An Architecture for Flexible Service Discovery in OCTOPUS

Tao Gu, H. C. Qian, J. K. Yao, H. K. Pung
Center for Internet Research, Department of Computer Science
National University of Singapore
Singapore
{gutao, qianhc, yaojiank, punghk}@comp.nus.edu.sg

Abstract —**Service discovery has been drawing much attention from researchers and practitioners. The existing service discovery systems, like SLP, Jini, UPnP and Salutation, provide basic infrastructures where services can announce their presence and users can locate these services across the network. However there are several key issues which are partially solved or have not been well addressed - such as scalability, availability, dynamics and support for multiple matching mechanisms. In this paper, we propose a design for a Service Locating Manager (SLM) system which addresses some of these issues. The SLM system adopts a dynamic hierarchical tree structure and service aggregation for scalability, availability and dynamics, and introduces multiple matching mechanisms which contain an attribute-based and a semantic matching engine. It provides a scalable, distributed, dynamic and robust solution to establish a flexible service discovery architecture. We describe our concepts, architecture and implementation, and present a performance study for our prototype.**

Keywords -Service Discovery; Scalability; Dynamic; Service Aggregation; Service Matching

1 INTRODUCTION

Today's Internet is going through a major change - from being a mere repository of information when it first started, to a vehicle of various services today. Services can be defined as any devices, or applications with well-known interfaces that perform computation or actions on behalf of client users. They can be deployed in various forms and with different levels of complexities. How to facilitate users in discovering these services is indeed a challenging task, judging from the diversity of services and the dynamics of users as well as service providers.

In recent years, many service discovery architectures arising from both industrial research such as SLP, Jini, UPnP, Salutation and UDDI, and academic research such as SDS and INS. Service Location Protocol (SLP) [1] is a language independent protocol for automatic resource discovery on local-area IP network. Services are described in the form of *service:URLs* which are composed of a service type and a set of attribute-value pairs. The SLP does service matching based on predefined service attributes. Jini [2] is a distributed service discovery architecture built on top of Java object and RMI [3] system. A service proxy object is registered with Jini Lookup Service (LUS) [2]. A client downloads the service proxy and invokes it to access the service which is identified by means of Java class hierarchy. Jini employs Java interface matching. As such, client is solely responsible for knowing the precise name of the Java class representing the service. The Jini architecture has the limitation on scalability as it does not provide any solutions to connect Jini federations which may reside in global networks. Universal Plug and Play (UPnP) [4] has a Peer-to-Peer architecture based on TCP/IP networks and is designed to accommodate home networks or small office networks. It uses the Simple Service Discovery Protocol (SSDP) for discovery of services, which can operate with or without a lookup service in the network. It has a simple XML [5] matching mechanism; however XML was defined only at syntactic level. Salutation [6] is an open standard of communication independent service discovery and session management protocol. The Salutation architecture defines an entity called the Salutation Lookup Manager that functions as a service broker for services in the network. The services are discovered based on a comparison of the required service types with the service types stored in the Salutation Lookup Manager directory. Universal Description, Discovery and Integration (UDDI) [7] is an emerging industry standard that defines a business oriented discovery mechanism to a global registry holding XML-based WSDL [8] service descriptions. It uses SOAP [9] that allows one program to invoke service interfaces across the Internet in a language independent and distributed manner. UDDI aims at global networks, but it is targeted towards web services [10] and has less dynamic support. Secure Service Discovery Service (SDS) [11] is a research level service discovery system developed at University of California, Berkeley. It has a client-repository-server type architecture and has a XML-based semantic matching mechanism. Service descriptions and queries are specified using XML. It has simple semantic matching capability based on XML. International Naming System (INS) [12] is a resource discovery and service locating system for dynamic and mobile networks developed at Massachusetts Institute of Technology. It uses a simple naming language based on attributes and values to achieve expressiveness, integrates name resolution and message forwarding that tracks change, and uses soft-state name discovery protocols that enable robust operation. INS has the limitation when it scales to large numbers of resources spread throughout wide-area networks. INS/Twine [13] achieves more scalability by partitioning the name space across resolvers by mapping names into numeric keys.

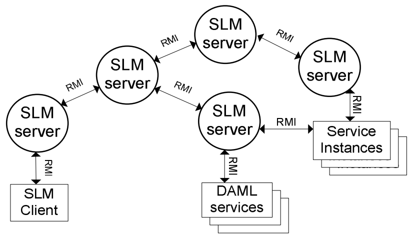

Figure 1. An overview of the SLM system

In this paper, we introduce the SLM service discovery system to enable users to search for and use the services available in the network more efficiently. It addresses some of key issues such as scalability, service availability, the dynamic nature of services joining and leaving, and support for multiple matching mechanisms which are partially solved or have not been well addressed in the existing service discovery infrastructures. It is also an important component in OCTOPUS [14] project which provides a middleware level support for collaborative multimedia applications.

The rest of this paper is organized as follows. Section 2 describes our design consideration and the SLM system architecture; in Section 3 we present our implementation; followed by performance measurements in Section 4; and finally, we summarize and conclude in Section 5.

2 Design Consideration and System Architecture

The SLM service discovery system consists of SLM servers, services and SLM clients as shown in Figure 1. A SLM server is a service information repository, providing SLM clients with access to all available services. SLM clients on behalf of end users can search for services. The system adopts a distributed hierarchical tree structure to organize SLM servers which may physically be located in wide-area networks. Compared to other connection structures such as mesh and ring, tree has the advantage of minimizing latency and achieving good scalability. It is also more flexible and expandable. The underlying network infrastructure of SLM system is based on TCP/IP network. Communications between two SLM servers, SLM clients and SLM servers, instances and SLM servers are based on Java RMI.

The SLM system has the following features: (1) the tree is structured in such a way that it can be re-configured for meeting the dynamicity of services and servers; (2) to ensure service instance is up-to-date, each service instance keeps a lease with its SLM server, and stale service information is deleted from its SLM server automatically upon the expiry of the lease; (3) to ensure service availability, multiple service instances can register to different SLM servers.

Services are deployed in various forms. A flexible service discovery system should enable the discovery of all available services conforming to a particular functionality or set of attributes. In existing service discovery systems, services are defined in syntactic level and they leverage on attributed-based or interface-based matching mechanisms, e.g., predefined service types and attributes in SLP or Jini's interface which is also based on attribute-value pairs. We believe that syntactic

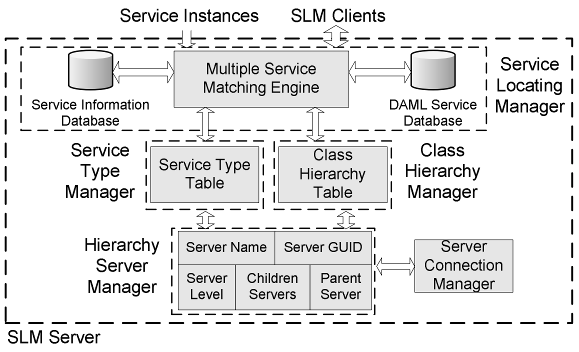

Figure 2. Components of a SLM server

level matching is not enough as the same service may be implemented by different interfaces. With the emergent trend of the Semantic Web [15], services will be defined in a semantic manner using semantic markup language – DAML+OIL [16] which builds on W3C standards such as RDF and RDF Schema [17]. Semantic matching also plays a key role in discovering services in a mobile environment due to the heterogeneity of service interfaces in such a domain. Therefore there is an increasing need to discover services in a semantic manner. We incorporate attribute-based and semantic matching mechanisms in the SLM system and also retain Jini's interface matching mechanism to provide users a flexible means to search for services.

2.1 Key Components

In this section, we describe the key components in our SLM system, focusing on the roles in the system and how they interact with each other.

A SLM Server consists of Service Locating Manager, Service Type Manager, Class Hierarchy Manager, Hierarchy Server Manager, and Server Connection Manager as shown in Figure 2. Their functionalities are described below.

Server Connection Manager: SLM servers communicate with each other through their Server Connection Managers. Initially, a Server Connection Manager uses Jini's multicast and unicast to locate other SLM servers, and then its Hierarchical Server Manager decides which SLM server to accept as its child server. After setting up a parent-child relationship between these two SLM servers, they can communicate with each other through RIM. A SLM Client or a service instance also has its own Server Connection Manager to communicate with the correspondent SLM server for service request or advertisement.

Service Locating Manager (SLM): SLM consists of a service information database, a DAML service database and a multiple service matching engine. It has the functionality of service information registration and service matching. The Multiple Service Matching Engine consists of an attribute matching engine and a semantic matching engine which will be described in Section 2.4. For attribute-based service descriptions, service instances are registered to the service information database and abstracted service type information will be updated in the Service Type Table. For semantic service descriptions, DAML services are reasoned and registered in the DAML service database and the Class Hierarchy Table will be updated.

Hierarchy Server Manager: Hierarchy Server Manager is responsible for server management. Its functionality includes building and updating the dynamic SLM server tree and checking server relationship. It creates and maintains the following information about a SLM server.

- Server Name: It contains the name of a SLM server.
- Server GUID (Global Unique Identifier): It is used to identify each SLM server. Each SLM server has one unique GUID.
- Server Level: It designates the level of a SLM server. The administrator will determine the appropriate level for each SLM server.
- Parent/Children Servers: It keeps a reference to its parent server and children servers.

Service Type Manager: This component will be invoked in the case of attribute-based service descriptions. In each SLM server, its Service Type Manager maintains a Service Type Table which keeps abstracts of service information that the SLM server can support. The abstracted service information is used for service aggregation. The Service Type Table will be updated when a new service instance joins a SLM server; the updating process will be propagated to its parent servers until reaching the root SLM server. Its Service Type Manager will also perform the updating process when there is a structural change to its children servers. The path for updating is always along its parent server.

Class Hierarchy Manager: This component will be invoked in the case of DAML service descriptions. In each SLM server, its Class Hierarchy Manager maintains a Class Hierarchy Table which keeps a set of hierarchy subclasses through which the SLM server can provide services. When a new DAML service ontology is introduced and a new DAML instance joins a SLM server, the supported subclasses will be updated in the Class Hierarchy Table. Its parent server also performs the updating process.

2.2 Dynamic Server Tree

In the SLM system, SLM servers are dynamically constructed and connected in a hierarchical tree topology. The relationship between two adjacent SLM servers is either a parent-to-child or a child-to-parent relation. To join the tree, a SLM server will first discover other SLM servers using its Server Connection Manager and select a suitable server as its parent. Its Hierarchy Server Manager will then setup a parent-child relationship between these two SLM servers. The two SLM servers also keep its proxy object of each other for further communication through RMI. In the child server, the abstracted service information will be aggregated to its parent server and the aggregation process will continue upwards. When a SLM server finds its parent server or its child server has closed the connection, it should tear down a parent-child relationship through its Hierarchy Server Manager. If one of its children servers is leaving, the parent server must update its Service Type Table and Class Hierarchy Table. If the parent server is leaving, all its children servers will set their parents as null or locate another SLM server as their parents. However, the Service Type Tables and Class Hierarchy Tables of these children servers remain unchanged.

```
<profile:Profile rdf:ID="MP3MusicService">
  <profile:serviceName>mp3musicService</profile:serviceName>
  ……
  <profile:input>
    <profile:ParameterDescription rdf:ID="MusicTitle">
      <profile:parameterName>musicTitle</profile:parameterName>
      <profile:restrictedTo rdf:resource="http://www.w3.org/2000/10/XMLSchema.xsd#string" />
      <profile:refersTo rdf:resource="http://lucan.ddns.comp.nus.edu.sg/octopus/daml/MP3.daml#musicTitle"/>
    </profile:ParameterDescription>
  </profile:input>
  <profile:input>
    <profile:ParameterDescription rdf:ID="CreditCardNumber">
      <profile:parameterName>creditCardNumber</profile:parameterName>
      <profile:restrictedTo rdf:resource="http://www.w3.org/2000/10/XMLSchema.xsd#decimal" />
      <profile:refersTo rdf:resource="http://lucan.ddns.comp.nus.edu.sg/octopus/daml/MP3.daml#creditCardNumber"/>
    </profile:ParameterDescription>
  </profile:input>
  <profile:ouput>
    <profile:ParameterDescription rdf:ID="MusicOuput">
      <profile:parameterName>music</profile:parameterName>
      <profile:refersTo rdf:resource="http://lucan.ddns.comp.nus.edu.sg/octopus/daml/MP3.daml#Music"/>
    </profile:ParameterDescription>
  </profile:ouput>
  ……
</profile:Profile>
```

Figure 3. Advertisement of a service in DAML-S

A service provider can register service instances to any SLM server. First, it discovers SLM servers using its Service Connection Manager. The service provider will select a suitable SLM server to register its service instances. The selection criteria can be based on server load, geographic region, administrative domain, and favorite.

2.3 Service Description and Service Aggregation

Services are described in various forms in our SLM system. A service provider can register a service instance represented by a service template, a semantic description or a Java/Jini object.

A service template consists of attribute-value pairs describing the properties of a service in terms of service type and associate attributes, including service provider, IP address and service description. Service providers can add additional attribute names and values when the need arises. It provides a high-level description of services, which typically would be presented to users when browsing a service registry. It will be used during the attribute matching which will be described in Section 2.4.

For semantic description, services are represented by DAML-S [18] Service Profile in terms of their capabilities and functionalities. DAML-S is a web service ontology based on DAML+OIL. It defines a set of classes and properties specifically for the description of web services within DAML+OIL. A DAML-S Service Profile consists of three types of information: a human readable description of the service, a specification of the functionalities that are provided by the service, and a list of functional attributes which provide additional information and requirements about the service that assist when reasoning about several services with similar capabilities. The functional specification of a service is represented in terms of inputs, outputs, preconditions and effects. An example of advertisement is shown in Figure 3. It

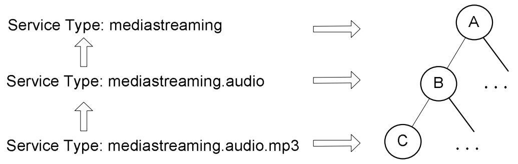
Service Type: mediastreaming
Service Type: mediastreaming.audio
Service Type: mediastreaming.audio.mp3
A
B
C
. . .
. . .

Figure 4. Service aggregation by service type

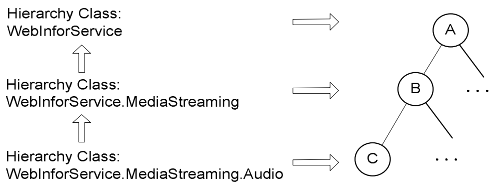
Hierarchy Class: WebInforService
Hierarchy Class: WebInforService.MediaStreaming
Hierarchy Class: WebInforService.MediaStreaming.Audio
A
B
C
. . .
. . .

Figure 5. Service aggregation by class hierarchy

shows a MP3 music service returns which music can be brought to a requester when presented with a music title and a credit card number.

To locate a service efficiently, it is important to find out a service path which leads to the destination server where the service instance is stored. In the SLM system, we introduce a service aggregation which allows each parent SLM server to keep summarized service information about its children servers. Since any parent SLM server knows about service information stored in all his children SLM servers, a service path can be found and the destination SLM server can be located quickly based on the hints provided by the service aggregation. Further more, the service aggregation can be used to minimize the cost of information exchange between servers as we adopt a hierarchical server structure. There are two types of service aggregation in the SLM system. In the case of attribute-based service representation, the service aggregation is achieved by aggregating service information through the abstraction of service type information. It allows the lower-tiered SLM servers to keep detailed service information and the upper-tiered SLM servers to save space by aggregating this specific information into a more generic type. An example is shown in Figure 4.

In the case of semantic service representation, a separate service aggregation is needed as services are defined using ontologies and object instances. It is achieved by profile-based class hierarchy. A DAML-S Service Profile is used to characterize a service for purposes of advertisement and discovery. We construct a hierarchy of subclasses of the *Profile* class to categorize a broad array of services that exists within a domain. Each subclass inherits the properties from its superclass. Each SLM server contains a set of class hierarchy in which it can provide these services; and its parent server stores the superclass. In this way, DAML service ontologies and instances can be distributed among SLM servers where a parent-child relationship exists based on the class hierarchy. Each SLM server maintains a Class Hierarchy Table indicates which services the SLM server can support. If there are any new DAML ontologies and instances registered or there is a structural change on its children servers, the Class Hierarchy Table will be updated. An example is shown in Figure 5.

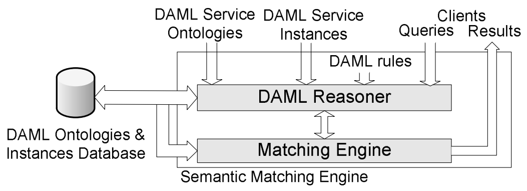

Figure 6. The architecture of the semantic matching engine

2.4 *Multiple Service Matching Engine*

In Jini architecture, a service is registered as a Java/Jini object to LUS. The Jini interface matching engine in LUS maps interfaces indicating the functionality provided by a service to sets of objects. In the design of our SLM system, we retain the interface matching employed by Jini. In addition, we introduce an attribute matching mechanism when services are described using service templates, as well as a semantic matching mechanism when services are described using DAML-S.

Attribute matching engine: In the attribute matching, both services and queries are defined using service templates. The attribute matching mechanism in our SLM system is based on a frame-based search engine which increases its precision compared to keyword-based search engines at the cost of requiring that all services be modeled as frames using templates. It will match services whose service type and attribute values equal to those in the query. The results for the search will be returned to the client in a ranked order if multiple service instances have been found. The ranking is determined by the number of matched attributes and the priority of attributes.

Semantic matching engine: The architecture of the semantic matching engine is showed in Figure 6. A service provider registers its DAML service ontologies and instances with any SLM server. The service ontological information and instances are presented as inputs to the ontology reasoner. The reasoner parses each statement in the ontologies and instances; checks the validity of each statement to ensure they conform to the ontology. Then the reasoner loads the ontology, the instances and relationship rules into its Knowledge Base. When a SLM client makes a query through the application GUI, the query is converted to a DAML-S description and parsed by the reasoner for checking of validity. If the checking succeeds, the reasoner will parse all DAML-S statements to RDF triples [17] and perform semantic matching. The service information will return to the client when a match is found.

3 IMPLEMENTATION

We have implemented the SLM system based on Java J2SE 1.3.1. For a rapid prototyping, we make use of Jini's multicast and unicast mechanism to build up the Server Connection Manager component, and use Jini's LUS to register SLM servers. The current version is based on Jini 1.2. A testbed for performance evaluation has also been built. Although the SLM system allows servers to be connected in a hierarchical tree, too many levels in the tree may increase the latency of service searching. It is recommended to limit the number of levels of SLM servers. In our experiments, we use three server levels as

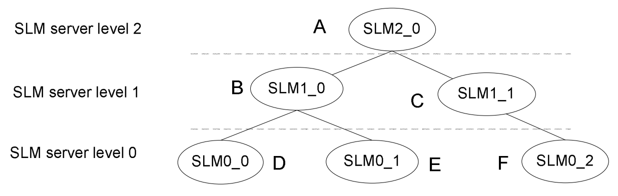

Figure 7. SLM server levels in our testbed

shown in Figure 7. Each SLM server ran in a 1.6G Intel Pentium 4 machine with 256M RAM.

A SLM server needs to configure itself before its startup. The configuration includes specifying server name, server level, unicast address, and downloading service type information from the global service type repository and common ontologies from the global service ontologies repository. We assume that all the available service types and common ontologies are stored in the global repository which is located somewhere in the network. When a new SLM server wants to join the SLM system, its Server Connection Manager will use Jini's multicast or unicast or both to locate the Jini federations. Then the SLM server registers itself to all the LUS found. It also needs to find other SLM servers to build up a hierarchical SLM server tree.

We have implemented the attribute matching engine and the semantic matching engine. For the attribute matching, a SLM client wishes to search for a service specifies the service type name and service attributes using a service template. For the semantic matching, we adopt JTP [19] as our DAML reasoner as it is an object-oriented modular reasoning system and it is easy to add-in a user specific reasoning modular. A query is specified by DAML-S description through the application GUI and then converted to Knowledge Interchange Format (KIF) [20] which JTP can recognize. Then its Server Connection Manager will use Jini's multicast or unicast to locate the nearby SLM servers and chooses one to process the user request. If the service type or the service class is supported by this server, the server will call the attribute matching engine or the semantic matching engine to process its query and return the results. Otherwise, the user request will be forwarded to its parent server and this process will continue until the desirable service type or service class is identified and the corresponding destination server is found. Finally, the corresponding SLM server will return the results to the client.

4 Performance

We have tested all the basic functionalities of our SLM system in the testbed. SLM servers can join and leave the system dynamically and smoothly. Services can also be added or deleted to/from any SLM server dynamically. The system can adapt to changes quickly. In our experiments, we set the default life time to 10 seconds for multicast and 2 seconds for unicast. All SLM servers can be discovered within the time limit. The average latency between parent and children servers is around 1.2 ms.

4.1 System Performance

We measured the attribute searching performance in two experiments. In Experiment 1, we created 400 service instances

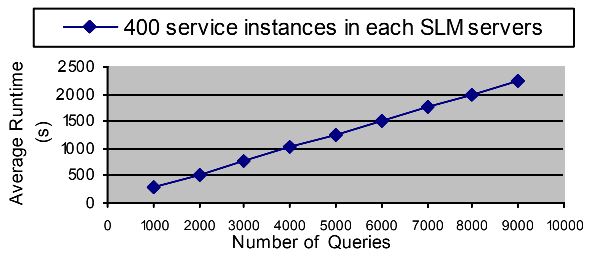

Figure 8. Average runtime for 400 service instances in each SLM servers

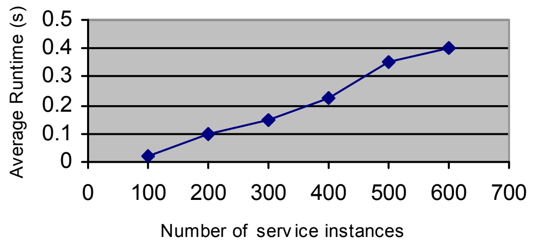

Figure 9. Average runtime of each search request for different number of service instances

in each SLM server. A SLM client on a separate machine made 10,000 requests continuously to Server D as indicated in Figure 7. For every 1000 requests, we measured the average runtime taken by our SLM system. We found the average runtime for each search request is around 250 ms. Figure 8 shows the average runtime as a function of number of search requests for server with 400 service instances. The average runtime for attribute searching is nearly proportional to the number of requests. The attribute matching mechanism performs well as the number of client requests increases. It also demonstrates the SLM system is reasonably scalable with respect to the number of users.

In Experiment 2, a SLM client on a separate machine made 10,000 requests continuously to Server D. We tested the attribute matching performance for different numbers of service instances running in each SLM server. Each time, a SLM server stored different number of service instances starting from 100 instances to 600 instances. We measured the average runtime of each search request for different number of instances as shown in Figure 9.

The runtime for attribute matching is also nearly proportional to number of service instances. When the SLM servers store more service instances, the attribute matching performance will gradually decrease. This can be explained below.

In our current implementation, all service instances are stored in memory of SLM server machines, which are running JVM (Java Virtual Machine). When number of service instances increases, the required memory also increases. The operating system is increasingly spending more time in memory swapping, which deteriorates performance. Adding more memory on the SLM server machines will help to improve the performance. In the future, we plan to store service

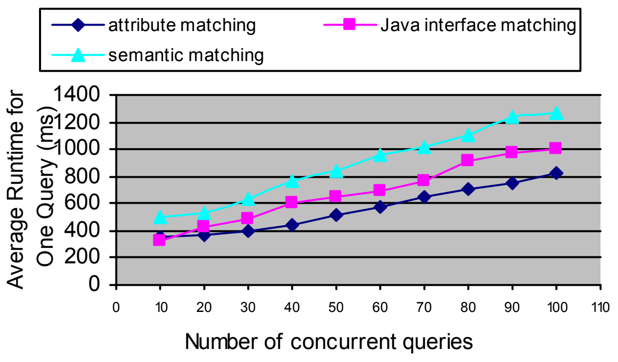

Figure 10. Comparison of multiple matching performance

instances into database that can deal with large volume of data and hopefully this can better the searching performance. We also plan to simulate our SLM system in wide-area networks to further test the scalability by using network simulation tools.

4.2 Performance of The Multiple Matching Engine

In Experiment 3, we measured and compared the performance of attribute matching, Java interface matching and semantic matching engines. We created 400 service instances using service templates, DAML-S descriptions and Jini interfaces separately. A SLM client on a separate machine made multiple concurrent queries to the SLM server, and we measured the average runtime of one query for the three matching engines. For each matching engine, the measurements were repeated for 10 times and the average results were collected.

As shown in Figure 10, among the three matching engines, the attribute matching and the Java interface matching performs better than the semantic matching. They have a similar trend that is the average runtime for one query increases proportionally to the number of concurrent queries. By further studying the factors affecting the average runtime of a search request for the semantic matching engine, we found the reasoning process has consumed a significant time in the overall matching process. It is the major factor which caused the performance difference compared with the other two engines.

We also find that the runtime for RMI operation is another important factor in terms of overall performance. As Java RMI is used in our system as a basic communication mechanism between a client and a server, hence, enhancing Java RMI operation will significantly improve the performance of all the three matching engines.

5 CONCLUSION

We have proposed and implemented a prototype of a service discovery system known as SLM. Tests and performance measurements indicate our SLM system is functional, and seems capable of providing an improved and flexible service discovery for networked services. The SLM system automatically adapts its behavior to handle dynamic changes of both SLM servers and services, hiding the complexities of internal mechanisms from users and service providers. Our SLM system is scalable (linearly) and dynamic as it adopts the dynamic tree structure, and is flexible as it has multiple service matching mechanisms. The idea of service aggregation is embedded in our SLM system, which is used for a faster searching and minimizing communication between SLM servers.

REFERENCES

[1] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2", IETF RFC 2608, June 1999.

[2] Jim Waldo, Ken Arnold, "The Jini Specification Second Edition", Addison-Wesley, 2nd edition, December, 2000.

[3] "Java Remote Method Invocation - Distributed Computing for Java", http://java.sun.com/marketing/collateral/javarmi.html

[4] UPnP White Paper, http://upnp.org/resources.htm/, June 2000.

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

[6] The Salutation Consortium, "Salutation Architecture Specification (part 1), version 2.1 edition", http://www.salutation.org, 1999.

[7] Thomas A. Bellwood, "UDDI - A Foundation for Web Services", Proceedings of XML Conference & Exposition. Orlando, Florida, December, 2001.

[8] Web Services Description Language (WSDL), W3C, http://www.w3c.org/TR/wsdl12.

[9] Simple Object Access Protocol (SOAP), W3C, http://www.w3.org/TR/SOAP

[10] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, Sanjiva Weerawarana, "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI", IEEE Internet Computing, March 2002.

[11] Steven E. Czerwinski, Ben Y. Zhao, Todd Hodes, Anthony D. Joseph, Randy Katz, "An Architecture for a Secure Service Discovery Service", Proceedings of the fifth Annual International Conference on Mobile Computing and Networks, Seattle, WA, August 1999.

[12] Adjie-Winoto, W., Schwartz, E., Balakrishnan, H., Lilley, J., "The design and implementation of an intentional naming system", Proceedings of ACM Symposium on Operating Systems Principles. (1999) 186–201.

[13] M. Balazinska, H. Balakrishnan, and D. Karger., "INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery", In Proceedings of the First International Conference on Pervasive Computing, pages 195–210, Zurich, Switzerland, August 2002.

[14] Cai Hong Zhang, Hung Keng Pung, Sae-Whong Suthon, "OCTOPUS: A Middleware for Multimedia Communication", Proceedings of the sixth IASTED International Conference Internet and Multimedia Systems and Applications (IMSA 2002), Kauai, Hawaii, USA, August, 2002.

[15] Tim Berners-Lee, James Hendler, Ora Lassila, "The Semantic Web", Scientific American, May 2001.

[16] Ian Horrocks, "DAML+OIL: a reason-able web ontology language", Proceedings of the Conference on Extending Database Technology (EDBT 2002), March 2002.

[17] Resource Description Framework, World Wide Web Consortium, http://www.w3c.org/rdf.

[18] Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne and K. Sycara, "DAML-S: Web Service Description for the Semantic Web", Proceedings of the First International Semantic Web Conference (ISWC), 2002.

[19] Fikes, R., Jenkins, J., & Frank, G., "JTP: A System Architecture and Component Library for Hybrid Reasoning", Proceedings of the Seventh World Multiconference on Systemics, Cybemetics and Informatics, Orlando, Florida, USA, July 2003.

[20] Genesereth, M. R. & Fikes, R. E., "Knowledge Interchange Format, Version 3.0 Reference Manual", Technical report, Knowledge Systems Laboratory, Stanford University, June 1992.