Service Discovery in the Future for Mobile Commerce

by *Dipanjan Chakraborty* and *Harry Chen*

Introduction

This article focuses on the effect of mobile computing on electronic commerce and discusses the technology behind the change that is taking place. It explains different ways of discovering services in a dynamic environment. The increased use of Personal Digital Assistants like PalmPilots provides users with the capability of carrying out electronic transactions from a mobile platform. This has coined in a new term "Mobile Commerce". Electronic Commerce can be defined in general as an aggregation of all transactions on the Internet. When these types of applications and transaction processing facilities need to be accessed from a wireless device, a new domain of research area and challenging problems come into the picture. As is immediately evident, Mobile Commerce has to deal with the new problem of finding location dependent data.

Services can be defined as facilities that are available to a computer. A computer connected to a distributed system has access to the printer, copier, scanner, etc. If it is a node wired to the Internet, it can access prices of books through web sites and carry on various other transactions. As businesses offer various types of heterogeneous services to their customers, finding a service efficiently becomes important. This article summarizes existing architectures for discovering services, compares and contrasts the existing infrastructures, and explains some of the inherent drawbacks. We also present the Ronin Agent framework and the work done in the XReggie project to improve upon the existing techniques of service discovery [4, 6].

Is it possible?

Jill is driving down the highway. The car has an onboard computer. She herself has a cellular phone and a PalmPilot. The car's onboard computer connects with computers in nearby cars using its built-in wireless device and exchanges information regarding the traffic ahead, accident information, etc. Jill is running out of gas and wants to find the nearest gas station nearby. Her onboard computer or her PalmPilot can do this job for her. The onboard computer connects to other onboard computers of cars passing in the opposite direction using adhoc networking protocols and tries to glean the information on

whether any car has refueled recently and also tries to know the location of the gas station. In the meantime, her PalmPilot is searching out for gas station agents in the vicinity. Soon, it gets a reply from one and figures out its position. The PalmPilot also finds a map agent who feeds the driving directions to the gas station into the palmtop.

Jill has no problem filling up.

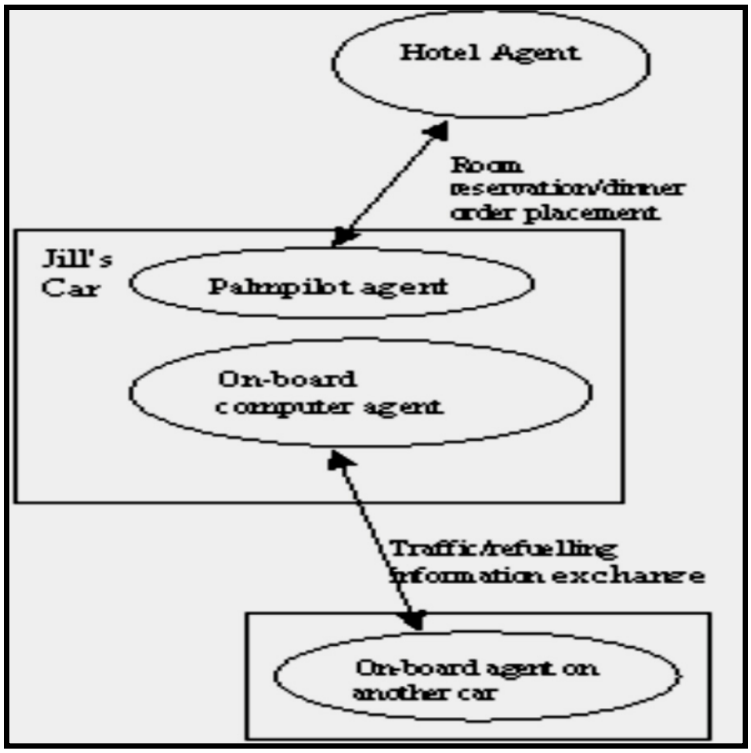

Figure 1: Interactive Services

She is driving the whole day and its almost evening. Jill decides to call it a day. Jill tells her PalmPilot to book a room at the nearest hotel. The palm device contacts the proxy agents for the local hotel/restaurant associations and gets the best possible deal on the room she wants. Jill wants her Mexican style dinner ready when she arrives at the hotel. The agent in her PalmPilot places this request to be executed by the hotel agent. Driving directions are obtained as usual. Figure 1 gives a brief overview of the communication going on within different agents during the whole episode.

Technical Analysis

There are some basic assumptions in realizing the above scenario. We assume that the level of networking and computing hardware will increase consistently in the recent future. We assume the emergence of PalmPilots and Bluetooth like systems, which would provide short-range moderate bandwidth connections at very low cost. We also assume the existence of widely deployed and easily accessible wireless Local Area Networks (LAN) and Wide Area Networks (WAN).

Bluetooth technology is essentially a cable replacement technology where two Bluetooth enabled devices can communicate efficiently with each other using radio frequency. It is a joint achievement of nine leading companies within the telecommunications and computer industry.

What technologies do we need to make the above scenario feasible? There should be service agents in the vicinity to receive the requests and process them. But another important question bubbling up in a computer scientist's mind would be "How does my PalmPilot know about the electronic facilities available around me?" Discovering these types of services dynamically is becoming increasingly important in the present scenario. In the near future, a service would be selected automatically for a job, taking into consideration its physical location, previous history, and various other semantic information. The discovery mechanism needs to move beyond trivial attribute or interface matching. The discovery mechanism would be much more knowledge based, and artificial intelligence would play a major role in service selection. While trying to build a efficient service discovery architecture, some of the high level questions a researcher might try to address are:

1. Efficiency of the discovery mechanism: How quickly and how cheaply can a service be discovered?

2. Reliability of the underlying communication mechanism: If the underlying communication mechanism is unreliable, then the protocol has to adjust accordingly.

3. Service lifetime management: In a dynamic environment, services might be available only for a bounded amount of time.

4. Security issues: malicious users should not be able to access secure services.

We venture out into the Internet and travel from network to network to find the existing service discovery protocols. We try to identify some of the inherent problems associated with the existing discovery infrastructures. Subsequently, we describe the work we have done in the service discovery arena to make service discovery more dynamic.

Service Discovery Survey

Service Location Protocol (SLP)

The Service Location Protocol (SLP) is a product of the Service Location Protocol Working Group (SVRLOC) of the Internet Engineering Task Force (IETF) [9, 10]. It is a protocol for automatic resource discovery on Internet Protocol based networks. SLP is a language independent protocol. Thus the protocol specification can be implemented in any language. It bases its discovery mechanism on service attributes, which are essentially different ways of describing a service. It can cater to both hardware and software forms of services.

The SLP infrastructure consists of three types of agents:

1. User Agents
2. Service Agents
3. Directory Agents

The User Agents acquire service handles for end user applications that request the services. The Service Agents are responsible for advertising service handles to the Directory Agents thus making services available to the User Agents. The Directory Agent maintains a list of the advertised services in a network. In the above scenario, the agent residing in the palm pilot is the user agent and the agent keeping track of hotel information is the service agent. SLP offers the following services:

1. Obtaining service handles for User Agents.
2. Maintaining the directory of advertised services.
3. Discovering available service attributes.
4. Discovering available Directory Agents.
5. Discovering the available types of Service Agents.

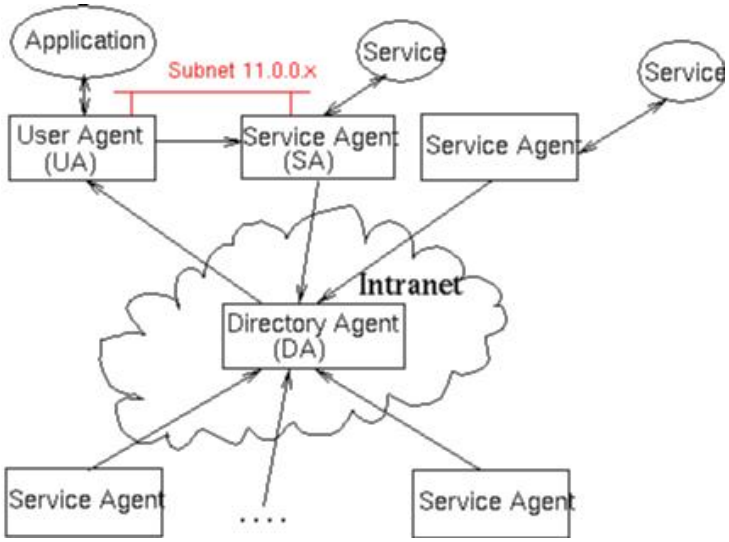

Figure 2: The basic transactions of the Service Location protocol:

A service is described by configuration values of the attributes possible for that service. For instance, a service that allows users to download audio or video content can be described as a service that is a pay-per-use real-time service or a free-of-charge service. The SLP also supports a simple service registration leasing mechanism that handles the cases where service hardware is broken but the services continue to be advertised.

Jini: A service discovery architecture based on Java

Jini is a distributed service-oriented architecture developed by Sun Microsystems [1]. Jini services can represent hardware devices, software programs, or a combination of the two. A collection of Jini services forms a Jini federation. Jini services coordinate with each other within the federation. The overall goal of Jini is to turn the network into a flexible, easily administered tool on which human and computational clients can find services in a flexible and robust fashion. Jini is designed to make the network a more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly.

One of the key components of Jini is the Jini Lookup Service (JLS), which maintains dynamic information about the available services in a Jini federation. Every service must discover one or more Jini Lookup Service before it can enter a federation. The location of the JLS could be known before hand, or they may be discovered using multicast. A JLS can potentially be made available to the local network or other remote networks (i.e. the Internet). The JLS can also be assigned to have group names so that a service may discover a specific group at its vicinity.

When a Jini service wants to join a Jini federation, it first discovers one or many JLSs from the local or remote networks. The service then uploads its service proxy (i.e. a set of Java classes) to the JLS. The service clients can use this proxy to contact the original service and invoke methods on the service. Since service clients only interact with the Java-based service proxies, this allows various types of services, both hardware and software services, to be accessed in a uniform fashion. For instance, a service client can invoke print requests to a PostScript printing service even if it does not have any knowledge about the PostScript language.

A user searching for a service in the network first multicasts a query to find the JLS in the network. If a JLS exists, the corresponding remote object is downloaded into the user's machine. The user then uses this object to find its required service. In Jini, service discovery is done by interface matching or Java attribute matching. If the JLS contains a valid service implementing the interface specified by the user, then a proxy for that service is downloaded in the user's machine. The proxy is used henceforth to call different functions offered by the service.

Universal Plug and Play (UPnP): Microsoft's solution to service discovery

Universal Plug and Play (UPnP), pushed primarily by Microsoft, is an evolving architecture designed to extend the original Microsoft Plug and Play peripheral model to a highly dynamic world of many network devices supplied by many vendors [12]. UPnP works primarily at lower layer network protocols suites (i.e. TCP/IP), implementing standards at this level. UPnP attempts to ensure that all device manufacturers can quickly adhere to the proposed standard without major hassles. By providing a set of defined network protocols UPnP allows devices to build their own Application Programming Interfaces that implement these protocols - in whatever language or platform they choose.

UPnP uses the Simple Service Discovery Protocol (SSDP) to discover services on Internet Protocol based networks. SSDP can be operated with or without a lookup or directory service in the network.

SSDP operates on the top of the existing open standard protocols, using the Hypertext Transfer Protocol over both unicast User Datagram Protocol and multicast User Datagram Protocol. The registration process sends and receives data in hypertext format, but has some special semantics.

When a service wants to join the network, it first sends out an advertise (or announcement) message, notifying the world about its presence. In the case of multicast advertising, the service sends out the advertisement on a reserved multicast address. If a lookup or directory service is present, it can record such advertisements. Meanwhile, other services in the network may directly see these advertisements as well. The "advertise" message contains a Universal Resource Locator (URL) that identifies the advertising service and a URL to a file that provides a description of the advertising service.

When a service client wants to discover a service, it can either contact the service directly through the URL that is provided in the service advertisement, or it can send out a multicast query request. In the case of discovering a service through the multicast query request, the client request may be responded by the service directly or by a lookup or directory service. The service description does not play a role in the service discovery process.

Salutation: A light weight network protocol independent service discovery protocol

Salutation is a service discovery and session management protocol developed by leading information technology companies [7]. Salutation is an open standard independent of operating systems, communication protocols, and hardware platforms. Salutation was created to solve the problems of service discovery and utilization among a broad set of appliances and equipment in an environment of widespread connectivity and mobility. The architecture provides applications with different services that are scattered all through out the network. It also contains functions to find out capabilities of remote services. Salutation provides features for an application to establish interoperable sessions with any remote service.

The Salutation architecture defines an entity called the Salutation Manager (SLM) that functions as a service broker for services in the network. Different functions of a service are represented by functional units. Functional Units represent essential features of a service (e.g. fax, print, scan etc). Furthermore, the attributes of each Functional Unit are captured in the Functional Unit Description Record. Salutation defines the syntax and semantics of the Functional Unit Description Record (e.g. name, value). SLM can be discovered by services in a number of ways such as:

1. Using a static table that stores the transport address of the remote SLM.

2. Sending a broadcast discovery query using the protocol defined by the Salutation architecture.

3. Inquiring the transport address of a remote SLM through a central directory server. This protocol is undefined by the Salutation architecture, however, the current specification suggests the use of SLP.

4. The service specifies the transport address of a remote SLM directly.

The service discovery process can be performed across multiple SLMs. A SLM can discover other remote SLMs and determine the services that are registered there. Service Discovery is performed by comparing a required service type(s), as specified by the local SLM, with the service type(s) available on a remote SLM. Remote Procedure Calls are used to transmit the required Service type(s) from the local SLM to the remote SLM and to transmit the response from the remote SLM to the local SLM. The SLM determines the characteristics of all services registered at a remote SLM by manipulating the specification of required service type(s). It can also determine the characteristics of a specific service registered at a remote SLM or the presence of a specific service on a remote SLM by matching a specific set of characteristics.

Critical comparison of existing service discovery techniques

The protocols described above essentially use typical attribute or interface matching to compare existing services in the network. SLP assumes the existence of an underlying Internet Protocol based communication mechanism and uses User Datagram Protocol (UDP) to communicate. Salutation in contrast is a network protocol independent architecture. The presence of a transport manager in salutation shields the underlying communication mechanism from the salutation manager. Thus salutation is expected to come into prominence when non Internet Protocol based networks come into vogue. UPnP works at lower level protocol suites and its specification is based on the assumption of an Internet Protocol type network underneath.

Both SLP, Salutation and UPnP are language independent protocols. The protocol specifications can be implemented in any language. Jini on the other hand is totally dependent on Java for its implementation. Java is a platform independent language which in turn makes Jini platform independent. In Jini, the interaction between services and users is defined in terms of an object interface, whereas in UPnP, the interaction is defined in terms of a network protocol. UPnP uses an Extended Markup Language (XML) based matching schema while discovering services. Jini on the other hand uses Java interfaces.

UPnP and Salutation essentially uses multicasting to find services or listen to broadcast service requests to find locations of desired services. Jini on the other hand uses a centralized scheme where all types of services are registered to a lookup service. Users contact the lookup service to discover existing services in the network. Service Location Protocol implements a centralized directory based scheme as well as a multicast scheme for discovering services. While centralized schemes of Service Location Protocol and Jini are more scalable, they are also prone to single point of failures. Salutation unlike Jini is a lightweight protocol and makes the least assumption of the underlying protocol stack and computing resources. Hence it can easily be ported to low power handheld devices. Jini on the other hand, due to its dependence on Java requires considerable computing resources to function properly.

Deficiencies in the existing service discovery architectures

All the above mentioned service discovery infrastructures and architectures like Service Location Protocol (SLP), Jini, Universal Plug and Play (UPnP), and Salutation have been developed to explore the service discovery issues in the context of distributed systems. While many of the architectures provide good base foundations for developing systems with distributed components in the networks, they do not adequately solve all the problems that may arise in a dynamic domain.

Some of the deficiencies are:

1. Lack of rich representation

   Services in Electronic Commerce are heterogeneous in nature. These services are defined in terms of their functionalities and capabilities. The functionality and capability descriptions of these services are used by the service clients to discover the desired services. Attribute matching is a very important component for finding the proper services in such an environment. The existing service discovery infrastructures lack expressive languages, representations and tools that are good at representing a broad range of service descriptions and are good for reasoning about the functionalities and the capabilities of the services. E.g., the service discovery protocols does not use any performance parameters for the existing services. They are satisfied with finding a service only. They do not consider whether the service would be able to serve the requester.

2. Lack of inexact matching

   In the Jini architecture, service functionalities and capabilities are described in terms of Java object interface types. Service capability matchings are processed in the object-level and syntax-level only. For instance, the generic Jini Lookup and other discovery protocols allow a service client to find a printing service that supports color printing, but the protocols are not powerful enough to find a geographically closest printing service that has the shortest print queue. The protocols do exact semantic matching while finding a service. Thus they lack the power to give a "close match" even if it was available.

Enhanced service discovery techniques for Mobile Commerce

Having identified some of the common deficiencies in the existing service discovery techniques, we present a brief description of our work aimed at resolving the issues. One can improve the service discovery mechanism by applying Artificial Intelligence tools and techniques to the existing systems. The XReggie project concentrates on improving the attribute matching beyond trivial syntax based matching. With Ad-hoc networks and Agent based approaches coming up, it will be extremely necessary to communicate between different agent groups in different networks who speak different languages. In the Ronin project we have addressed this problem and provided a framework for such communications.

The Ronin Agent Framework

The Ronin Agent Framework is a Jini-based distributed agent development framework [2,4]. Ronin introduces a hybrid architecture, a composition of agent-oriented and service-oriented architecture, for deploying in dynamic distributed systems. Among many of the distinguishable features of the framework, Ronin offers a simple but powerful agent description facility that allows agents to find each other and an infrastructure that encourages the reuse of the existing non-Java Artificial Intelligence applications.

In Ronin, agents are described using two types of attributes: the Common-Agent attributes and the Domain-Agent attributes. These are two sets of Jini service attributes that are associated with each Ronin agent in the Jini Lookup Service. Common-Agent attributes defines the generic functionalities and capabilities of an agent in a domain-independent fashion. Domain-Agent attributes defines the domain specific functionalities of agents. The framework defines the semantic meaning of each Common-Agent attribute, but it does not define the semantic meaning of any Domain-Agent attribute.

The Ronin description facility can enhance the Jini service discovery infrastructure for Mobile Commerce in the following ways:

1. As all agent-oriented Jini services, Ronin agents share a set of Common-Agent attributes. They can discover other services solely based on the domain independent functionalities of the services.

2. Once a service has discovered another service using the Ronin description facility, it is possible for these two services to form basic communication negotiation based on the semantic meanings of the Common-Agent attributes. For instance, if a service has knowledge about the type of Agent Communication Language (ACL) that the other service speaks, they can start their conversation by following the predefined standard Agent Communication Language negotiation protocol.

The Ronin framework encourages the reuse of the existing non-Java Artificially Intelligent applications. The philosophy is that developing infrastructures, such as the service discovery in Mobile Commerce, that are highly adaptive, interactive, interoperable and autonomous requires more than just the Jini architecture. Developing infrastructures that are more "intelligent" often requires sophisticated AI tools and techniques, such as inference engines, knowledge representation systems, constraint programming etc. For instance, it would be useful for a Jini Lookup Service to be able to reason about the capabilities of its registered services, enabling the JLS to make more "intelligent" service lookup recommendations.

The Ronin framework provides a Jini Prolog Engine Service (JPES) [3]. This service provides remote Prolog engine services to Jini-enabled components in the network. The JPES provides a distributed logical programming tool to the Jini-enabled services. An enhanced Jini Lookup Service can use the JPES to reason about the capabilities of services and can make more "intelligent" recommendations.

Adding descriptive powers to services through XReggie

The project XReggie investigates how Jini and similar systems can be taken beyond their simple syntax-based service matching approaches, adding expressive powers to the service descriptions. XReggie adds the facilities to describe service functionalities and capabilities using Extended Markup Language. XReggie allows service discovery to be performed in a more structured and descriptive fashion [6]. At the heart of the XReggie is the enhanced Jini Lookup Service that provides "service location" for Jini-enabled services. XReggie can help service clients to discover services in a manner essentially unchanged from the existing Jini Lookup and Discovery infrastructure. In addition, it allows services to advertise their capabilities in a well-structured descriptive format using Extended Markup Language. Furthermore, XReggie allows services to be discovered using the Extended Markup Language descriptions.

Conclusion

This article describes service discovery architectures that would come into prominence with the advent of Mobile Commerce. As more people start using laptops and palm pilots, the need for connecting them to the Internet for ubiquitous sharing and accessing of data will increase. The stationary computers connected to the Internet are no longer considered sufficient for "any time" and "anywhere" computing. Mobile Commerce and its proliferation is obviously leading us to a more refined and flexible way of accessing facilities from anywhere. A location based service oriented model is one of the most important necessities in such a distributed environment. A good service discovery infrastructure will play an important role in such an environment. We have reviewed a number of existing service discovery infrastructures. We found that all these infrastructures suffer from one common problem, namely lack of the descriptive languages and tools that are good at representing a broad range of service descriptions and are good for reasoning about the functionalities and the capabilities of the services. Lack of infrastructures for defining common ontologism across enterprises and customers is another key problem in the existing service discovery infrastructures.

We have developed the Ronin Agent Framework and XReggie that aimed to enhance the performance of the Jini service discovery infrastructure in the future Mobile Electronic Market. The Ronin framework enhances the Jini service discovery infrastructure by providing a simple but powerful description facility and encourages the reuse of the non-Java tools in the Jini environment. The XReggie project enhances the Jini service discovery infrastructure by extending the existing Jini Lookup Service to handle service registration and matching using the well-structured Extended Markup Language descriptions.

References

1

Arnold, Wollrath, et al. *The Jini Specification*. Reading, MA, USA: Addison-Wesley, 1999.

2

Bluetooth White Paper. http://www.bluetooth.com/developer/whitepaper/whitepaper.asp.

3

Chen, Harry. "Developing a Dynamic Distributed Intelligent Agent Framework Based on the Jini Architecture." Master's thesis, University of Maryland Baltimore County, January 2000.

4

Chen, Harry. Jini Prolog Engine Service (JPES). http://gentoo.cs.umbc.edu/jpes/.

5

Chen, Harry. Ronin Agent Framework. http://gentoo.cs.umbc.edu/ronin/.

6

IETF. Service Location Protocol v.2 Draft.

7

Liang, Xu. "Using Jini and XML to Build a Component Based Distributed System." Technical Report, University of Maryland Baltimore County, 2000.

8

Rekesh, John. "UPnP, Jini and Salutation - A look at some popular coordination framework for future network devices." Technical Report, California Software Lab, 1999. http://www.cswl.com/whiteppr/tech/upnp.html.

9

The Salutation Consortium Inc. Salutation Architecture Specification (Part 1), version 2.1 edition, 1999.

10

SLP White Paper. http://playground.sun.com/srvloc/slp_white_paper.html.

11

Understanding E-Service: Chapter 2 of the Internet. http://www.hp.com/e-services/understanding/chapter2/.

12

University Plug and Play Device Architecture Reference Specification. Microsoft Corporation.

Biography

Dipanjan Chakraborty can be reached at dchakr1@cs.umbc.edu. He is a Ph.D. student and a Research Assistant in the Department of Computer Science & Electrical Engineering, University of Maryland Baltimore County. His research interests include Mobile and distributed computing, Electronic Commerce, Network Protocols, Ad-hoc Networks and Distributed Systems. He is a student member of ACM.

Harry Chen can be reached at hchen4@cs.umbc.edu. He is a Ph.D. student and a Research Assistant in the Department of Computer Science & Electrical Engineering, University of Maryland Baltimore County. His research interests include Multi-Agent System, Mobile Computing, Pervasive Computing, Electronic Commerce, and Distributed Systems. The URL to his home page is http://www.acm.org/~hchen1/