Enhancing the Bluetooth Service Discovery Protocol

Sasikanth Avancha, Anupam Joshi, Timothy Finin
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore, MD 21250
E-mail: {savanc1, joshi, finin}@csee.umbc.edu

Abstract

Bluetooth Service Discovery Protocol (SDP) enables a client application on a device to discover information about services on other Bluetooth devices. Every service is represented by a profile, identified by a 128-bit Universally Unique Identifier (UUID). A match occurs on a peer device if and only if at least one UUID specified by the client is contained in one or more of its service records. We believe that the advantages of UUID-based matching to support service discovery are restricted to ad-hoc Bluetooth networks consisting of resource constrained devices. The more common case for applications using Bluetooth networks, is the existence of one or more resource rich devices (e.g., the Compaq iPAQ) in the network. This calls for a matching mechanism that uses *semantic information* **associated with services and attributes to decide the success or failure of a query. We present an enhanced version of Bluetooth SDP that supports** *semantic matching* **and provides service registration. We evaluate the performance of this enhanced version of SDP and compare it with regular SDP. We show that enhanced SDP performs comparably to regular SDP in terms of Round Trip Time and matching time.**

I. Introduction

Bluetooth [12] is a short-range wireless technology that allows devices to exchange data and voice in real-time. The Bluetooth Special Interest Group (SIG) is responsible for developing specifications of this technology. Bluetooth consists of a set of protocols that constitute a protocol stack. The current version of the specifications defines the Service Discovery Protocol (SDP), RFCOMM (for cable replacement), Logical Link Control and Adaptation Protocol (L2CAP), Host Controller Interface (HCI), Link Manager Protocol (LMP), Baseband and Radio (RF).

The focus of this paper is SDP [7]. SDP provides a means for applications on a device to discover services on peer devices and to determine characteristics of the discovered services. Bluetooth SDP has been designed for efficient service discovery on resource constrained devices. The underlying assumption of this design is that devices in a Bluetooth network are resource constrained, primarily in terms of memory and computing power. Thus, Bluetooth SDP provides a simple matching scheme using 128-bit UUIDs. Client applications on devices attempting to discover service(s) must specify the correct service UUID(s) in discovery requests sent to peer devices. The Bluetooth SDP server on each device providing information about services, matches the UUID in incoming discovery requests against existing service UUIDs. SDP also allows service attribute UUIDs to be specified along with service UUIDs. Thus, the server attempts to match against attribute UUIDs if a match on the associated service UUID succeeds.

The UUID-based matching mechanism is well-suited to service discovery in pure ad-hoc Bluetooth networks. However, it is often the case that Bluetooth networks consist of one or more devices that are resource rich. In such cases, a more comprehensive matching mechanism based on *semantic information* associated with services and attributes will result in a more flexible service discovery protocol. Semantic information includes priorities and expected values of service attributes. For example, a client may specify that it requires a color printer that also provides duplex printing, with the additional constraint that the color printing capability overrides the duplex printing capability.

We have designed an enhanced Bluetooth SDP prototype. In this prototype, the matching mechanism uses semantic information associated with services and attributes in its decision making process. In addition, service providers can register services with a designated *service registrar*. The prototype consists of two main components to provide semantic matching and service registration:

- A comprehensive, well-defined *service ontology* described in RDF
- An XSB-based knowledge base and reasoning engine

The service ontology contains the vocabulary associated with a service in an ad-hoc network environment. Thus, clients must use the service ontology to describe services and attributes in service discovery requests. Similarly, service providers are required to use this ontology to describe services and attributes in service registration requests. We envisage the use of DAML to describe this ontology in future versions of the prototype.

The rest of the paper is organized as follows. Section II describes features and limitations of existing service discovery protocols, including Bluetooth SDP. We also discuss other efforts at semantic matching for service discovery and other approaches to improve Bluetooth SDP in this section. In section III we discuss the design of the enhanced SDP. Section IV provides an overview of the implementation. In section V, we describe the experiments designed to evaluate the performance of the enhanced SDP and regular SDP, and discuss the results. We conclude the paper and discuss future work in section VI.

II. Background

Service discovery is a process involving a client, a service provider and a lookup/directory server. Service registration and lookup or matching are important components of any service discovery protocol. The simplest matching mechanism matches string. The client specifies a string representing a service or an attribute in its discovery request and the lookup server attempts to match that string with the strings in its repository. In existing protocols, the client provides very little, if any, additional information regarding the services it is attempting to discover. Well-known service discovery protocols, that use string matching or a slight variant, include the Service Location Protocol [18], Jini [1], Universal Plug and Play [13] and Salutation [17].

We have discussed earlier, the limitations of UUID-based matching. The limitations of string matching are no different. In addition, these protocols are limited by the fact that clients and service providers do not share a common, well-defined, formal ontology. SLP and Salutation attempt to capture some ontological information using fixed data structures, while Jini implicitly forces the use of an ad-hoc ontology by ensuring that client queries contain interfaces and attributes. The lack of a well-defined ontology for service descriptions could result in false matching. Thus, despite being more powerful than Bluetooth SDP, these protocols do not solve the problem of making service discovery more flexible and powerful. These issues are discussed in detail in [5]. The string matching limitation is partially addressed in [10], which discusses a mechanism to enhance the Jini lookup service Reggie (Sun Microsystems), to match against service components described as XML [3] Document Model (DOM) objects. Clients attempting to discover a service send an XML DOM object consisting of various constraints that must be satisfied by the service. The modified Jini lookup server, called XReggie, attempts to match the queries to services that satisfy the client-specified constraints.

One of the requirements of the design of Bluetooth SDP is interoperability with existing service discovery protocols. This has led to attempts to map Bluetooth SDP to existing service discovery protocols. One such effort is described in [14]. This white paper describes possible ways in which to map the Bluetooth SDP to the Salutation protocol. One method is to map Salutation APIs to Bluetooth SDP transaction types. Thus, the Salutation protocol forms a layer above SDP. The latter can extract the information from the APIs and process it according to the corresponding transaction type.

III. System Design

The two main components of the prototype system are the **service ontology** and the **XSB knowledge base & reasoning engine**. The process of semantic matching is dependent on a well-defined ontology. In section III-A, we discuss the importance of the ontology and how it is used to perform semantic matching. In order to use the ontology effectively and reason about the information provided in it, a powerful reasoning engine is required. We have implemented the reasoning engine in the XSB logic programming language. In section III-B, we discuss the reasoning engine. Figure 1 shows the various components of the enhanced SDP.

A. Ontology Design

We have designed an ontology to describe services in an ad-hoc networking environment like Bluetooth. Describing services using an ontology is superior to UUID-based descriptions, because the former method provides a structure that makes it possible to reason about and derive knowledge from the given descriptions. It is also important to note that relationships between different entities in the system can be described more clearly in an ontological description. This also allows for better reasoning and inferring about the knowledge.

The service ontology is described in RDF [11] and RDF-Schema (RDFS) [4]. RDF is a powerful language to describe semantic information. It uses the syntax of the Extensible Markup Language (XML) [3]. One of the requirements for successful semantic matching is a simple, yet powerful language for describing resources. It should be easy to parse, easy to manipulate and easy for a reasoning engine to use. It should be scalable enough to handle any number of resources in a given ontology. RDF meets all these requirements. In addition, it serves as the basis for the DARPA Agent Markup Language (DAML) [9] – a more powerful semantic language for representing knowledge. Figure 2 shows a partial view of the ontology. We note that the priority field is used to decide the ordering of attributes, so that the highest priority attribute is used to determine the correct service instance to match against. The client may leave the *priority* and/or the *value* field unspecified. In such cases, the server uses a set of predefined priority values for the attributes.

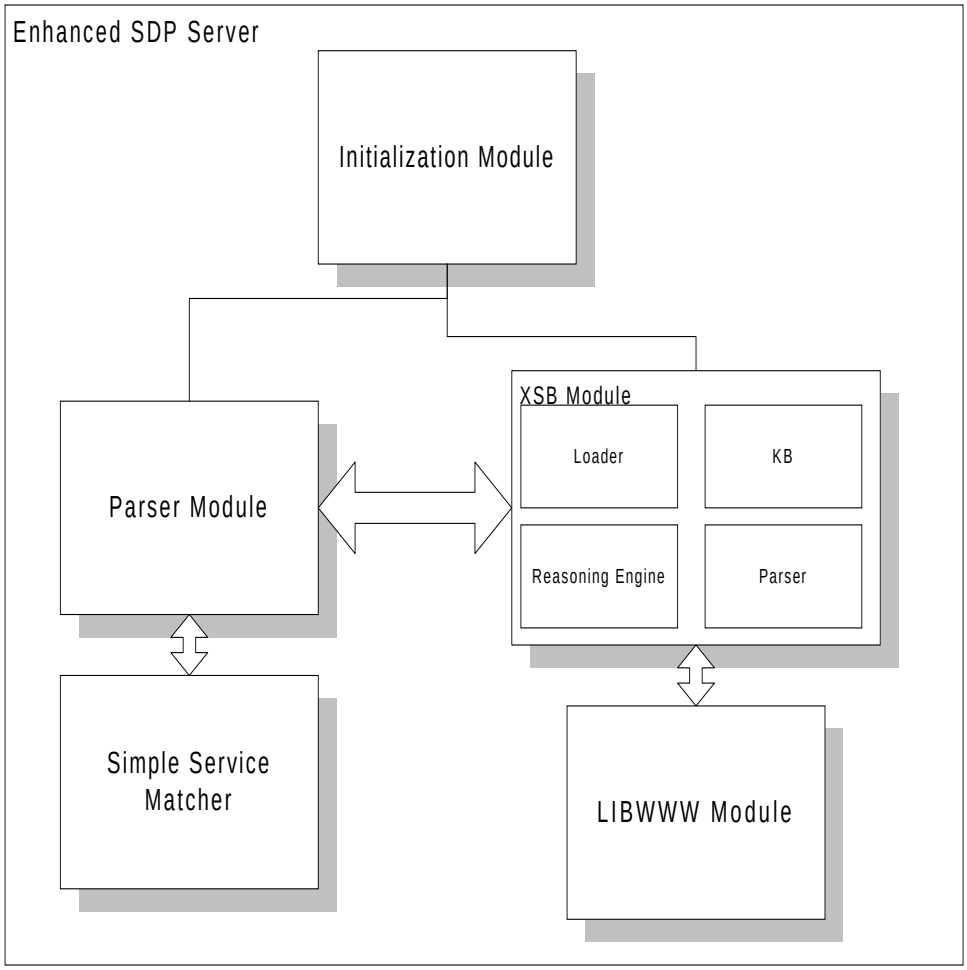


Fig. 1. Components of enhanced Bluetooth SDP

B. XSB-based Reasoning Engine

XSB [16] is a research-oriented Logic Programming system for Unix and Windows-based systems. Its predecessors were PSB-Prolog and SB-Prolog. In using XSB as the reasoning engine for our work, we have taken advantage of a powerful indexing technique for asserted code called *unification factoring* that can improve program speed and indexing for compiled code. Large amounts of factual data imply that the data must be loaded into the knowledge base before the program can use it. The designers of XSB claim that the method of asserting clauses in XSB is the fastest method of all Prolog systems [16]. The key to the correct functioning of our reasoning engine is a knowledge base with sufficient and complete information about service instances. XSB has also been designed to interface easily with the C language. It also provides an interface to the `libwww` [15] package that is used to parse RDF into a format suitable for XSB to treat as predicate clauses.

IV. Implementation Overview

In this section, we discuss the service registration process and the semantic matching process. We used the Bluetooth stack for Linux developed by Axis Communications Inc., to implement the enhancements. The stack is developed in C.

Service registration in this system is similar to that in systems like Jini. A service provider registers information about service instances on a device designated (by some external mechanism) as the *service registrar*. The service provider can issue three types of requests: *register, renew* and *cancel*. The *register* request consists of three parts: a unique identifier (UID) for the service instance, the requested service lease time, and an RDF description of the service instance. The *renew* request is similar, except that the RDF description contains only the service name. The *cancel* request consists only of the UID and service name.

The service registration process consists of the following steps:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs=""
         xmlns="">

<rdfs:Class rdf:ID="Service">
  <rdfs:subClassOf rdf:resource="resource"/>
</rdfs:Class>

<rdfs:Class rdf:ID="ServiceProvider">
  <rdfs:subClassOf rdf:resource="resource"/>
</rdfs:Class>

<rdf:Property rdf:ID="ProviderName">
  <rdfs:domain rdf:resource="ServiceProvider"/>
  <rdfs:range rdf:resource="AttribPriorityNValue"/>
</rdf:Property>

<rdf:Property rdf:ID="ContactURI">
  <rdfs:domain rdf:resource="ServiceProvider"/>
  <rdfs:range rdf:resource="AttribPriorityNValue"/>
</rdf:Property>

...

<rdf:Property rdf:ID="ProvidedBy">
  <rdfs:domain rdf:resource="Service"/>
  <rdfs:range rdf:resource="ServiceProvider"/>
</rdf:Property>

<rdfs:Class rdf:ID="AdHocNetworkService">
  <rdfs:subClassOf rdf:resource="Service"/>
</rdfs:Class>

<rdf:Property rdf:ID="NetworkTechnology">
  <rdfs:domain rdf:resource="AdHocNetworkService"/>
  <rdfs:range rdf:resource="AttribPriorityNValue"/>
</rdf:Property>

<rdf:Property rdf:ID="DeviceType">
  <rdfs:domain rdf:resource="AdHocNetworkService"/>
  <rdfs:range rdf:resource="AttribPriorityNValue"/>
</rdf:Property>

<rdf:Property rdf:ID="CPUType">
  <rdfs:domain rdf:resource="AdHocNetworkService"/>
  <rdfs:range rdf:resource="AttribPriorityNValue"/>
</rdf:Property>

<rdf:Property rdf:ID="OperatingSystem">
  <rdfs:domain rdf:resource="AdHocNetworkService"/>
  <rdfs:range rdf:resource="AttribPriorityNValue"/>
</rdf:Property>
...
...
</rdf:RDF>
```

Fig. 2. Service Ontology

- The service provider sends a *register* request over an existing L2CAP connection to the *service registrar*. (The default Maximum Transmission Unit (MTU) on both devices is 672 bytes, as specified in [8]. Thus, L2CAP will not transmit packets greater than 672 bytes. Although it is possible to use a larger default value, no mechanism to determine the optimum value exists. We therefore modified SDP at the kernel level to segment large SDP packets into 672 byte packets. To ensure that all segments are successfully transmitted, the kernel level SDP sleeps for a short time via `interruptible_sleep_on_timeout`.)
- The registrar extracts the UID, lease time and RDF description from the request. It then parses, validates and loads the RDF description into the knowledge base. It periodically checks for expiration of the lease time and unloads the RDF description on expiry to de-register the service.

A. Semantic Service Matching

The semantic service matching process begins when an SDP client sends an SDP query to the SDP server. The query message consists of only one part: the RDF description of the query. The client must use the same ontology as the SDP server, We assume that this requirement is enforced. An example RDF query is shown in figure 3. The SDP server executes the following steps:

- Extract and parse RDF query description.
- Validate service description in query.
- Read and create a list of all specified attributes, their values and priorities, if any.
- If no attributes have been specified by the client, values of all direct attributes (i.e., not inherited) of one or more service instances of the validated service are returned to the client. In addition, the value of the `ContactURI` attribute is returned.
- If the list of attributes is not empty, then it is sorted according to priority.
- Validate each attribute in the list against the service ontology and attempt to match the value specified by the client (if

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:sdp_sem="">
<sdp_sem:Printer rdf:ID="Any">
        <sdp_sem:PrintColorQuality rdf:parseType="Resource">
          <sdp_sem:Priority>3</sdp_sem:Priority>
          <sdp_sem:Value>color</sdp_sem:Value>
        </sdp_sem:PrintColorQuality>
        <sdp_sem:PrintFileType rdf:parseType="Resource">
          <sdp_sem:Priority>6</sdp_sem:Priority>
          <sdp_sem:Value>Postscript</sdp_sem:Value>
        </sdp_sem:PrintFileType>
        <sdp_sem:PrintOutputFormat rdf:parseType="Resource">
          <sdp_sem:Priority>7</sdp_sem:Priority>
          <sdp_sem:Value>Letter</sdp_sem:Value>
        </sdp_sem:PrintOutputFormat>
        <sdp_sem:PrintResolution rdf:parseType="Resource">
          <sdp_sem:Priority>4</sdp_sem:Priority>
          <sdp_sem:Value>1440</sdp_sem:Value>
        </sdp_sem:PrintResolution>
        <sdp_sem:ServiceCost rdf:parseType="Resource">
          <sdp_sem:Priority>1</sdp_sem:Priority>
          <sdp_sem:Value>x</sdp_sem:Value>
        </sdp_sem:ServiceCost>
</sdp_sem:Printer>
</rdf:RDF>
```

Fig. 3. Example of RDF query

any) to one or more service instances. If no exact match occurs, one or more *closest* values of the corresponding attribute are returned.

- Finally, the value of the `ContactURI` attribute corresponding to the appropriate service instance(s) will be returned.

V. Performance Evaluation of Service Registration and Semantic Matching

In this section, we discuss the performance of the service registration and semantic matching mechanisms. We compare the performance of semantic service matching with that of simple service matching in the current Bluetooth SDP. We use time and request size as the metrics in all experiments.

The general setup for all experiments is as follows. The enhanced SDP server runs on a Linux based server and the SDP client on a Linux-based laptop. Both systems use the Bluetooth protocol stack developed by Axis Communications, Inc., and the Bluetooth hardware module developed by Ericsson. In all experiments the laptop is in master mode and the server in slave mode. In each experiment, we measure the time taken for a given registration or discovery request. Each request is sent 100 times and the registration or discovery time is measured. Request size increases by the number of bytes required to specify a single attribute. We have used the *PrinterService* ontology, shown in figure 3 in these experiments.

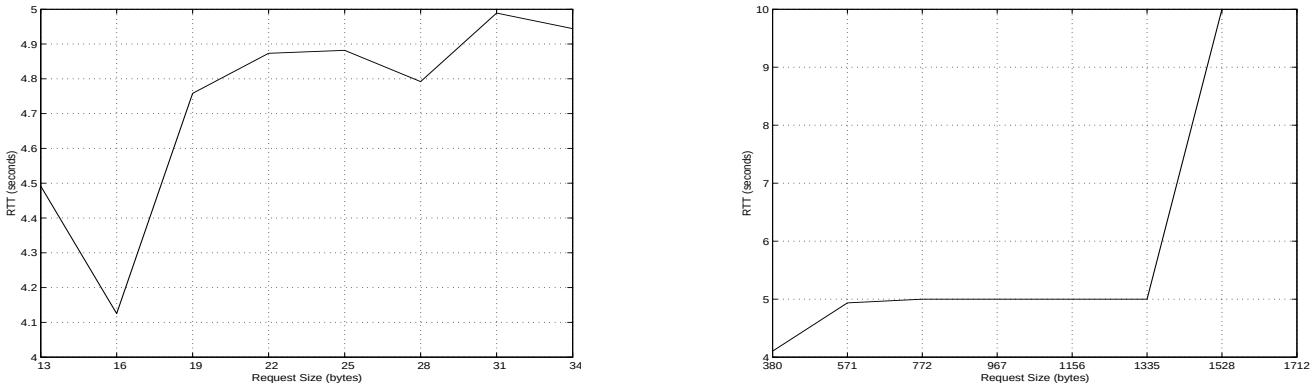
A. Experiment 1

In this experiment, we compare the performance of simple and enhanced SDP in terms of Round Trip Time (RTT) for service discovery. The RTT is measured on the client and includes the time taken to transmit the request, perform matching and receive the response. Figures 4a and 4b show the graphs for service discovery using simple and enhanced SDP. Figure 4a shows that the average RTT stays in the 4s-5s range with increasing request sizes.

In figure 4b, we observe that the RTT changes in distinct steps. This is due to segmentation of large semantic requests, as discussed in section IV. We have chosen a delay of 300ms for which the kernel level SDP sleeps between segment transmissions. We experimented with values ranging from 200ms to 400ms for this delay and determined 300ms to be an optimal value. The first two request sizes are shorter than the default MTU of 672 bytes, therefore they experience no transmission delay. However, in the graph we do not notice a large difference between the 571-byte request and the 772-byte request. We attribute this to the extra time spent by XSB in performing garbage collection tasks. The last two request sizes experience a delay of at least 600ms to transmit 3 segments.

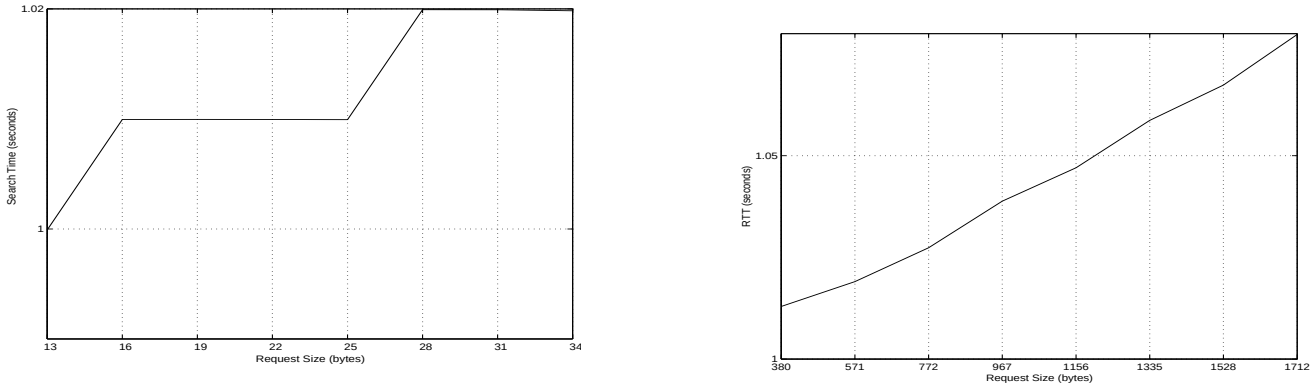
B. Experiment 2

In this experiment, the server measures the time taken exclusively for the matching functions to execute. In the case of the simple SDP, this involves validating and searching for the service and attribute UUIDs. In the case of semantic matching, this



(a) (b)

Fig. 4. Service Discovery Request RTT (a) Simple SDP (b) Enhanced SDP



(a) (b)

Fig. 5. Service Discovery Request Matching time (a) Simple SDP (b) Enhanced SDP

involves parsing the request, validating it and then performing the match. Figures 5a and 5b show the graphs of the time taken to perform UUID-based and semantic matching.

We observe that there is very little difference between time taken for simple and semantic matching. We believe that this is an important result because the penalty paid for using a complex matching technique with a heavy reasoning engine like XSB is quite insignificant. This experiment confirms the fact that the higher RTT values observed for large semantic request sizes is due mainly to transmission time as opposed to matching time.

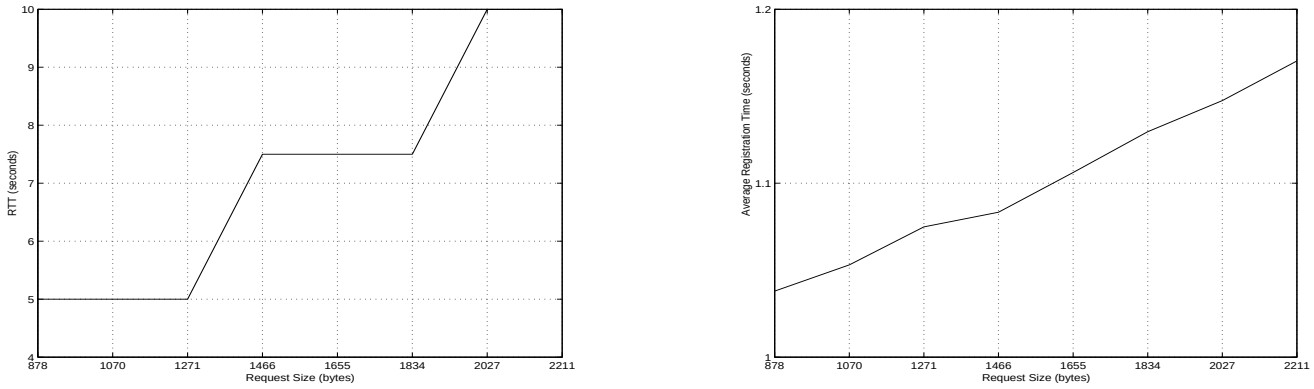
C. Experiment 3

In this experiment, we evaluate the performance of service registration. Figure 6a shows the graph of registration request size versus RTT. We observe that, as expected, the RTT increases in a step-wise fashion as registration request sizes increase. The RTTs of first and second groups of requests differ by about 2.5s, as do RTTs of the second and third groups.

Figure 6b shows the graph of registration request size versus registration time. The server measures the time taken to process registration requests. This includes parsing the RDF data, validating it, loading it into the knowledge base and returning the response.

VI. Conclusions and Future Work

We have designed and evaluated a prototype of an enhanced Bluetooth SDP system. We have shown that, in Bluetooth networks consisting of one or more resource rich nodes, the performance of semantic matching is very nearly the same as that of the existing matching mechanism. We conclude that, in such networks, semantic matching is better than simple matching.



(a) (b)

Fig. 6. Service Registration (a) RTT (b) Registration time

We have also shown that the performance of service registration is satisfactory. Both service registration and semantic matching require a well-defined ontology in order to understand and validate requests. We have successfully developed an ontology for describing services specific to ad-hoc network environments. A powerful, yet efficient reasoning engine is also required for service registration and semantic matching. We have developed an initial version of this engine that uses XSB and RDF.

Future work in this area includes the use of a DAML-based ontology to allow the specification of additional constraints like cardinality and also describe other relationships like *inverseOf* and *equivalent*. In addition, we would also like to improve segmentation and reassembly of large SDP requests to reduce transmission delay. In the future, we will develop prototypes that use other Prolog variants and perform experiments on them.

REFERENCES

[1] K. Arnold, A. Wollrath, B. O'Sullivan, R. Scheifler, and J. Waldo. *The Jini Specification*. Addison-Wesley, Reading, MA, USA, 1999.

[2] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers. `http://www.rfc-editor.org/rfc/rfc2396.txt`, 1998.

[3] T. Bray, J. Paoli, and C. Sperberg-MacQueen. Extensible Markup Language. `http://www.w3.org/TR/1998/REC-xml19980210`, 1998.

[4] D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0 - W3C Recommendation. `http://www.w3.org/TR/2000/CR-rdfschema-20000327`, 2000.

[5] D. Chakraborty and H. Chen. Service discovery in the future for mobile commerce. In *ACM Crossroads*. ACM Press, Winter 2000.

[6] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *Proc. ACM MOBICOM*, pages 24–35, 1999.

[7] D. Farnsworth (Ed.). Service Discovery Protocol. `http://www.bluetooth.com`, 2001.

[8] J. Inouye (Ed.). Logical Link Control and Adaptation Protocol. `http://www.bluetooth.com`, 2001.

[9] J. Hendler. DARPA Agent Markup Language. `http://www.daml.org`, 2000.

[10] L. Xu, A. Joshi. Using Jini and XML to build a component based distributed system. Technical report, University of Maryland Baltimore County, USA, 2000.

[11] O. Lassila and R. Swick. Resource Description Framework. `http://www.w3.org/TR/1999/REC/rdf-syntax-19990222`, 1999.

[12] R. Mettala. Bluetooth Protocol Architecture. `http://www.bluetooth.com/developer/whitepaper/`, 1999.

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

[14] B. Miller and R. Pascoe. Mapping Salutation architecture APIs to Bluetooth Service Discovery Layer. `http://www.bluetooth.com`, 1999.

[15] H. Nielsen and T. Berners-Lee. Libwww - the W3C Protocol Library. `http://www.w3.org/pub/WWW/Library`, 2000.

[16] K. Sagonas, T. Swift, and D. S. Warren. XSB as an efficient deductive database engine. In *Proc. SIGMOD*. ACM, 1994.

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

[18] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service Location Protocol. `http://www.rfc-editor.org/rfc/rfc2165.txt`, 1997.