DReggie: Semantic Service Discovery for M-Commerce Applications

Dipanjan Chakraborty, Filip Perich, Sasikanth Avancha, Anupam Joshi
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
{*dchakr1, fperic1, savanc1, joshi*}*@cs.umbc.edu*

Abstract

The emergence of mobile devices and wireless networks has created a new path in the field of e-commerce: "m-commerce". Significant research is needed, in the field of service discovery, to support m-commerce applications. Various new applications, that would use services available to a mobile device from both the fixed network backbone and peer mobile devices in its proximity, are being developed. M-commerce applications have the challenging task of discovering services in a dynamically changing environment. The DARPA Agent Markup Language (DAML) is an important tool in the process of developing the Semantic Web. In this paper, we present a dynamic service discovery infrastructure that uses DAML to describe services and a Prolog reasoning engine to perform matching using the semantic content of service descriptions. We believe that the architecture we have designed is a necessary component of any m-commerce infrastructure.

1. Introduction

The last few years has seen an exponential growth in the usage of wireless-enabled handheld devices. This suggests a change in the way mobile devices are being used now-a-days. In addition to being used as standalone personal organizers, these devices are being increasingly used to access the Internet. For example, most Palm Pilots are being shipped with AOL. This has started a new field of e-commerce: m-commerce. In the near future, we envision Palms using short range ad-hoc networking technologies like Bluetooth, to interact with each other, possibly to support m-commerce applications. In such an ad-hoc environment, we envision one or more devices providing and using services.

As businesses offer various types of services to their customers, service discovery will become 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.) by 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.

In an ad-hoc environment, *a priori* information and/or description of the services is, more often than not, unavailable. This is because the location of devices would be changing continuously. It is also impractical to assume that the mobile device would have an enumerated list of all possible services and their interfaces. Therefore, a flexible service discovery infrastructure is an important base foundation of the future m-commerce market.

Currently existing service discovery techniques use simple interface-based or attribute-based matching. Service discovery is effectively done at a syntactic level. We argue that syntactic level matching and discovery is inefficient in a mobile environment. This is due to the heterogeneity of service interfaces in such a domain. For example, we can have the same service implement different interfaces which could result in the failure of a syntactic match if the service query does not match with any interface. Therefore, there is a need to discover services in a semantic manner. An effort has been going on in the World Wide Web to provide richer and machine understandable descriptions of web pages [3]. The DAML initiative is an effort to design an ontology to describe metadata about websites. We believe that this ontology can also be used to describe domain specific m-services. We can use the features of DAML to reason about the capabilities and functionality of different services.

The rest of the paper is organized as follows: In section 2 we describe some existing service discovery architectures that we have surveyed. In section 3 we describe the drawbacks of these existing service discovery mechanisms with respect to the demands of the future service-oriented e-market. We also describe the basics of intelligent service discovery protocols. In section 4 we describe the DARPA Agent Markup Language and its utilities. In section 5 we describe the DReggie - a Jini based service discovery infrastructure which uses the reasoning services offered by DAML to do a semantically richer service discovery. We present our conclusions in section 6.

2. Existing Service Discovery Infrastructures

In this section, we describe current service discovery infrastructures, that have become popular are widely used on

wired networks. Some of these infrastructures, like SLP and Jini are platform and communication protocol dependent.

The Service Location Protocol (SLP) [7] is a language-independent protocol for automatic resource discovery on IP networks utilizing an agent-oriented infrastructure. The basis of the SLP discovery mechanism lies on predefined service attributes, which can be applied to universally describe both software and hardware services. The architecture consists of three types of agents: User Agent, Service Agent and Discovery Agent. The User Agents are responsible for discovering available Directory Agents, and acquiring service handles on behalf of end-user applications that request services. The Service Agents are responsible for advertising the service handles to Directory Agents. Directory Agents are responsible for collecting service handles and maintaining the directory of advertised services. SLP uses multicasting for service registration and discovery, and unicasting for service discovery responses from Directory/Service Agents.

Jini [2] 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 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 network 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 (UPnP)[10] extends the original Microsoft Plug and Play peripheral model to support service discovery provided by network devices from numerous vendors. UPnP works and defines standards primarily at the lower-layer network protocol suites, so that the devices can natively, i.e., language and platform indecently, implement these standards. UPnP uses the Simple Service Discovery Protocol (SSDP) for discovery of services over IP networks, which can operate with or without a lookup service in the network. In addition, the SSDP operates on the top of the existing open standard protocols utilizing HTTP over both unicast (HTTPU) and multicast UDP (HTTPMU). When a new service wants to join the network, it transmits an announcement to indicate its presence. If a lookup service is present, it can record this advertisement to be subsequently used to satisfy clients' service discovery requests. Additionally, each service on the network may also observe these advertisements. When a client wants to discover a service, it can either contact the service directly through the URL that is stored within the service advertisement, or it can send out a multicast query message, which can be answered by either the directory service or directly by the service.

Salutation [1] is an open standard, communication, operating-system, and platform-independent service discovery and session management protocol. The goal of Salutation is to solve the problem of service discovery and utilization among a broad set of appliances and equipment in a wide-area or mobile environment. The architecture provides applications, services and defines a standard method for describing and advertising their capabilities, as well as locating and determining other services and their capabilities. In addition, the Salutation architecture defines an entity called the Salutation Lookup Manager (SLM) that functions as a service broker for services in the network. The SLM can classify the services based on their meaningful functionality, called Functional Units (FU), and the SLM can be discovered by both a unicast and a broadcast method. The services are discovered by the SLM based on a comparison of the required service types with the service types stored in the SLM directory. The service discovery process can be performed across multiple Salutation managers, where one SLM represents its client while communicating with another SLM, to discover services.

3. Drawbacks of Existing Techniques

We believe that existing service discovery architectures like SLP, Jini, UPnP and Salutation, although popular, have a few limitations which makes them unsuitable for wide deployment in the m-commerce domain. We briefly discuss and analyze these limitations.

- **Lack of Rich Representation:** Services in the eMarket are heterogeneous in nature. These services should be defined in terms of the their functionality and capabilities. The functionality and capability descriptions of these services should be used by the service clients to discover them.

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 functionality and the capabilities of the services [4].

• **Lack of Constraint Specification and Inexact Matching:** In the existing service discovery infrastructures, it is impossible to find services which require a specific attribute value that can change based on the dynamic content of the environment. In addition, service functionality are described at the syntax level or object level. This makes it difficult to apply approximate matching rules. For example, if the client is attempting to discover a black and white printer, an approximate matching rule would succeed on finding a color printer.

• **Lack of Ontology Support:** Services in the m-commerce arena need to interact with clients and other services across enterprises. Service descriptions and information need to be understood and agreed among various parties. In other words, well-defined common ontology must be present before any effective service discovery process can take place. Common ontology infrastructures are often either missing from or are 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. However, service descriptions in UPnP does not play a role in the service discovery process.

4. DAML: The DARPA Agent Markup Language

DAML is a semantic language being developed by a consortium of U.S. located academic and business researchers to address the WWW's limitations in providing machine-readable and more importantly machine-interpretable information over the Internet.

Consequently, the goal of DAML is to enable the transformation of the present human-oriented Web, which is largely used as a text and multimedia repository only, into a Semantic Web as envisioned in [3]. This process involves the augmentation of the web pages with additional information and data that are expressed in a way that facilitates their understanding by machines.

The development of DAML is based upon the capabilities of an already existing syntactic language, the eXtensible Markup Language (XML)[8]. The Resource Description Framework and Resource Description Framework Schema (RDFS)[6] are XML applications that provide a number of preliminary semantic facilities required for the realization of the Semantic Web vision.

XML was developed by the World Wide Web Consortium (W3C) as a standard for alternative data encoding on the Internet that was primarily intended for machine processing. The XML standard provides the necessary means to declare and use simple data structures, which are stored in XML documents and are machine-readable. However, since XML is defined only at the syntactic level, machines cannot be guaranteed to unambiguously determine the correct meaning of the XML tags used in a given XML document, and consequently XML cannot be used as a language for representing any complex knowledge.

The W3C has thus developed RDFS with the goal of addressing the deficiencies of XML by adding formal semantics on the top of XML. These two standards provide the representation frameworks for describing relationships among resources in terms of named properties and values, which are similar to representation frameworks of semantic networks and rudimentary frame languages as in the case of the RDF Schema. However, both standards are still very restricted as a knowledge representation language due to the lack of support for variables, general quantification, rules, etc.

The DAML project is an attempt to build upon XML and RDFS to produce a language that is well suited for building the Semantic Web. It uses the same mechanism for representing data and information in a document as XML, and it also provides all the elementary rules and definitions similar to RDFS. In addition, DAML provides rules for describing further constraints and relationships among resources including cardinality, domain and range restrictions as well as union, disjunction, inverse and transitivity. DAML is, therefore, intended to be a universal Semantic Web markup language that is sufficiently rich to provide machines not only with the capability to read data but also to interpret and make inferences from the data. DAML will enable the development of intelligent agents and applications, which can independently (without human involvement) retrieve and manipulate information on the Internet.

5. DReggie Overview and Implementation

The project DReggie is an attempt to enhance the matching mechanisms in Jini and other service discovery systems. The key idea in DReggie is to enable these service discovery systems to perform matching based on *semantic information* associated with the services. As is well-known, service matching in existing systems is strictly syntactic (i.e., string matching). Semantic information of services consists of their extensive descriptions including, but not limited to, capabilities, functionality, portability and system require-

ments. Semantic service matching introduces the possibilities of fuzziness and inexactness of the response to a service discovery request. In the DReggie system, a service discovery request contains the description of an "ideal" service - one whose capabilities match exactly with the requirements. Thus, matching now involves comparison of requirements specified with the capabilities of existing services. Depending on the requirements, a match may occur even if one or more capabilities does not match exactly. For example, the request could specify that the service must be able to execute on an Intel Pentium II based system. If a service the ability to execute on a Pentium III based system is found, a match has occurred. Service descriptions in the DReggie system are marked up in DAML. The semantic matching process that uses these descriptions is performed by a reasoning engine. At the heart of DReggie is an enhanced Jini Lookup Service (JLS) that enables smart discovery of Jini-enabled services. DReggie retains the matching mechanism currently employed by the Jini Lookup and Discovery infrastructure. Clients attempting to discover services have the option to utilize either of the two matching mechanisms.

When a service registers with the enhanced Jini Lookup Server (JLS), it registers a DAML description as well as its interface. The DAML description stored in the lookup server and updated whenever the service registers again. A client attempting service discovery creates a DAML description of the desired service including various constraints. The DReggie lookup server consists of a simple Java-based matching module and an advanced Prolog-based reasoning engine. In our initial implementation of the infrastructure, the simple Java-based module has been integrated with the JLS. The Java-based module performs attribute matching and limited constraint-based matching. The Prolog-based reasoning engine is capable of more complex matching based on DAML service descriptions. The capability and functioning of the Prolog module is described in section 5.2. In our future work, we aim to integrate the Prolog engine as a module with the Jini Lookup Server to discover m-services more efficiently and successfully.

It is assumed that the request that a client issues uses the same ontology that a service uses to describe itself. This is a very important assumption. It does restrict service matching to one particular ontology, but enables more knowledge based discovery of services. In the future, it is quite likely that there would be translator services that would enable translation of one ontology to another and hence enable cross-ontology service discovery. The DAML match in JLS handles constraints such as client requirements, cost, mobility etc. It ensures that a client discovers only those services that it is capable of executing in terms of hardware or software requirements.

In order to compare DAML descriptions in the Jini Lookup Server, we have made modifications to Reggie - Sun's implementation of Jini Lookup Service. Reggie provides all the functions and methods defined in the Jini Core package, besides some extra functionality. Modifications and additions to this package include:

- Addition of an entry class that accepts a well-formed DAML description as an argument
- Addition of a class that implements the functions required to compare two DAML instances
- Modification to a registrar class to enable DReggie to determine the appropriate matching mechanism to use
- Modification to a lookup class to allow DReggie to find information regarding the appropriate parser package to use

5.1. DAML Representation of Services

We have used DAML as a language to represent the semantic description of the capabilities and requirements of m-services. As mentioned in previous section, XML was developed as a standard for alternative data encoding on the Internet that was primarily intended for machine processing. However, XML was defined only at the syntactic level. This restriction implies that although a machine could have the necessary capability to retrieve information, the same machine may not be able to precisely identify the intended interpretation of the XML syntax presented in the retrieved information. For example, a machine may incorrectly determine that two instances of the same tag, e.g. <DUE-DATE>, in two different XML documents provide the same information, although, in the first document it could represent a due date to ship a product, while in the latter document it could represent a due date to return a library book. Conversely, a machine may not be able to correctly determine that two different tags, e.g. <AUTHOR> and <WRITER>, in fact represent the identical information. Thus DAML has been used to facilitate uniform interaction and discovery of heterogeneous services. We have created a DAML ontology to describe m-services in terms of their functionality, capability, platform requirements and other attributes. It is available online at `http://www.daml.umbc.edu/ontologies/dreggie-ont.daml`.

We show below a partial view of the ontology we created to describe services. It is mostly self-explanatory; the service component (the Component class) forms the root of the hierarchy. Capability and functionality descriptions are added as properties of the Component class.

```
<?xml version="1.0" ?>

<rdf:RDF
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml = "http://www.daml.org/2001/03/daml+oil#"
  xmlns = "http://daml.umbc.edu/ontologies/dreggie-ont#"
>
...
<daml:Class rdf:ID="Component">
  <rdfs:label>Component</rdfs:label>
</daml:Class>
<daml:Property rdf:ID="Description">
  <rdfs:domain rdf:resource="#Component" />
```

```
</daml:Property>
<daml:Property rdf:ID="ServiceName" >
  <rdfs:domain rdf:resource="#Description" />
  <rdfs:range rdf:resource="#String" />
</daml:Property>
<daml:Property rdf:ID="ServiceAlias" >
  <rdfs:domain rdf:resource="#Description" />
  <rdfs:range rdf:resource="#String" />
</daml:Property>
...
<daml:Property rdf:ID="Capability">
  <rdfs:domain rdf:resource="#Description" />
  <rdfs:range rdf:resource="#CapabilityClass" />
</daml:Property>
<daml:Class rdf:ID="CapabilityClass" />
...
<daml:Property rdf:ID="ServiceCapability">
  <rdfs:domain rdf:resource="#CapabilityClass" />
  <rdfs:range rdf:resource="#Type" />
</daml:Property>
<daml:Property rdf:ID="Requirements">
  <rdfs:domain rdf:resource="#Description" />
  <rdfs:range rdf:resource="#RequirementsClass" />
</daml:Property>
<daml:Class rdf:ID="RequirementsClass" />
<daml:Property rdf:ID="CPURequirement">
  <rdfs:domain rdf:resource="#RequirementsClass" />
</daml:Property>
<daml:Property rdf:ID="MemoryRequirement">
  <rdfs:domain rdf:resource="#RequirementsClass" />
</daml:Property>
...
<daml:Property rdf:ID="Mobility">
  <rdfs:domain rdf:resource="#Description" />
  <rdfs:range rdf:resource="#MobilityClass" />
</daml:Property>
<daml:Class rdf:ID="MobilityClass" />
<daml:Property rdf:ID="ClientMobility">
  <rdfs:domain rdf:resource="#MobilityClass" />
  <rdfs:range rdf:resource="#Type" />
</daml:Property>
<daml:Property rdf:ID="ServiceMobility">
  <rdfs:domain rdf:resource="#MobilityClass" />
  <rdfs:range rdf:resource="#Type" />
</daml:Property>
...
...
</rdf:RDF>
```

5.2. Prolog Reasoner for Semantic Discovery

The DAML service ontology is the basis of the semantic matching process for different services. This ontological information is parsed, converted into a form that the Prolog engine uses to perform matching and loaded into its Knowledge Base (KB). The engine also accepts DAML profiles of services from service providers. This information is also parsed and loaded into its KB. In addition to this information, the engine loads DAML relationship rules (e.g., inverse, equivalence, disjoint etc.). We have designed and developed rules that use the ontology, service profile information and the parsed query from the client to perform matching based on relationships between attributes and their values. These rules are loaded into the KB as the final step of the start-up phase.

DAML, by design, imposes a class hierarchy on all objects. Thus, every class is a subclass of some object at the root of the hierarchy and inherits all its attributes. In addition, every class has some attributes of its own. Thus, the DAML based service ontology defines this hierarchical relationship between various components of a service and their attributes. The Prolog engine uses this ontology for validation of service profiles at the time of registration and validation of both services and attributes at the time of service queries. The motivation for this kind of online validation is that although the information provided/requested is well-formed DAML, the relationships may be ill-formed.

The actual working of the Prolog engine for semantic matching is best described by the following scenario:

- The DAML service ontology description and a PrinterService profile based on this ontology are presented as inputs to the Prolog engine. First, the ontology is parsed and loaded. Then, the PrinterService profile is parsed. For each parsed DAML statement, a validity check is performed to ensure that the relationship in the statement conforms to the ontology. If this check fails, the profile cannot be registered and an error is returned. If the check succeeds the statement is loaded into the KB. We assume that the engine has already loaded DAML-specific and service-specific rules into its KB.
- Assume that a service query for the PrinterService arrives as a DAML description. The attributes described in the query indicate that a color printer using LaserJet technology is required. Assume that the PrinterService profile describes two types of printers: a color printer using InkJet technology and a black-and-white printer using LaserJet technology. Further assume that the client specifies that the color attribute has higher priority over the technology attribute.
- The Prolog engine first parses the DAML query and as before, validates each statement. The most important check is for the service being requested. If this check fails, an error indicating unavailability of the service is returned. However, attribute validity failures do not generate errors. They are silently discarded and only values for valid attributes are returned. Based on the priority rules, the engine tries to find the closest match to the color attribute. It matches the color printer based on the InkJet technology and returns information about this service irrespective of the fact that the query also requested a LaserJet printer.

The reasoning engine is currently under development and testing as part of a project to enhance the Bluetooth Service Discovery Protocol (SDP). We briefly describe the working of the current version of the reasoning engine. In the following paragraph, ”Server” refers to the device that contains information about services and ”Client” is a device attempting service discovery.

- Server initialization code starts up, loads the knowledge base and starts the reasoning engine.
- Client establishes a connection to the Server after device discovery.
- Client sends a ”semantic service discovery request” message to the Server. The message contains, among other information like connection id, transaction id etc., a DAML query, as described in the scenario above.

- Server receives the message, parses it and determines it to be a ”semantic service discovery request” .
- In the Server, the function to process the semantic search request is called. This function creates a temporary file and writes the query into it. It also creates an output file name for the reasoning engine to write its results out to. Finally, it passes this information to another function that actually handles the query.
- The query handling function makes two Prolog function calls in order to handle the query. The first function is used to parse the DAML data into triples [6]. The second function invokes the Prolog predicate in the reasoning engine to perform the semantic search.

6. Conclusions

Service discovery will play an important role for successful development and deployment of m-commerce applications. In this paper, we have discussed some existing commercial service discovery mechanisms. We have presented DReggie (modified Reggie - an implementation of the Jini specification by Sun Microsystems), a service discovery infrastructure that addresses drawbacks of existing discovery mechanisms. DAML is expected to change the way people and machines browse the WWW. We have shown that it can be used to change the way services are described and discovered in the wired and the wireless world. We conclude that combining an enhanced Reggie, DAML and a powerful reasoning engine based on Prolog will enable semantic service discovery for m-commerce applications.

References

[1] The Salutation Consortium Inc 1999. Salutation architecture specification (part 1), version 2.1 edition. World Wide Web, `http://www.salutation.org`.

[2] Ken Arnold, Ann Wollrath, Bryan O'Sullivan, Robert Scheifler, and Jim Waldo. *The Jini specification.* Addison-Wesley, Reading, MA, USA, 1999.

[3] Tim Berners-Lee, James Handler, and Ora Lassila. The Semantic Web. In *Scientific American*, may 2001.

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

[5] UC Berkeley Computer Science Division. `http://ninja.cs.berkeley.edu`.

[6] Resource Description Format. World Wide Web Consortium (W3C), `http://www.w3c.org/RDF`.

[7] E. Guttman, C. Perkins, J. Veizades, and M. Day. ”rfc2068: Service location protocol, version 2”. ftp://ftp.isi.edu/in-notes/rfc2608.txt, 1999.

[8] Extensible Markup Language. World Wide Web Consortium (W3C), `http://www.w3c.org/XML`.

[9] Bluetooth White Paper. World Wide Web, `http://www.bluetooth.com/developer/whitepaper`.

[10] UPnP White Paper. World Wide Web, `http://upnp.org/resources.htm`.