GridLite: A Framework for Managing and Provisioning Services on Grid-Enabled Resource Limited Devices

Raj Kumar, Xiang Song
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2005-146
August 9, 2005*

Grid, Web services, handheld device, resource constrained device, music management

GridLite is an extensible framework that provides services to users on ubiquitous, resource-limited devices within a Grid infrastructure. It uses a server infrastructure for provisioning of persistent services, and smart helper services running on the "lite" devices which tap into this infrastructure. One of the goals of GridLite research is to define a Grid architecture which manages these devices such that their resource constraints are minimized by the intelligent Grid infrastructure. This is done by defining new services for managing various resources. In this paper, we discuss the background, requirements and architecture of GridLite. We also demonstrate a proof of concept by implementing GridLite architecture in our test bed.

* Internal Accession Date Only
Approved for External Publication

GridLite: A Framework for Managing and Provisioning Services on Grid-Enabled Resource Limited Devices

Raj Kumar[1], Xiang Song[1,2]

[1]*Hewlett-Packard Company, OST/HP Labs, 1501 Page Mill Rd, Palo Alto, CA 94304*
[2]*Georgia Institute of Technology, College of Computing, 801 Atlantic Dr, Atlanta, GA 30332*
raj.kumar@hp.com, xiang.song@hp.com

Abstract

GridLite is an extensible framework that provides services to users on ubiquitous, resource-limited devices within a Grid infrastructure. It uses a server infrastructure for provisioning of persistent services, and smart helper services running on the "lite" devices which tap into this infrastructure. One of the goals of GridLite research is to define a Grid architecture which manages these devices such that their resource constraints are minimized by the intelligent Grid infrastructure. This is done by defining new services for managing various resources. In this paper, we discuss the background, requirements and architecture of GridLite. We also demonstrate a proof of concept by implementing GridLite architecture in our test bed.

1. Introduction

Grid computing [1, 2] has made rapid strides during the last few years from their first use in the scientific computing domain to enterprise Grids deploying commercial applications. We believe the next phase of Grid computing will deal with making grid services available to resource constrained appliances such as handheld devices, PDAs, smartphones, and sensors to name a few.

Provisioning Grid services on such appliances can be a daunting task because of severe limitations in system resources such as processing power, memory, storage, network bandwidth, and battery power. Additional difficulties include stripped down runtime environments, APIs, and applications.

GridLite is an extensible framework that provides services to mobile users on ubiquitous, resource-limited devices within a Grid infrastructure. The GridLite architecture is adaptive, intelligent, and uses emerging industry standards such as XML, Web services, WSRF [3], and WSDM [4]. The context of GridLite applies to all appliances that are connected to a LAN, WAN or the public Internet. GridLite uses a server infrastructure for provisioning of persistent services, and smart helper services running on the "lite" devices which tap into this infrastructure. It builds on the research done in the areas of ubiquitous computing and Grid computing during last several years. One of the goals of GridLite is to provide mobile users ubiquitous access to Grid resources, higher productivity and entertainment. The client side of GridLite implementations includes PDAs such as iPAQs, smart phones and other handheld devices, and the server side includes Linux and Windows servers.

Another goal of GridLite research is to define a Grid architecture which manages these devices such that their resource constraints are minimized by the intelligent GridLite infrastructure. This is done by defining new services for managing various resources. For example, one of our services deals with the limited storage in a handheld device (tens of megabytes for both memory and storage). In order to provide the illusion of an unlimited storage, we have implemented a storage management service which manages a device's storage. When the available storage falls below a predefined threshold (based on a policy engine), a background service flushes unused data content to a server in the Grid infrastructure. Another service we have implemented manages applications on the handheld device. If a user needs to launch an application, the service automatically checks for the existence of the application and downloads it from the Grid infrastructure if it is not currently on the device. We also use predictive algorithms to anticipate use of an application in the near future, which could be based on, for example, recurring usage, and potentially download the application before the user actually requires the service. Similarly, if an application is predicted to not be used in the near future, it may be uninstalled to free up resources for other applications.

In addition to architecting new management services for devices, we are defining several utility services which provide value to the end user. One service we have implemented deals with music distribution and

management in the context of a Grid infrastructure. A handheld device user would be able to define multiple profiles of music preferences to the Grid infrastructure. The user would also be able to specify patterns of their use in future. Once this is done, various services operating in the Grid infrastructure (servers), and the handheld device would work collaboratively to provide the user a streamlined music experience without the user being aware of the limitations of the handheld device.

The contributions of this paper include: (1) define a framework for managing and provisioning services on Grid-enabled resource limited devices, (2) demonstrate a test bed where some core services in GridLite are implemented and (3) show in the music service that GridLite can extend the notion of SOAs (Service Oriented Architectures) [9] down to the smallest of devices.

In section 2, we will outline the technology landscape, and show how it drives the goals and requirements for GridLite research. Section 3 will then discuss GridLite architecture in detail and section 4 will present the test bed we built to demonstrate the functionalities of GridLite. We will show the related works in section 5 and draw the conclusions in section 6.

2. Technology landscape and GridLite requirements

Recent years have seen the growth of Grid computing, and ubiquitous/pervasive computing as distinct domains of research. Grid computing has advantages over traditional computing technology in that it provides a virtual aggregation of many resources and manages them dynamically at runtime. The concept of ubiquitous computing allows users to use computing facilities whenever and wherever they want. When these two technologies converge, there should be synergistic benefits by addressing complimentary needs of users: Grids could potentially solve the distributed resource management problem whereas ubiquitous computing could provide services to mobile users on handheld devices which are also managed by the Grid infrastructure. We propose, in this paper, GridLite middleware that extends Grid computing to ubiquitous devices. It provides all the features of Grid computing to handheld devices, like information services, security services, data services and resource management services. We implement the prototype of GridLite in some typical ubiquitous devices – handheld devices.

There are different kinds of handheld devices currently in the market. Apple iPod, HP iPAQ and Samsung smartphone are a few examples. iPod is a single function music player and typically has tens of gigabyte of storage. Users could download music files into iPod and play it wherever they want. iPAQ on the other hand is a PC-like multi-function device. It just behaves as a trimmed PC and can do many things that a PC can do. The storage of iPAQ is tens of megabytes. Smartphone has less functionality than iPAQ but is GPRS-enabled. It could be always connected through the cell phone network. Besides making telephone calls, it could also perform some basic processing on documents, emails and media files.

GridLite basically focuses on the latter two devices. Compared to iPods, iPaqs and smartphones are multi-functional, network-enabled and programmable devices. With the underlying infrastructure and server side support, these devices could provide data to the user ubiquitously. GridLite-enhanced iPaqs and smartphones will give the user more flexibility and freedom to use their data.

Handheld devices have some constraints when they are in use. The typical limitations of these devices are:

- Limited memory and storage (2MB-64MB)
- Limited battery power
- Limited computing power
- Limited network bandwidth
- Limited display and input capabilities

GridLite architecture addresses several of these concerns as described in the next section.

Furthermore, the processors, operating systems and APIs for each handheld device vary a lot. It makes the development on these handheld devices more difficult. We chose to use java for development since java is a platform independent programming language. We have the same java virtual machine, IBM J9 [5], for PocketPC/Smartphone OS, Linux and Windows XP. The development is much easier when the same code can be running on JVMs for different platforms. There are mainly three flavors of Java, designed for different platform environment: J2EE, J2SE and J2ME. We choose J2ME [6] for GridLite since it is designed for resource constrained handheld and mobile devices. Within J2ME, various configurations and

profiles are designed for different kind of devices. Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC) are two configurations Sun currently supports. They provide the basic functionality for a particular range of devices that share similar characteristics, such as network connectivity and memory footprint [6]. For example, CLDC is basically for low-end devices that typically have less than 2MB storage, while CDC supports high-end devices that have tens of megabyte storage. A profile supports a narrower category of devices within the framework of a chosen configuration. Profiles provide a complete set of APIs for particular devices (for example, user interface API).

GridLite is not designed for a particular class of devices but general resource limited mobile devices. Thus, it could be implemented in various configurations/profiles combination. In the test bed we built, we chose CDC + Personal Profile for HP iPAQ with PocketPC operating system. Note that it is not the only way to implement GridLite.

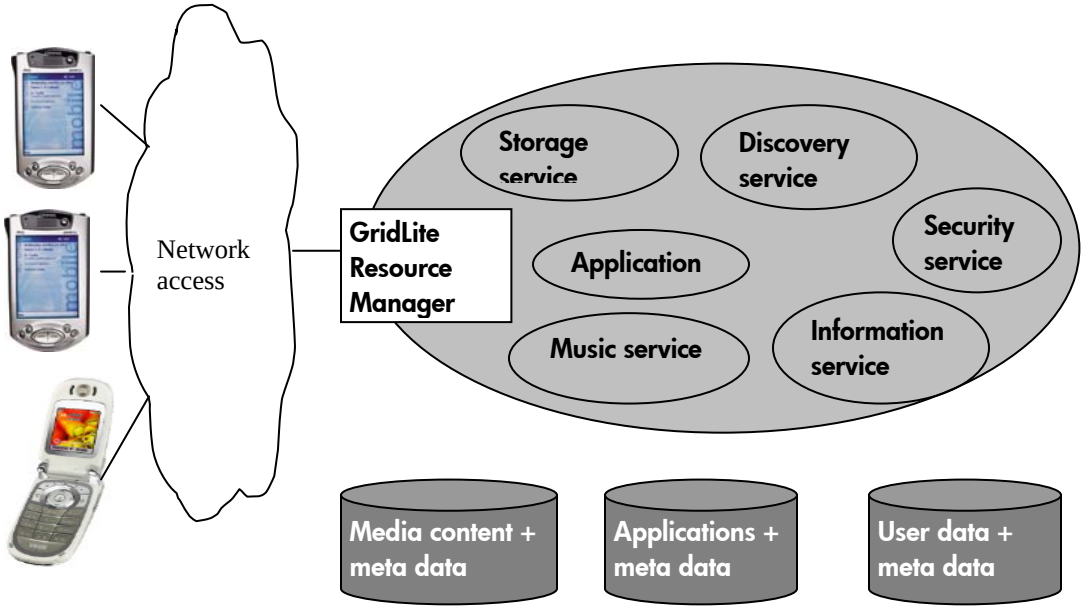

Figure 1. GridLite Infrastructure

3. GridLite architecture

As mentioned before, GridLite is an extensible framework that provides Grid services to users ubiquitously on handheld, mobile or embedded devices. The basic infrastructure is shown in Figure 1.

Basically, GridLite has a server side running in a wired infrastructure, providing all the Grid services to the clients. Client side could be on any ubiquitous device like iPaqs or cell phones. Clients will contact the server through the server portal – GridLite Resource Manager (GRM). The GRM will allocate the proper resources for servicing the clients' requests within the server side.

Each service in GridLite consists of two parts: client side intelligent helper service and server side service. The services could be classified into three categories based on the functionality. Core services are the services critical to all other services. For example, storage management service is a core service. Utility services are services that are designed for some particular domain. They need the support from core services to run. Music service is an example of a utility service. Business services are services designed to implement certain business processes.

Figure 2 shows the software stack of GridLite. Besides the hardware, OS and runtime environment, there are three service layers on the server side and three corresponding helper service layers on client side. Like OSI model, each helper service layer on client side talks to the corresponding server layer directly without knowing the underlying infrastructure. The application layer on client side refers to the applications already installed on the devices. GridLite services may invoke these applications to satisfy user requests.

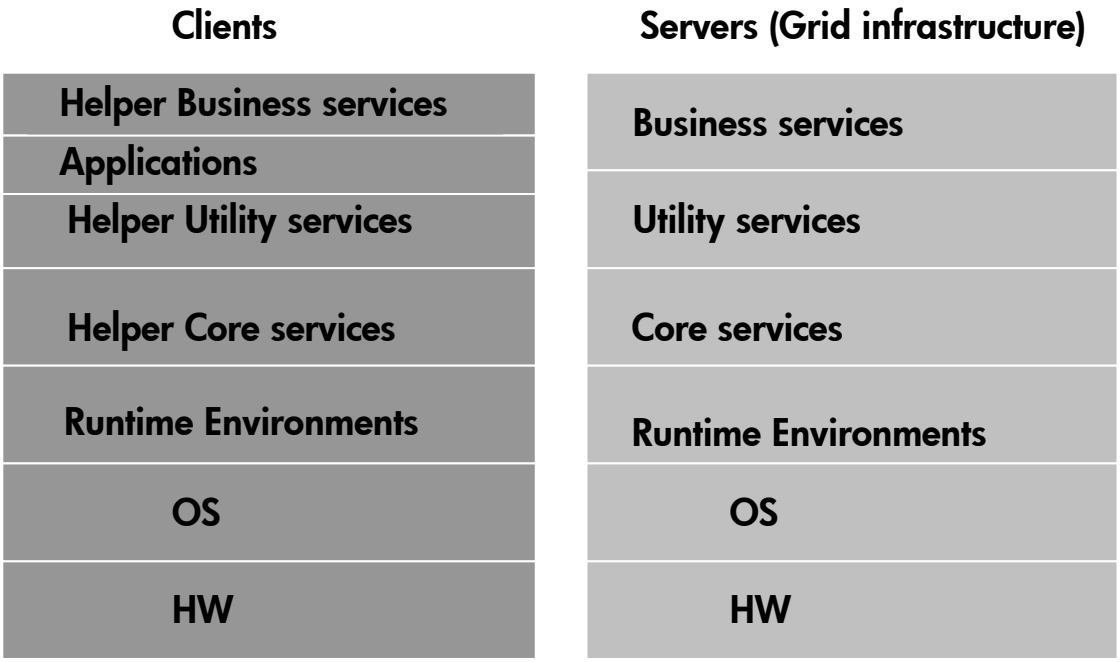

Figure 2. GridLite Software Stack

A lot of services could be built on top of GridLite infrastructure. Storage management service is able to provide users illusion of an unlimited storage on handheld devices. Users are freed from mundane operations such as copying files back and forth in order to free enough storage space. Application management service could manage the applications installed on the devices intelligently so that users can just launch the applications without knowing where they are. Music service will provide music entertainment to users and ease the users from the hassle of downloading music before listening. For all the services, the grid infrastructure running on the server side is transparent to the users. Users use the services as if they are running locally on the devices.

We have implemented the above three services in GridLite and will discuss them in detail in section 4. There are several other services that could be implemented in GridLite framework. Here is a list of other services we consider useful:

- Billing services
- Notification service
- Capacity planning service
- Data mining service
- Caching service
- Health service
- Photo service

4. Test bed and provisioned services

As stated in section 3, GridLite architecture supports a variety of services. We built two core services, and one utility service to show how GridLite could be used: application management service, storage management service and music service. In this section, we will discuss the development and runtime environment, use cases and implementation details of these services.

4.1 Development and runtime environment

4.1.1 Hardware

Our test bed consists of one or more Windows or Linux servers which provide the wired infrastructure. For wireless devices, we use HP 4300 series and 5500 series iPaqs with WiFi 802.11 wireless card installed. The wireless network is the standard IEEE 802.11b within HP Labs. We also test our services using Samsung i600 smart phone with GPRS connection (Verizon) to Internet.

4.1.2 Software

On the server side, we have Apache Tomcat 5.0 and MySQL server 4.0 installed as windows services, running in Microsoft Windows XP Professional with Service Pack 1. On wireless devices, IBM J9 Java virtual machine is installed as a runtime environment (on top of PocketPC OS on iPAQ and Smartphone OS on cell phone).

IBM J9 supports multiple versions of Java on devices. We use J2ME CDC configuration and Foundation/Personal profile for our services. Most of the services are built in Java, but in some special cases we use native C/C++ library to implement some basic functionality.

For development purposes, we also install Microsoft Embedded C++ 4.0 and PocketPC 2003 SDK (Smartphone 2003 SDK) for building native C/C++ library. IBM Websphere Device Developer is used for developing Java program on the devices. These development tools are not required for running our services.

4.2 Storage Management Service

4.2.1 Functionality

In the storage management service, we provide illusion of unlimited storage space for the user. The user can just use the disk space as if it is unbounded. With the help of wired infrastructure, the storage service could upload/download the user's files back and forth, depending on the specified policy (e.g. LRU) and the available disk space. Users will be freed from the "out of space" error and enjoy the infinite storage provided by our services.

Storage management service could be used by other services in case those services need disk space. We will show how the storage management service is integrated into music service later in this section.

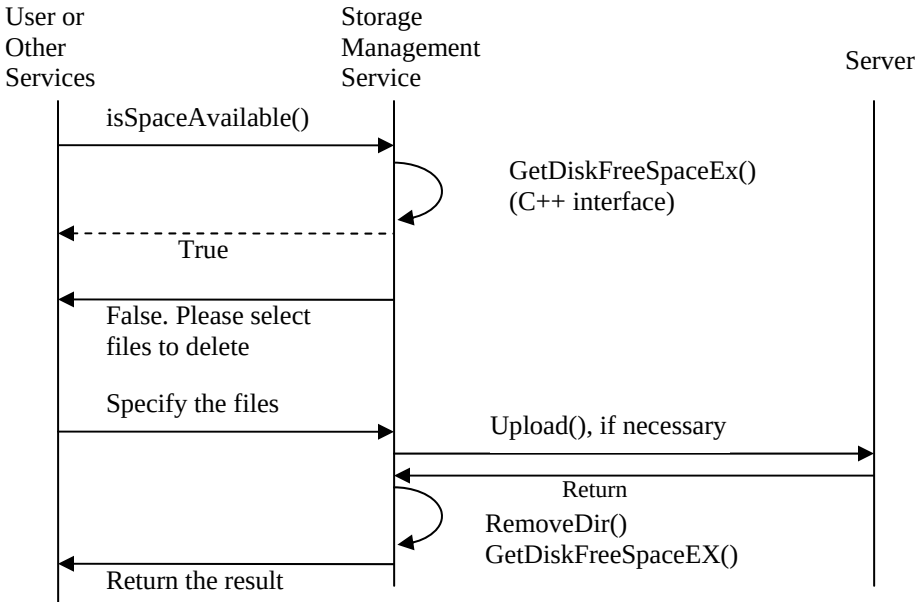

Figure 3. Sequence Diagram of Storage Management Service

4.2.2 Implementation

In the following figure, we show the methods we used in the storage management service on both the wired and wireless devices. Note that we cannot check available disk storage in Java, so we use C++ API provided by Microsoft PocketPC 2003 SDK to do it and then pass the result to Java using JNI (Java Native Interface).

4.3 Application Management Service

4.3.1 Functionality

Application management service allows users to launch applications freely without worrying about where it is located. Application management service provides a wrapper service for each application and creates an icon for each application wrapper. What the user needs to do is just click the icon. The wrapper service will find out if the application is installed locally on disk. If yes, it will launch it immediately. Otherwise, the wrapper service will download and install the application before launching it.

4.3.2 Implementation

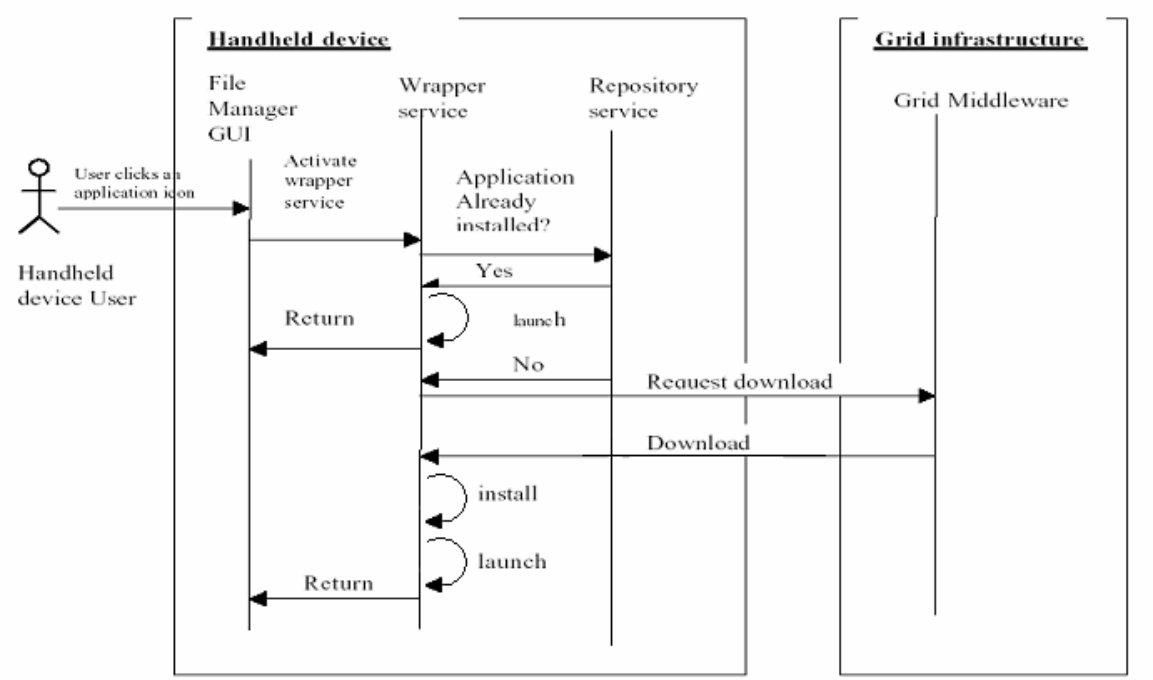

Figure 4. Sequence Diagram of Application Management Service

Figure 4 shows how the application service works. We use C methods to create processes in PocketPC OS since J2ME CDC doesn't provide this functionality.

4.4 Music Service

4.4.1 Functionality

Unlike the other two services, music service is not a core service but a utility service. It is applicable to a particular domain – service-oriented entertainment. It has a server side infrastructure, which allows users to create accounts, purchase music and create profiles, and a client side agent running on PDA, which downloads and plays music on demand. The service-oriented model gives the users more freedom because the music being played doesn't need to be stored locally when users start the service. All the information is stored on the server side and users could listen to the music whenever and wherever they want in a Wi-Fi enabled campus.

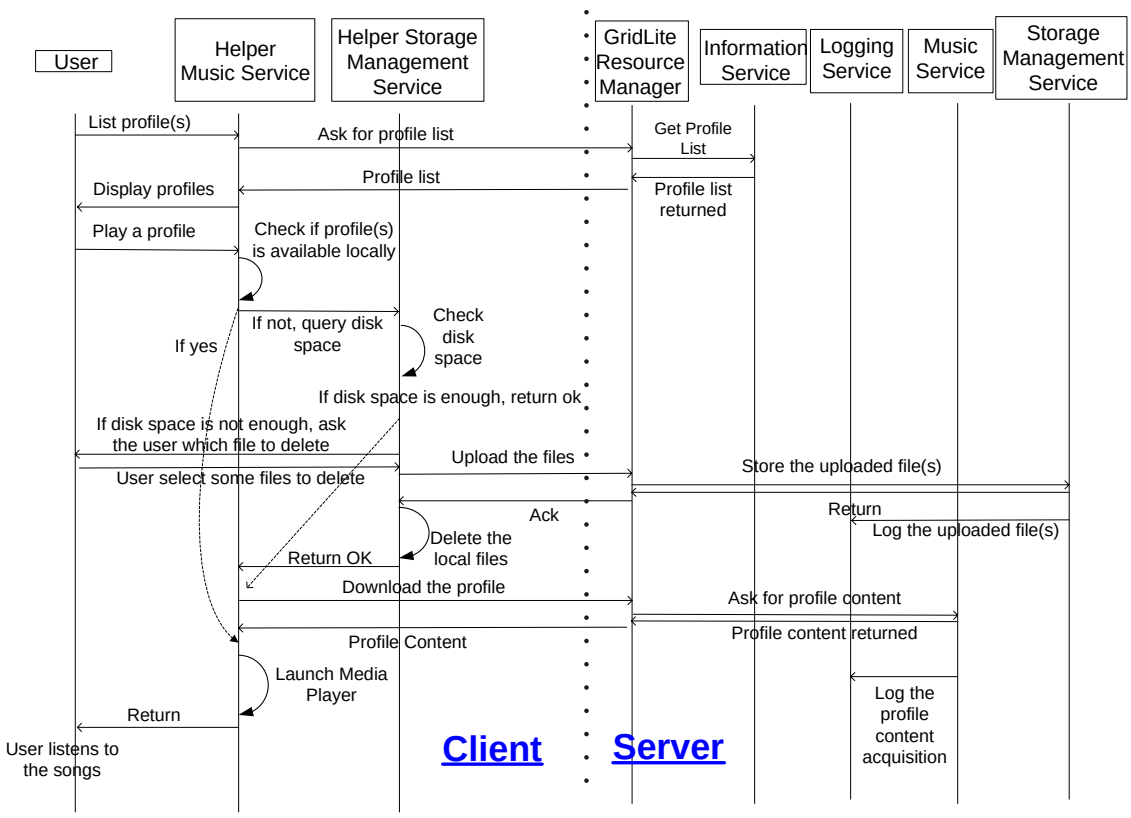

Figure 5. Sequence Diagram of Music Service

4.4.2 Implementation

Figure 5 shows how our music services work. Basically, it combines the idea of the core services and provides a real service to enhance the users' experience. A detailed implementation of this service is beyond the scope of this paper.

Since the devices are resource constrained, the download-and-play model may not be suitable for them. A better way is to play music on the fly. In this case, the pre-fetching of some music files is a good idea to enhance the user's experience. However, if we use windows media player, we have no control over the playback. For example, we don't know when the song is finished or where the user pauses the song. That makes the pre-fetching very difficult. In order to circumvent this problem, we have architected a custom mp3 music player that is tightly coupled to the music service. As the media content is consumed by the player, it periodically communicates to the music service the amount of media content still left to play. The music service uses this information to implement a closed loop prefetch of media content from the grid infrastructure which is synchronised with the consumption of the media content. Currently this custom player is functional on a laptop client, but needs more work on the iPAQ.

5. Related Work

Obje Software Architecture, developed by PARC research, is an interconnection technology that enables digital devices and services to easily interoperate over both wired and wireless networks [7]. It provides a "meta-standard" that will allow people to connect devices more easily, and on the fly. But this technology doesn't have Grid at the back-end. Thus it cannot provide the services that Grid supports. As these resource-limited devices become increasingly pervasive, not only the connection, but also the management of these devices becomes important. GridLite incorporates Grid technologies into devices to fulfill this requirement.

Apple iPod + iTunes is a music distribution service currently in the market [8]. Compared to our music service, iPod is a standalone device (never connected) and can do only primitive operations. The business

model of iPod + iTunes is buy-and-save model: There will be plenty of storage available on iPod. When people buy a song from iTunes, they can download it into the iPod for future use. However, our music service in GridLite could do much more to enhance the user's experience. While we can implement an iTunes-like model in GridLite, we have another choice of the model—a service-oriented architecture (SOA) [9] —which is implemented in our test bed. Our music service allows people to buy music and store the information on the sever side. They are able to download and listen to the music on demand without worrying about where it is stored. We provide services in terms of managing a user's media content as well as the handheld device, not just the music downloading, to people to enhance their experience and ease of use.

Visual Radio [10] is a small device which could provide interactive audio services to people through a traditional radio channel and a parallel cellular network. People are able to get detailed information about the song they are listening to and buy the song or concert tickets through a special offer. However, compared to our music services, Visual Radio only has connected operation. There is no client-side management support or various server-side services. The music service in GridLite, on the other hand, supports both standalone and connected operations, and has significant infrastructure support. It could provide more services than Visual Radio such as customized profiles of songs and location-based music recommendation. With the management of client-side devices, all the background services are transparent to users so that they just use the intelligent device as if it has unlimited resources.

File sharing is another method to deal with storage limitation on handheld devices. Users can manually copy files back and forth between devices and remote server. PocketPC OS doesn't support file sharing directly, but some third party software is available to enable network mapping and sharing, such as PocketLAN [11], GotoMyPC [12] and Total Commander [13]. However, because of lack of Grid infrastructure support on the backend, these software services cannot support automatic file management as our storage management service does.

While we have implemented utility services dealing with music distribution and management, the scope of GridLite is much broader and covers many other application domains. As far as we can tell, GridLite is a first attempt where a grid infrastructure actively manages a resource constrained device to enable it to tap into various services provided by the Grid.

6. Conclusion and Future Work

In this paper, we have proposed GridLite, an extensible framework that provides Grid services to ubiquitous users on resource constrained appliances (mobile, handheld or embedded devices). GridLite complies with industry standards like XML, SOAP, WSDL and WSRF. Devices powered by GridLite are able to provide intelligent services to users which tap into the grid infrastructure to provide illusion of unlimited resources. GridLite also enables automatic management of client side resources by the grid infrastructure.

We built some core services in GridLite like storage management service, application management service and utility services like a music management and distribution service, to show the functionalities of GridLite in real usage. Developers can build many other services in GridLite and take advantage of it to better service end users.

As grid computing and ubiquitous computing technologies continue to evolve, this would enable provisioning of new services to users on resource constrained devices using a grid infrastructure. We believe GridLite is a first effort that attempts to exploit both of these areas of research with interesting results.

In the near future, we will investigate how the GridLite framework can work with Web Service Resource Framework (WSRF) [3]. With the release of all six specifications of WSRF, we should be able to integrate WSRF into GridLite and make GridLite accessible through this common standard.

Acknowledgement

We would like to acknowledge the support, and key inputs to this investigation from Greg Astfalk. We would also like to acknowledge our discussions on web services, and WSRF with Latha Srinivasan and Jem Treadwell.

References

[1] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". *International J. Supercomputer Applications*, 15(3), 2001.
[2] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration" *Open Grid Service Infrastructure WG, Global Grid Forum*, June 22, 2002.
[3] Karl Czajkowski, Donald F Ferguson, Ian Foster, Jeffrey Frey , Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, William Vambenepe
"Web Service Resource Framework", http://www.globus.org/wsrf/specs/ws-wsrf.pdf
[4] OASIS Web Services Distributed Management (WSDM) TC http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm
[5] IBM Websphere and J9 Java Virtual machine
http://www-306.ibm.com/software/wireless/wsdd/
[6] Java 2 Platform, Micro Edition (J2ME); JSR 68 Overview http://java.sun.com/j2me/overview.html
[7] The Obje™ Software Architecture, PARC research,
http://www.parc.xerox.com/research/csl/projects/obje/default.html
[8] Apple iPOD + iTunes, http://www.apple.com/itunes/
[9] Ahmed Elfatatry, Paul Layzell, "Negotiating in service-oriented environments", *Communications of the ACM*, Volume 47, Issue 8 (August 2004)
[10] Visual Radio http://www.visualradio.com/
[11] PocketLan http://www.pocketgear.com /software_detail.asp?id=2825
[12] GotoMyPC https://www.gotomypc.com/
[13] Total Commander http://www.ghisler.com/pocketpc.htm
[14] JLayer MP3 decoder http://www.javazoom.net/javalayer/javalayerme.html
[15] MPEGJava http://www.joeshmoe.com/mpegjava/
[16] Winfoot SOAP http://www.wingfoot.com/products.jsp
[17] I. Foster (ed), J. Frey (ed), S. Graham (ed), S. Tuecke (ed), K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, S. Weerawarana, "Modeling stateful resource with Web services" http://www-106.ibm.com/developerworks/library/ws-resource/ws-modelingresources.htm
[18] Telescience: A Collaborative Environment for TelemicroscopyTM and Remote Science, https://donor.ucsd.edu/
[19] gLite project: http://glite.web.cern.ch/glite/