a resource-rich platform.

The next chapter will discuss the system evaluation using the Petri Nets tool.

Chapter 7

System Evaluation

7.1 Introduction

The information-access and the work-assistant scenarios discussed early demonstrate that mobile devices are required to be integrated into the Grid environment to perform complicated tasks. In the previous chapter, we discussed the design and implementation of a system architecture to provide enhanced Grid access for mobile devices. This chapter describes the approach adopted to evaluate the system architecture, and the results of that evaluation.

Generally speaking, there are three methods of estimating the overall system performance, a comparison evaluation, a simulation evaluation and a user evaluation. In chapter six, we have built several sample application scenarios and the experimental results indicate that using the middleware on the Grid gateway can improve the system performance by performing complex tasks significantly faster for both computing-intensive and transmitting-intensive operations compared to the direct communication between mobile devices and Grid services. In this chapter, we will concentrate on the simulation method to evaluate the system performance.

This chapter begins with the brief introduction of the simulation approach and Petri Nets. Petri Nets are a powerful modeling tool for describing and studying a variety of systems. Before the simulation experiments, several interaction models have been built using non-Markovian Stochastic Petri Nets, which indicate possible communication mechanisms between mobile devices, Grid gateways and Grid services. The simulation goal is to estimate the system response time under different communication paradigms, which represent a set of possible rules to be followed in the data exchange and the task execution synchronization. In the mobile Grid environment, it will take users more time to access Grid services through their handheld devices than through typical desktop terminals. This is because of the extra communication overhead between mobile devices and Grid gateways, and the additional processing time required by mobile devices.

However, a reasonable system architecture should reduce the delay to an acceptable limit.

The evaluation experiments are divided into three sections. In section one, we model the communication paradigms for the static Grid client and two kinds of mobile clients, the objective is to demonstrate the performance difference between these clients. In section two, we model two possible interaction mechanisms between mobile users, Grid gateways and the service-oriented Grid environment. The evaluation purpose is to determine which mechanism should be adopted for the system architecture and whether one mechanism is more suitable for a set of application scenarios. In section three, an integrated system model is built and based on the model, the distribution of system response time and the system performance again multiple mobile clients are measured. In all of the evaluation experiments, the time of the completion of the whole operation process is used as an index of performance.

7.2 Simulation Approach

In the Oxford English Dictionary, simulation is described as [160]:

"The technique of imitating the behavior of some situation or system (economic, mechanical, etc.) by means of an analogous model, situation, or apparatus, either to gain information more conveniently or to train personnel."

In other words, simulation is the technique of imitating or representing a model of the real things, state of affairs, or processes, so that the behavior of the system under specific conditions may be studied.

Traditionally, the formal modeling of systems has been implemented through a mathematical model, which attempts to find analytical solutions enabling the prediction of the behavior of the system from a set of parameters and initial conditions. Computer simulation is often used as an adjunct to, and substitution for modeling systems when analytic solutions are not possible. There are many different types of computer simulation, but one common feature they all have is the generation of a sample of representative scenarios for a model in which a complete enumeration of all possible states would be prohibitive or impossible [161].

An integrated simulation process usually has four stages:

1. Building simulation models with provided formal methods.
2. Doing an initial test to ensure that the models are credible.
3. Executing the models in order to obtain results.

4. Analyzing the results to provide a basis to make decisions on for system modification or further development.

Figure 7.1 shows a basic structure of a simulation system [162] [163]. Entities are concrete elements either temporary or permanent which are used to represent objects in the real world. Logic relationships connect various entities together and are the key part of a simulation model, because they define the overall behavior of the model.

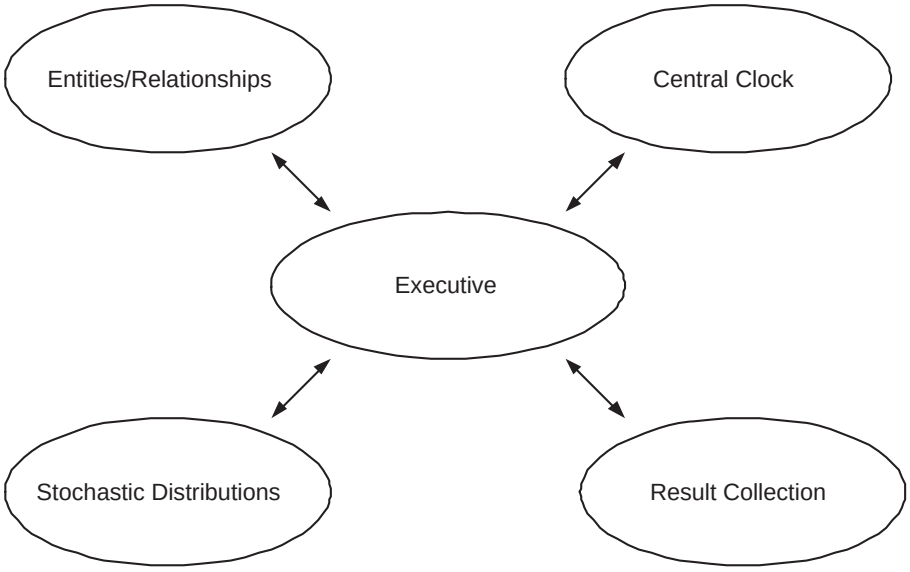

FIGURE 7.1: Structure of a Simulation System (from [162])

The simulation executive is another key part of any simulation systems, providing the dynamic and time based behavior of the model. It is responsible for determining system time advance and controls the logic relationships between entities. In the system, time is kept track of by a central clock. Although the executive and the system clock are critical for a simulation system, they are straightforward to implement and have relatively simple behavior.

The random number generator and the result collection are another two important parts of a simulation system. The random number generator is used to approximate the typical stochastic behavior of the real world. The result collection part offers the user a method of utilizing the simulation tool to analyze the simulated system. Simulation tools can typically display raw results and provide straightforward graphs and charts.

In the field of computer science, simulation is an important tool in the development cycle of large-scale distributed systems, because selected behavior of the system can be reproduced. The simulation can be applied for different specific system development purposes: during the initial system design stage, the system specification can be simulated in order to demonstrate its suitability to accomplish the planned tasks; during the system function testing stage, the environment within which the real system will operate can be simulated so that the real system can be tested by a set of stimulation conditions

before its deployment; during the system performance testing stage, the performance evaluation of the real system can be achieved by simulating a variety of working environment configurations and workloads which may not be available before the actual system [164].

The common simulation objectives include [164] [163]:

- Functional estimation: evaluation of the system specification.
- Performance estimation: timed evaluation of the system specification.
- Real system functional testing: functional simulation of the system working environment.
- Partial system simulation: incremental testing of critical parts of the system.
- System workload simulation: performance stressing of the system.

Discrete event simulation is a prevalent and powerful computing technique for understanding the behavior of systems [165]. As distinct from continuous simulation, in which time is controlled by continuous variables expressed as differential equations [166], the operation of a discrete event system is represented as a chronological sequence of events. Each event occurs at an instant in time and marks a change of state in the system [167].

As a complement of a general simulation system, a discrete event simulation system includes a new component - the event list, which maintains at least one list of simulation events [162]. Normally, an event has a start time, the code that describes the performance of the event itself, and possibly an end time. Events are scheduled dynamically as the simulation proceeds.

In the discrete event simulation, a set of system states is specified for the system, and the evolution of the system can be viewed as a sequence of the following form:

$$\prec s_0, (e_0, t_0), s_1, (e_1, t_1), s_2, ... \succ$$

where the s_i's are system states, the e_i's are system events, and the t_i's are nonnegative numbers representing event occurrence time. At the beginning, the system starts at time0 in the state s_0; then the event e_0 occurred at time t_0 taking the system to the state s_1; then the event e_1 occurred at time t_1 taking the system to the state s_2; and so on. The occurrence of the event is assumed to take zero time.

The discrete event simulation of distributed systems has long been studied and used. In the past, a number of languages have been proposed, with the special purpose of writing discrete event simulation programs for the timed simulation of systems. These languages allow rapid development compared with general-purpose programing languages (e.g.

C++, Fortran) as well as providing primitives and library functions supporting the development of simulators. However, a prospective user still has to spend considerable time turning the conceptual model of the system into a correct program. The structure of the real system is also not always reflected in the simulation program.

Providing a user-friendly interface is an important trend of modern simulation development. A number of simulation tools have been implemented for the simulation and analysis of systems with the high-level model description techniques - examples are GreatSPN [168], ProModel [169] and so on. These tools save the user the task of writing and debugging a simulation program but are more limited and restricted to specific applications.

7.3 Petri Nets

The concept of Petri Nets was introduced in Carl Adam Petri's dissertation in 1962 [170]. It is a powerful modeling tool used to represent and study discrete distributed systems. As a system modeling language, Petri Nets describe the structure of a distributed system as a place/transition network with annotations. As a graphical tool, Petri Nets are similar to flow charts and block diagrams and can provide visual communication support. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models controlling various behaviors of the system [171].

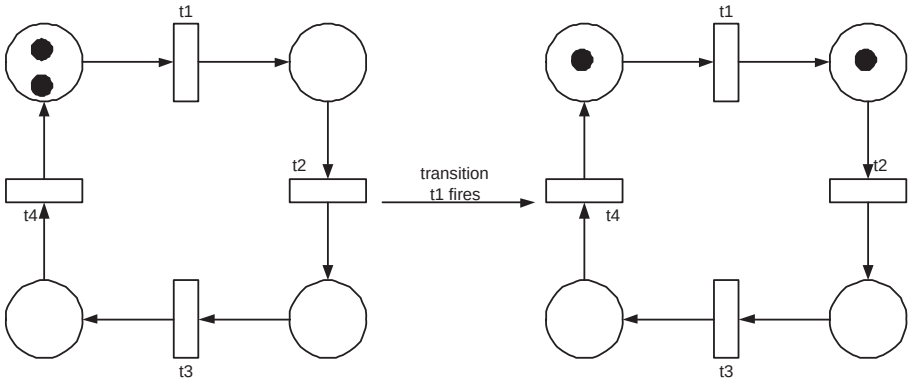

FIGURE 7.2: A Petri Net before and after the firing of transition t1.

Figure 7.2 shows the example of a basic Petri Net. A Petri Net consists of places, transitions, and arcs. Places usually contain tokens, the number of which in each place indicates the possible states of the modeled system. Transitions model activities which can occur in the system and thus change the state of the system. Arcs connect places and transitions. Input arcs start at a place and end at a transition, while output arcs are from a transition to a place. Transitions can be allowed to fire if there are enough tokens available in their input places, which suggests that all of preconditions for activities are

satisfied. When the transition fires, tokens from its input places are removed and its output places are added by a number of tokens.

As a formal definition, a Petri Net is a finite sequence which includes five kinds of elements (S, T, F, M_0, W):

- S is a set of places.
- T is a set of transitions.
- F is a set of arcs (flow relations).
- M_0 is an initial marking, which indicates the initial state of a system (the number of in each place).
- W is a set of arc weights, suggesting how many token are removed from a place by a transition, or how many tokens are produced by a transition to add its output places.

Basic Petri Nets have many extensions. In order to study the performance and the dependability issues of systems, a timing concept has to be defined into the model. In the usual condition, a firing delay is associated with each transition to specify the waiting time that is required to enable the transition when all of input places of the transition contain a token.

Stochastic Petri Net (SPN) [172] [173] is a type of timed Petri Nets, which enhances the system modeling power by associating exponentially distributed random firing times (waiting times) with the transitions. When using an exponential random distribution to mark the time attribute of Petri Nets, the reachability graph of Petri Nets can be translated into a Markov chain.

However, the exponential assumption was gradually becoming the restriction in the practical application of SPN models. It is frequently necessary to model transitions whose occurring time is not exponentially distributed in a wide range of conditions. The existence of deterministic or other non-exponentially distributed event times leads to stochastic Petri Nets that are non-Markovian attributes [174]. Generalized Stochastic Petri Nets (GSPN) [175], Stochastic Reward Nets (SRN) [176], and Markov Regenerative Stochastic Petri Nets (MRSPN) [177] are the main extensions of Stochastic Petri Nets, which upgrade both the theoretical background and the potential areas of applications. For example, GSPN allows the transitions of the underlying PN to be immediate as well as to be a firing rate (Figure 7.3).

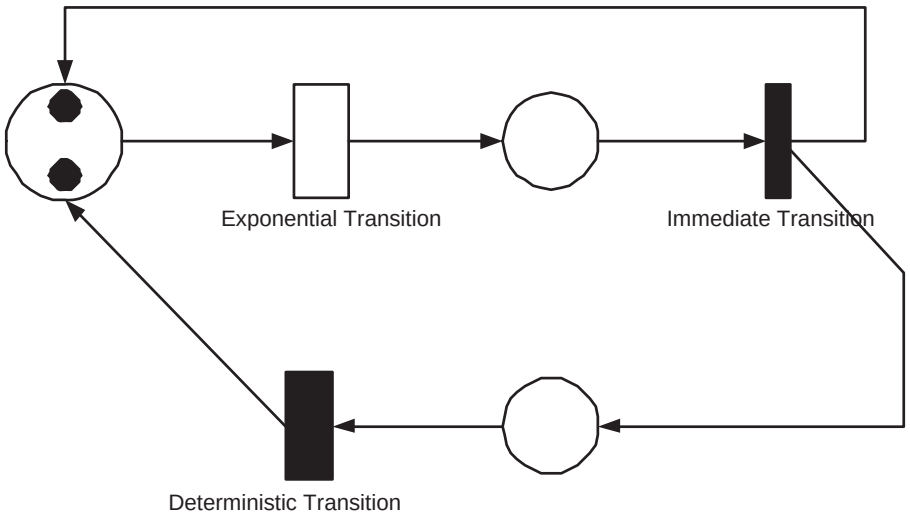

FIGURE 7.3: A Example of Genelialized Stochastic Petri Net.

7.4 System Evaluation

As we have discussed in the above section, Petri Nets represent a useful and popular tool for performance, dependability and performability analysis of complex systems. Although the usual definition of Stochastic Petri Nets is based on the assumption that all the firing times of transitions are exponentially distributed, many extensions which consider nonexponentially distributed events to the basic model have appeared to enhance the modeling power. In this section, in order to evaluate the system performance, we use Stochastic Petri Nets to model various communication paradigms between mobile devices and the Grid environment.

Much amount of effort has been devoted to define and implement automatic solution tools for analyzing PN models. WebSPN [178] is a modeling tool for the analysis of non-Markovian stochastic Petri nets. It provides a friendly interface to allow users to specify the Stochastic Petri Nets model to be simulated by simply drawing it on the screen. An analysis engine is also integrated with the interface which can be used to analyse the net and compute the measures required by the user. We present and analyse different interaction models using the WebSPN tool in this section.

The system evaluation work can be divided into three sections as set out below.

7.4.1 Static Grid Client vs. Mobile Grid Client

In this evaluation section, two kinds of interaction models are built: the static Grid client and the mobile Grid client. For the mobile Grid client scenario, we model both the "with deputy" and "without deputy" communication paradigms between mobile devices and Grid services (corresponding to two kinds of mobile clients built in the test application

scenarios at chapter six). The objective is to demonstrate the performance difference between the static Grid client, the "with deputy" mobile Grid client, and the "without deputy" mobile Grid client. In this evaluation section, the time taken by both the Grid invocation and the Grid service discovery is kept constant because we assumes their execution time is not related with the client type. Also, in order to simplify the problem analysis, the task to be solved is assumed to require only one Grid service invocation. A key purpose in the evaluation experiments is to estimate the interaction delay of the "with deputy" mobile client model, which is caused by the communication between the mobile device and the deputy middleware as well as the additional processing to invoke Grid services. We use the completion time of the whole process as an index of the system performance and compare the delay between mobile clients with and without using the deputy middleware. In any case, the delay should be within acceptable limits compared to the static Grid client. Otherwise, the proposed system architecture would not be a reasonable candidate for the mobile client accessing Grid services.

7.4.1.1 Models of Petri Nets

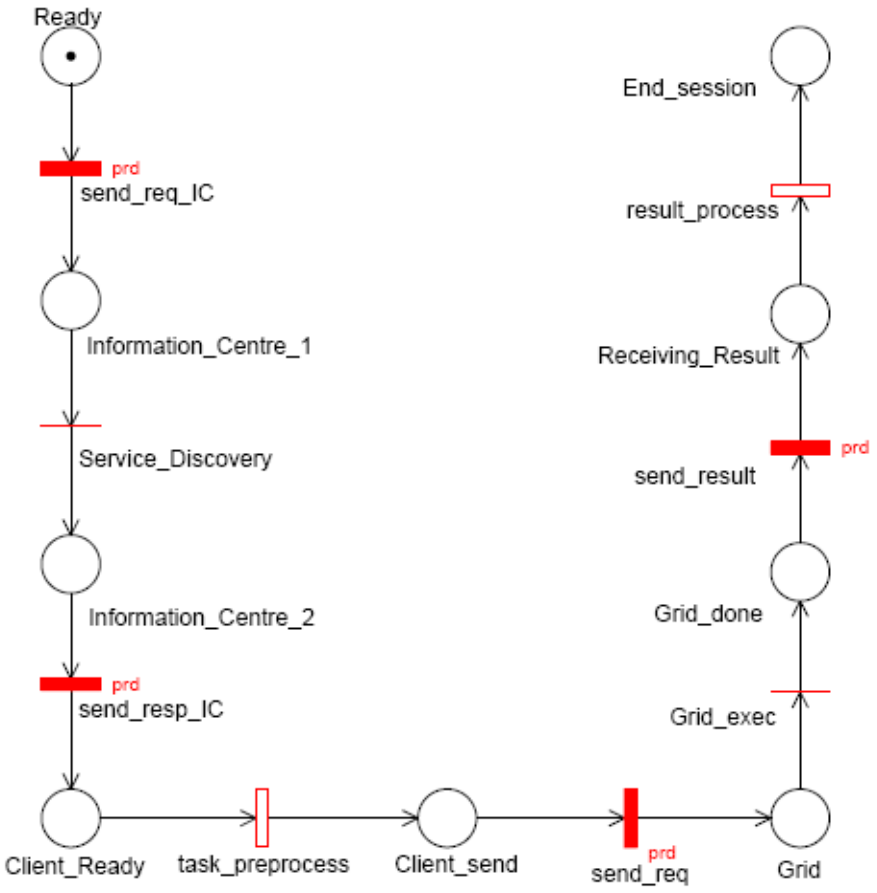

FIGURE 7.4: The Static Grid Client Model.

Figure 7.4, Figure 7.5, and Figure 7.6 show the static Grid client model, the "with deputy" mobile Grid client, and the direct access mobile Grid client.

For the static Grid client model, at the beginning, the place *Ready* contains a token, indicating that a static Grid client is ready to access Grid services. The transition *send_req_IC* models the time required for communicating with the service information centre and discovering desired Grid services. The information centre sends the response back to the static client after the immediate transition *service_discovery*, modeled with the *send_resp_IC* transition. The place *Client_ready* represents that the static client has located a Grid services from the information centre and is ready to submit a request for the next step. Before submitting the Grid service request, the client needs to preprocess its task, such as decomposing tasks, requesting input parameters and so on, which is modeled by the *task_preprocess* transition. The place *Client_send* means the client is ready to submit the Grid service request. The time of submitting the service request is modeled through the transition *send_req*. When it fires, the token enters the place *Grid*, which indicates that the Grid environment has accepted the service request from the client. The transition *Grid_exec* represents the time required for invoking Grid services. At this stage, it is considered as an immediate transition. Once the Grid returns the result (place *Grid_done*), the static client collects the result (transition *send_result*) and processes the results (transition *result_process*) for the task completion. The token entering the place *End_session* means one Grid request is over.

The work flow of the "with deputy" mobile Grid client is different to that of the static Grid client. The place *Ready* contains a token at the beginning. Transition *send_req* models the time of submitting a task request to the Grid gateway. After the Grid gateway initializes a new deputy object for this mobile client (transition *deputy_init*), the mobile Grid client can send the task to the its deputy object, which is modeled by the transition *send_task*. The deputy object needs to preprocess the submitted task and locate required Grid services from the service information centre, before invoking Grid services. Because of the central range of Grid service registry, the service information centre middleware and the mobile deputy middleware are deployed on different platforms, which can interact with each other through a high-speed network. These series of operations are modeled through the transition *task_preprocessing*, *send_req_IC*, *service_discovery* and *send_resp_IC*. When the transition *send_service_req* fires, it means Grid services are invoked to accomplish the user task. The transition *Grid_exec* remains an immediate transition. Once the Grid returns the result (place *Grid_done*), the deputy object collects the result (transition *result_retrieval*), completes the task using the result (transition *result_process*), and sends the final result back to the mobile client (transition *send_result*). The token entering the place *End_session* means that an integrated operation sequence is over.

The work flow of the "without deputy" mobile Grid client model is same with that of the static Grid client model. The difference is the type of the client: in the "without

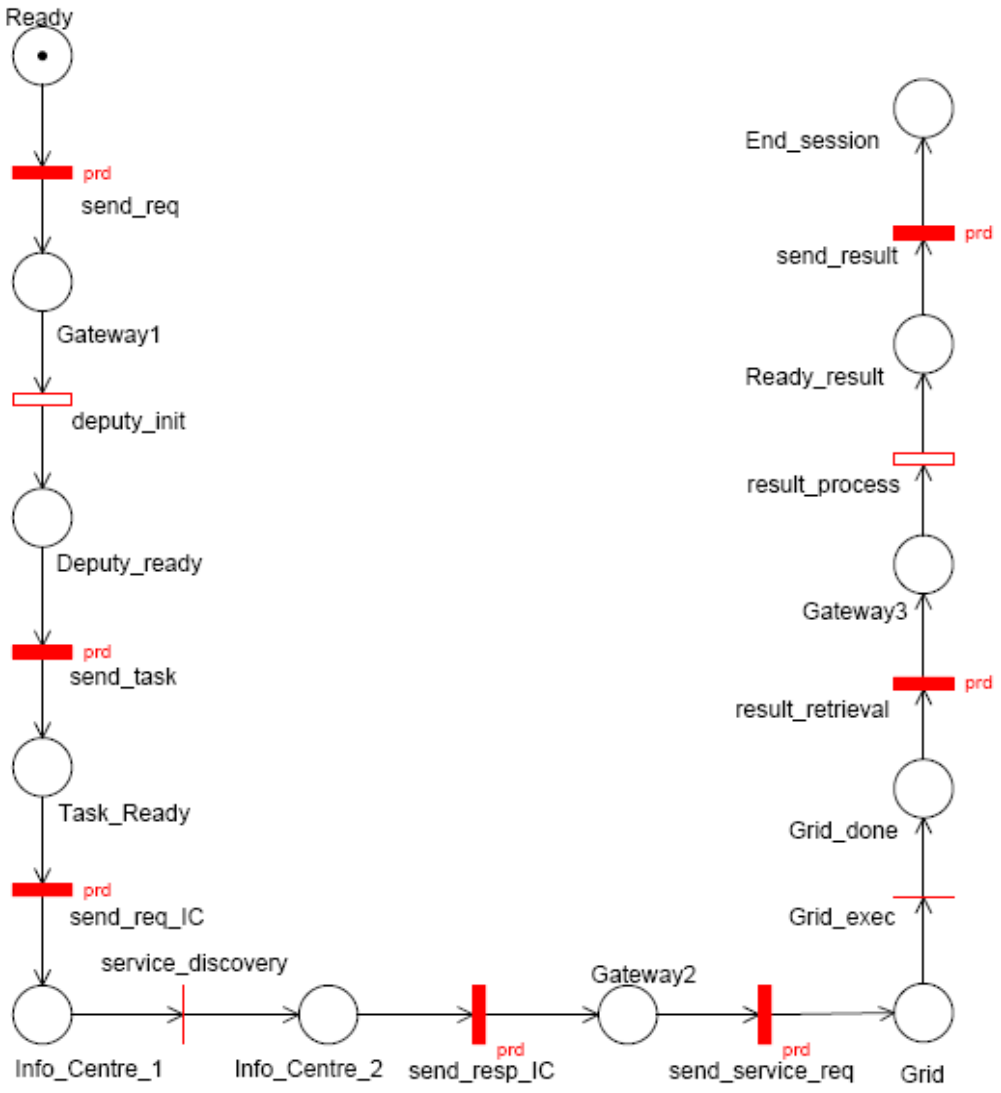

FIGURE 7.5: The "with deputy" Mobile Grid Client Model.

deputy" mobile Grid client model, mobile devices are user interfaces to the Grid; while in the static Grid client model, user terminals are the traditional Grid clients, such as desktops.

7.4.1.2 Parameters used in the Petri Net Model

In order to evaluate the above Petri Net models, we have adopted the following numerical parameters, some of which are consistent with the ones used in [179]:

- Size of a request: The static Grid client sends two types of request. The service discovery request is submitted to the service information centre (D_{req_IC}), and

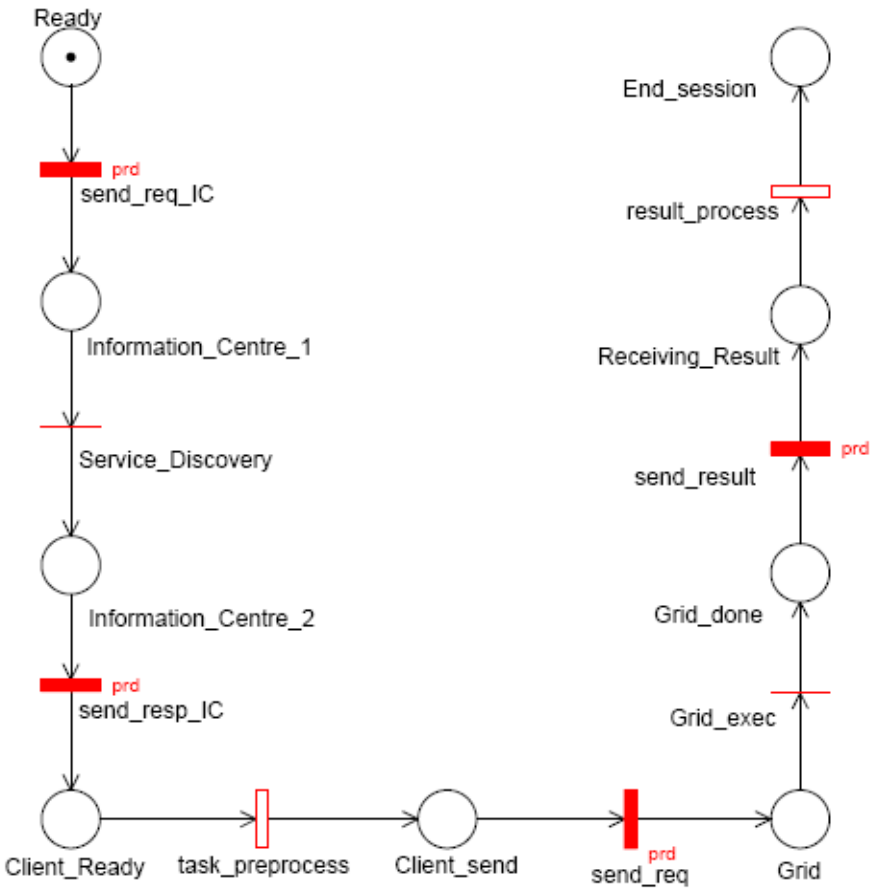

FIGURE 7.6: The "without deputy" Mobile Grid Client Model.

the other is the Grid request (D_{req}). In the mobile Grid client models, besides the initial request and the service discovery request, the mobile device needs to submit its task to the Grid gateway . The size of the service discovery request and the initial request can be assumed to be small and constant, while the size of submitting an encapsulated task to the deputy object is larger because it may include codes for the task description and execution.

- Time for preprocessing the task ($\lambda_{task\_preprocess}$): This processing can take place either in the Grid gateway ("with deputy" mobile client model) or in the service client itself ("without deputy" mobile client and static Grid client model). The exact value for this parameter and its distribution will greatly depend on the type of the application and hardware characteristics of the device equipped with. For this reason, the distribution of preprocessing tasks is fixed as exponential.

- Time for processing the result ($\lambda_{result\_process}$): Similar to the time for preprocessing tasks, its exact value greatly depends on the type of the application (type and size of the result) and the device hardware capabilities. The distribution is again fixed as exponential.

- Size of the result: There are two types of result in above models. One is the result directly from the Grid service invocation (D_{result}), and the other is the final result existing in the "with deputy" mobile Grid client model ($D_{finalresult}$). The final result from the Grid gateway to the mobile client depends on the type of the task submitted by the client, which could be a small numerical value or a larger file. Hence, a minimum and a maximum value are considered. The result directly from the Grid service invocation usually has a larger size.

- Throughput of the communication network: Two kinds of transmission rates are presented in the models. The wireless network has been assumed to have lower throughput (TH_{low}), while the Grid environment and the static machine (the Grid gateway and the static Grid terminal) are assumed to be connected with a high-speed network (TH_{high}).

The above parameters are used to calculate the values assigned to the transitions in the Petri Net models, according to the formulas depicted in Table 7.1.

7.4.1.3 Numerical Evaluation of the Model

The numerical values assigned to above parameters for this stage of evaluation are indicated in Table 7.2.

The firing rates associated with the transition of pre-processing the task, initializing a deputy object and processing the result from the Grid have been fixed to λ=5req/s (1req/s for the mobile device).This factor is not only application dependent but also dependent on the computation power of the machine that contains the middleware. The high-speed network connection between the Grid service provider and the Grid gateway or the static Grid client has been assigned a value of TH_{high} = 10Mbps.

Based on these assigned values, the Petri Net models are evaluated by using the WebSPN tool, by which both exponential and non-exponential firing rates can be assigned to transitions.

Figure 7.7 shows a diagram of the average system response time based on the variation of TH_{low} for both the static Grid client model and two mobile Grid client models. The results of this experiment indicate the following points:

- The average system response time of the mobile Grid client model is greatly affected by the wireless network throughput between mobile devices and other external resources (Grid gateway or Grid services). For the "with deputy" mobile Grid client model, the delay caused by the communication between mobile devices and Grid gateways as well as the additional processing required to invoke Grid

Static Grid Client Model		
Transition Name	Type	Expression
send_req_IC	Deterministic	D_{req_IC}/TH_{high}
send_resp_IC	Deterministic	D_{resp_IC}/TH_{high}
task_processing	Exponential	$\lambda_{task\_preprocess}$
send_req	Deterministic	D_{req}/TH_{high}
send_result	Deterministic	D_{result}/TH_{high}
result_process	Exponential	$\lambda_{result\_process}$
Mobile Client (with "deputy")		
Transition Name	Type	Expression
send_req	Deterministic	D_{req}/TH_{low}
deputy_init	Exponential	λ_{deputy_init}
send_task	Deterministic	D_{task}/TH_{low}
send_req_IC	Deterministic	D_{req_IC}/TH_{high}
send_resp_IC	Deterministic	D_{resp_IC}/TH_{high}
service_req	Deterministic	D_{req}/TH_{high}
task_processing	Exponential	$\lambda_{task\_preprocess}$
send_result	Deterministic	D_{result}/TH_{high}
result_process	Exponential	$\lambda_{result\_process}$
send_finalresult	Uniform	$[D_{min}, D_{max}]/TH_{low}$
Mobile Client (without "deputy")		
Transition Name	Type	Expression
send_req_IC	Deterministic	D_{req_IC}/TH_{low}
send_resp_IC	Deterministic	D_{resp_IC}/TH_{low}
task_processing	Exponential	$\lambda_{task\_preprocess\_MD}$
send_req	Deterministic	D_{req}/TH_{low}
send_result	Deterministic	D_{result}/TH_{low}
result_process	Exponential	$\lambda_{result\_process\_MD}$

TABLE 7.1: Parameters used in the Petri Net Model.

services is within acceptable limits. The delay is no more than 2.5s assuming that the TH_{low} is equal to and larger than 50kbps (and no more than 10s when the throughput is 10kbps).

- The increased speed of the wireless network causes improvements in the average system response time. Obviously, the result when the speed is higher than 200Kbit/sec are close. This is because in such conditions, the task processing time

Parameter	Description	Value
D_{req_IC}, D_{resp_IC}	Dimension of req and resp to Info Centre	1KB
D_{req}	Dimension of client request	1KB
$\lambda_{task\_preprocess}$	Task preprocessing rate	5req/s
$\lambda_{task\_preprocess\_MD}$	Task preprocessing rate of mobile devices	1req/s
D_{result}	Dimension of result	50KB
$\lambda_{result\_process}$	Result processing rate	5req/s
$\lambda_{result\_process\_MD}$	Result processing rate of mobile devices	1req/s
D_{task}	Dimension of task request	5KB
λ_{deputy_init}	Rate of initializing a deputy object	5req/s
D_{min}, D_{max}	Dimension of the final result	1KB, 10KB
TH_{high}	throughput of high speed network	10Mbps

TABLE 7.2: Numerical Values assigned for this stage of evaluation.

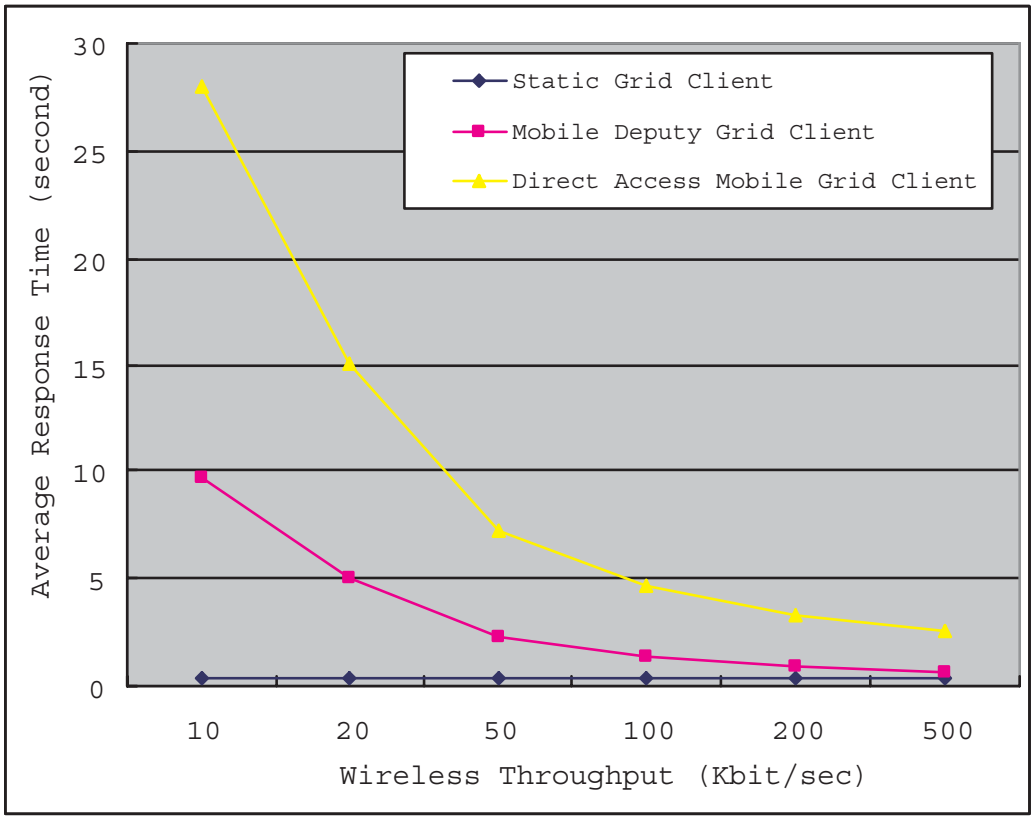

FIGURE 7.7: Average System Response Time vs. TH_low.

of the Grid gateway and the device is predominant and much greater than the communication delay between mobile clients and external resources. However, as the technological advance, the bandwidth for mobile devices continues to increase

so that the processing time of the Grid gateway and the device are gradually becomming the most important factor of affecting the overall system performance.

- In the "without deputy" mobile Grid client model, the average response time is much larger than that of other two models. Mobile devices can not be direct Grid clients unless they communicate to the Grid environment with a high-speed wireless network (more than 200Kbps) and under these conditions their processing capability increases remarkably (high $\lambda_{task\_preprocess}$ and $\lambda_{result\_process\_MD}$). The reason for the performance difference between the two mobile Grid client models is that the Grid gateway handles most of the preprocessing work and the communication overhead with the Grid service provider. Mobile devices under the "with deputy" model need to organize the service/task request and be ready for result visualization.

- In last chapter, three test applications have been built to investigate the performance difference of two kinds of mobile clients. However, due to the experimental restriction, the system response time was only measured when the connection between mobile devices and the Grid environment is a high-bandwidth wireless network (WiFi). We expect that the performance difference will increase as the connection changes to a low-bandwidth network (e.g. GPRS). The simulation results confirm our estimation. When the network throughput goes down, the "with deputy" mobile client provides a better system performance compared to the "without deputy" mobile client. Furthermore, both real application tests and simulation evaluations show that the "with deputy" client has a potential to improve the system performance. This consistency proves that our Petri Nets models built for the communication paradigm between mobile devices, Grid gateways, and Grid services are credible.

Figure 7.7 demonstrates that the "device-gateway-Grid" system architecture is a reasonable candidate architecture for mobile clients accessing Grid services because the delay caused by the communication between mobile devices and Grid gateways as well as the additional processing to access Grid services is within accepted limits. However, the delay is still more than three seconds when the throughput of the wireless network is low (10kbps or 20kbps). We believe reducing the size of the task and the final result can improve the system performance. Figure 7.8 shows the average system response time based on the throughput of the wireless network when the size of the task is reduced to 2KB and the maximum size of the final result is reduced to 5KB. The system performance improves when the task and result sizes reduce.

By comparing the average system response time of both the static Grid client model and two mobile Grid client models, we can conclude that mobile devices are not ideal clients for accessing Grid services, because of poor computation and connection capabilities. The "device-gateway-Grid" system architecture is a reasonable candidate for enabling

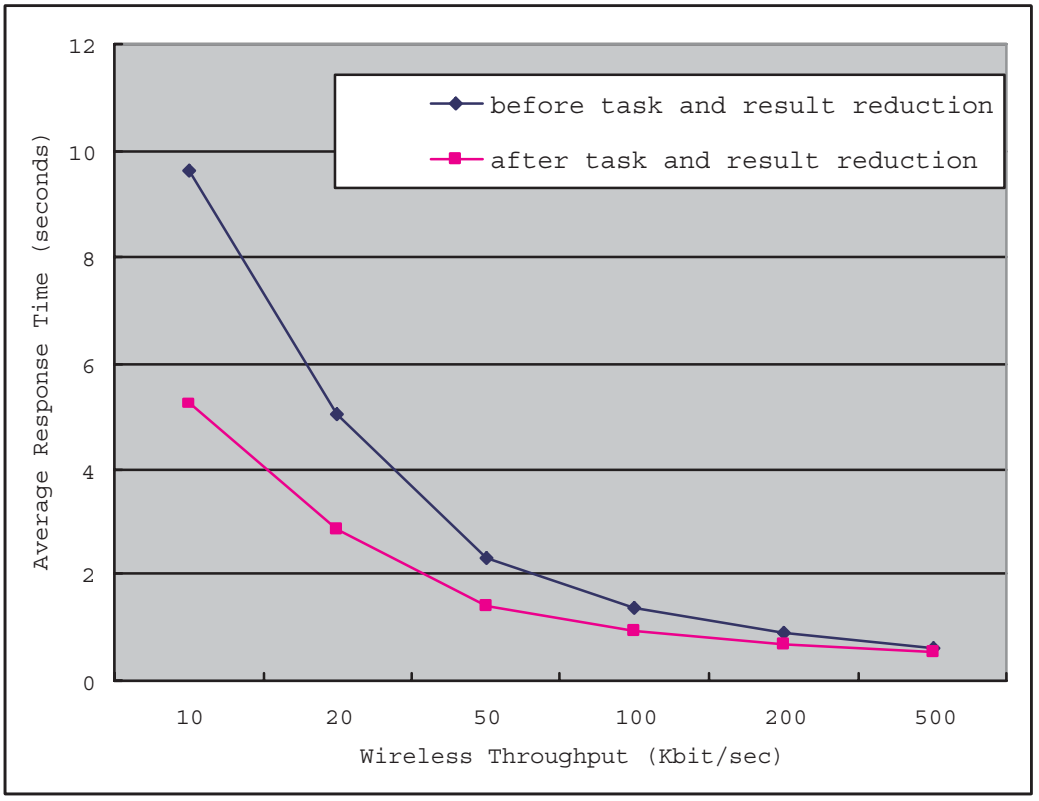

FIGURE 7.8: Average System Response Time vs. TH_low.

mobile devices to invoke Grid services because mobile devices can offload their complicated tasks to their deputy objects and deputy objects handle most of preprocessing work and the communication overhead with the Grid service providers in a relative resource-rich platform. Even in a not-ideal case (where the throughput of the wireless network is no more than 50kbps), the delay is still within an acceptable limit (for no real-time application scenarios), less than 2.5 seconds. These experiments have also demonstrated that reducing the size of the task and the results is an approach to improving the system performance greatly when the wireless network bandwidth is low.

7.4.2 Procedure-oriented vs. Task-oriented

In the above three Petri Nets models, a task was also assumed to require one Grid service invocation only, because the purpose was to estimate the delay caused by the communication overhead and the additional processing. The time taken by Grid processing was also kept constant, an immediate transition. However, both of these constrains are not realistic for real applications. In order to evaluate the system performance under these practical considerations, the basic Petri Net models discussed in the previous section have to be extended.

There are two possible mechanisms to execute a complex tasks which includes several procedures of Grid service invocation:

- One opinion is first to decompose the complex task into several procedures, and then send the procedure to the Grid gateway one by one. Each procedure includes at most one request for invocation of the Grid service. After the result of the executing procedure is returned, the client sends the next procedure to the gateway. This mechanism is termed the "procedure-oriented" interaction strategy.
- The other opinion is to send the whole task to the Grid gateway at once. After submitting the task, the client will be in a idle state and waiting for the task result. This mechanism is named the "task-oriented" interaction strategy.

Both of these interaction strategies have their advantages and disadvantages. For the "procedure-oriented" approach, there is no need to send a large amount of task codes to the gateway over an unreliable and low-bandwidth wireless network. However, it has to monitor each procedure execution status and submit the next procedure at the appropriate time. For the "task-oriented" strategy, the time and resources of mobile devices are saved so that users can process other jobs on their portable devices after submitting their task. However, sometimes they have to accept a failure when uploading large size files over their poor connection.

Hence, in this evaluation section, the purpose is to solve the following problems:

- Which interaction strategy should be used to implement the system architecture?
- Is it possible that one interaction strategy is more suitable for a set of application scenarios than the other?

7.4.2.1 Models of Petri Nets

Two models are built for both the "procedure-oriented" and "task-oriented" interaction strategies. To simplify the analysis, the preparatory operation (e.g. locating services and initializing the deputy object) are not included in the models. The start point of the models is the state that mobile clients are ready to send procedures or tasks.

Figure 7.9 shows the "procedure-oriented" Petri nets model. In the beginning state of the model, it is assumed that a task has already been decomposed into five procedures and the client is ready to submit these procedures to the Grid gateway. As long as the transition *send_procedure* is started, the number of tokens in the place *Client_ready* decreases. Transition *send_procedure* models the time required for sending a procedure to the gateway over the wireless network, which is enabled if a token is also in the place *Ready*.

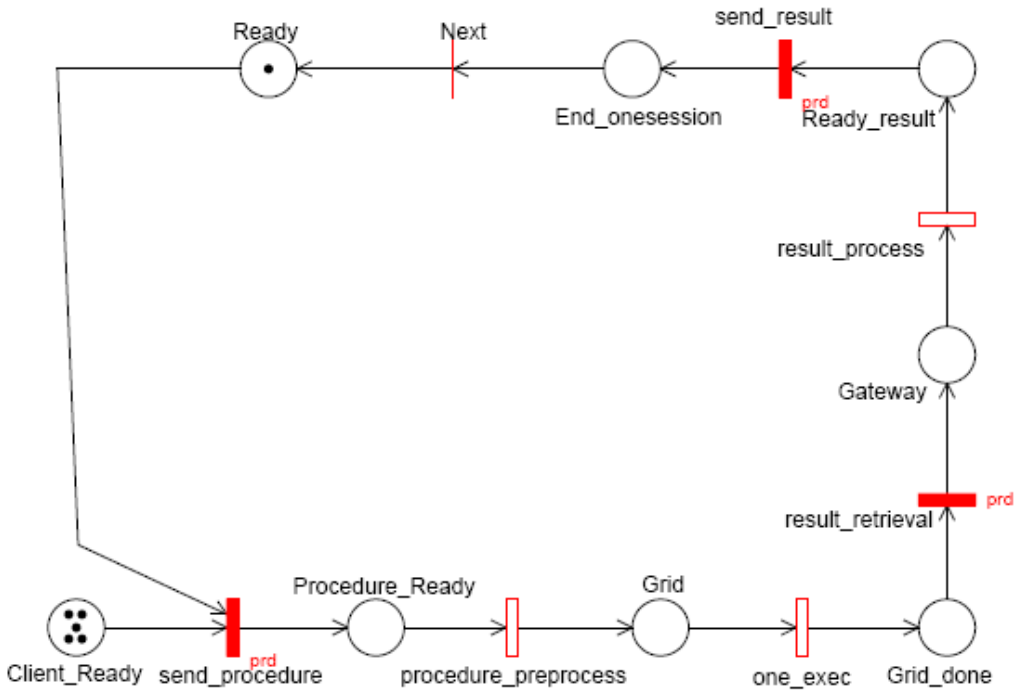

FIGURE 7.9: The "Procedure-oriented" Model when N=5.

After the Grid gateway receives a procedure, it preprocesses the procedure (transition *procedure_preprocess*) and submits the Grid service request. The time of invoking Grid services is modeled by transition *one_exec*, and once it fires, one token enters place *Grid_done*, which indicates the result is returned from the Grid service. The Grid gateway will retrieve the result (transition *result_retrieval*), process the result (transition *result_process*), and send the result of this procedure back to the client (transition*send_result*). Place *End_onesession* means the mobile client receives the reply of the submitted procedure.

Once one operation is over, the immediate transition *Next* fires and the token enters place *Ready*. If there is at least one token in place *Client_Ready*, transition *send_procedure* is enabled again so that a new operation sequence starts. The model has an absorbing state, in which the number of place *Client_Ready* is zero and the number of place *Ready* is one. Hence, the average system operation time of the model can be measured by evaluating the mean time to the absorption state.

Figure 7.10 shows the "task-oriented" Petri nets model. This is different to the "procedure-oriented" model because mobile clients offload the whole task onto their deputy objects. Deputy objects will decompose tasks into several procedures and execute them one by one. Mobile clients only need to start the session for the first time, and the rest depends on the deputy object in the Grid gateway. This may lead to two advantages: the communication between mobile clients and the Grid gateway is slow and unreliable, but

after submitting the task, the communication will no longer be the performance bottleneck of the whole system; the connection between the client and the gateway is not required to be continuous, and clients may even turn to other work at the time of the task execution. However, this model increase the workload of the Grid gateway, such as monitoring the task execution, and transferring the task and intermediate results to another Grid service providers.

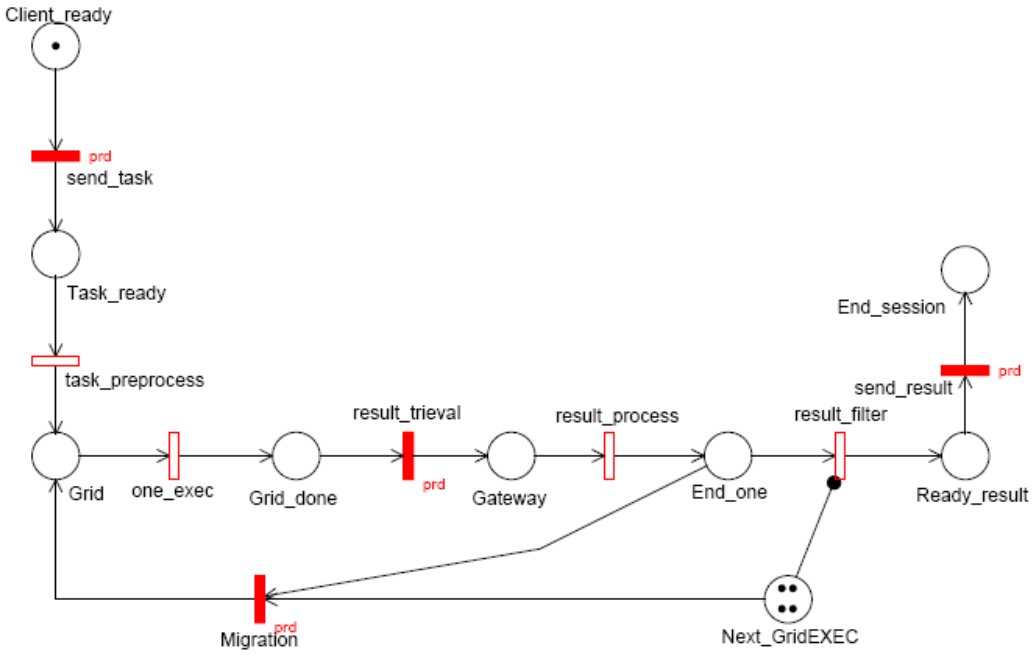

FIGURE 7.10: The "Task-oriented" Model when N=5.

In the model, the task is sent to the Grid gateway at the beginning, which is modeled by transition *send_task*. The Grid gateway will first preprocess the uploaded task (transition *task_preprocessing*), and then start the Grid service invocation. After the Grid service invocation is finished, the gateway retrieves the result (transition *result_retrieval*) and processes the intermediate result (transition *result_process*), which indicates one step of the task is completed (place *End_one*). At this stage, two possibilities can happen: either the task execution is completed, which enables transition *result_process*, or the Grid gateway submits another Grid service request. As long as there are tokens in the place *Next_GridExec*, transition *result_process* will be disabled and transition *data_code_transfer* will be fired to continue executing the task. The initial number of the token in place *Next_GridExec* is N-1 (N is the number of the procedures), which indicates that N-1 procedures remain to be executed.

7.4.2.2 Parameters used in the Petri Net Models

A number of numerical parameters in both the "procedure-oriented" and "task-oriented" work flows are required to be defined in order to evaluate constructed Petri Net models. Some of these parameters have already been discussed in the evaluation section of the "static Grid client vs. mobile Grid client".

- Size of a procedure ($D_{procedure}$): At the beginning of the "procedure-oriented" model, the mobile client needs to send a procedure to the Grid gateway. The size of the procedure should be larger than the size of a common request (D_{req} in the last section) and smaller than the size of the whole task, because it contains codes and input data for one service invocation.

- Size of a task (D_{task}): The whole task is required to be submitted to the deputy object on the Grid gateway at the beginning of the "task-oriented" Petri Net model. It is assumed that the task size is much larger than that of the procedure because one complex task can be decomposed into several executing procedures. The transfer of the task from the mobile client to the Grid gateway may become the bottleneck of the system. It is therefore an alternative opinion that the task is not stored at the client but a remote machine, so that the Grid gateway can collect it when receiving the "executing" command from the mobile client.

- Time for preprocessing the task/procedure ($\lambda_{task\_preprocess}/\lambda_{procedure\_preprocess}$) and processing the result from the Grid ($\lambda_{result\_process}$): These parameters have been discussed in the last section. Because their values greatly depend on the type of the application and the characteristics of the machine hardware equipments, their distribution is assumed exponential.

- Size of the result (D_{result}, $D_{finalresult}$): The same as the parameters used in the above section.

- Time for Grid execution (λ_{one_exec}): Transition *one_exec* models the time of Grid gateway invoking Grid services one time. This is similar to preprocessing the task or processing the result, so that the exact value greatly depends on the type of the invoked services and the Grid resources to be accessed. Its distribution is therefore assumed to be an exponential distribution.

- Size of code and data transfer (D_{trans}): When one procedure execution is over, the Grid gateway will submit another Grid service request and transfer the executing object to the Grid environment again as well as the results obtained from previous invocation steps. The size of the transferred code and data could be same as the size of a procedure, or larger. Hence, a minimum and a maximum value are adopted here and the following expression is used to calculate them.

$$D_{trans\_min} = D_{procedure}$$

$$D_{trans\_max} = D_{procedure} + (N-1) * D_{result}$$

The above discussed parameters are used to calculate the values assigned to the transitions in Petri Net models, based on the formulas depicted in Table 7.3.

"Procedure-oriented" Model		
Transition Name	Type	Expression
send_procedure	Deterministic	$D_{procedure}/TH_{low}$
procedure_preprocess	Exponential	$\lambda_{procedure\_preprocess}$
one_exec	Exponential	λ_{one_exec}
result_retrieval	Deterministic	D_{result}/TH_{high}
result_process	Exponential	$\lambda_{result\_process}$
send_result	Deterministic	D_{result}/TH_{low}
"Task-oriented" Model		
Transition Name	Type	Expression
send_task	Deterministic	D_{task}/TH_{low}
task_preprocess	Exponential	$\lambda_{task\_preprocess}$
one_exec	Exponential	λ_{one_exec}
result_process	Exponential	$\lambda_{result\_process}$
data&code transfer	Deterministic	D_{trans}/TH_{high}
result_process	Exponential	$\lambda_{result\_process}$
send_finalresult	Uniform	$[D_{min}, D_{max}]/TH_{low}$

TABLE 7.3: Parameters used in the Petri Net Model.

7.4.2.3 Numerical Evaluation of the Model

The numerical values assigned to above parameters for this section of evaluation are indicated in Table 7.4.

The firing rate of Grid execution has been fixed to $\lambda = 10req/s$, which is larger than that of other processing work done by the Grid gateway, because the Grid service execution has the same influences for both of interaction strategies. The size of the procedure has been fixed to 2KB, and the size of the task is fixed to 10KB in this stage of evaluation. The sizes may larger for some more complex procedures and tasks. However, we do not consider this condition because if mobile users need to upload a large-size file to Grid gateways through an unreliable low-bandwidth wireless network, the overall system performance will be mostly affected by the code-transferring time. As discussed in the above chapter, for complex tasks we recommend that mobile users submit a URL (the location where the task executing codes are located) instead of the executing codes.

Parameter	Description	Value
$D_{procedure}$	Dimension of client request	2KB
D_{task}	Dimension of client request	10KB
$\lambda_{task\_preprocess}$	Task preprocessing rate	5req/s
$\lambda_{procedure\_preprocess}$	Procedure preprocessing rate	5req/s
λ_{one_exec}	Rate of Grid execution	10req/s
D_{result}	Dimension of Grid result	50KB
$\lambda_{result\_process}$	Result processing rate	5req/s
D_{min}, D_{max}	Dimension of the final result	1KB, 10KB
TH_{high}	throughput of high speed network	10Mbps

TABLE 7.4: Numerical values assigned for this stage of evaluation.

Based on these assigned values, both the "procedure-oriented" and "task-oriented" models are executed. The purpose is to measure the system response time and draw a conclusion that which interaction strategy is more suitable for the implementation of the system architecture.

At first, the throughput of the wireless network (TH_{low}) is fixed at 50kbps, and Figure 7.11 shows the chart of the average system response time based on the variation of the number of the Grid request (N). As expected, the "procedure-oriented" model has shorter system response time when the value of N is small, only 1.345 seconds when N is 1. As the value of N increases, the time of the "procedure-oriented" model increases linearly, the slope (the rate at which the system response time changes with respect to a change of the number of Grid request) of which is 1.35 or so. For the "task-oriented" model, the initial system response time is longer than that of the "procedure-oriented" model. However, as the value of N increases from 1 to 10, the average system response time increases exponentially but very slowly, only 97.8 percent going up in the experiment, so that when the value of N rises to three, the response time of the "procedure-oriented" model exceeds that of the "task-oriented" model.

The reason of behind the change in system performance is the interaction strategy between mobile clients and Grid gateways. For the "procedure-oriented" model, because of the smaller size, a relatively short time is required to upload the procedure to the gateway. However, for every Grid request, mobile clients need to transfer the procedure over their relatively-slow wireless network and perform the same operation, which leads to the linear increase in the system response time. For the "task-oriented" model, the bottleneck of the system performance is uploading the task to the gateway at the beginning of the task execution. However, once the deputy object has accepted the task, mobile clients do not need to do anything no matter how many Grid services are

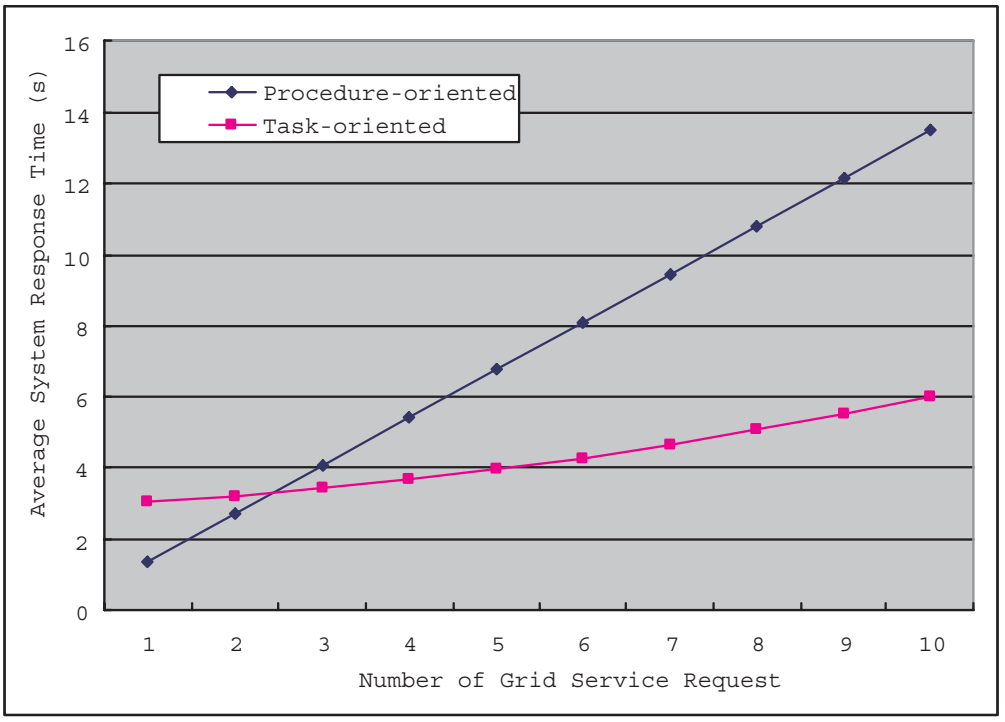

FIGURE 7.11: Average System Response Time vs. Number of Grid Request when TH_{low} = 50kbps.

required to be invoked. Hence, the system performance does not decrease much as the number of the underlying Grid requests increase.

In Figure 7.11, a threshold point (between N=2 and N=3) for the Grid request can be observed. When N is equal to or lower than 2, the "procedure-oriented" model has a better system performance than the "task-oriented" model; when N is equal to or larger than 3, the result is reversed and the "task-oriented" model has a better overall system performance.

The throughput of the wireless network is a key parameter for these models because it determines the procedure and task uploading time (it was fixed at 50Kbps in the above experiment). In order to estimate its effect for both types of interaction models, we measure the system response time under different wireless network throughputs. Figure 7.12, Figure 7.13 Figure 7.14, and Figure 7.15 show diagrams of the average system response time based on the variation of the number of the Grid request when TH_{low} is 20kbps, 100kbps, 200kbps and 500kbps respectively.

By observing the results of the above five experiments, the following conclusions can be derived:

- The average system response time of both two models decreases as the bandwidth of the wireless network becomes higher.

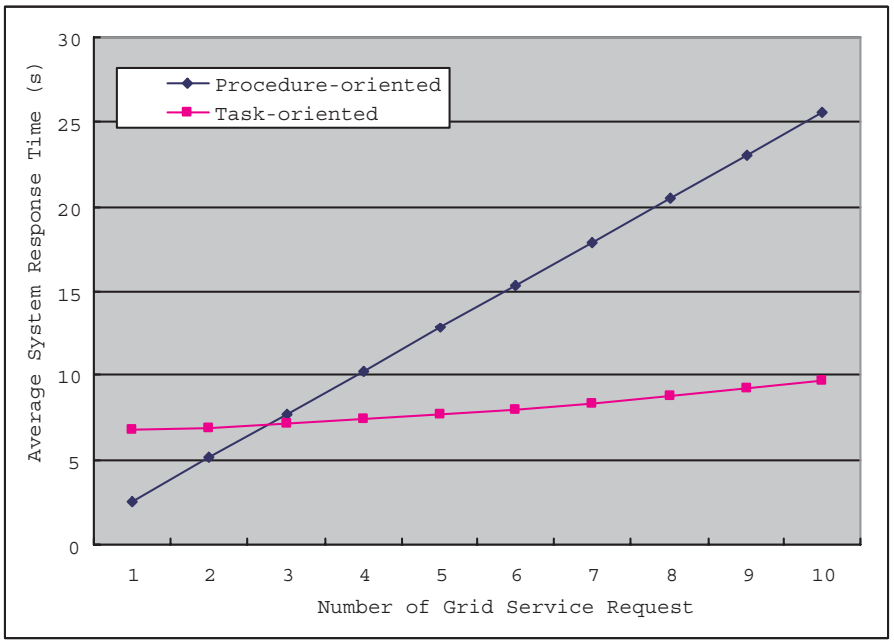

FIGURE 7.12: Average System Response Time vs. Number of Grid Request when TH_{low} = 20kbps.

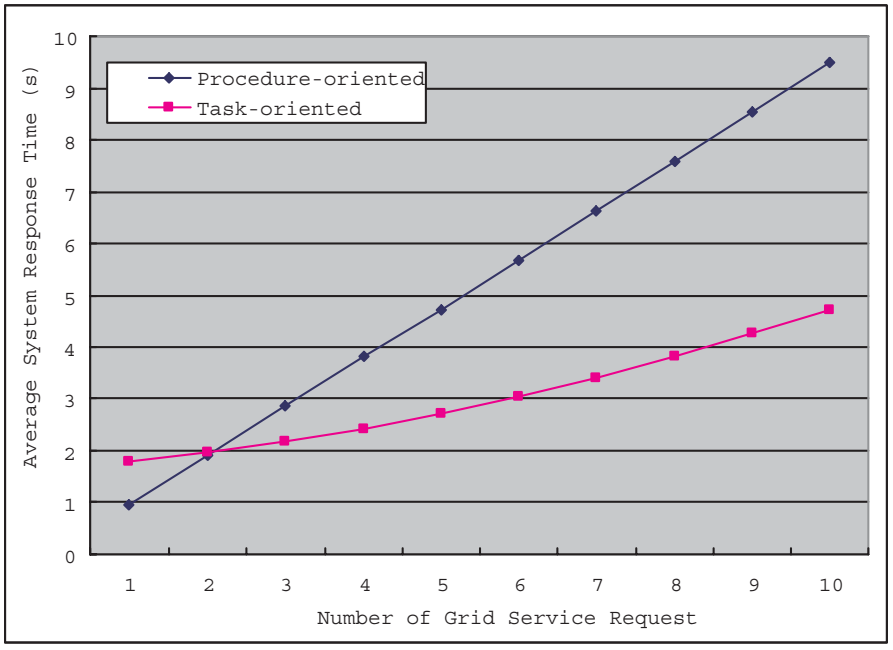

FIGURE 7.13: Average System Response Time vs. Number of Grid Request when TH_{low} = 100kbps.

- The average system response time of the "procedure-oriented" model always increases linearly as the number of Grid requests for executing a task from mobile clients increases. However, the speed of the increase (the slope of the line) becomes slower as the bandwidth of the wireless network becomes higher. Table 7.5 shows the slope value under different values of TH_{low}.

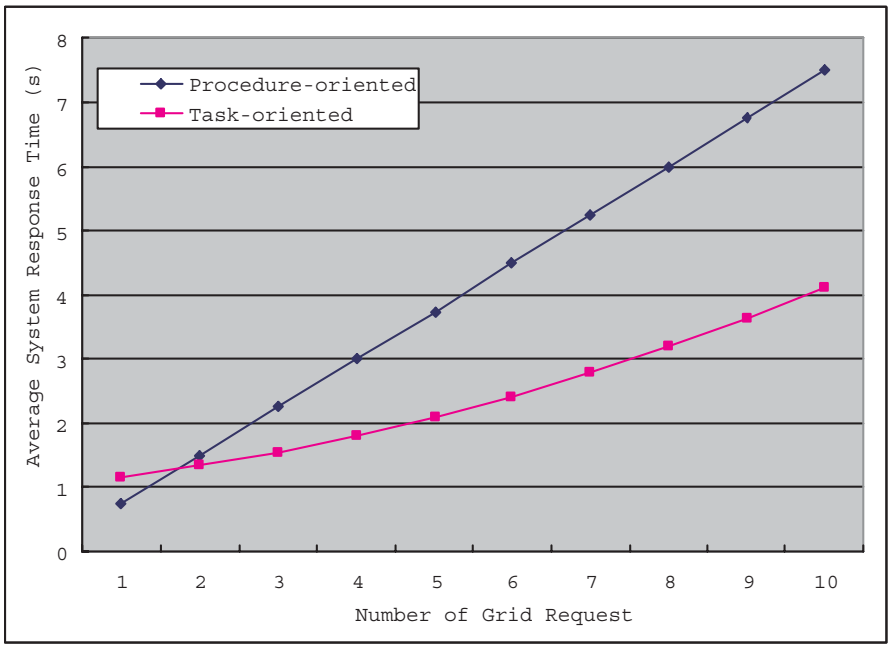

FIGURE 7.14: Average System Response Time vs. Number of Grid Request when TH_{low} = 200kbps.

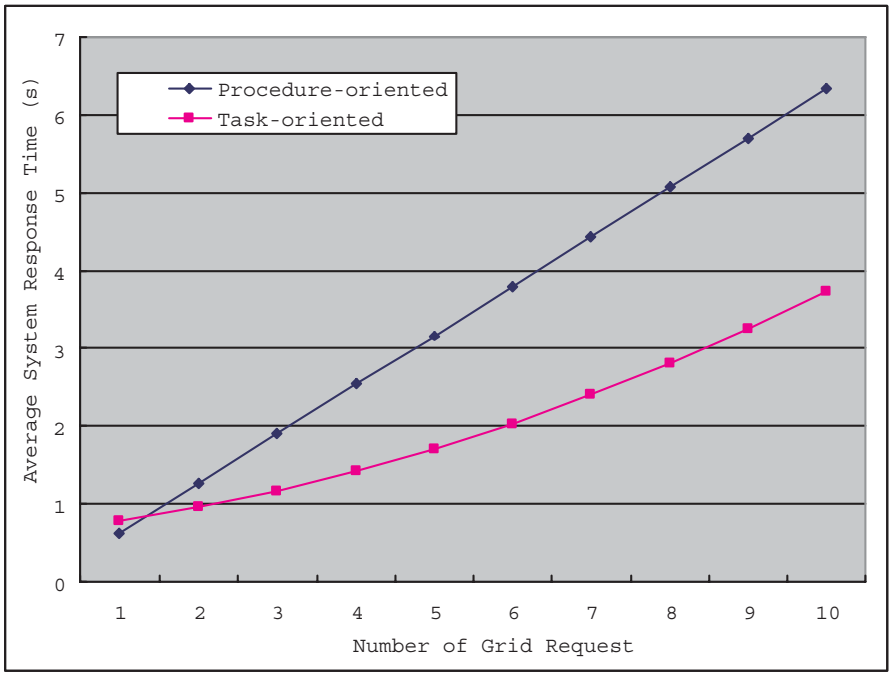

FIGURE 7.15: Average System Response Time vs. Number of Grid Request when TH_{low} = 500kbps.

- The average system response time of the "task-oriented" model increases exponentially as the number of Grid request for executing a tasks from mobile clients increases. However, the increasing speed of curves becomes faster when the bandwidth of the wireless network becomes higher. Table 7.6 shows the increasing ratio under different TH_{low}.

	20kpbs	**50kbps**	**100kbps**	**200kbps**	**500kbps**
Slope=Δt/Δr (s)	2.56	1.35	0.95	0.75	0.63

TABLE 7.5: Increasing Line Slope Value of the "Procedure-oriented" Model under Variation of TH_{low}.

	20kpbs	**50kbps**	**100kbps**	**200kbps**	**500kbps**
Ratio=t_{10}-t_1/t_1	0.44	0.98	1.66	2.55	3.75

TABLE 7.6: Increasing Ratio of Curves of "Task-oriented" Model under Variation of TH_{low}.

- When the number of Grid request is small, the "procedure-oriented" model always has a better overall performance than the "task-oriented" model. When the number of Grid requests increase, the system response time of the "task-oriented" model becomes lower. Hence, there is always a crossing point (threshold point) in every result diagram, which represents the boundary of the most efficient system performance for the two models. The experiments show that the threshold point is not relevant to the variation of the wireless network bandwidth, the value of which is approximately 2.

Figure 7.16 shows the diagram of the variation of the threshold point under different ratios between the procedure size and the task size. In the experiment, the bandwidth of the wireless network is fixed at 50kpbs and the size of the procedure is kept at 2KB. From the result of the experiment, it is clear that the value of the threshold point is relevant to the ratio between the size of the task and the procedure. When the ratio is low, the value of the threshold point is small (no more than 2 when the ratio is 3); as the ratio increases, the value of the threshold point goes up (almost 4 when the ratio is 10).

As we have discussed at the beginning of this evaluation section, both "procedure-oriented" and "task-oriented" interaction strategies have their advantages: the "procedure-oriented" strategy does not require the user to accept the risk of failure in uploading a large executing object to the Grid gateway over the unreliable and low-bandwidth wireless network; while the "task-oriented" strategy saves the time and resources of mobile devices so that users can process other jobs with their mobile devices after submitting the task. The evaluation demonstrates that it cannot be determined which interaction strategy is more suitable for implementing the system architecture because both of them can provide a better system performance under different conditions.

For most mobile clients, the "task-oriented" interaction strategy appears to be a better option because it preserves the limited resources of the device and increases the working efficiency. Furthermore, as the connection capability of mobile devices improves (the bandwidth is more than 50kbps), uploading larger files through the wireless network of

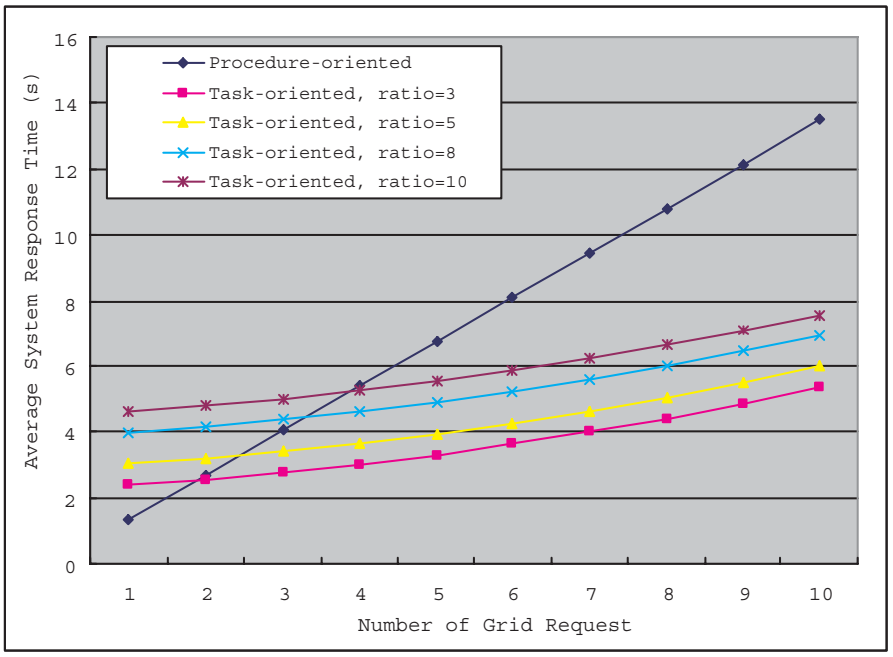

FIGURE 7.16: Variation of Threshold Point under Different Ratio between Size of Task and Procedure when TH_{low} = 50kbps.

mobile devices is gradually becoming common. Hence, the "task-oriented" interaction strategy should be the first choice for implementing the system architecture. However, it must be noted that the "procedure-oriented" strategy provides a better performance than the "task-oriented" strategy when the wireless network bandwidth is low and a task is not required to invoke Grid services many times.

There is a system performance threshold point between these two strategies when other model parameters are fixed. If the number of the Grid service request is lower than the value of this point, the "procedure-oriented" strategy has a better performance; if the number of the Grid service request is higher than the value of the point, the "task-oriented" mechanism has a better performance. Through the experiments, it can be concluded that the system performance threshold point does not depend very much on the wireless network bandwidth, but the ratio between the size of the task and the procedure. Hence, when designing an application based on the system architecture, the detailed task description and executing codes should be concise, especially when the task can be performed without requesting Grid services many times. Otherwise, it is not worth providing a convenient interaction strategy for users at the price of sacrificing the overall system performance.

7.4.3 The Integrated System Analysis

In the first evaluation section, the completion time of the "static Grid client", the "with deputy mobile Grid client", and the "without deputy mobile Grid client" models is

measured and it is concluded that the "device-gateway-Grid" system architecture is a reasonable candidate for mobile devices accessing Grid services because of the acceptable communication delays. In section two, two possible interaction strategies are evaluated in order to determine which one is more suitable for the system architecture and it is concluded that the "task-oriented" should be first choice of implementing the system architecture although it may become the system bottleneck if the task size is not well controlled. In this evaluation section, an integrated system model is built which is used to analyze the performance of an comprehensive operation between service consumers and service providers.

7.4.3.1 Models of Petri Nets

Figure 7.17 shows the Petri Net model for an integrated process of task execution. The following points explain this complex model:

- Mobile clients can interact with Grid gateways and Grid services using either the "procedure-oriented" or the "task-oriented" strategy. This behavior is described by a pair of immediate transitions, each of which is associated with a probability that the event described takes place. If p_r is the probability of adopting the "procedure-oriented" strategy, 1-p_r represents the probability of adopting the "task-oriented" strategy. In the Petri Net models, a firing probability p_r is associated to the transition *SendingProce_Model*, and a firing probability 1-p_r is associated to the transition *SendingTask_Model*. Hence, when a token enters the place *Deputy_Ready*, it will enable either the transition *SendingProce_Model* or the transition *SendingTask_Model* randomly.

- In order to simplify the model, it is assumed that every task submitted to the Grid gateway requires to access Grid services three times, or a mobile client needs to send three executing procedures to achieve the task. This assumption is made because of two reasons: a normal task execution does not require to invoke Grid services many times; and the value of the system performance threshold point between two possible interaction strategies is between two and three when the bandwidth of the wireless network is from 20Kbps to 100Kbps. The behavior is described by the value of the token number in the place *Num_Proc* and the place *Next_Gridexec* of the model, which is initialized by three and two respective.

- Although this model seems complex, it still has an absorption state, and the mean time to the absorption state can be measured. However, in contrast to the models built in the earlier evaluation sections, there are now two conditions which identify the absorption state because two interaction strategies are described in the model. If the mobile client sends one executing procedure one time, the condition for the absorption state is (#Gateway_Ready1 == 1) and (#Num_Proc == 0). If the

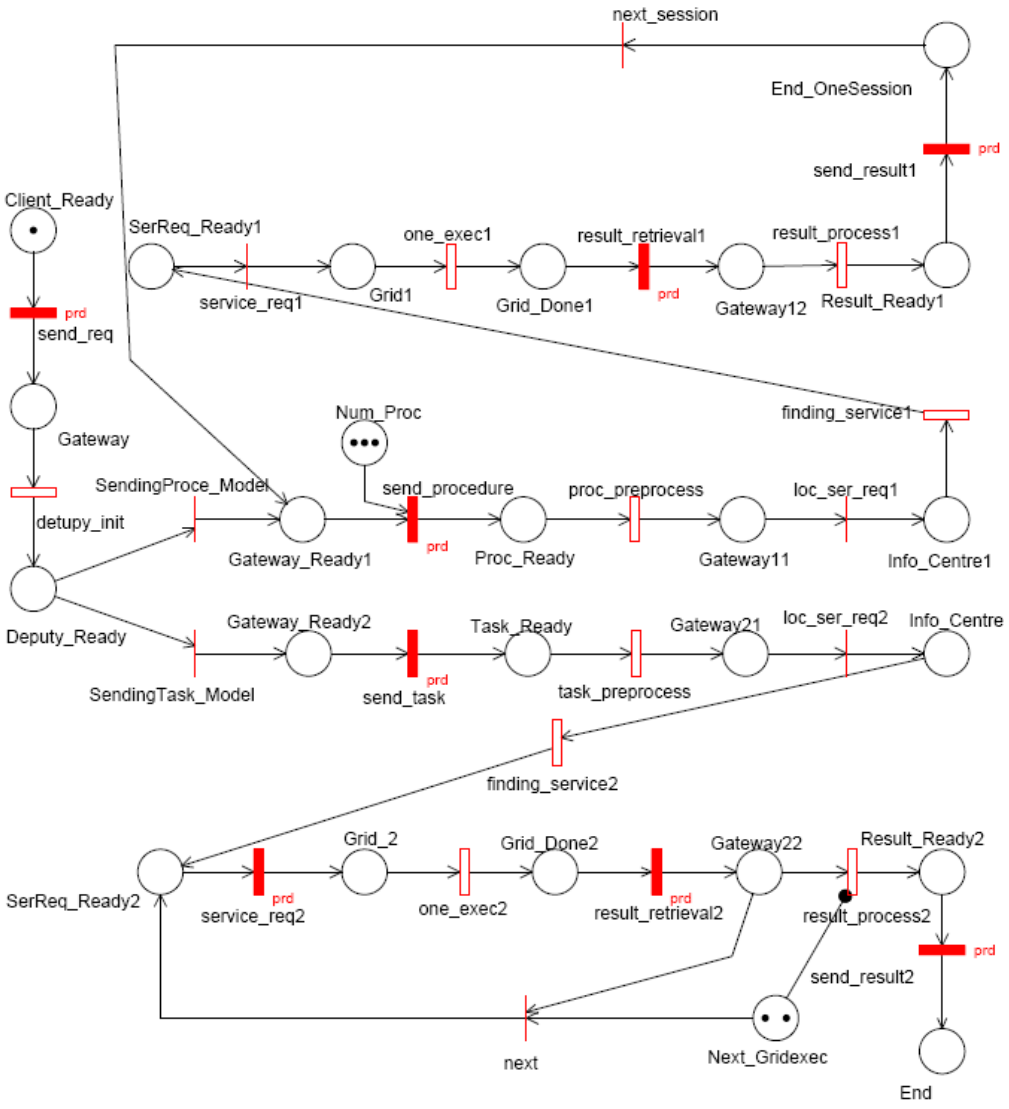

FIGURE 7.17: Integrated System Model.

client sends the whole task to the Grid gateway one time, the condition for the absorption state is (#End == 1).

7.4.3.2 Numerical Evaluation of the Model

The parameters which are defined in the model have already been discussed in the above evaluation sections. The numerical values assigned to these parameters are almost the same with those of the earlier experiments. Two values should be noticed: the bandwidth of the wireless network is fixed at 100Kbps, and the size of a task is set as 10KB. Based on these values, the system performance is measured. However, in contrast to the former

experiments, the distribution of the system response time is measured, rather than the average system response time, because we believe the distribution of the response time provides more accurate information about the system performance.

Figure 7.18 shows the distribution of the system response time. The graph shows that a probability of 0.9 of the system response time is reached in a time $t_{0.9}$≈4.15s and a probability of 0.999 of the system response time is reached in a time $t_{0.999}$≈6.96s.

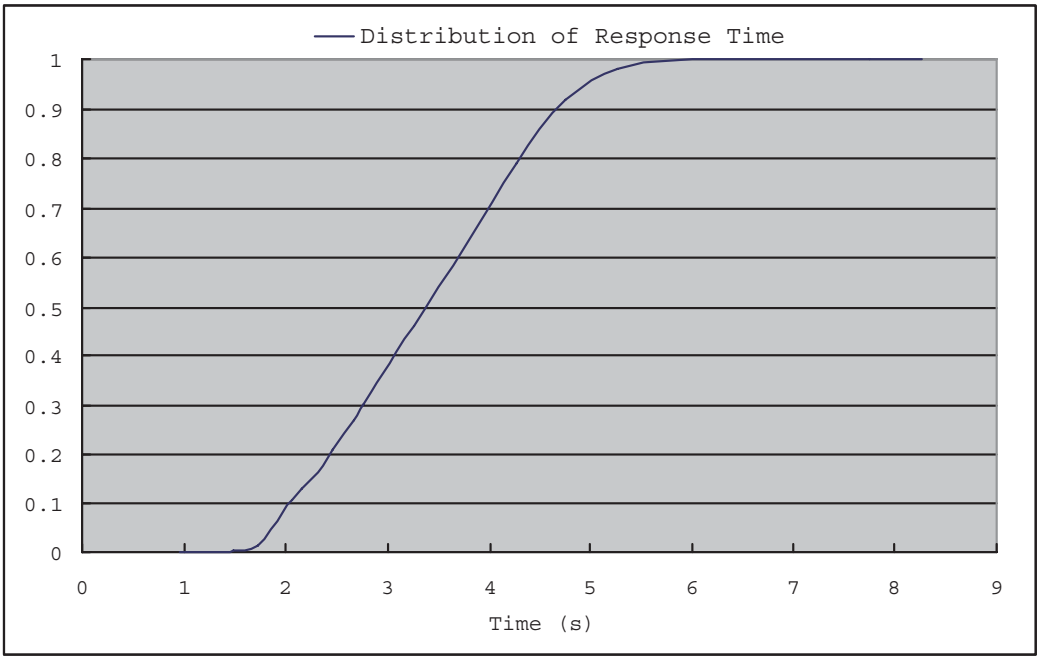

FIGURE 7.18: Distribution of System Response Time.

In order to test the system scalability, we measure the average system response time when a number of mobile clients request to invoke Grid services. Here, the term "gateway capacity" is defined, which indicates the maximum number of mobile clients that can submit tasks simultaneously to a Grid gateway. If the current task execution number is below the "gateway capacity", new mobile clients can be assigned the connection contracts with the Grid gateway immediately after they submit the request. Otherwise, they have to wait until a current task execution is completed and required resources of the Grid gateway are released.

The following Petri Nets pseudo sub structure (Figure 7.19) simulates a Grid gateway that can accept five client requests at the same time. In this structure, the transition *task_execution* indicates an integrated task execution course, which is modeled detailedly in the Figure 7.17. The inhibitor arc between the transition *queue* and the place *Start* signifies that only when a task execution is completed can the next token enter the execution course. The token number in the place *Ready* shows the number of clients which are requesting to offload tasks to Grid gateway.

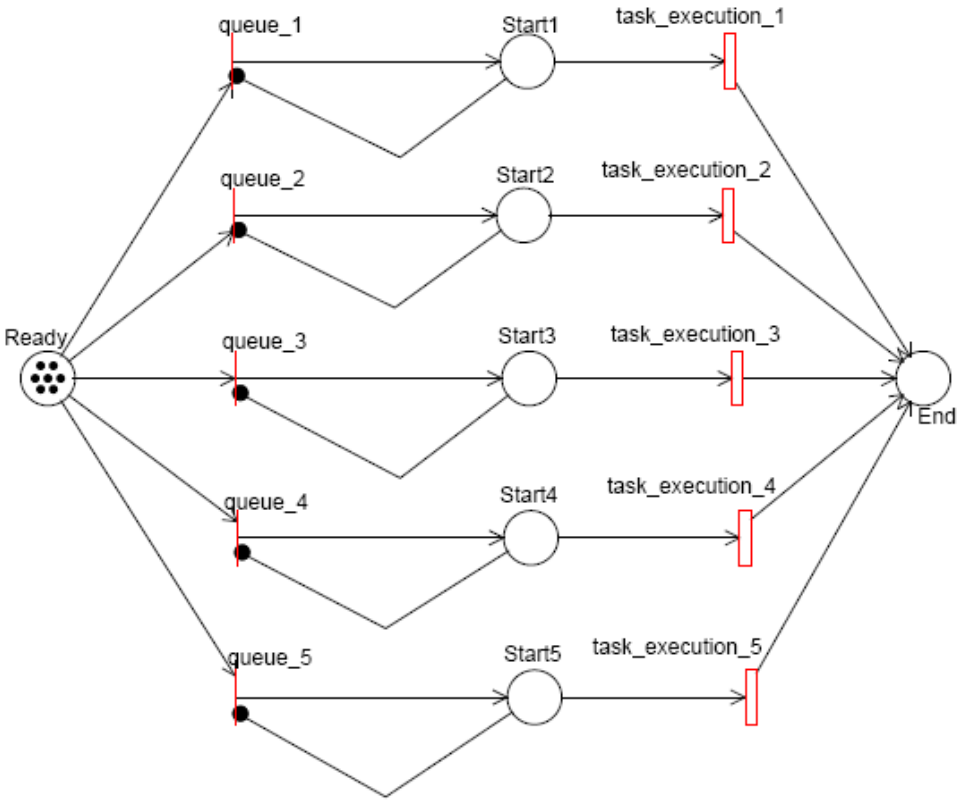

FIGURE 7.19: Petri Nets Structure (gateway capacity = 5)

We measure the average system response time against a number of mobile client requests, varying the gateway capacity from 2 to 9. Figure 7.20 shows the result:

The experiment result shows that as the number of mobile clients increases, the system response time goes up too. Before the experiment, we expected that the response time not to be tightly proportional to the number of mobile clients. The experiment result confirms our expectation. Furthermore, the result also indicates that the system performance is highly correlated to the value of "gateway capacity". As the value of the gateway capacity increases, the average system response time decreases remarkably. Even when mobile clients need to queue because there are no processing slots, the average additional required time still goes down as the gateway capacity value rises. Table 7.7 shows the average additional time under different values of gateway capacity (from 2 to 9).

Gateway Capacity	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Additional Time (s)	1.64	1.09	0.82	0.657	0.54	0.47	0.41	0.37

TABLE 7.7: Required Task Processing Time for an Additional Mobile Client when Gateway Capacity is from 2 to 9.

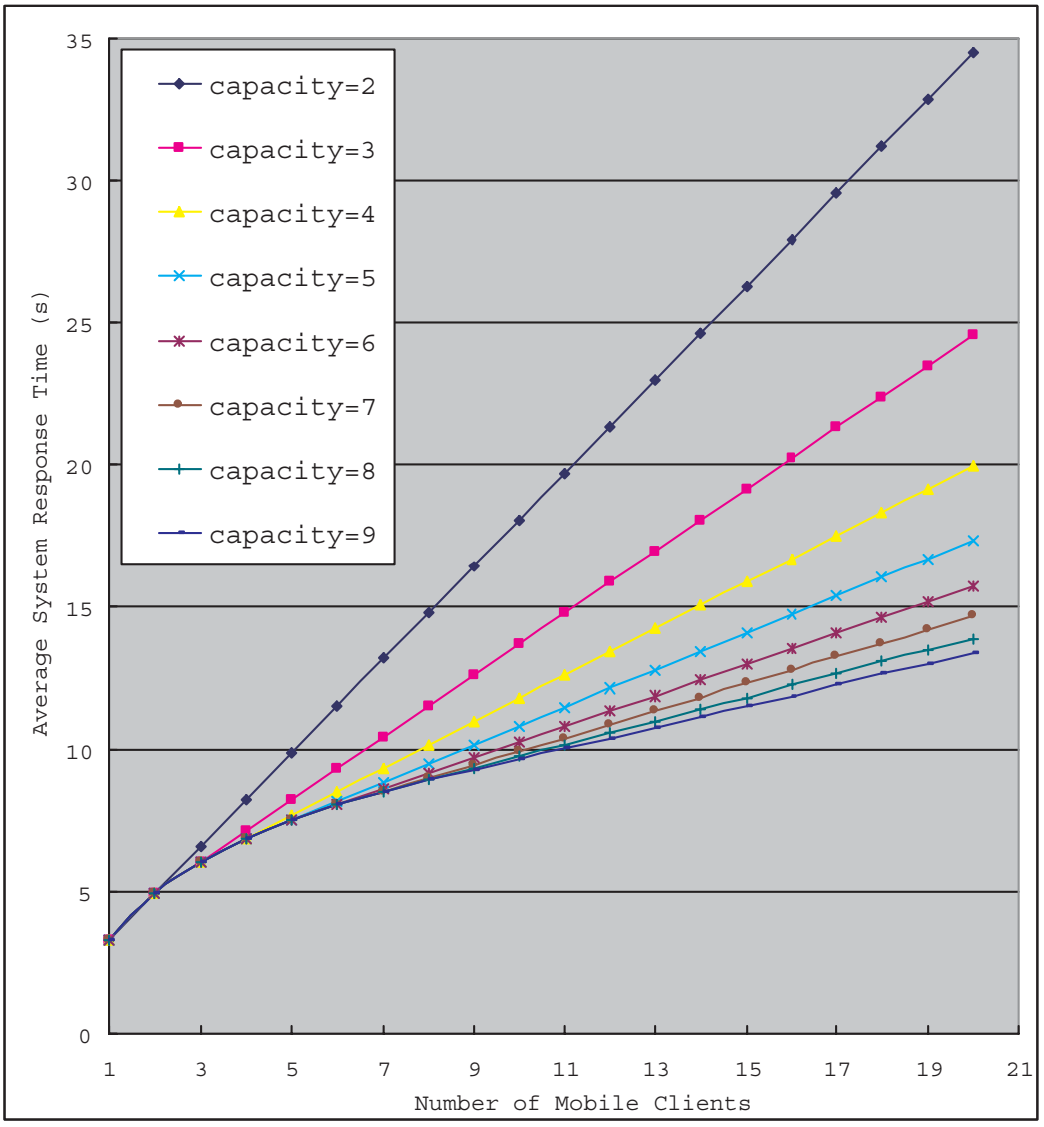

FIGURE 7.20: System Completion Time as the Number of Mobile Clients Increases.

7.5 Summary

This chapter has evaluated the system performance with the simulation method. A number of interaction models have been built using non-Markovian Stochastic Petri Nets to indicate interaction mechanisms between mobile devices, Grid gateways, and Grid services. Based on these models, the system response time is measured. The evaluation experiments are divided into three sections. From the experimental results, the following conclusions can be derived:

- The "with deputy" mobile client has the potential to improves the system performance, which consists with the conclusion acquired from real application tests. The "device-gateway-Grid" system architecture is a reasonable candidate for enabling

mobile devices to invoke Grid services because the delay caused by the communication between mobile devices and the deputy middleware on Grid gateways as well as the additional overhead for accessing the Grid environment is within acceptable limits.

- "Task-oriented" is the first choice for implementing the system architecture. However, the task description and other related executing codes required to be uploaded to gateways should be small and concise, especially when the task does not require to invoke Grid services many times and the wireless network bandwidth is low.

- "Gateway capacity" is an important parameter for the scalability of the system architecture. A number of service clients may submit requests to a Grid gateway simultaneously. With the rising of the value of gateway capacity, the average time of achieve these tasks decrease. Hence, in order to improve the system performance in the condition of heavy load, new technologies should be adopted to increase the "gateway capacity".

Chapter 8

Conclusions and Future Work

Pervasive mobile devices are gradually becoming prevalent in our everyday life, enabling users in the physical world to interact with the digital world smoothly and conveniently. Grid computing provides a vision of accessing distributed resources on demand. Semantic web technologies have the potential to enable automation and interoperability and are now being adopted within the practice of Grid computing. This semantic Grid provides a high degree of easy-to-use and seamless automation to facilitate flexible collaborations and computations on a global scale. The integration of the semantic Grid and ubiquitous devices will encourage a new concept of "Ambient Intelligence", the notion of intelligence in the surrounding environment supporting the activities and interactions of users.

8.1 Future Work

The research work presented in earlier chapters needs to be extended in several ways before it is utilized in practice. The following outlines the potential work that can be carried out in the future.

8.1.1 Grid Service Composition

A prerequisite for mobile devices to perform tasks by using Grid services is that required Grid services must be discovered. In chapter five, a semantic approach for service description and discovery is presented, based on which a Grid service matching middleware has been built. The semantic service matching middleware is integrated in the system architecture so that other middleware can locate Grid services through its programming interface.

However, one Grid service sometimes cannot provide all the desired functionalities for achieving complicated tasks. Several Grid services are required to be discovered, integrated and invoked in order to achieve an object. This series of procedures (termed a "Service Composition") can be implemented manually: users invoke different services respectively and put corresponding results together. However, manual service composition is not an attractive approach for future application scenarios: mobile users should be able to submit their tasks and receive the final results automatically.

As discussed in chapter five, semantic service description is the foundation of a service discovery mechanism. We also believe that a prerequisite of the service composition is an unambiguous computer interpretable form of the service description. Because a comprehensive Grid service description has been built with the extended OWL-S ontology, research into Grid service composition can start in the future based on the Grid service description.

Inputs, outputs, preconditions and effects (IOPEs) are important functional attributes for a service, which represent the constrains and results of the service invocation. A service composition mechanism can be implemented using a task-solving workflow through reasoning about the constrains and results of services advertised in the service registry. There are a number of different approaches to service composition, most of which utilize the atomic and composite process model of the OWL-S language. For example, the Golog system [180] [181] models services as actions with IOPEs and uses OWL-S composite processes to represent general procedures of performing tasks. Hierarchical Task Network (HTN) planning [182] [183] is another technology to perform web service composition. All of these provide valuable references for future work.

8.1.2 Security and Privacy Considerations

The current system architecture does not take into account important security and privacy considerations. We assume a pre-existing trust relation between mobile users, Grid gateways and Grid service providers because in the test applications, the mobile user uses his own desktop as the Grid gateway and the required Grid services have been deployed in the server provided by the research group. However, if the system architecture is to be deployed in practice, appropriate provision must be made for security and privacy.

Build-in security is also important for implementing a semantic service matching mechanism because there are a number of advertised services on a service registry but a service should only be capable of being discovered by authorized users. At present, we have encoded the service security information into the service description ("service range" attribute) and assumed every mobile user is able to get an access level which will be used to determine whether advertised services can be discovered by the user. In the future,

we plan to extend this service discovery restriction to allow the security considerations to be evaluated during the service discovery process.

8.1.3 Further Consideration for System Design and Evaluation

The goal of the research work is to build a system architecture to provide enhanced Grid access for mobile devices, which requires to meet a great number of challenges. This thesis concentrates on solving the problems about seamless interaction between devices and computing environment, flexible Grid service discovery and offloading tasks from mobile devices to intermediate machines. Other issues such as mobile user modelling, the usability and user interface beyond screen size of devices, the optimization of detailed task formats for specific applications, and the context knowledge reuse as well as the device context (e.g. screen size) should be considered when improving the system design in the future.

Three sample applications have been built in order to test the functionalities of the system architecture before it is practically used. The result of the experiments based on the "Mobile Shopping" and "Searching for Information" test applications demonstrate that using the mobile deputy middleware to invoke Grid services improves the system performance compared to accessing Grid resources directly from mobile devices. This conclusion has been confirmed by evaluating the system performance with the simulation method. The integrated system model built using Petri Nets indicates that "Gateway Capacity" is an important parameter for the scalability of the system architecture.

Further experiments are also required before the system architecture can be used in practice. For example, because of the restrictions of the current experiment environment, we do not measure the system performance when the connection between mobile devices and Grid gateways is not based on IEEE 802.11 networking, but uses other networking protocols (e.g Bluetooth or General Packet Radio Service (GPRS)).

In the current system evaluation experiments, a number of parameters (e.g. bandwidth and processing power of mobile devices) are configured to be consistent with capabilities of existing mobile devices. However, because of technological advance, next generations of mobile devices will be at least as capable as a desktop in many aspects. Hence, during the performance evaluation in the future, technological parameters should be assigned values to show the practical capabilities of mobile devices, for example, a higher value of $\lambda_{task\_preprocess\_MD}$ and TH_low.

After testing experiments, Grid gateways should be deployed in a real service-oriented Grid environment so that mobile users can utilize them to invoke Grid services. This will demonstrate the effectiveness of the middleware designed. Every Grid gateway has its "Gateway Capacity". When the number of mobile users exceeds the "Gateway Capacity", mobile users have to wait to obtain a processing slot. Approaches to improving

the system performance in this aspect should be investigated. A possible solution is the application of a load balancing method that will transfer the user request to another Grid gateway which has available processing slots.

8.2 Conclusions

The central motivation of this thesis is the trends in the modern computing technology: consumer electronics devices require more integration and computation. While new mobile devices improve their capabilities, executing complex applications on mobile devices remains a challenging objective because of the limited resources compared to their static counterparts. One feasible solution is to enable mobile devices to be integrated into the Grid environment so that Grid services can enhance their capabilities to execute complex tasks.

The integration between mobile devices and Grid services enables us to establish a new field and a number of challenges are required to be addressed to bridge the gap between these two new computing models. Through reviewing other research work in this field, it is found that most current research projects concentrate on implementing a framework which brings mobile devices into the Grid environment and few of them demonstrate solutions for other important requirements (e.g. service matching, context awareness). In our research work, Semantic Web technologies have been adopted to address the issues of seamless interaction and Grid service discovery.

The seamless interaction between mobile users and the service-oriented Grid computing environment is required because of the dynamic nature of users, devices and services. To realize this vision, the computer systems need to have the ability of understanding the context information of the computing environment. In chapter four, a shared context model is defined using the ontology approach and various context information can be stored in a public knowledge base. The context-aware framework provides functionalities of context aggregation, context querying and context reasoning. The experiments demonstrate that the context-aware framework can assist applications to determine the appropriate actions according to information (e.g. user status, device profile) obtained through context querying and reasoning.

Grid service discovery is the prerequisite of enhancing mobile device capabilities by utilizing Grid services. Existing service discovery mechanisms do not support flexible matching between service advertisements and requests, and users can only locate required services based on the syntactical equivalence of keywords or strings. A semantic approach to service description and discovery is discussed in chapter five. Service attributes have been represented with the extended OWL-S ontology language and the service matching engine implemented using the ontology reasoning system provides query

interfaces for user or other middleware to locate required services. The semantic service matching mechanism facilitates a flexible discovery in a centralized range of service directory and offers the ranking information that enables users to select appropriate services from the discovery results. The experimental results show that significant improvement in overall system functionality has been obtained with an acceptable increase in the service discovery time.

Based on the context-aware framework and the semantic service matching middleware, a system architecture has been developed to provide an enhanced Grid access for mobile devices, overcoming both the slow processing capability of mobile devices and restricted data transmission through an unreliable and bandwidth-limited wireless network. In the system architecture, the static distributed Grid services and mobile devices are interconnected by the Grid gateway, which provides a relatively resource-rich execution environment and stable network connectivity. The following characteristics have been supported in our system architecture that integrates mobile devices into the service-oriented Grid environment:

- Reliable task execution: the mobile deputy middleware deployed on the Grid gateway is responsible for accepting tasks and invoking required Grid services on behalf of mobile devices, which improves the potential system performance compared to executing tasks directly from devices themselves. The process of task execution is transparent for mobile users. Offline processing is supported so that users are able to disconnect from the Grid gateway after submitting tasks and reestablish the connection to collect the result.

- Seamless interaction: various context information is stored in the context information centre middleware. The knowledge base provides a querying interface which enables the appropriate decisions to be made for further actions. For example, the result will be returned to users in a format which is suitable with the device profile.

- No prior knowledge of Grid services: mobile users do not need to know the prior knowledge of deployed Grid services. The system locates and selects suitable services for task execution based on the task requirement as well as the access level of users.

- Device adaptation: a variety of mobile devices with different hardware and software equipments can be integrated into the system. The deputy object created hides the diversity of mobile devices. Mobile devices do not need to install a heavyweight client software. The minimum requirement is a standard web browser.

The proposed system architecture was evaluated with both comparison and simulation. Three sample scenarios have been implemented and the experimental results demonstrate that using the system architecture can improve the system response time significantly when executing a complex task. In chapter seven, the Petri Nets simulation approach has been used to evaluate the system performance. The results indicate that the "device-gateway-Grid" system architecture is a reasonable candidate for mobile clients accessing the Grid environment because the delay caused by the communication between mobile devices and Grid gateways as well as the additional processing to utilize Grid services is within acceptable limits. "Procedure-oriented" and "task-oriented" are two possible interaction strategies between mobile devices and Grid services, and both of them have advantages and disadvantages. The system performance threshold point has been evaluated and the results show that the task description and executing codes should be made compact if using the "task-oriented" interaction strategy, especially when the task does not require many Grid services. Through investigating the system scalability with respect to the number of mobile clients, it is concluded that "Gateway capacity" is an important parameter to improve the system performance.

As a summary, the contributions of our research work include:

- A context-aware framework has been built to support the seamless interaction between mobile users and computing environment.
- A semantic service discovery middleware has been built to provide a flexible service matching and offer ranking information for service selection.
- Based on the context-aware framework and the service discovery middleware, a system architecture has been developed to provide enhanced Grid access for mobile devices.
- The interaction between mobile devices, Grid gateways and Grid services has been modeled using Non-Markovian Stochastic Petri Nets. System performance has been evaluated with the simulation method.

In conclusion, this thesis discusses a system architecture that enables mobile users to utilize Grid resources with their handheld devices in order to perform complex tasks. The middleware in the system architecture hides the diversity of heterogeneous mobile devices, enables the required Grid services to be discovered in a flexible way, and provides a reliable task execution mechanism. Although there is no doubt that there are still a number of challenges to be overcome before the long-term vision of bringing Grid services into the world of "Ambient Intelligence" comes into existence, we believe the system architecture designed, implemented, and evaluated in this thesis represents an important step to realizing a variety of new computing models (in particular Grid computing) in the pervasive and mobile computing world.

Bibliography

[1] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 340 Print Street, Six Floor, San Francisco, CA, USA, 1999.

[2] M. Baker, R. Buyya, and D. Laforenza. The grid: International efforts in global computing. In *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, Roma, Italy, August 2000.

[3] I. Foster, C. Kesselman, and S. Teucke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[4] J. Bunn and H. Newman. *Data-intensive Grids for High-Energy Physics*. Berman F., Fox G.E., Hey A.J.C (eds): Grid Computing: Making the Global Infrastructure a Reality, Wiley, 2002.

[5] S. Hastings, T. Kurc, S. Langella, U. Catalyurek, T. Pan, and J. Saltz. Image processing for the grid: A toolkit for building grid-enabled image processing applications. In *3rd International Symposium onCluster Computing and the Grid*, pages 36–43, Tokyo, Japan, May 2003.

[6] W.E. Johnston. Computational and data grids in large-scale science and engineering. *Future Generation Computer Systems*, 18(8):1085–1100, 2002.

[7] V. Breton, A.E. Solomonides, and R.H. Mcclatchey. A perspective on the healthgrid initiative. In *2nd International Workshop on Biomedical Computations on the Grid*, pages 434–439, Chicago, USA, April 2004.

[8] C. Allison, S.A. Cerri, A. Gaeta, M. Gaeta, and P. Ritrovato. Services, semantics and standards: Elements of a learning grid infrastructure. *Applied Artificial Intelligence*, 19(9-10):861–879, 2005.

[9] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):99–104, 1991.

[10] D. Saba and A. Mukberjee. Pervasive computing: A paradigm for the 21st century. *IEEE Computer Society*, 36(3):25–31, 2003.

[11] M. Satyanarayanan. Swiss army knife or wallet. *IEEE Pervasive Computing*, 4(2):2–3, 2005.

[12] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[13] D. DeRoure and J. Hendler. E-science: The grid and the semantic web. *IEEE Intelligent System*, 19(1):65–71, Jan/Feb 2004.

[14] D. DeRoure, J. Frey, D. Michaelides, and K. Page. The collaborative semantic grid. In *Proc. 2006 International Symposium on Collaborative Technologies and Systems*, pages 411–418, Las Vegas, USA, 2006.

[15] D. DeRoure, N. Jennings, and N. Shadbolt. The semantic grid: Past, present and future. *Procedings of IEEE*, 93(3):669–681, March 2005.

[16] T. Hey and A.E. Trefethen. The uk e-science core programme and the grid. In *Proceedings of the International Conference on Computational Science-Part I*, pages 3–21, London, UK, 2002.

[17] D. DeRoure, T. Hey, and A.E. Trefethen. A global e-infrastructure for e-science - a step on the road to ambient intelligence. *Chapter of "True Visions: Tales on the Realization of Ambient Inteligence", Edited by E.H.L. Arts and J.L.Encarnacao, Springer*, pages 209–229, 2006.

[18] V. Hingne, A. Joshi, T. Finin, and E. Houstis. Toward a pervasive grid. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 8–12, Washington, DC, USA, April 2003.

[19] M. Satynanrayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communication*, 8(4):10–17, 2001.

[20] B. Noble. System support for mobile, adaptive applications. *IEEE Personal Communications*, 7(1):44–49, 2000.

[21] G. Banavar. Challenges: An application model for pervasive computing. In *the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 266–274, Boston, Massachusetts, USA, August 2000.

[22] D. Saba, A. Mukherjee, and S. Bandopadhyay. *Networking Infrastructure for Pervasive Computing: Enabling Technologies and Systems*. Kluwer Academic, 320 pp., ISBN 1-4020-7249-X, Norwell, MA, USA, October 2002.

[23] J. Flinn, D. Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, pages 61–66, Washington, DC, USA, May 2001.

[24] J. Flinn, S. Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *Proceedings of the 22nd Intl. Conf. on Distributed Computing Systems*, pages 217–226, September 2002.

[25] R. Balan, M. Satyanarayanan, S. Park, and T. Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services*, pages 273–286, San Francisco, CA, USA, 2003.

[26] R. Balan, J. Flinn, M. Sstyanarayanan, S. Sinnamohideen, and H. Yang. The case for cyber foraging. In *10th ACM. SIGOPS European Workshop*, pages 411–428, Saint-Emilion, France, May 2002.

[27] S. Goyal and J. Carter. A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *Sixth IEEE Workshop on Mobile Computing Systems and Applications*, pages 186–195, Lake District National Park, UK, December 2004.

[28] Y. Suu and J. Flinn. Slingshot: Deploying stateful services in wireless hotspots. In *the 3rd Annual Conference on Mobile Systems, Applications, and Services*, pages 79–92, Seattle, USA, June 2005.

[29] Y. Suu and J. Flinn. Portable storage support for cyber foraging. In *2005 International Workshop on Software Support for Portable Storage*, San Francisco, CA, USA, March 2005.

[30] S. Gitzenis and N. Bambos. Mobile to base task migration in wireless computing. In *IEEE Intl. Conf. on Pervasive Computing and Communication*, pages 187–196, Orlando, FL, USA, March 2004.

[31] I. Mohomed, A. Chin, J.C. Cai, and E. Lara. Community-driven adaptation: Automatic content adaptation in pervasive environments. In *Sixth IEEE Workshop on Mobile Computing Systems and Applications*, pages 124–133, English Lake District,UK, December 2004.

[32] L. Smarr and C. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, 1992.

[33] A. Grimshaw, A. Ferrari, G. Lindahl, and K. Holcomb. Metasystems. *Communications of the ACM*, 41(11):46–55, 1998.

[34] I. Foster and C. Kesselman. Globus: a metacomputing infrastructure toolkit. *Available online at ftp://ftp.globus.org/pub/globus/papers/globus.pdf*, 1997.

[35] I. Foster. What is grid? a three point checklist. *Available online at http://www-fp.mcs.anl.gov/ foster/Articles/WhatIsTheGrid.pdf*, 2002.

[36] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *International Conference on Network and Parallel Computing*, pages 2–13, Columbus, USA, August 2006.

[37] I. Foster, H. Kishimoto, and A. Savva. The physiology of the grid: An open grid services architecture for distributed systems integration. *Technical report, Open Grid Service Infrastructure WG, Global Grid Forum*, 2002.

[38] H. He. What is service-oriented architecture. *Available online at http://webservices.xml.com/lpt/a/1292*, 2003.

[39] T. Ciardiello. Data exchange - simple object access protocol. *IEE Water Event*, Ref No: 2005-11083:161–176, 2005.

[40] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web service definition language 1.1. *Available online at http://www.w3.org/TR/wsdl*, 2001.

[41] Jagannadham, D. Ramachandran, V. Kumar, and H.N. Harish. Java2 distributed application development (socket, rmi, servlet, corba) approaches, xml-rpc and web services functional analysis and performance comparison. In *International Symposium on Communications and Information Technologies*, pages 1337–1342, Sydney, Australia, March 2007.

[42] K. Czajkowsku, D. Ferguson, I. Foster, J. Frey, S. Graham, I.r Sedukhin, D. Snelling, S. Tukcke, and W. Vambenepe. The ws-resource framework. *Available online at http://www.globus.org/wsrf/specs/ws-wsrf.pdf*, 2004.

[43] S. Graham and J. Treadwell. Web services resrouce properties version 1.2. *Available online at http://doc.oasis-open.org/wsrf/2005/03/wsrf-WS-ResourceProperties-1.2-draft-06.pdf*, 2005.

[44] J. Frey and S. Graham. Web services resrouce lifetime version 1.2. *Available online at http://doc.oasis-open.org/wsrf/2005/03/wsrf-WS-ResourceLifetime-1.2-draft-05.pdf*, 2005.

[45] T. Maguire and D. Snelling. Web services service group version 1.2. *Available online at http://doc.oasis-open.org/wsrf/2005/03/wsrf-WS-ServiceGroup-1.2-draft-04.pdf*, 2005.

[46] S. Tuecke, L. Liu, and S. Meder. Web services base faults version 1.2. *Available online at http://doc.oasis-open.org/wsrf/2005/03/wsrf-WS-BaseFaults-1.2-draft-04.pdf*, 2005.

[47] S. Graham, P. Niblett, D. Chappel, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Seduhkin, D. Snelling, S. Tuecke, W. Vanbenepe, and B.W. eihl. Published-subscribe notification for web services. *Available online at http://www-106.ibm.com/developerworks/library/ws-pubsub/WS-PubSub.pdf*, 2005.

[48] F. Manola and E. Miller. Rdf primer. *Available online at http://www.w3.org/TR/REC-rdf-syntax/*, 2004.

[49] O. Lassila and R.R. Swick. Resource description framework (rdf) model and syntax specification. *Available online at http://www.w3.org/TR/1999/REC-rdfsyantax-19990222/*, 2005.

[50] D. Brickley and R. Guha. Rdf vocabulary description language 1.0: Rdf schema. *Available online at http://www.w3.org/TR/rdf-schema/*, 2004.

[51] B. Chandrasekaran, J. Josephson, and R. Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, 1999.

[52] D. Connolly, F. Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L.A. Stein. Daml+oil reference description. *Available online at http://www.w3.org/TR/daml+oil-reference*, 2001.

[53] D. McGuinness, R. Fikes, J. Hendler, and L. Stein. Daml+oil: An ontology language for the semantic web. *IEEE Intelligent Systems*, 17(5):72–80, 2002.

[54] D.L. McGuinness and F. Harmelen. *OWL Web Ontology Language Overview*. Web Ontology Working Group, in Proposed Recommendation for OWL, 2003.

[55] M. Smith, C. Welty, and D. McGuinness. Web ontology language guide version 1. *Available online at http://www.w3.org/TR/owl-guide*, 2003.

[56] D. Martin, M. Burstein, and J. Hobbs. Owl-s: Semantic markup for web services. *Available online at http://www.w3.org/Submission/OWL-S*, 2004.

[57] F. Baader, D. McGuinness, D. Nardi, and P. Patel-Schneider. *Description Logic Handbook - Theory, Implementation and Applications*. Cambridge University Press, 2003.

[58] A.K. Dey. Understanding and using context. *Personal And Ubiquitous Computing*, 5(1):4–7, 2001.

[59] M. Uschold and M. Gruniger. Ontologies: Principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[60] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *First International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 34–39, Nottingham, UK, Spetember 2004.

[61] H. Chen, T. Finin, and A. Joshi. Semantic web in the context broker architecture. In *Proceedings of Second Annual IEEE International Conference on Pervasive Computing and Communications*, pages 277–286, Orlando, FL, USA, March 2004.

[62] F.L. Gandon and N.M. Sadeh. Semantic web technologies to reconcile privacy and context awareness. *Web Semantics Journal*, 1(3):93–155, 2004.

[63] A. Ranganathan, R.E. McGrath, R.H. Campbell, , and M.D. Mickunas. Ontologies in a pervasive computing environment. In *Workshop on Ontologies and Distributed Systems*, pages 209–220, Mexico, September 2003.

[64] D. Brickley and L. Miller. Foaf vocabulary specification 0.9. In *RDF Web Namespace Document, RDFWeb, xmlns.com*, May 2007.

[65] E. Dumbill. Finding friends with xml and rdf. In *IBM developerWorks, XML Watch, xmlhack.com*, June 2002.

[66] F. Pan and J.R. Hobbs. Time in owl-s. In *Proceedings of AAAI-04 Spring Symposium on Semantic Web Service*, Stanford University, California, March 2004.

[67] D.B. Lenat and R.V. Guha. Building large knowledge-based systems; representation and inference in the cyc project. *Addison-Wesley Longman Publishing Co., Inc*, 1989.

[68] D.A. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In *Proceedings of Third International Knowledge Representation and Reasoning Conference*, pages 165–176, Cambridge, MA, USA, 1992.

[69] H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments. *Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3):197–207, 2003.

[70] J. Man, Q. Chen, X. Deng, and Y. Qiu. The design and implementation of shared ontologies for smart space application. In *Proceedings of International Conference on Machine Learning and Cybernetics*, pages 125–131, Guangzhou, China, August 2005.

[71] L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *Second International Semantic Web Conference*, pages 402–418, Sanibel Island, Florida, USA, September 2003.

[72] L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin, and A. Joshi. Authorization and privacy for semantic web service. In *Spring Symposium on Semantic Web Services*, pages 50–56, Palo Alto, CA, USA, March 2004.

[73] X. Wang, J.S. Dong, C.Y. Chin, S.R. Hettiarachchi, and D. Zhang. Semantic space: An infrastructure for smart spaces. *IEEE Pervasive Computing*, 3(3):32–39, 2004.

[74] D. DeRoure, N.R. Jennings, and N.R. Shadbolt. Research agenda for the semantic grid: A future e-science infrastructure. In *Technical Report UKeS-2002-02, National e-Science Centre*, December 2001.

[75] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen. Contextphone: A prototyping platform for context-aware mobile applications. *IEEE Pervasive Computing*, 4(2):51–59, 2005.

[76] D.F. Zucker, M. Uematsu, and T. Kamada. Content and web services converge: A unified user interface. *IEEE Pervasive Computing*, 4(4):8–11, 2005.

[77] E. Toye, R. Sharp, A. Madhavapeddy, and D. Scott. Using smart phones to access site-specific services. *IEEE Pervasive Computing*, 4(2):60–66, 2005.

[78] R.S. Kalawsky, S.P. Nee, I. Holmes, and P.V. Coveney. A grid-enabled lightweight computational steering client: a .net pda implementation. *Philosophical Transactions of the Royal Society*, 363(1833):1885–1894, 2005.

[79] J.M. Brooke, P.V. Coveney, J. Harting, S. Jha, S.M. Pickles, R.L. Pinning, and A.R. Porter. Computational steering in realitygird. In *Proceedings of the UK e-Science All Hands Meeting*, pages 885–888, Nottingham, UK, September 2003.

[80] O. Storz, A. Friday, and N. Davies. Towards 'ubiquitous' ubiquitous computing: an alliance with 'the grid'. In *First Workshop on System Support for Ubiquitous Computing Workshop in association with Fifth International Conference on Ubiquitous Computing*, Seattle, US, October 2003.

[81] D.E. Millard, A. Woukeu, F. Tao, and H.C. Davis. Experience with writing grid clients for mobile devices. In *1st International ELeGI Conference*, Naples, Italy, March 2005.

[82] M. Nidd. Service discovery in deapspace. *IEEE Personal Communications*, 8(4):39–45, 2001.

[83] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *17th ACM Symposium on Operating Systems Principles*, pages 186–201, Kiawah Island,SC, December 1999.

[84] M. Balazinska, H. Balakrishnan, and D. Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery. In *International Conference on Pervasive Computing*, pages 195–210, Zurich, Switzerland, August 2002.

[85] S. Czerwinski, B.Y. Zhao, T. Hodes, A. Joseph, and R. Katz. An architecture for a secure service discovery service. In *Fifth Annual International Conference on Mobile Computing and Networks*, pages 24–35, Seattle, US, August 1999.

[86] C.D. Knutson, E. Hall, and D. Vawdrey. Bluetooth [wireless connectivity]. *IEEE Potentials*, 21(4):28–31, 2002.

[87] J. Waldo. *The Jini Specifications*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[88] Salutation. Salutation architecture specification. *Available online at http://www.salutation.orgspecordr.htm*, 1999.

[89] M.L. Diagne, T. Noel, and J. Pansiot. Extension of service location protocol for ipv6 communicationmobility. In *IEEE Pacific Rim Conference on Communications, Computers and signal Processing*, pages 495–497, Victoria, BC, Canada, August 2001.

[90] J. Allard, V. Chinta, S. Gundala, and G. Richard. Jini meets upnp: an architecture for jini/upnp interoperability. In *Proceedings of Symposium on Applications and the Internet*, pages 268–275, Orlando, FL, USA, January 2003.

[91] F. Zhu, M. Mutka, and L. Ni. Splendor: a secure, private, and location-aware service discovery protocol supporting mobile services. In *First International Conference on Pervasive Computing and Communications*, pages 235–242, Fort Worth, TX, USA, March 2003.

[92] F. Zhu, M. Mutka, and L. Ni. Prudentexposure: a private and user-centric service discovery protocol. In *Second International Conference on Pervasive Computing and Communications*, pages 329–338, Orlando, FL, USA, March 2004.

[93] F. Zhu, M. Mutka, and L. Ni. A private, secure, and user-centric information exposure model for service discovery protocols. *IEEE Transactions on Mobile Computing*, 5(4):418 – 429, April 2006.

[94] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[95] Y. Huang, C. Xu, H. Wang, Y. Xia, J. Zhu, and C. Zhu. Formalizing web service choreography interface. In *21st International Conference on Advanced Information Networking and Applications Workshops*, pages 576–581, Niagara Falls, ON, Canada, May 2007.

[96] M.B. Juric, B. Mathew, and P. Sarang. *Business Process Execution Language for Web Services : BPEL and BPEL4WS*. Packt Publishing, Boston, MA, USA, 2004.

[97] M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel. Semantic business process management: A vision towards using semantic web services for business process management. In *IEEE International Conference on e-Business Engineering*, pages 535–540, Beijing, China, October 2005.

[98] M. Adacal and A. Bener. Mobile web services: A new agent-based framework. *IEEE Internet Computing*, 10(3):58–65, May 2006.

[99] A. Terracina, S. Beco, T. Kirkham, J. Gallop, I. Johnson, D. Macrandal, and B. Ritchie. Orchestration and workflow in a mobile grid environment. In *Proceedings of the Fifth International Conference on Grid and Cooperative Computing Workshops*, pages 251–258, Hunan, China, Oct. 2006.

[100] S.W. Loke. Context-aware artifacts: Two development approaches. *IEEE Pervasive Computing*, 5(2):48–53, 2006.

[101] D. Siewiorek. Sensay: A context-aware mobile phones. In *7th International Symposium of Wearable Computers*, pages 248–249, White Plains, NY, USA, October 2003.

[102] H. Chen, T. Finin, and A. Joshi. Semantic web in the context broker architecture. In *IEEE Intl. Conf. on Pervasive Computing and Communication (PerCom)*, pages 277–286, Orlando, FL, USA, March 2004.

[103] N.M. Sadeh, E. Chan, and L. Van. Mycampus: An agent-based environment for context-aware mobile services. In *First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Italy, July 2002.

[104] S. Wong and K. Ng. A middleware framework for secure mobile grid services. In *Proceedings of the sixth IEEE International Symposium on Cluster Computing and the Grid Workshops*, pages 8–11, Singapore, May 2006.

[105] S. Wong and K. Ng. Security support for mobile grid services framework. In *Proceedings of the International Conference on Next Generation Web Services Practics*, pages 75–82, Seoul, South Korea, September 2006.

[106] J.Vaucher and A. Ncho. Jade tutorial and primer. *Available online at http://www.iro.umontreal.ca/ vaucher/Agents/Jade/JadePrimer.html*, 2003.

[107] S.M. Park, Y.B. Ko, and J.H. Kim. Disconnected operation service in mobile grid computing. In *Proceeding of the International Conference on Service Oriented Computing*, pages 499–513, Italy, December 2003.

[108] M. Ciampi, A. Coronato, and G. DePietro. Middleware services for pervasive grid. In *Proceedings of International Symposium on Parallel and Distributed Processing and Application 2006*, pages 485–498, Italy, December 2006.

[109] R.R. Sambasivan, A.X. Zheng, E.Thereska, and G.R. Ganger. Categorizing and differencing system behaviours. In *Proceedings of the second International Workshop on Hot Topics in Autonomic Computing*, pages 9–13, Jacksonville, FL, USA, June 2007.

[110] R. Anthony. Policy-centric integration and dynamic composition of autonomic computing techniques. In *Proceedings of the fourth International Conference of Autonomic Computing*, pages 2–3, Jacksonville, FL, USA, June 2007.

[111] S. McIlraith and T. Son. Adapting golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning*, pages 482–493, Toulouse, France, April 2002.

[112] D. McDermott. Estimated-regression planning for interaction with web services. In *Proceedings of the sixth International Conference on AI Planning and Scheduling*, pages 204–211, Toulouse, France, April 2002.

[113] M. Wooldridge. *An Introduction to Multi Agent Systems*. John Wiley and Sons, ISBN 0 47149691X, October 2002.

[114] I. Foster, N. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 8–15, NY, USA, Aug. 2004.

[115] T. Phan, L. Huang, and C. Dulan. Challenge: Integrating mobile wireless devices into the computational grid. In *Proceedings of the eighth International Conference on Mobile Computing and Networking*, pages 271–278, Atlanta, Georgia, USA, September 2002.

[116] Bhagyavati and S. Kurkovsky. Wireless grid enables ubiquitous computing. In *the International Conference on Parallel and Distributed Computing Systems*, pages 399–404, Honolulu, Hawaii, USA, August 2003.

[117] S. Kurkovsky, Bhagyvatim, and M. Yang. Medeling a grid-based problem solving environment for mobile devices. In *the International Conference on Information Technology: Coding and Computing*, pages 05–07, Las Vegas, Nevada, USA, April 2004.

[118] M.Riaz, S.L. Kiani, A. Shehzad, and S. Lee. Bringing handhelds to the grid resourcefully: A surrogate middleware approach. In *International Conference on Computational Science and its Applications*, pages 1096–1105, Singapore, May 2005.

[119] S.L. Kiani, M. Riza, S. Lee, T. Jeon, and H. Kim. Grid access middleware for handheld devices. In *Europe Grid Computing 2005, LNCS 3470*, pages 1002–1011, Amsterdam, Netherlands, July 2005.

[120] H. Jameel, U. Kalim, A. Sajjad, S. Lee, and T. Jeon. Mobile-to-grid middleware: Bridging the gap between mobile and grid environments. In *Europe Grid Computing 2005, LNCS 3470*, pages 932–941, Amsterdam, Netherlands, July 2005.

[121] V. Borges, J. Dias, A. Rossetto, and M. Dantas. Summit: An architecture for mobile devices to coordinate the execution of applications in grid environments. In *Proceedings of 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 217–222, June 2007.

[122] D. Chu and M. Humphrey. Mobile ogsi.net: Grid computing on mobile devices. In *Fifth IEEE/ACM International Workshop on Grid Computing*, pages 182–191, Pittsburgh, USA, November 2004.

[123] P. Ghosh, N. Roy, and S.K. Das. Mobility-aware efficient job scheduling in mobile grids. In *The Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 701–706, Rio de Janeiro, Brazil, May 2007.

[124] S. Choi, I. Cho, K. Chung, B. Song, and H. Yu. Group-based resource selection algorithm supporting fault-tolerance in mobile grid. In *Proceedings of Third International Conference on Semantics, Knowledge and Grid*, pages 426–429, China, October 2007.

[125] S. Isaiadis and V. Getov. Integrating mobile devices into the grid: Design considerations and evaluation. In *Euro-Par*, pages 1080–1088, Lisbon, Portugal, August 2005.

[126] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. In *Proceeding of the ACM/IEEE MobiCom*, pages 59–68, Seattle, Washington, USA, August 1999.

[127] A.K. Dey and G.D. Abowd. Towards a better understanding of context and context-awareness. In *Workshop on The What, Who, Where, When, and How of Context-Awareness, part of Conference on Human Factors in Computing Systems*, pages 304–307, Karlsruhe, Germany, May 1999.

[128] I.C. Millard, D. DeRoure, and N. Shadbolt. The use of ontologies in contextually aware environments. In *First International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 42–47, Nottingham, UK, September 2004.

[129] M. Fernadez-Lopez. Overview of methodologies for building ontologies. In *IJCAI 99 Workshop on Ontologies and Problem Solving Methods*, Stockholm, Sweden, August 1999.

[130] B. Swartout, P. Ramesh, K. Knight, and T. Russ. Toward distributed use of large-scale ontologies. In *Symposium on Ontological Engineering of AAAI*, pages 138–148, Standard, CA, USA, March 1997.

[131] M. Fernadez-Lopez. Methontology: From ontological art towards ontological engineering. In *Symposium on Ontological Engineering of AAAI*, pages 33–40, Stanford, CA, USA, March 1997.

[132] S.G.M. Koo, C. Rosenberg, H.H. Chan, Y.C. Lee, A. Vilavaar, and A. Wenzel. Location-based e-campus web services: From design to deployment. In *First IEEE International Conference on Pervasive Computing and Communication*, pages 207–215, Fort Worth, TX, USA, March 2003.

[133] J. Hightower and G. Borriello. Locating systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, August 2001.

[134] M. Li and Z. Peng. Interface management in enterprises. In *Proceedings of ICSSSM '05. 2005 International Conference on Services Systems and Services Management*, pages 344–346, Chongqing, China, June 2005.

[135] J.A. Robinson, N. Wirth, and R. Kowalski. Prolog tutorial. *Available online at http://cs.wwc.edu/KU/PR/Prolog.html*, 1997.

[136] J. Broekstra, A. Kampman, and F. Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *Proceedings of International Semantic Web Conference*, pages 54–68, Sardinia, Italy, January 2002.

[137] B. McBride. Jena: Implementing the rdf model and syntax specification. In *Proceedings of Semantic Web Workshop*, Hongkong, May 2001.

[138] E. Prud and A. Seaborne. Sparql query language for rdf. *Available online at http://www.w3.org/TR/rdf-sparql-query/*, 2008.

[139] A. Seaborne. A programmer's introduction to rdql. *Available online at http://jena.sourceforge.net/tutorial/RDQL/*, 2004.

[140] N.F. Noy, M. Sintek, S. Decker, M. Crubezy, R.W. Fergerson, and M.A. Musen. Creating semantic web contents with protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[141] L. Clement, A. Hately, C. Riegen, and T. Rogers. Uddi version 3.0.2. *Available online at http://www.uddi.org/pubs/uddi-v3.htm*, 2004.

[142] N. Srinivasan, M. Paolucci, and K. Sycara. Semantic web service discovery in the owl-s ide. In *Proceedings of the 39th Hawaii International Conference on System Sciences*, pages 109–111, Hawaii, USA, January 2006.

[143] S. Majithia, A. Shaikh, O.F. Rana, and D.W. Walker. Reputation-based semantic service discovery. In *Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 297–302, Italy, June 2004.

[144] L. Li and I. Horrock. A software framework for matchmaking based on semantic web technology. In *In. Proc. 12th Int World Wide Web Conference Workshop on E-Service and the Semantic Web*, pages 331–339, Noosa, Australia, July 2003.

[145] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1(1):27–46, 2003.

[146] D. Martin, M. Paolucci, S. Mcilraith, and M. Burstein. Bringing semantics to web services: The owl-s approach. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, CA, USA, July 2004.

[147] M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara. Semantic matching of web services capability. In *Proceedings of the First International Semantic Web Conference*, pages 333–347, Sardinia, Italy, January 2002.

[148] M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara. Importing the semantic web in uddi. In *Proceedings of E-Services and the Semantic Web*, pages 225–236, Toronto, Canada, May 2002.

[149] J. Heflin. Owl web ontology language use cases and requirements. *Available online at http://www.w3.org/TR/2004/REC-webont-req-20040210/*, 2004.

[150] A. Bandara, T. Payne, D. DeRoure, and T. Lewis. A semantic approach for description and ranked matching of services in pervasive environments. In *Applications of Semantic Technologies*, Germany, Sep. 2007.

[151] A. Schwering. Hybrid model for semantic similarity measurement. In *Published by Lecture Notes in Computer Science*, pages 1449–1465, Oct. 2005.

[152] A. Tverski. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

[153] David Martin, Mark Burstein, Ora Lassila, Massimo Paolucci, Terry Payne, and Sheila McIlraith. Describing web services using owl-s and wsdl. *Available online at http://www.daml.org/services/owl-s/1.0/owl-s-wsdl.html*, 2003.

[154] A. Bandara, T. Payne, D. DeRoure, and T. Lewis. A pragmatic approach for the semantic description and matching of pervasive resources. In *3rd International Conference on Grid and Pervasive Computing*, China, May 2008.

[155] V. Haarslev and R. Moller. Racer: a core inference engine for the semantic web. In *Proceedings of Second International Workshop on Evaluation of Ontology-based Tools*, pages 27–36, Sanibel Island, FL, USA, October 2003.

[156] P. Livet. What is transparency? *Available online at http://psyche.cs.monash.edu.au/symposia/metzinger/Livet.pdf*, 2005.

[157] F. Cali, M. Conti, and E. Gregori. Ieee 802.11 protocol: design and performance evaluation of anadaptive backoff mechanism. *IEEE Journal on Communications*, 18(9):1774 – 1786, 2000.

[158] J. Schneider, B. Linnert, and L. Burchard. Distributed workflow management for large-scale grid environments. In *Proceedings of the International Symposium on Applications on Internet*, pages 229–235, Washington, DC, USA, January 2006.

[159] OMIIUserGuide. Omii server stack. *Available online at http://www.omii.ac.uk/docs/*, 2007.

[160] J. Simpson, E. Weiner, and M. Proffitt. *Oxford English Dictionary (Second Edition)*. Oxford University Press, 1989.

[161] P. Fishwick. What is simulation. *Available online at http://www.cis.ufl.edu/ fishwick/introsim/node1.html*, 1995.

[162] P. Ball. Introduction to discrete event simulation. In *DYCOMANS Workshop II: Management and Control: Tools in Action*, pages 367–376, Monterey, California, May 1996.

[163] W. Kreutzer. *Systems Simulation - Programming Styles and Languages*. Addison-Wesley, 1986.

[164] A.M. Law and W.D. Kelton. *Simulation modeling and analysis - third edition*. McGraw-Hill Book Company, 2000.

[165] J. Banks, J. Carson, B. Nelson, and D. Nicol. *Discrete-event system simulation - fourth edition*. Pearson, 2005.

[166] J.F. Klingener. Programming combined discrete-continuous simulation models for performance. In *Proceedings of the 28th conference on Winter simulation*, pages 833–839, Coronado, California, United States, December 1996.

[167] S. Robinson. *Simulation - The practice of model development and use*. Wiley, 2004.

[168] M.A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing, 1995.

[169] C.R. Harrell and R.N. Price. Simulation modeling using promodel technology. In *Proceedings of the Winter Simulation Conference*, pages 192–198, San Diego, CA, United States, December 2002.

[170] C.A. Petri. Communication with automata. *New York:Griffiss Air Force Base. Tech. Rep. RADC-TR-65-377*, 1(1), 1966.

[171] K. Mcleish. What is a petri net. *Available online at http://www.cse.fau.edu/ maria/COURSES/CEN4010-SE/C10/10-7.html*, 1999.

[172] G. Rozenberg. *Advances in Petri Nets*. Springer, 457 pages, 1995.

[173] M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models*. Johns Hopkins University Press, Baltimore, 1981.

[174] M.K. Molloy. *On the Integration of delay and throughput measures in distributed processing models*. Phd Thesis, UCLA, 1981.

[175] M.A. Marsan, G. Balbo, and G. Conte. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[176] G. Ciardo, J. Muppala, and K.S. Trivedi. Spnp: Stochastic petri net package. In *IEEE International Workshop on Petri Nets and Performance Models*, pages 142–151, Kyoto, Japan, December 1989.

[177] H. Choi, V.G. Kulkarni, and K. Trivedi. Markov regenerative stochastic petri nets. *Performance Evaluation*, 2(1-3):337–357, 1994.

[178] A. Bobbio, A. Puliafito, M. Scarpa, and M. Telek. Webspn: Non markovian stochastic petri net tool. In *18th International Conference on Application and Theory of Petri Nets*, Williamsburg, VA, USA, June 1997.

[179] A. Puliafito, S. Riccobene, and M. Scarpa. Which paradigm should i use?: An analytical comparison of the client-server, remote evaluation and mobile agents paradigms. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 278–292, August 2001.

[180] S. Mcilraith and T. Son. Adapting golog for composition of semantic web services. In *Proceedings of the eight International Conference on Knowledge Representation and Reasoning*, pages 482–493, Toulouse, France, April 2002.

[181] S. Mcilraith, T. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[182] D.S. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 968–973, Sydney, Australia, 1999.

[183] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating daml-s web services compostion using shop2. In *Proceedings of the Second International Semantic Web Conference*, pages 195–210, Florida, USA, September 2003.

Publications

This Appendix includes the papers which have been published.