Service Discovery in the Future Electronic Market

Harry Chen, Dipanjan Chakraborty, Liang Xu, Anupam Joshi, Tim Finin
{hchen4, dchakr1, lxu2, joshi, finin}@cs.umbc.edu

Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250

Abstract

The trend the electronic market is taking, aided by the concommitant development of mobile devices suggest a major change from the way electronic commerce is done today. The increased use of PDAs, laptops has showed us a new horizon of proliferation in the electronic market. When e-commerce services and transaction processing facilities need to be accessed from a wireless device,new and challenging research problems come into the picture. Discovering services dynamically will become increasingly important in the mobile e-commerce scenario. In the near future, a service would be selected automatically for a job, taking into consideration its physical location, context related and other semantic information. To support such a picture, the existing discovery mechanisms need to move beyond trivial attribute or interface matching. They would be much more knowledge based. In this paper, we present a summary of the existing service discovery protocols and the work that we have done in the service discovery area in our quest to make service discovery more dynamic.

Introduction

As thousands of companies and customers are wired to the fast growing Internet, the existing electronic market (eMarket) is primed for its next evolution. In the future, instead of interacting with the "do-it-yourself" web storefronts, customers will be able to communicate with the businesses via an automated service-oriented eMarket model(Hewlett-Packard 2000). In such eMarket model businesses not only can offer services and products through the static desktop computers, but also can also make their services and products available to the customers anytime and anywhere. For example, many investment institutions like Ameritrade and Morgan Stanley Dean Witter are offering investment services via mobile devices. Online bookstore Amazon.com is offering services via cellular phones.

As businesses offer various types of services to their customers, service discovery in the future eMarket becomes very important. Services are deployed in various forms and with different levels of complexities. Some services might be more software-oriented, such as personal banking via cellular phone or finding a closest restaurant in a local area. Some services might be more hardware-oriented, such as submitting pictures from a digital camera to a color printer or controlling home devices (VCR, Security Systems etc.) using a mobile device remotely. No matter what forms the services take, it is important for customers to be able to find the desired services effectively and correctly. A good service discovery infrastructure is one of most important base foundation to the future eMarket.

This paper is organized into five sections. In the first Section we describe some of the existing service discovery architectures that we have surveyed. In the second Section we describe what we believe to be the common weaknesses of the existing service discovery architectures for building the future service-oriented eMarket. In particular, we concentrate our discussion on the Jini architecture. In the third Section we describe the basics of intelligent service discovery protocols. In the fourth Section we describe the enhancements that we believe can be applied to the Jini architecture, allowing the Jini architecture to be better suited as the service discovery foundation to the future eMarket. We also describe our on-going research work in the XReggie project and development of the Ronin Agent Framework. Finally, the conclusion is given in the last section.

A Survey on different Service Discovery Architectures

In this section we briefly describe four recent industry supported efforts to create standards for service discovery. Due to lack of space, we do not present other systems such as Bluetooth or Ninja.

Service Location Protocol

The Service Location Protocol (SLP) is a product of the SVRLOC Working Group of the Internet Engineering Task Force (IETF). (E. Guttman 1999) It is a protocol for automatic resource discovery on IP networks. SLP is a language independent protocol. It bases its discovery mechanism on service attributes and can cater to any form of service, whether it is hardware or software.

The SLP infrastructure consists of three types of agents: *User Agent*, *Service Agent* and *Directory Agent*. The User Agents acquire service handles for end user applications that request for services. The Service Agents are responsible for advertising service handles to the Directory Agents, making services available to the User Agents. The Directory Agents collect together service handles and maintain the directory of advertised services. The core functionalities of the SLP are the following:

- Obtaining service handles for User Agents.
- Maintaining the directory of advertised services.
- Discovering available service attributes.
- Discovering available Directory Agents.
- Discovering the available types of Service Agents.

A service is described by configuration values for the attributes which are 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

Jini is a distributed service-oriented architecture developed by Sun Microsystems. Jini services can be realized to 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 the dynamic information about the available services in the Jini federation. Every service must discover one or more JLS 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 be potentially made available to the local network (i.e. the local LAN) 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 groups in its vicinity.

When a Jini service wants to join a Jini federation, it first discovers one or many JLS from the local or remote networks. The service then upload its service proxy (i.e. a set of Java classes) to the JLS. This proxy can be used by the service clients to contact the original service and invoke methods on the service. Service clients interact only 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 has no knowledge about the PostScript language.

Universal Plug and Play

Universal Plug and Play (UPnP), pushed primarily by Microsoft, is an evolving architecture that is designed to extend the original Microsoft Plug and Play peripheral model to a highly dynamic world of many network devices supplied by many vendors. UPnP works primarily at lower layer network protocol suites (i.e. TCP/IP), implementing standards at this level. This primarily involves addition to the suite, certain optional protocols which can be implemented natively by devices. The keyword here is "natively". UPnP attempts to make sure 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 APIs that implement these protocols - in whatever language or platform they choose.

UPnP uses the Simple Service Discovery Protocol (SSDP) for discovery of services on IP 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 HTTP over both unicast UDP (HTTPU) and multicast UDP (HTTPMU). The registration/query process sends and receives data in HTTP format, but has 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 observe these advertisement. The advertise message contains an URL that identifies the advertising service and an URL to an XML file that provides a description of the advertising service.

When a service client want 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. While 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 XML service description does not play a role in the service discovery process. (Rekesh John 1999)

Salutation

Salutation is a service discovery and session management protocol developed by leading information technology companies. Salutation is an open standard, independent of operating systems, communication protocols and hardware platforms. The 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, services and defines a standard method for describing and advertising their capabilities, as well as finding out the capabilities of others. The architecture also enables applications, services and devices to search for a particular capability, and to request and establish interoperable sessions with them.

The Salutation architecture defines an entity called the Salutation Manager (SLM) that functions as a service broker for services in the network. Services may be subdivided by meaningful functionality call Functional Unit, representing some essential feature (e.g. Fax, Print or Scan) 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 the following:

- Using of a static table that stores the transport address of the remote SLM.
- Sending broadcast discovery query over the transport using the protocol defined by the Salutation architecture.
- Inquiring the transport address of a SLM from a central directory server. This protocol is undefined by the Salutation architecture. However, the current specification suggests the use of Service Location Protocol (SLP) in conjunction with SLM. (Sal 1999)
- The service specifies the transport address of a remote SLM directly.

The service discovery process can be performed across multiple Salutation managers. One SLM can discover other remote salutation managers and determine the services registered there. Service Discovery is performed by comparing a required services 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. Through manipulation of the specification of required Service type(s), the SLM can determine the characteristics of all services registered at a remote SLM, the characteristics of a specific service registered at a remote SLM, and the presence of a service on a remote SLM by matching a specific set of characteristics.

Defeciencies in existing Service Discovery Architectures

Echoing the demands for service discovery infrastructures, architectures like Service Location Protocol (SLP) (SVRLOC Working Group 1997), Jini (Arnold *et al.* 1999), Universal Plug and Play (UPnP) (Mic 1999) and Salutation (Sal 1999) 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 network, we argue that they are not sufficient for building the future eMarket.

- **Lack of Rich Representations:**

  Services in the eMarket are heterogeneous in nature. These services are defined in terms of the their functionalities and capabilities. The functionality and capability descriptions of these services are used by the service clients to discover the desired services. 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. (Chen 2000) E.g. the service discovery protocols do not use any performance parameters for the existing services. They are satisfied with finding out a service only. They do not consider whether the service would be able to serve the requester.

- **Lack of Constraint Specification and Inexact Matching:**

  In the Jini architecture, service functionalities and capabilities are described in Java object interface types. (Arnold *et al.* 1999) Service capability matchings are processed in the object-level and syntax-level only. For instance, the generic Jini Lookup and 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 out a service. Thus they lack the power to give a close match even if it was available.

- **Lack of Ontology Support:**

  Services in the eMarket need to interact with clients and other services across enterprises. Service descriptions and information need to be understood and agreed among various parties. In another word, well-defined common ontology must be present before any effective service discovery process can take place.

  We found that common ontology infrastructures are often either missing from or not well represented in the existing service discovery architectures. Architectures like Service Location Protocol, Jini and Salutation do provide some sort of mechanisms to capture ontology among services. However, these mechanisms like Java class interfaces and ad-hoc data structures are difficult to be widely adapted by the industries to become standards. In the Universal Play and Plug (UPnP) architecture, service descriptions are represented in XML (eXtensible Markup Language), which provides a good base foundation for developing extensible and well-formed ontology infrastructure (Mic 1999). However, service descriptions in UPnP does not play a role in the service discovery process (Rekesh John 1999).

  In the existing Jini architecture, ontologies are captured in the level of Java object interface types. We believe this ad-hoc representation is an unfeasible approach for developing common ontology. Information captured in the Java programming language level is difficult to be understood by non-Java entities. Furthermore, the express power of the Java programming language can only capture a limited amount of information about the services. Therefore, this is not a feasible approach to describe information in the world of the eMarket.

Applying AI Techniques to the Service Discovery Infrastructures

One way to enhance the service discovery infrastructures is to apply AI techniques to the existing systems specifically in the areas of knowledge representation, reasoning and intelligent agents (Chen 2000). As a part of our research experiments to find a suitable solution for building a service discovery infrastructure for the future eMarket, we have developed the Ronin Agent Framework and XReggie to enhance the service discovery infrastructure in the existing Jini architecture. The Ronin Agent Framework is a Jini-based distributed agent development framework (Chen 2000). Ronin introduces a hybrid architecture, a composition of agent-oriented and service-oriented architecture, for deploying dynamic distributed systems. Ronin provides 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 AI applications. 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 (Liang Xu 2000).

Enhancing the Jini Service Discovery for Future eMarket

Jini offers a simple and robust service discovery infrastructure for building highly dynamic distributed systems. However, using Jini as the building block for the future service-oriented eMarket that are adaptive, interactive, interoperatable and autonomous, we need to extend and modify the existing Jini architecture.

Adding Agent Descriptions and making AI Tools available through Ronin

The Ronin Agent Framework is a Jini-based distributed agent development framework. Ronin introduces a hybrid architecture, a composition of agent-oriented and service-oriented architectures, for deploying 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 AI applications.

The Ronin description facility provides two disjointed sets of agent attributes: the *Common Agent Attributes* and the *Domain Agent Attributes.* (Chen 2000) The Common Agent Attributes and the Domain Agent Attributes are two sets of Jini service attributes that are associated with each Ronin agent in the Jini Lookup Service. The Common Agent Attributes defines the generic functionalities and capabilities of an agent in domain-independent fashion. The Agent Domain Attributes define the domain specific functionality of an agent. 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 in the eMarket in the following ways:

- 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.
- 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 pre-defined standard ACL negotiation protocol.

The Ronin framework encourages the reuse of the existing non-Java AI applications. The philosophy is that developing infrastructures, such service discovery in the eMarket, that are highly adaptive, interactive, interoperatable and autonomous requires more than just the Jini architecture. Developing infrastructures that are more "intelligent" often requires sophisticated AI tools and techniques, such inference engine, knowledge representation system, 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 present Ronin framework provides a Jini Prolog Engine Service (JPES). (Harry Chen 2000) 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.

XReggie: Supporting Rich and Constraint based Inexact Matching in Jini

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, and augmented to build a recommender for distributed components in a pervasive computing enviroment.

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 XML. Furthermore, XReggie allows services to be discovered using the XML descriptions and be matched at a semantic level.

XReggie adds the facility to describe service functionalities and capabilities using XML. A service is described using XML in terms of its capabilities, requirements and service attributes.

When a service registers with the JLS, it registers an XML description entry as well. When a client wants to use a service, the client creates a XML DOM object that describes the desired service along with its constraints. The XML match in JLS handles constraints such as requirements, cost, mobility etc. XReggie allows service discovery to be performed in a more structured and descriptive fashion. It ensures that a client receives only those components which it is capable of executing in terms of hardware or software requirements.

Conclusion

The trend of e-commerce is towards a service-oriented model service discovery infrastructure will play an important role in such an environment. We have reviewed a number of existing service discovery infrastructures. All these infrastructures suffer from some common problems such as lack of descriptive 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. Lacking of infrastructures for defining common ontology across enterprises and customers is another key problem in today's service discovery architectures.

We have developed the Ronin Agent Framework and XReggie that aimed to enhance the performance of the Jini service discovery infrastructure in the future eMarket. 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 AI tools in the Jini environment. The XReggie project enhances the Jini service discovery infrastructure by extending the existing JLS to handle services registration and matching using the well-structured XML descriptions. More information about our work can be found at http://www.cs.umbc.edu/dna.

References

Arnold, K.; Wollrath, A.; O'Sullivan, B.; Scheifler, R.; and Waldo, J. 1999. *The Jini specification.* Reading, MA, USA: Addison-Wesley.

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

E. Guttman, C. P. 1999. RFC 2608: Service Location Protocol v.2 Draft. Technical report, Sun Microsystems.

Harry Chen. 2000. Jini Prolog Engine Service (JPES). Available online from `http://gentoo.cs.umbc.edu/jpes/`.

Hewlett-Packard. 2000. Understanding E-Service: Chapter 2 of the Internet. Available online from `http://www.hp.com/e-services/understanding/chapter2/`.

Liang Xu. 2000. Using Jini and XML to build a component based distributed system. Technical report, University of Maryland Baltimore County.

Microsoft Corporation. 1999. *Universal Plug and Play Device Architecture Reference Specification*, version 0.9 edition.

Rekesh John. 1999. UPnP, Jini and Salutaion - A look at some popular coordination framework for future network devices. Technical report, California Software Labs. Available online from `http://www.cswl.com/whitepaper/tech/upnp.html`.

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

SVRLOC Working Group. 1997. SLP White Paper. Available online from `http://playground.sun.com/srvloc/slp_white_paper.html`.