ARCHITECTURE AND PROTOCOLS FOR SANGAM COMMUNITIES AND SANGAM E-SERVICES BROKER

By

ARUN SWARAN JAGATHEESAN

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

To my parents (Dr. Bhanumathi and Dr. Jagatheesan) and teachers.

ACKNOWLEDGMENTS

The motivation and support provided by many individuals resulted in this thesis. Their encouragement, discussions, and advice will continue to help me.

Firstly, I would like to thank my advisor, Dr. Abdelsalam (Sumi) Helal who has always remained a source of motivation. He gave me the opportunity to work on this thesis. His enthusiasm, guidance and tireless patience with any issue helped me in this research work. I would like to express my gratitude to my co-advisor, Dr. Stanley Su who recommended me for this research when I mentioned my interest. His support was inspiring during this work.

I am also thankful to Dr. Sanguthevar Rajasekaran who kindly agreed to serve on my supervisory committee. It has been a great honor to be a part of a research team and work with Raja Krithivasan and Jie Meng. Sharon Grant, Mathew Belcher and all of friends in the Database and Harris Research Labs deserve a special mention for their support.

I would like to acknowledge Professor Shanmugam who took the initiative and talked to my parents regarding my higher studies. Finally, I would like to thank my family members for their support in all my endeavors.

TABLE OF CONTENTS

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

ARCHITECTURE AND PROTOCOLS FOR SANGAM COMMUNITIES AND SANGAM E-SERVICE BROKER

By

Arun Swaran Jagatheesan

August, 2001

Chairman: Dr. Abdelsalam (Sumi) Helal
Major Department: Computer & Information Science & Engineering

The concept of e-services is poised to bring about rapid and significant change in society by giving birth to virtual tools that will completely change some of the day-to-day activities. The evolution of numerous e-services will make automated service discovery and binding concepts very important in future distributed computing scenarios. Some of these scenarios include workflow, e-business, self-organizing ad hoc networks and pervasive computing. With automated service discovery, devices or users may automatically or semi-automatically discover e-services and services may advertise their existence in a dynamic way.

The motivation of this research work is to provide a highly scalable, interoperable and end-to-end architecture along with the protocols for constraint-based brokering of services leading to the proliferation of distributed and ubiquitous computing using web services. *Sangam communities* are hierarchical communities formed by the service

brokers along with the service providers and service requestors. Instead of inventing its own protocol stack, Sangam relies on standards-based technologies like TCP/IP, HTTP, XML, SOAP and UDDI to create services based communities. The *Sangam protocols* (or Sangam) used by the participants of the Sangam communities are proposed as a step toward *Universal Service Interop Protocols* with regard to service discovery and binding.

Sangam extends UDDI by forming communities of service brokers. The service description is done using service interfaces (or templates) using Web Services Description Language (WSDL). Service Providers provide binding information of these templates using service implementation documents in WSDL. Service Requestors use the service templates to make a service query to the broker. The service query can have certain service constraints regarding the service requested. If the broker could find a service provider who can meet the service constraints of the service query, the binding information to use the service is returned to the requestor. The requestor can directly contact the service provider to use the service.

This thesis contributes the Sangam architecture and protocols. An e-services broker implementation based on the architecture and protocols is presented. The usage of Sangam e-services broker in workflow has been studied and is presented as a sample scenario.

CHAPTER 1
INTRODUCTION

"Imagination is more important than knowledge"–Albert Einstein.

Thesis Objective

The recent advancements in e-commerce, pervasive computing and user personalization give a futuristic vision of ubiquitous e-services. The purpose of this research is to enable proliferation of pervasive computing by providing the architecture and protocols for applications or devices to dynamically discover and bind to e-services. One objective of this thesis is to study the requirements and challenges of dynamic discovery and binding techniques. A survey of related research is needed. Architecture and the associated protocols required for using ubiquitous e-services, using any device anywhere, must be formulated.

Once the architecture is developed, service descriptions of any e-service may be published. The implementation of the design provides facilities for e-services to register themselves with a service broker and for applications to discover desired service. The discovered service can be invoked programmatically.

Motivation

One of the fundamental needs of any software application in the future is to provide highly personalized services to users by using the external services available to the application. A user's home, car and other mobile devices may be connected, forming a pervasive environment. In such an environment, applications must be able to take

advantage of data and services offered by the other devices. Also applications must be able use the e-services available for users. For example, a car must be able to take advantage of the traffic tracking service; and a house inventory system must be able to take advantage of the grocery-order service from a super-market. Also, business enterprises must be able to discover other business services, which can provide a profitable supply chain. This business service discovery process will help in the automation or semi-automation of business-to-business (B2B) integration. In the scenarios mentioned above, we need to dynamically discover services that are interoperable and meet the requirements of the user before binding the service to the user application.

The motivation for this research is to identify shortcomings in the current approaches to dynamic service discovery and binding for ubiquitous e-services. We also need to propose and implement our approach to use well-accepted standard technologies and protocols, whenever possible. Another motivation for this research is to facilitate current and future research in areas of Workflow [MEN 00a, HEL 01a] and Pervasive Computing at the Database Systems R&D Center and the Harris Networking and Communications Lab where this research is conducted.

Need for Dynamic Service Discovery and Binding

The advancements in various fields of computing have indicated the need for dynamic service discovery and binding of network services and e-services. We summarize some of the emerging trends.

Service-Oriented Programming

Service-Oriented Programming (SOP) builds on Object-Oriented Programming (OOP), adding the premise that problems can be modeled using services that an object can provide or receive [BIE 01]. Unlike the client server model, in SOP clients are not tied to a particular server. The service providers can be changed dynamically. In SOP, services publish and use other services in a peer-to-peer manner. This requires service discovery. It must be noted that the process of service discovery could also ideally be a service by itself.

Advancements in distributed computing and Internet computing suggest that future applications might be based on SOP. Also, mobile devices, which have less processing and storage space, might require external services for their functionality and may use SOP for modeling. This in turn requires service discovery and binding.

Pervasive Computing

Pervasive computing at its core is about three things [BAN 00]: First, it concerns how users use (mobile) computing devices within their various environments to accomplish their tasks. Second, it concerns how applications are created and deployed to accomplish various tasks. Third, it concerns how the environment is enhanced by the ubiquity of new resources in computing.

In a pervasive environment, the participant devices must know all the services available from other devices. An application, rather than being viewed as software that is written to exploit a device's capabilities, must be viewed as a means by which the user in a given environment can perform a task. Thus, the software or the means to perform a task must be accessible from the device on which it runs and from other devices of the environment in which it runs. This requires the devices to know all the members of the

environment with which they can interact and to discover services that are interoperable to them.

Dynamic E-business

The broad integration of systems among enterprises and across value chains provides collaborative commerce. The next stage of e-business, *dynamic e-business* [IBM01] involves integration of systems across Intranets, Extranets and the Internet so that the integration is dynamic. This gives flexibility needed for changes in the market and for business decisions.

Although many businesses are online and willing to interact with others, the fundamental challenge is to discover other possible businesses that are interoperable and to integrate with any of the desired business. Until recently, integration generally involved customized applications, which were hard-wired to only a few of the business partners. This restricts the flexibility of changes in the business rules and decision-making. A dynamic e-business, is needed that can discover and bind with any interoperable business service based on the business requirement at any given time.

Composite Services

Once a service is well understood and documented, instead of inventing the wheel again with a new service that performs the same task, *service reuse* must be made. This encourages the same service description for a given task. Higher-level services can be constructed using the stable services available [MEN 00b].

The ongoing research work [KRI 01] suggests a methodology to compose complex services using the simple e-services. The approach used here is to discover simple e-services and compose higher-level services on-the-fly based on requests. The binding here can be either dynamic (at the run time) or can be hard-wired if the service

designer wants to use only a particular service. The ability to compose more complex services increases as standardized means of lookup or discovery are achieved.

Ad-hoc Networks

Requirements for configuration and reconfiguration of network devices have changed a lot in recent years. The exploding growth in networked digital devices in diverse "real world" and "hostile" environments such as industries, cars and homes, has increased the need to simplify network administration for all networks. *Zero Administration* networks, which need almost no maintenance or administration, are envisioned. A variety of new protocols [MCG 00, GUT 99a] have been proposed, which attempt to provide automatic discovery and configuration of network devices.

Workflow Systems

Most of the activities in our day-to-day work involve many tasks that need to be executed in a specific order to complete the activity. The tasks involved in an activity may vary based on the results of the other tasks executed. This dynamic sequence of execution of tasks to complete an activity leads to a workflow. Cichocki et al. [CIC 00] studied some of the workflow systems. Workflow using e-services forming an Internet Workflow is explored by Helal et al. [HEL 01a].

The workflow engine manages the sequence of execution of tasks in the workflow. In an Internet-based workflow, the workflow engine needs to decide on the tasks necessary to complete a workflow and has to determine the appropriate service providers able to complete the task. A task might be completed using the services provided by different service providers in various possible combinations. The interoperability of different service providers has to be taken into account before binding

with the service. An e-service broker can be used to discover the service and the required type of service binding. An e-service broker for workflow is discussed in Chapter 4.

Proposed Approach

We focus on discovery of interoperable and more relevant e-services offered by businesses, as one of our motivations is to support brokering of e-services in e-business and dynamic workflow environments. This approach could also be used for services offered by the devices in a pervasive environment using a scaled-down local version of the broker used here.

Services need to be described and published (we explain the process of publishing a service later in detail) in a standard format before they can be discovered or brokered. This thesis explores a standard way of describing service using WSDL. A service broker is used in our approach for discovery of services. The broker maintains a registry of services and their interfaces that can be used to invoke the service.

We use the term "*service provider*" to denote the business or the device or the process that provides a service. Similarly, "*service requestor*" is used in this thesis to denote the business or the device or another service that makes a request for the service. A request for the binding information about a service is referred to as "*service query*". The service broker can also have other requests or queries regarding the interfaces or service templates (explained in Chapter 4). The broker uses the registry formed using the services published to generate result sets for the service queries from the service requestors.

As our approach toward brokering of ubiquitous and interoperable e-services, we propose Sangam as the architecture for forming a *hierarchical brokering community*.

Distributed applications and pervasive devices can use Sangam to dynamically find services and bind to them. Sangam is a single solution to the problems related to scalability, interoperability and flexibility in dynamic discovery and binding. As we show in our study of the existing protocols and emerging standards in the next chapter, we currently don't have a single solution for all these problems. Some protocols can be useful in the local network, but cannot be scaled to the Internet. Some protocols can be used in the Internet but lack flexibility or interoperability. Sangam brings the best of these different technologies using a combination of the concepts of the existing technologies. The Sangam suite of protocols (Chapter 3) has both stateless and state-based protocols.

Sangam uses a community of brokers who use a *publish-and-query* mechanism to broker e-services. The protocols used are based on HTTP and XML, which today are synonymous with "Internetworking" and "interoperability" making Sangam the next generation of ubiquitous protocol to help in many fields of computing. Sangam has been kept very simple at the bottom layer where most of the interaction takes place. Also Sangam's bottom layers are stateless protocols making them easy to implement in small devices.

Thesis Contributions

Based on our research on service discovery and brokering, we make some contributions:

- Sangam: We propose architecture design for scalable and hierarchical brokering that can be used in various fields of computing. This architecture can be used with *Sangam Protocols* to create communities of brokers.
- We present an approach for programming for M2M (Machine-to-Machine) business using Service-Oriented Programming concepts.
- We suggest a methodology to embed constraints in service description.

- We provide a brief survey of related technologies.
- We also present an implementation of a Sangam-based broker and a sample scenario for service discovery.

Thesis Organization

This thesis is organized into six chapters. The technologies that relate to service discovery and e-service brokering are discussed briefly in Chapter 2. The architecture of Sangam and its protocols for e-service brokerage are discussed in Chapter 3. An implementation of Sangam Broker for workflow is discussed in Chapter 4. Sangam API and its usage in programming along with screen shots are shown in Chapter 5. We summarize our research and give our conclusions in Chapter 6.

CHAPTER 2
RELATED TECHNOLOGIES

"Enthusiasm is contagious. You could start an epidemic!"–Unknown.

This Chapter discusses service discovery technologies that relate to this thesis. The first section of this chapter discusses the various terminology associated with e-service discovery. The next section discusses some of the existing protocols used for service discovery. The last section of this chapter discusses emerging standards and technologies (such as SOAP, UDDI and WSDL).

Terminology Used in Thesis

E-Services

E-services can be defined as the distributed services that can be accessed via the Internet through standard protocols. Although this definition is generic, business services offered as e-services exhibit many advanced properties [SAH 01]. Some of the generic properties of e-services:

1. E-services are described using some standard formats or languages
2. They are generally published in well-known places like registries or brokers for the prospective users to know about it.
3. One of the foremost requirements for an e-service to be useful is that its prospective consumers must be able to discover it from the registries.
4. E-services can be invoked via the Internet by forming instances or proxies of the service.

5. E-services are *composable*. Two or more e-services can be combined together to form new services and applications.

Devices and Services

Devices and services [MCG 00] are the entities that participate in service discovery. “Devices” include conventional computers, network devices, mobile appliances, printers etc. “Services” include any service available over the internal network or an e-service that can be accessed over the Internet.

Service Lookup and Discovery

We use “lookup” and “discovery” as shown in [MCG 00]. The term “lookup” is used to refer to the process of locating a specific object or resource. Lookup may be made by exact name or address, or by some matching criteria. Lookup is initiated by a requestor and requires the existence of some directory or agent to answer the request. Lookup is usually done in a statically configured environment. Examples of lookup services are DNS [MOC 87], LDAP and CORBA Naming Service.

In contrast, “discovery” is a more spontaneous process in which many entities discover the other entities on the network and present themselves to other entities. Discovery protocols have the overall goal of making digital networks easier to create and use. Many “lookup” services do not support discovery. The important features of a discovery protocol are as follows:

- Selection of specific types of service.
- Almost zero human administration required.
- Interoperability across manufacturers and platforms.

Static and Dynamic Binding

Service discovery and binding can be either *static* or *dynamic*. In *static binding*, information about the required service is discovered (using a service broker), during the

design time or compile time. In most cases, the designer of the service request locates a service that is interoperable and meets the required functionality. Once a desired service is located, the components of the requestor's application are modified to make a request to the service located. Thus the request is hard-wired using the software to a particular service implementation. Since the software logic is hard-wired and points to a single service, the application need not discover the service again. This approach eliminates the calls to the broker and may be suitable for applications, where only a pre-selected service will be used.

The request made by the application to the selected service must have the necessary information expected by the service. Once the selected service gets a request, it processes the request and sends a response (if any) as the output of the request. If there is a response to the request, the service requestor must be aware of the format of response and must extract the required information from the response message. The service response may be either synchronous or asynchronous.

Dynamic binding involves binding of services at runtime. Dynamic binding has some elements in common with static binding. But, it has some unique objectives. Dynamic binding focuses on adapting to the environment formed by the different applications and business rules. Locating services at run time that will provide maximum benefits for a given environment based on the business rules of the user initiates dynamic binding. The system must be able to access a broker, with knowledge of the available services and their service providers. The application (service requestor) might invoke the service whenever required.

Existing Service Discovery Protocols

In this section we survey the existing protocols that may be used for service discovery.

Lightweight Directory Access Protocol (LDAP)

The LDAP is an IETF standard providing a scalable hierarchy of name spaces, through which services can advertise and clients can locate services [WAH 97]. The LDAP by itself does not specify any discovery protocols but can be used for service announcements and requests. The LDAP is a "heavy weight" protocol, which would not be optimal for implementation on small devices.

CORBA Trader Service

The CORBA Trader Service (TS) provides an infrastructure for advertisement and request of services by attributes. This makes it possible for any device or service that has a CORBA implementation, proxy, or wrapper to use the Trader Service to advertise its attributes. Requests may be made as constraint expressions. The CORBA TS supports federation of trader services into a single logical service.

The CORBA TS was not selected for use in this thesis for several reasons. Implementations of CORBA on different platforms are needed and CORBA protocols may be difficult to implement in small devices. Configuration of CORBA may be rather difficult. Using IDL to describe each component interface delays the development time. Sangam protocols use the concept of communities, which is similar to federation in CORBA or JINI.

Universal Plug-and-Play (UPnP)

Universal Plug and Play [MIC 00] is designed primarily for the discovery and enumeration of devices that are attached to a network. It provides a way for devices to be

discovered and used directly by other devices in a P2P (peer-to-peer) environment with or without the use of a PC. The UPnP messaging protocol has three parts: Announce, Discovery and Response to Discovery. The Simple Service Discovery Protocol (SSDP) is used within UPnP to discover services. SSDP uses HTTP over UDP. Services are described in XML.

UPnP is a discovery service. But, its usage is proposed for small office or home computer networks, where it enables peer-to-peer mechanisms for auto configuration of devices, service discovery and control of devices.

Service Location Protocol (SLP)

Service Location Protocol is an IETF standard for "spontaneous" discovery of services [GUT 99b]. SLP establishes a framework for resource discovery that uses a set of agents to operate. A User Agent (UA) is the client of a service and performs service discovery on behalf of the client software. A Service Agent (SA) advertises the location attributes of a service. The Directory Agent (DA) is used to place service information into a repository and for lookup.

SLP can be implemented in several configurations. SLP provides "passive discovery" in which, the DAs periodically multicast service advertisements. In "active discovery", the UAs and the SAs multicast all the SLP requests to the network. Service Templates specify the attributes for a particular service type. SAs advertise services according to these attribute definitions so UAs must make requests for services using these same definitions. This ensures interoperability among vendors.

SLP can be used in a single LAN to an enterprise network. But, one of the drawbacks in SLP is that it relies on multicast discovery mechanisms. This prohibits the usage of SLP in Internet where very high scalability is required.

JINI Lookup Service

Jini Lookup service provides a set of Java-based mechanisms and APIs for service lookup and spontaneous discovery [EDW 99]. Jini is similar to SLP in many ways. Clients can discover the existence of a Jini Lookup Server in a manner that is similar to Discovery Agents (DA) is SLP. A Java Client Program begins "discovery" with a multicast of UDP/IP to locate instances of the Jini lookup service. In SLP, the service discovery returns a URL denoting the location of the service, whereas Jini returns a service object that offers direct access to the service.

Jini uses exact matching algorithm on Java serialized objects. This will restrict the matches made. Also, Jini requires service templates based on Java objects. This is not always efficient. More over, Jini lacks the scalability required for massive e-business applications on the Internet and is tied up with the programming language Java.

At the end of the survey, we find that these existing protocols are not scalable for the Internet to the requirements of system to perform brokerage of e-services on the Internet. In the next section, we analyze some of the emerging standards and methodologies for e-service brokering (At the time of writing this thesis, some of these methodologies are yet to be accepted by the standards committee).

Emerging Standards and Methodologies

Since Web based services or e-services are becoming the building blocks for constructing distributed Web-based applications, tremendous interest has been generated in the industry to find a standard means to describe and discover e-services before using them in applications. Also the industry realizes the need to have open Internet standards for description and discovery of Web Services.

There are tools [KRI 01] and techniques for building e-services either using object technologies or by making wrappers for legacy applications. Once e-services become available for business, other businesses need to discover them and use them in their applications. Since the focus of this thesis is in coming up with standard protocols and methodologies for discovery of e-services, we take a look at the emerging technologies, which are related to service discovery, description and invocation.

Microsoft .NET Framework

.NET is Microsoft's Framework to build web applications and web services that can connect to appliances, web services and legacy applications on the Internet. .NET lays its foundation on a Common Language Runtime (CLR) that can be used to run applications written in any of the supported languages. This runtime supplies many services that help simplify code development ad application deployment. The .NET Framework also includes a set of class libraries that developers can use from any programming language.

.NET does not by it self address the business conventions required for the automatic service discovery and business-to-business interactions. It relies on other technologies like UDDI (discussed later) which could be used in the programs to discover and integrate with e-services.

Simple Object Access Protocol

Simple Object Access Protocol (SOAP) [BOX 00] is a XML/HTTP protocol for accessing services, objects and servers in a platform-independent manner. E-service discovery has to be highly interoperable. XML is used to represent data for interoperability. Even though SOAP directly does not help in service discovery, it is

useful in the transport of XML documents and provides an ideal way for invocation of services including the discovery service that might be offered by a broker.

SOAP is a lightweight, platform agnostic protocol which is highly scalable compared other remote method and object broker protocols like RMI and CORBA. Moreover it is built on accepted standards like HTTP and XML.

E-Speak

Hewlett-Packard Company (HP) has been investigating on e-services and other related technologies and has contributed e-Speak [ESP] and Cool Town [COOL]. HP's e-Speak platform consists of Service Framework Specification (SFS) and the e-Speak Service Engine. E-Speak is based on open standards. HP's SFS defines the technical and business conventions required for dynamic business interaction marketplace. SFS creates a new standard for dynamic business process collaboration on the Internet.

The e-Speak service engine can take a service query and search all repositories of service descriptions available to it. Searches are made on local repository and a global repository of the addresses for all e-speak engines belonging to the same community. But, the broader goal of e-speak is to implement cross compatibility among different marketplace enablers and the lookup abstraction in e-speak is separated out of the core functionality as a separate service. The current e-speak advertising service depends mostly on LDAP for its service registration and lookup. It also has a built-in multicast mode for operating in a LAN. Sangam uses a similar concept of communities by using local broker and hierarchical global brokers.

Universal Description, Discovery and Integration (UDDI)

The Universal Description, Discovery and Integration (UDDI) project is a comprehensive, open industry initiative by the software industry. The project creates a

platform-independent, open framework for e-businesses to interact with each other. The UDDI enables businesses to:

1. Discover each other.
2. Define their interface using which other businesses can interact with them
3. Share categorized information in a global repository of e-services.

The UDDI specifications [UDD 00] define a way to publish and discover information about Web services. UDDI describes a conceptual cloud of Web services and a programmatic interface that defines a simple framework for describing any kind of Web service. The specification consists of several related documents and XML schema, which define SOAP based programming protocol for registering and discovering Web services.

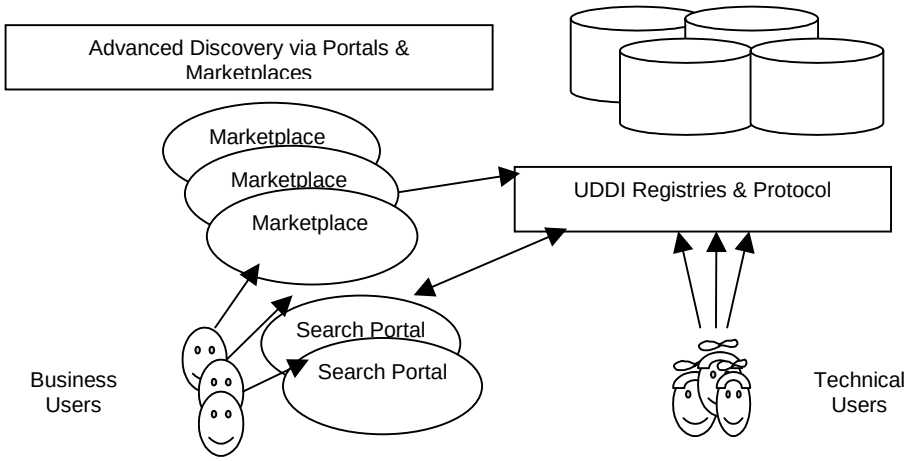

Figure 2-1. Use of UDDI registries for service discovery (Courtesy: www.uddi.org)

Figure 2-1 taken from the UDDI technical specification shows the relationship among the UDDI specifications, XML schema and UDDI business registry cloud. As shown in the figure, Business Users can use brokers or marketplaces, which internally

use the UDDI registries to find other businesses. Also technical users can have applications that either publish or use the data from UDDI.

Using UDDI discovery services, businesses can register about their web services to other businesses. This information can be added to the UDDI business registry either via a Web site or by using tools that can use the programmatic service interfaces described in the UDDI Programmer's API Specification [BOU 00]. It is important to note that UDDI does not form a full-featured discovery service. UDDI is designed to complement the existing online marketplaces and search engines by providing them with standard formats for programmatic or automatic business and service discovery. Businesses may use any advanced discovery portals or marketplaces which lookup UDDI.

As we will see later, Sangam Protocols stack uses UDDI protocols. Since it is being widely accepted by almost all the major B2B vendors, we expect it to become a standard in the near future for basic e-service categorization and discovery.

Service Description Languages

Service description languages are needed to in dynamic service discovery architecture to know information about service location, service availability, parameters and messages involved in the service.

Extensible Markup Language (XML)

The XML (eXtensible Markup Language) [BRA 98] is used to format semi structured information containing both content and semantic meaning. A markup language is a mechanism to identify structures within a document in an easy and

interoperable way. An XML document can have DTD or Schema associated with it. XML provides a convenient and highly effective way to encode service specification.

Even though XML can be used to describe Web services, we will need standard DTD's or schemas to describe them. Without standard tags or schema information, every one might have their own DTD's or schemas.

Web Services Description Language (WSDL)

The WSDL [CHR 01] has been submitted as a suggestion for describing services for the W3C XML activity on XML Protocols by Ariba, IBM and Microsoft. "WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint".

A WSDL document defines "*services*" as collections of network endpoints or "*ports"*. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: "*messages"*, which are abstract descriptions of the data being exchanged, and "*port types*" which are abstract collections of "*operations*". The concrete protocol and data format specifications for a particular port type constitutes a reusable "*binding*". A port is defined by associating a network address with a reusable binding, and collections of ports define a service. An explanation and example of a WSDL document is given in Chapter 4.

The SOAP services can be easily described using WSDL documents. Our implementation uses WSDL documents for describing e-services. It must be noted that WSDL by it self can not specify enriched constraints like the inter-attribute constraints.

This is a requirement in some cases for fully automating discovery and the negotiation that might follow between the service provider and service requestor.

After our preliminary survey on the emerging technologies that might become standards in the near future, we find that each one of them has technologies of our interest in service discovery for ubiquitous e-services. While UDDI may be a possible way to maintain our registry, we don’t want all our services to be registered to the public UDDI cloud. Also, we need to consider if UDDI can be used in out Universal Interop Protocol Stack formed using Sangam Protocols.

Now that we are familiar with the protocols and technologies available for service discovery, we will discuss about the project Sangam for e-service brokerage and its overall architecture in the next chapter.

CHAPTER 3
SANGAM: E-SERVICES BROKERAGE

"Birds of a feather flock together"–Proverb

This chapter discusses about the Sangam project which is a combined R&D effort of the Database Systems R&D Center and the Harris (Pervasive) Networking and Communications Lab at the University of Florida. We give a brief description of Sangam and its approach to service brokering in the first section of this Chapter. The next section is on Service Description. After that we propose the architecture of Sangam. The next section describes the of Sangam Protocols.

Sangam

Introduction to Sangam Project

The Sangam project was started to contribute Architecture and the Protocols for the brokering of e-services. The focus of the brokering here is in the fields of dynamic Internet workflow and pervasive computing.

Why "Sangam"?

The word "Sangam" in most languages of the Indian subcontinent means "the confluence of various entities which benefit from each other." The mission of this project as mentioned in Chapter 1 is the confluence of service providers and service requestors who can benefit from each other.

Sangam's Approach

Sangam uses a combination of the good concepts, which are similar to those of some of the existing technologies. It also uses both state based and stateless protocols in its different layers.

We felt that a registry of e-services is needed for the broker to perform service-discovery. The look-up services discussed in Chapter 2 (like CORBA and LDAP) cannot be used in our case, as we need spontaneous discovery. Jini was not preferred as it cannot scale up and will be tied up to programming language. UDDI was decided to be used as it is emerging to become a standard with lot of support from different companies. Also, UDDI gives a set of protocols based on SOAP and relies on XML for interoperability.

But UDDI alone does not solve the problem. We cannot publish all our services especially the private services to the public UDDI cloud. We need local or private UDDI nodes. So, our brokers will use individual UDDI nodes to maintain a registry of services known to them. Since UDDI does not form a fully featured service discovery, the brokers using UDDI must have some more logic to rank or select services based on the service requestor's query. The service query from the requestor may have some constraints regarding the service. These constraints may be used for the selection of service(s).

By using the local UDDI, the service requests of a Business Organization, which have their own internal service providers, can be satisfied. This forms a poll of service providers and service requestors who are local. This pool is similar to JINI's federation formed by the registration of all the internal service providers. The service requests for these services are satisfied locally.

On the other hand, when services which are not available within the local registry are requested, an external service request has to be made to another broker who

specializes in that category of service requests. This external broker called the *domain broker* is specialized in a particular domain and has a greater knowledge of the service providers of that particular domain. This domain broker is part of community of service brokers who also specialize in the same domain. Thus even if the domain broker can not satisfy a service request by itself, it will pass on the request to the community members of that domain who will have a service that can match the request. This concept of communities is similar to e-Speak communities, where the service engine contacts other service engines in a community. The advantages gained by forming communities include very large scalability and great flexibility in choosing brokers. Sangam thus shares some concepts from UDDI, JINI and e-Speak.

Sangam uses service templates like JINI. But the service templates of Sangam are not Objects that can restrict usage based on the programming language. Sangam uses WSDL (based on XML Schema) to describe its templates. These templates are similar to UDDI service interfaces represented using WSDL. The advantage gained here is interoperability of these descriptions. Both the service providers and the service requestors use templates to advertise or request for a service. Thus the service template becomes the strategic standard document based on which, the service provider and requestor perform brokering and transactions.

During our analysis of service description we found that in most cases it may be advantageous or necessary to embed service constraints along with the description. These constraints can be later be used to match the service providers with the service requestors. In the next section of this chapter we mention a methodology to describe the service constraints along with service.

Service Description and Brokering

Description of a service has to be made before it can be it can be published or discovered. Sangam uses the concept of service templates as used in JINI or UDDI interfaces. Instead using objects as template (like JINI), Sangam uses documents defined based on XML Schema. A service description is divided into two parts: *Template (Interface) description* and *Binding description.* Each service implementation will be associated with a generic template description and its own binding description.

Template Description

Template (or Interface) description of service has information regarding the names of the operations that can be invoked in a service and their respective input and output parameters. For example consider a *calculator e-service* that has two input operations *calculate* and *generateRandom*. The *calculate* operation may accept three parameters, namely: operator, operand-1, operand-2. But the *generateRandom* operation does not accept any parameter. When a developer (service requestor) wants to use the *calculator e-service* in his/her application, he or she has at build time has to know the name of the service, the name of the service operation and its operands. Also the developer needs information on the data type of the operands. All these information is provided in the Template description. It must be noted that another service provider who wants to implement the same service can use the same template description. So, a given service template could have many service implementations, by different service providers. Hence service requestor applications once designed based on an template (say "calculator_interface") can use any service implementation (from any service provider) that is based on the same template ("calculator_interface"). The only difference in service

description between two service implementations that follow the same template is their binding description.

Binding Description

Binding Description for a service gives information on where the service is located and how it can be accessed. Each service implementation will have a binding description. The binding information gives descriptions like the protocol type(s), port(s), URL etc to access the service.

An application can use this information to find a desired protocol that can be used to invoke the service programmatically. So at runtime, when an application needs a service, it queries the broker with a service template and requests for the implementation of the particular service template. The broker returns the URI of the service provider who can provide the desired service.

In our implementation, we used WSDL to describe e-services. A description of WSDL is given in the next chapter. In the next sub-section we will see how constraints can be embedded in the service description and used in the e-service brokering.

Constraints Description

It is often necessary to have description about constraints regarding the attributes (parameters) of an e-service. Even though service implementations having same template description have same attributes for both input and output, they need not have the same constraint description. If consider the previous example (calculator_interface), even though the template may be the same for two implementations by different service providers (X and Y), X might output only positive numbers for the operation "*generateRandom*". Where as, Y might be able to generate both positive and negative numbers. If this information is not present in the service description, the service requestor

won't be aware of this. The input parameters may have some constraints. Apart from constraints on a single attribute, there can be inter-attribute constraints. Again consider the previous example (calculator_interface). There can be an inter-attribute constraint, the sum of operand-1 and operand-2 must be greater than 100. These service provider constraints can be described along with the binding description.

Apart from the constraints from the service providers (as we have seen in the above examples), the service requestor can also have some constraint on the requested service to be discovered. For example, the service requestor would like to have a service implementation of "calculator_interface" which can accept operands, which are negative. These service requestor constraints can be used in the service query.

Thus, broker needs to satisfy both the service provider and service requestor constraints. As indicated in [SU 87], constraints can be used in ranking different service providers who can provide the same service. The constraints of the service provider and the service requestor can be tested for satisfaction using Constraint Satisfaction Processor mentioned in [HAM 98].

We can describe constraints on the input or output parameters of a service using some constraint definition language. (In our implementation WSDL can be extended to describe constraints).

Sangam Architecture

As explained in previous section Sangam's Architecture is based on communities of brokers. The *Sangam Community* consists of local brokers, domain brokers and Super brokers. The local brokers are the e-service brokers, who represent a set of service

providers and service requestors. The community as shown in Fig 3-1 consists of *Service Layer*, the *Brokering Layer* and the *Super Brokering Layer*.

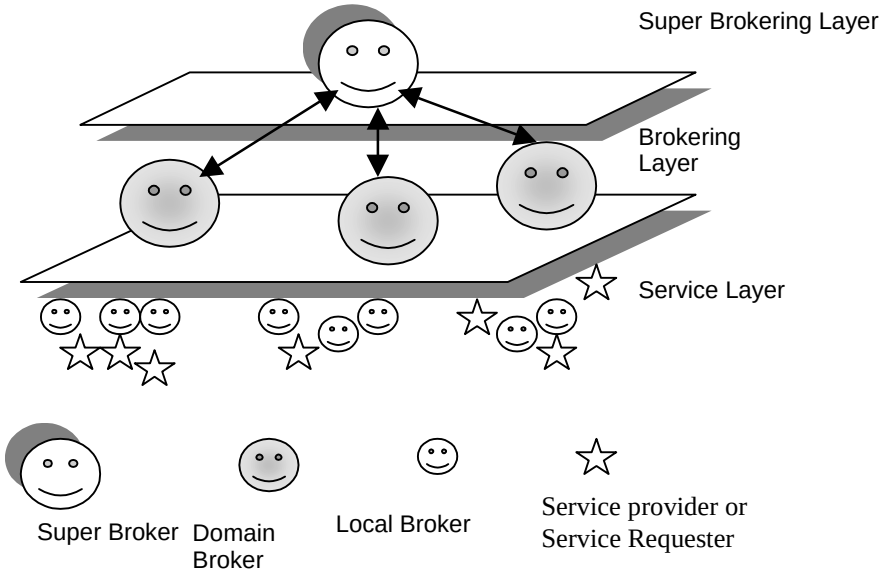

Figure 3-1. Hierarchical brokering architecture of Sangam

Service Layer

The *Service Layer* is the bottom layer in the community architecture hierarchy. This layer consists of service providers and the service requestors who interact with the local broker of this layer. This layer is called service layer as it contains the local brokers who represent the service provider and the service requestor. The service provider in this layer registers with the local broker with the description about the services it can provide. The service description registered (or published) with the local broker contains information on what template the provider has used to implement the service and what are its end-points to access the service. A service provider might register with any number of local brokers at the service layer.

The service requestor in service layer makes a service query to the local broker. The service query asks the broker to give one (or a list of) service provider(s) who

implement a certain service that conform to a certain service template. The service query may also provide some constraints that need to be used in making the selection of the service provider. The local broker in the service layer searches its registry for any service provider who has an implementation of the service template requested in a service query. Among the service providers who implement the requested template, the broker selects the service providers whose constraints (if any) and the constraints of the service requester (if any) are satisfied. The list of all the selected service providers may be returned to the service requestor or only a few service providers ranked by a certain order may be returned to the service requestor based on the request.

Brokering Layer

If the broker is not able to find a service provider who implements the requested template and satisfies the service constraints of the requestor, the request is sent to the *Domain broker* of the *Brokering layer*. The domain broker specializes in knowledge of service providers about a certain business domain. Every domain broker in the brokering layer registers with the domain Super broker of the community. When the domain broker gets a service query request, it tries to satisfy the request using the knowledge the available service providers. On the other hand if the domain broker is not able to find a service provider for the query, the request is sent to other brokers in the community using the Sangam protocols. The capable domain brokers in the brokering layer provide the requesting domain broker with information needed to obtain the service. The domain broker uses the information about the new information discovered to reply to the service query. Apart from replying to query, the domain broker updates it knowledge base with the information about the new service discovered. Thus this slowly leads to the expansion of knowledge among domain brokers. Also subscriptions can be made by brokers to get

notification when a capable service becomes available. The *brokering protocol* governs how brokers in the *Brokering Layer* advertise on behalf of the service providers; how service request are satisfied by the domain broker and how domain brokers self-organize themselves in the community.

Super Brokering Layer

The *Super Brokering Layer* is the top layer of the community. In this layer, the Superbroker helps in the self-organization of the community using the *Super Brokering protocol,* which governs how super brokers can communicate and coordinate over common templates among their communities.

The Sangam Community is easily scalable to a very large size by accepting new brokers into the community. When a broker joins or leaves the community the Super Broker updates the knowledge among the members of the community. Super Brokers have control over load balancing within their community and support greater reliability.

Researchers have designed different organizational structures for brokering systems, such as peer-to-peer multi-brokering architecture of *InfloSleuth* [NOD 99], the partitioned repository design in *Blackboard Systems* [CAV 92], centralized message routing design in *JAT Lite* [JAT 99], and the multi-facilitator mediation in OAA [MAR 99]. The hierarchical Sangam Community is designed to meet the current requirements of scalability, interoperability and highly customized brokering. Architecture similar to Sangam was proposed earlier in our previous work regarding agent communities in [HEL 01b]. Unlike the previous architecture using agents, Sangam uses UDDI at the service layer and builds on the existing standard protocols of the Internet. Sangam does not require any change in the underlying local broker system, which may be formed by private UDDI nodes.

The communication and interaction protocols specified in the Sangam Protocols are capable of avoiding single point failure, unlike those in the *JAT Lite* and the blackboard systems. Leader election protocol may be used in the event of failure of the Superbroker to elect a new Superbroker in the community. Each broker only needs to have the information about the Superbroker it is registered with, and communicates with other brokers only upon necessity. This reduces the amount of messages flowing among the brokers within the community. Also, the community becomes self-organizing, adapting itself based on the requests for service and the availability of brokers.

In the next section we take a look at the protocols of Sangam and how they allow the brokers in the different layers to interact with each other to help in service discovery.

Sangam Protocols

We have seen in the previous section that the Sangam architecture has a hierarchical architecture of brokers in the *Service Layer, Brokering Layer and the Super Brokering Layer*. Brokers in each layer interact with the other layer or among its peers in the same layer using the Sangam Protocols (referred in short as "Sangam"). Sangam consists of the *Service Protocols*, *Brokering Protocols* and the *Super Brokering Protocols*.

The protocols of Sangam were kept as simple as possible to allow others to understand and use these protocols (even they are not aware of the hierarchical architecture) in their systems. The protocols do not have any implementation dependency on programming language or operating systems. Sangam protocols are based on message exchange. The messages contain XML documents. The messages are SOAP messages

using HTTP. The usage of standards like SOAP, XML, UDDI and standard business categories, make Sangam the answer to e-business brokering.

Service Protocols

The Service Protocols illustrate the base level commitment and responsibility of the local brokers to the community. The local brokers to interact with the domain broker using the service protocol. The service protocols provide the following functionalities to the local brokers:

- Register/Unregister public services.
- Browse the categories supported by the domain broker.
- Browse the service templates present in the domain broker.
- Publish/Unpublish the service (binding) information based on an existing service template.

It must be mentioned that the service provider and service requestor can use the same service protocols to interact with the local broker. The interaction protocol among the local broker, the service provider and the service requestor are kept open. The reason for this is to allow existing brokerage systems to join Sangam communities without altering their own system with their clients. But, it is highly recommended to use UDDI based protocols along with the WSDL interface and service descriptions. The service providers register with the local broker and supply binding information needed for their services to the local broker either using UDDI based protocols or using any of their own proprietary protocol. This populates the service registry maintained by the local broker. The local broker can use this information in the brokering community if needed.

The local broker uses the Sangam service protocols to publish its public e-services (which can be accessed by any one) to the corresponding domain brokers based on the category information of the services. Whenever a public e-service is unpublished or

updated from the local broker, the local broker has to update the information in the community by contacting the corresponding domain broker with whom the service would have been previously registered.

Some of the Sangam messages in the Service Layer include "*getCategories*", "*putSerivce*", "*putServiceProvider*", "*removeService*", "*removeSerivceProvider*", "*getTemplate*" and "*getService*". Any local broker and domain broker confirming to Sangam Protocol Specification must implement all these basic Sangam messages. These messages of the Service Protocol are covered in detail along with their structure in Chapter 4.

Apart from these basic messages, the domain broker may provide some additional SOAP messages, which may be viewed as value added services like "getBestService". Sangam intentionally does not give any specification for these. The domain brokers may choose their own messages for value-added services in their own registry. Once the local broker has made a service query to the domain broker using the Service Protocol, the domain broker sends the query results as response back to the local broker. If the Domain Broker does not have the knowledge to reply to the service query, it uses the *Brokering Protocol* to contact with the other peer domain brokers or the Super Broker of the Community.

Brokering Protocols

The *Brokering protocols* describe a cooperative multi-brokering system, which provides the solution for interoperation among brokers in a dynamic and heterogeneous environment of service providers and requestors. Each domain broker performs basic brokering functionality, such as service discovery, dynamic service composition and knowledge sharing with the community. Domain brokers representing a set of e-service

providers can advertise their e-service templates and send capability queries to other domain brokers. The brokering protocols regulate the joining and leaving of brokers from a community. It also governs e-service knowledge discovery and sharing of acquired knowledge.

The Brokering Protocols define a set of messages among domain brokers to discover other service templates and services. Unlike the Service Protocol messages, which are stateless, the messages of the brokering protocol involve necessary state information. The reason for having state based protocol is to have more information about the domain brokers getting into the community and being selective in admitting other domain brokers into the community. This involves a small over-head at the Super-Broker of the community. But, it's a compromise to have highly valuable and trust-worthy brokers in the community.

The sub-protocols of the Brokering Protocol include Joining Protocol, Departure Protocol, Knowledge Discovery and Exchange Protocol and Capability Advertisement Protocol. These protocols are modified from BBACP [HEL 01a] used for agent communities to fit into the required Sangam protocols for Universal Interoperability of e-services.

Joining protocol.

The domain broker uses the *Joining Protocol* to join a community; it includes a logic conversation between the domain broker and the Superbroker. The conversation policy is a sequence of rule-based communication mapped into a finite state machine as shown in Figure 3-2. Each domain broker, in its application to join the community, specifies the service templates and categories it can deal. After receiving a registration attempt (State 1), the Superbroker conducts multi-faceted membership evaluation of the

applicant for credibility, possible value addition to the existing community, knowledge and capability relevancy among others. This evaluation results in *acknowledge* (State 1 –> State 3) or reject (State 1 –> State 6).

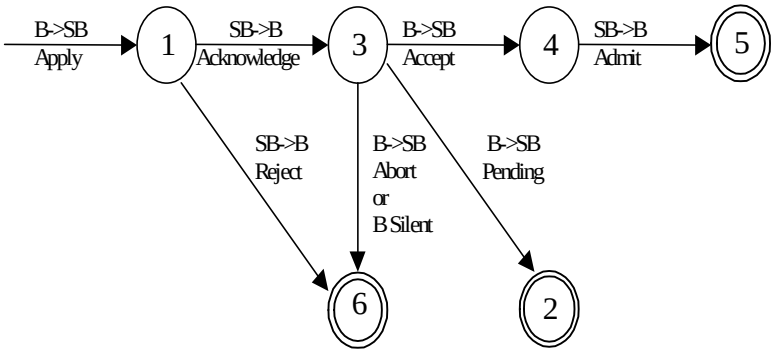

Figure 3-2. Finite state machine of joining protocol

The acknowledgement message from Superbroker to the applicant domain broker contains the necessary information used by the domain broker to join the community and also the default or basic service templates the domain broker might need in the community.

When the domain broker receives the acknowledgement from the Superbroker (State 3), it automates its behavior, if needed to fit into the community. The domain broker now autonomously makes a final decision to join the community or not based on the knowledge it can gain from the community. If the domain broker makes a positive decision, an acceptance message is sent to the Superbroker (State 3 –> State 4). The finite state 5 is reached, when the Superbroker confirms the grant of membership upon receiving acceptance message. If the domain broker decided not to join the community, an abort message is sent to the Superbroker (State 3 –> State 6).

If a decision cannot be made by the domain broker whether to join the community or not, a pending message is sent to the Superbroker. In order to allow the applicant to continue the member application in the future, the Superbroker must maintain the knowledge of access history. The state information in the conversation policy is maintained at the Superbroker side. During application processing in future, the Superbroker first checks the access history, following the conversation policy based on the state information. If the Superbroker does not receive any replies within a specified period, the synchronous conversation will move to the finite state of application failure (State 6).

Upon approval of a successful registration, the Superbroker updates information about the new broker in its repository. Failure of registration could encourage the domain broker to engage in additional learning activities about the community. The domain broker could also be modeled as an autonomous agent that can make autonomous decisions in the community.

Departure protocol.

The departure protocol is used to inform the Superbroker that the domain broker won't be able to provide the advertised e-service(s) any more. The leaving broker initiates the leaving procedure by sending a SOAP "*LeavingCommunity*" message to the Superbroker. The knowledge and services that would become unavailable because of the departure of this domain broker is updated in the community by a SOAP message "*UnpublishBroker*" by the SuperBroker to the member domain brokers in the community. Upon receiving the "*UnpublishBroker”* message, the Superbroker and the other brokers remove the services advertised by the leaving broker from their repositories. The

Superbroker makes the necessary changes in its member repository, member credit history and service quality about the leaving broker.

Knowledge discovery and exchange protocol

The individual domain brokers join the community so that they can gain some knowledge from the community and use it to serve service queries from the local brokers. When a local broker can not satisfy a service request, it contacts the domain broker of that service, which tries to find the requested service within its own brokered system (among the local brokers in its hierarchy. If the domain broker is not able to satisfy a service request within its brokered system, the request along with the constraints is sent to the community (using the superbroker) as an "*Ask*" message to other domain brokers (of the same domain) within the community. The capable domain brokers in the community, who can satisfy that service, provide the requester with the information about the service. This process leads to discovery and expansion of knowledge within the requesting domain brokers. The amount of knowledge discovered is based on the brokers' objectives. The domain broker may request for information of a single service provider using "*Ask"* or a list of providers who satisfy the service attributes using the "*Ask-All"* message.

A domain broker can also request to be informed if some specific service with certain constraints becomes available in the community. This form of knowledge discovery is called *subscription*. A broker may broadcast a "*Subscription*" to the community, which is stored by each member broker. If a broker (or a service provider it represents) is capable of serving the subscribed service, the broker will reply to the subscription and start a conversation with the subscriber. A broker can also unsubscribe its subscriptions.

Knowledge discovery protocol follows a finite state machine. If a request is satisfied and a discovery is made, the broker updates it self about the acquired capabilities. In case of many unsatisfied requests for the same service query, the broker may send a subscription for that service to the Superbroker.

Capability advertisement protocol

The domain broker can advertise to the community about the e-services that can be provided by its local brokers. When the Superbroker receives an advertisement, it checks for any subscriptions that can be satisfied and informs the subscribers. When a member agent needs to make a change in the service advertised, it sends an unadvertise message to Superbroker and makes an update. Advertisements are basically the means of getting business for the e-service providers.

Super Brokering Protocols

The *Super Brokering Protocol* governs how a Superbroker communicates and coordinates in an *Internet Enterprise* of communities, each specializing in a particular business domain. This protocol covers the community to community interaction and interoperability. Again since the community speaks SOAP on HTTP, it will be easy for the Superbroker to interact with another. The Superbroker Protocol is also responsible for Superbroker election and maintenance of the community. This set of protocols is under future work.

Now that we have seen the architecture and protocols of Sangam, we will take a look at the architecture of a local broker in the Services Layer of the Sangam Architecture we have proposed.

Sangam Local Broker

The Sangam Local Broker is used for basic brokering of e-services. It forms the basic building block of the Sangam Communities. The architecture of the Local Broker is shown in Fig 3-3.

The architecture shown here is for the extended local broker, which can perform brokering based on constraints. It also has the extended ability of ranking the matches. The enumeration of different components and their description is given below. It must be noted that implementations of Sangam Broker can use any programming language on any platform.

System Components

In Figure 3-3 the system diagram is shown with the components used to provide a constraint based brokering by Sangam. A brief note on some of the components is provided.

Service provider interface This interface is provided to receive requests and send responses back to the service provider. Sangam expects this interface to handle SOAP based messages. However, other protocols could also be used for exchange of data.

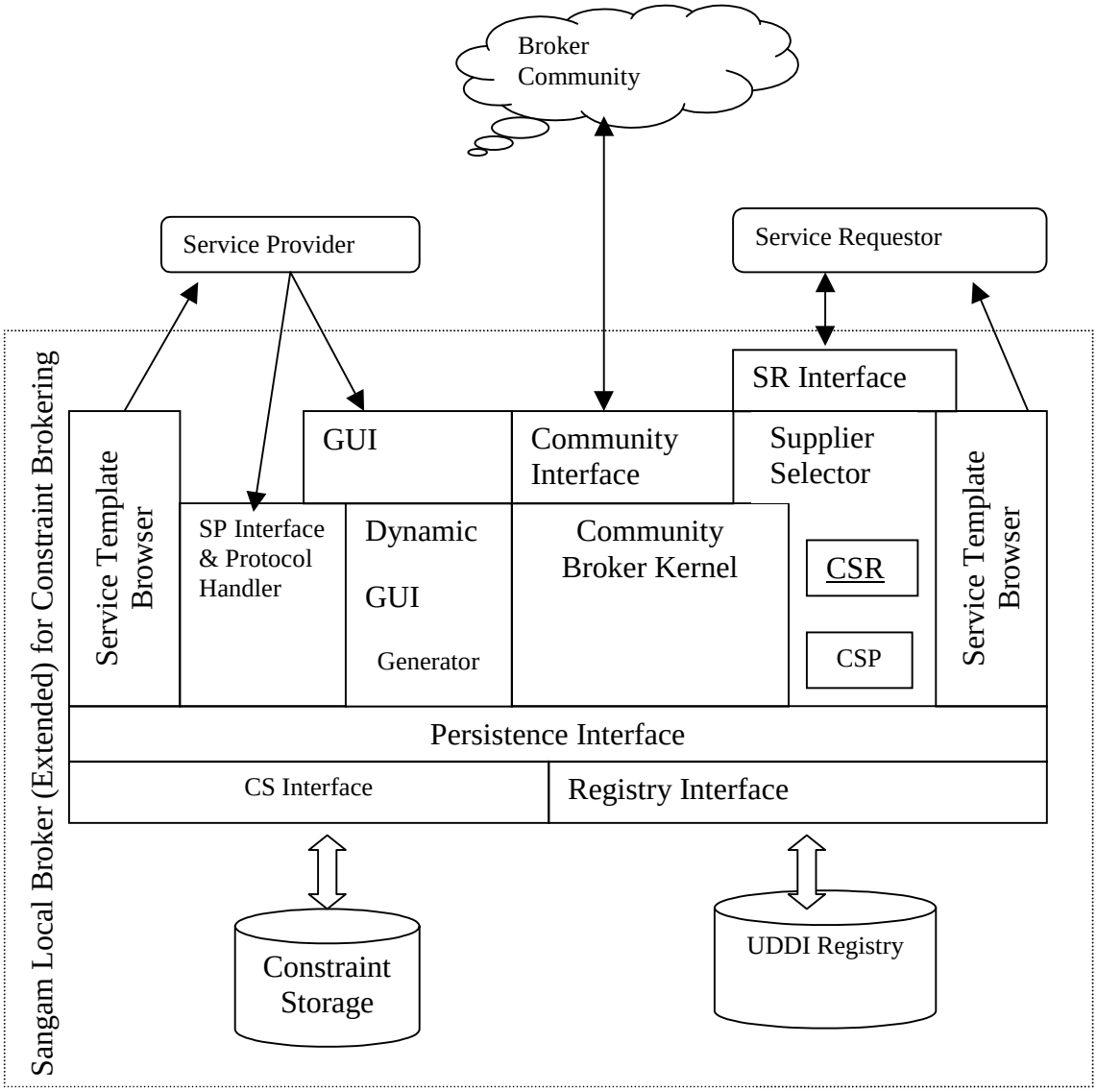

Figure 3-3. Sangam local broker with all possible components.

Service template browser The Service Template Browser is used to browse the existing service templates in the broker. The service provider and the service requestor use this during the build time.

Service requestor interface This Interface is provided means for the service requestor applications to interact with the broker (programmatically using SOAP messages).

Graphical User Interface Even though this is not an essential component, this component is provided for human user input and interaction with the broker. It is not necessary for Sangam broker to have a User Interface (UI).

Constraint Storage This module is used to store constraints. This is required only if the broker is going to deal with constraints of the services brokered. An implementation of this architecture may use either an OODBMS or an RDBMS for storage description of constraints (as either objects or XML data). It must be mentioned that this

Local UDDI Registry Sangam implementations of Local broker use a local UDDI Registry for storing information about e-services. The information stored is basically the location of the WSDL description files and information about the service providers.

Persistence Interface. This interface is needed to by the other components to query the UDDI Registry and the Constraint Storage. This interface will also be used to join the query results from the Registry and the Constraint Storage.

Service Provider Selector (SPS) This component is responsible for:

1. Search for service providers who can provide a requested service
2. Filter the service providers to find only the service providers whose constraints are compatible with the constraints of the service requestor.
3. Inform CBK if no service match could be found locally and a service has to be discovered from the community.

4. Search the local Service Providers for service queries from other brokers using the CBK.

Community Broker Kernel (CBK) Broker Community Kernel takes care of the interaction with the rest of the Sangam community. It forwards service requests to domain brokers. In the community if service is not form in local registry.

One another component is business domain service, which gives the location of various business domain communities.

Community Interface This module acts as the interface between the external broker community and the CBK. This module is responsible for sending SOAP messages. It also receives the SOAP messages and parses them.

CSP Constraint Satisfaction Processor (CSP) is used to compare two constraints and confirm if they satisfy each other. It is used in brokering to check as match of constraints of the service provider and service requestor.

CSR Constraint Satisfaction Ranker (CSR) is used to project the degree of satisfaction one constraint by another. This component is used to rank the services selected for a service query

Components–Functionality Sequence

This sub-section gives the various functionality of the local broker along with the components involved to achieve the functionality in each step.

Service provider registration

The Service Provider registration involves a registration of a service provider in the local broker's registry.

1. The Service Provider may fill up a form (*GUI)* or send a SOAP message and registers into the broker.

2. The *SP Interface & Protocol* Handler of the broker may or may not accept the new service provider based on the broker's business rules.
3. If the registration is accepted, the information provided by the service provider is stored in the *UDDI Registry* using the *Persistence Interfaces*

Service template browser

Service Templates standardize each e-service type implementation. A service provider who wants to register an e-service has to provide template description for an existing service template known to the broker. Thus, a service requestor who develops his application based on a service template can access any service implementation of the template using the same request parameters. The *Service Template Browser* is used to browse through all the service templates known to the broker. This is usually done at build time by a developer or designer to know the description of the available service templates.

Service registration

Service providers register the e-services that can be accessed by the service requestors of who are associated with the broker. The process sequence (refer figure 3-4) described here assumes that the service templates used for registration are already present with the broker.

1) Service provider browses the existing service templates present in a broker using the *Service Template Browser* and decides which category and service template best fits the service the to be made available as an e-service.
2) Service provider makes sure the e-service that has been implemented confirms to the service template selected.

3) Service provider uses the *GUI* or a SOAP message to the broker for registering necessary WSDL URI binding. The protocol handles check the WSDL for validity.

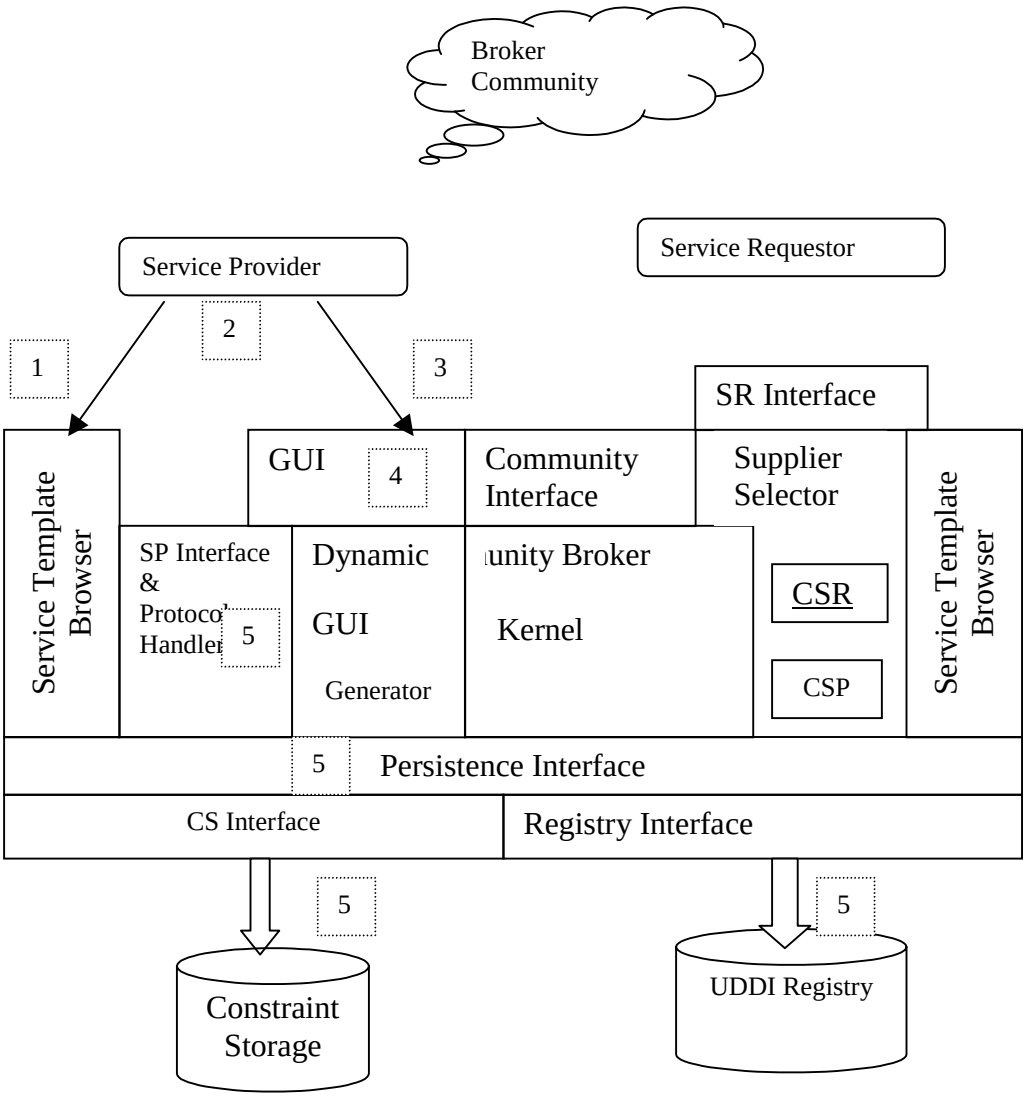

Figure 3-4 Service registration process sequence and components involved.

4) The information passed by the service provider is validated by the *SP Interface*, which updates the *UDDI Registry and Constraint Storage* with the information about the new e-service.

Service request

Service requestors make request for a service based on an existing service template. They may also specify some constraints that can be associated with the desired service.

1) The service request from the service requestor is a SOAP request. This request is parsed and validated by the *Service Requestor Interface.*
2) The request parameters are sent to the *Supplier Selector,* which finds the entire set of members who can provide the requested service and then finds the best service provider within the set based on the request constraints.
3) The binding description required to invoke the selected service is sent as reply to the request.
4) If no service provider could be found, the service request is forwarded to *Community Broker Kernel (CBK),* which uses the brokering community to get information about the requested service. The information received from the community is added to the UDDI Registry and the Constraint Storage. It is also used to generate the response to the service query.

Supplier selector

The Supplier Selector is the one of core components for constraint brokering. It finds the best service provider who can provide a requested service satisfying the service constraints. The Supplier Selector contains the sub-components CSP (Constraint Processor) [HAM 98], used to check validity of constraints and the CSR (Constraint Ranker), used to rank the services which satisfy the constraint.

1) The Constraint given by the requestor, the request constraint is compared with the of the existing service constraints present in the *Constraint Storage* using the CSP. The satisfying services are selected.
2) The selected services are ranked using the CSR based match in the desired constraints.

Community kernel

The Community Kernel is the heart of the Community based brokering. It will contain the components of required for community role of the broker including components that may be required for this broker to become a Super Broker.

In this chapter we have seen the service description, Sangam Architecture, Sangam Protocols and the components of a Sangam Broker. In the next chapter we will look at Sangam Broker for Workflow. We will also know more about an implementation of Sangam Broker.

CHAPTER 4
SANGAM BROKER FOR WORKFLOW

"Well done is better than well said." – Benjamin Franklin

In this chapter we will take a look at the implementation of Sangam Broker for workflow systems.

Dynamic Workflow

In the near future, many of the day-to-day activities and e-business processes will be offered as individual e-services and will act as virtual tools for our daily work. These existing e-services can be combined and reused, to create new value-added services. In such a case, the new service can be viewed as the result of execution of a workflow of existing and/or newly created services.

Workflow Management (WFM) is used in the automation of process oriented tasks. A workflow model is predefined for every business process and consists of a number of activities and a set of conditional transitions that link these activities. [MEN 00a] presents a design of ad-hoc workflow system and shows that ad-hoc workflow modeling is appropriate for e-business processes. The advantage of ad-hoc workflow is that systems can be built dynamically and flexibly. The system is flexible to the changes in the business rules and the business environment.

Since business processes today are becoming available as e-services, we need to take advantage of this "elecronification" of business processes and wire them dynamically. This reduces the cost and time in completing the regular business processes.

In order to use the emerging e-services to create a dynamic Internet based workflow, rather than just putting the vowel "*e*" in front of the word "workflow"; we need to have well defined description, discovery and integration of web-services.

In dynamic Internet-based workflow the sequence of the e-services combined to complete an activity and the service providers selected to perform the services are determined dynamically. The Workflow engine dynamically determines the individual tasks that will be used to accomplish an activity. Workflow systems can benefit by using e-services broker to dynamically find and bind to appropriate service providers for the individual tasks to be completed. Thus, the service broker enables late binding of services. The brokering service is therefore critical in determining the workflow.

Main Components

Figure 4-1 shows the components in the workflow scenario and their interaction. The Service provider is the owner of a service, which is deployed as an e-service. Its description can be accessed from the registry. Service Broker maintains a searchable repository containing service descriptions from the service providers. The Workflow engine is a service requestor looking for appropriate service providers.

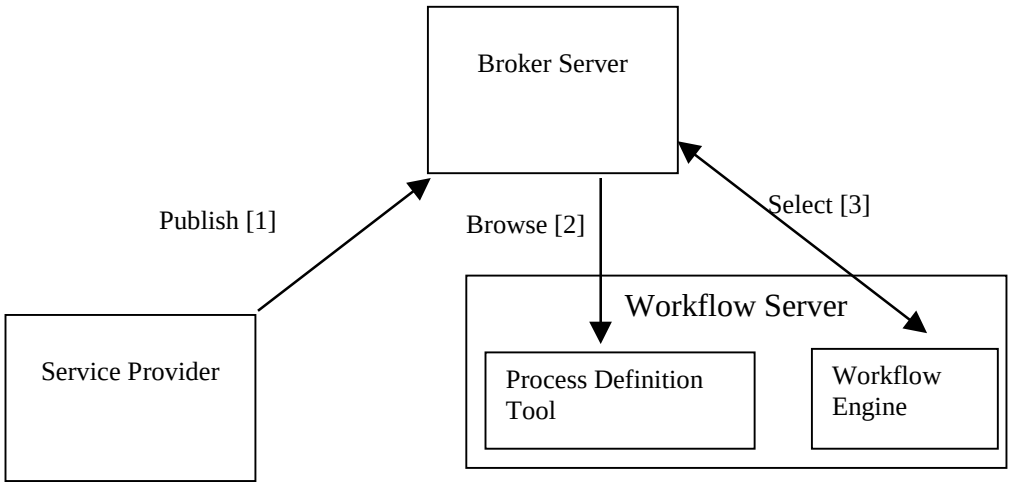

Figure 4-1. Service discovery and binding in workflow system

The operations performed here are:

- *[1] Publish/Unpublish*. Service providers advertise (publish) their e-services to one or more registries. They may also unpublish the advertisements of service from the registry.
- *[2] Browse*. The designer of the workflow browses service templates available in the Broker Server to find the service templates that can be used in the design of the workflow model (explained in the next section).
- *[3] Select.* At run time the Workflow engine sends e-service requests to the Broker server to get desired service provider(s) who satisfy these e-service request (s). The e-service requests are defined based on e-service template(s). The e-service request may also have constraints that require the service provider to have capabilities that can satisfy the constraints. The Broker Server uses its knowledge of the service providers and to find the possible binding information for the workflow.

E-service Description Using WSDL

Description of e-services is very important as it helps the service to be discovered. The description has to give the required information on how to bind and invoke the service. Also it must have a high level information about the service and its category.

In our implementation, we use WSDL (Web Services Description Language) [CHR 01] to describe the e-service interfaces. WSDL is essentially an XML IDL (Interface Definition Language) that can describe the functions and interface of a service. It is an XML format for describing network services as a set of endpoints operating on messages contains either document-oriented or procedure-oriented information. WSDL allows the operations and messages to be described abstractly and then bound to concrete protocol and end point. WSDL is extensible to allow description of endpoints.

The service description is divided into two parts: *E-Service Template Description* and the *E-Service Binding Description*. The *E-Service Template Description* gives description on the methods and arguments that of a service. This information will be needed by workflow developer to programmatically use the e-service. *E-Service Binding Description* contains the location of the service implementation and the details on the protocol and port used to access the server, which hosts the service.

WSDL allows specification of both the e-service template and the binding information. The template description and the binding description are separated as two WSDL documents: *WSDL template document* and *WSDL binding document*. This is advantageous, as the service providers need to only provide the WSDL binding to access their service (if a WSDL template document exists already in the broker for the service). Also, the workflow engine has to know only about the templates and use them in their

design time. At run time, the workflow engine can get the binding information required for any service template from the broker.

A WSDL Document

A WSDL document uses the following elements in the definition of network services:

- Types – A container for data type definitions using some type system (like XSD)
- Message – Abstract typed definition of the data being communicated.
- Operation – Abstract description of an action supported by the service.
- Port Type – Abstract set of operations supported by on or more endpoints.
- Binding – Concrete protocol and data format specification for a particular port type.
- Port – a single endpoint defined as a combination of a binding and network address.
- Service – A collection of related endpoints.

It is important to observe that WSDL is not a definition language. WSDL recognizes the need for rich type systems for describing message formats, and supports the XML Schemas specification. WSDL allows using other type definition languages via extensibility.

Service Discovery for Workflow

In this sub-section we describe all the processes involved in a service discovery step-by-step from publication till binding used for workflow. We will start with the publication of Service Interface.

Publish Service Interface

The first step before we publish our Service Implementation is the publication of the Service Interface. As mentioned in the previous section, the service interface is the abstract definition of the e-service.

1. *Define an Interface document.* A simple example of the WSDL-Interface document. A WSDL Interface document has to be published in the broker first.
2. *Determine Category.* The category of the e-service has to be determined. It must be mentioned that the two primary forms of categorization with UDDI are UNSPSC (Universal Standard Products and Services Classification) and NAICS (North American Industry Classification System). Sangam uses NAICS internally for the classification of services. Sangam can internally map the categories and the users need not lookup for service category codes.
3. *Publish to Broker.* The service interface definition is published in the broker. The publisher of the interface has to just supply the URI information of the WSDL. The broker reads the WSDL document, validates the definition and updates it to the UDDI registry it maintains.

Publish Service Implementation

Once a web service is ready to be published, it can use Sangam SOAP messages to publish the information. The steps involved here assume that the service provider information is already published in the registry.

1. *Category determination.* The publisher of the web service has to determine the category of the e-service. The Sangam message used to browse the categories is “getCategories”.
2. *Interface document.* The Interface document based on which the service implementation has been made has to be found from the broker. The Sangam message used here is “getInterface”.

3. *Implementation WSDL.* A WSDL document describing the binding for the service implementation has to be prepared.
4. *Publish Service.* The service implementation document is published using the Sangam message “putService”.

Find Service

A Workflow engine or the service requestor who wants to find a service and use it in their application follows these steps.

1. *Find Interface document.* The workflow engine has to first find the service interface templates before designing the client applications or the workflow models. The service interface document contains the operations that the client application must support and the arguments that are used in the service. The Sangam message used here is “getInterface”.
2. *Build to use the e-service.* Based on the information received from the interface document, the messages and the arguments needed to use a service are built in the application. The workflow model designer makes sure that the application that will use the e-service has all these parameters and is interoperable with the service. This happens during the build time.
3. *Use the service at Runtime.* At Runtime, the application contacts the broker with the information on the Service Interface template and requests the broker to find a service that matches the Particular Interface using the Sangam message “getService”. This gives the URI of the service that the application has to use to bind with the service provider.

This is a simple sequence of the steps from the start to use the service by the workflow engine. A sample of the WSDL template document is given in Figure 4-2. The implementation package and the screen shots of the web interface for the Sangam broker are explained in the next chapter.

```
<?xml version="1.0"?>
<definitions name="WebServiceExample-interface"
targetNamespace="http://www.dbcenter.cise.ufl.edu/~aswaran/SuperBroker/Test/WebSe
rviceExample-interface.wsdl"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

 <documentation> This interface file defines template for a simple webservice The input
message has a parameter of type string by name inMsgText. The output message has a
parameter of type string by name outMsgText.
</documentation>

  <message name="InMessageRequest">
    <part name="inMsgText" type="xsd:string"/>
  </message>
  <message name="OutMessageResponse">
    <part name="outMsgText" type="xsd:string"/>
  </message>
    <portType name="WebServiceExample">
    <operation name="echoMessage">
      <input message="InMessageRequest"/>
      <output message="OutMessageResponse"/>
    </operation>
  </portType>
  <binding name="WebServiceExampleBinding"
          type="WebServiceExample">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="echoMessage">
      <soap:operation
soapAction="http://www.WebServiceExample.com/EchoMessage"/>
      <input>
        <soap:body use="encoded"
                namespace="urn:web-service-example"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                namespace="urn:web-service-example"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
</definitions>
```

Figure 4-2. Sample WSDL template (Interface) document

CHAPTER 5
SANGAM API AND IMPLEMENTATION

"The value an idea is in the using of it" – Thomas A. Edison

In this chapter, we take a look at the Sangam API and some screen shots of our e-service broker. We also give the implementation details about the software used to build the system.

Sangam API

Sangam API can be used to build a minimal Local Broker confirming with the Sangam Architecture specifications. The API was written in Java and uses some of the J2EE [J2E] technologies. This API comes bundled with modules that can be used to perform the basic brokering. The components and their corresponding functionality with respect to the Sangam architecture are explained in this section.

Sangam Protocol Handler

The Sangam Protocol Handler is the module that processes the SOAP based Sangam Protocol messages. This module receives the messages from the service provider and the service requestor. The components included in this module are:

- *Category Browser*. The category browser is used to browse the business categories supported by the broker.
- *Publisher*. This component is used to publish Service Provider, Service Interface and Service Implementation. These components can be invoked by various SOAP messages. The structures of some of these SOAP messages are provided in this

chapter. The messages used for publishing to the broker include "publishServiceProvider", "publishServiceInterface" and "publishServiceImplementation"(The tasks performed by these messages are self-explanatory).

- *Find Service Template (Interface).* This component finds the service interfaces for a business category. Its SOAP response is an URI list of Service Interface WSDL documents.
- *Find Service.* This component finds the service implementation for a specific business category that supports a service template. The response is a list of service descriptions along with the various ports and protocols to access the respective services. This message when used by an application can be parsed and the URI that is interoperable with the requestor can be contacted programmatically without requiring any human intervention to access the service.

Service Template Browser

This module mentioned in the Sangam Architecture for a Local broker is used to browse all the templates that are currently available with the Local broker. It can be invoked using the message "getTemplates".

Sangam Parser

The API also contains a Sangam Parser. The Sangam messages used in the community or in the service layer can be parsed using this parser. This parser is a tool for the developers using Sangam to extract the required information from the messages received from the broker. This parser is an extension of the Apache XML Xerces Parser.

Sangam SOAP Client

The applications of Sangam need to send or receive SOAP messages for the protocols to interact with the broker. The API provides a functionality to send and receive SOAP messages. The received messages can be piped to the parser to extract information.

Local UDDI Node

Sangam uses an UDDI registry internally. It has components to make proxies of this registry and interact with it directly. This module is extended using the local UDDI from IBM.

User Interface

Sangam can be accessed either directly using the applications or using the web-based interface. The web-based interface allows easy access some of the Sangam functionality. The interface is created using Java Servlets that internally use the other components of the Sangam API to interact with the broker.

System Design

This sub-section discusses the software components used to implement the system and its configuration. This implementation uses N-tier architecture design. The Interface as shown in Fig. 5-1 is provided using HTML pages and acts as the presentation layer of the system. The next layer is the web-interface layer. This layer is implemented using Servlets on Apache Tomcat. This interface layer interacts with the application layer that hosts the web applications. This layer is implemented using the Java applications on the IBM WebSphere Applications Server. Using a database provides the persistence to the entire system. We have used IBM DB2 as our database.

Currently the entire system is running on an IBM-PC (machine Beijing in the database center) on a Windows NT platform for test purposes. The N-tier architecture of the system makes it possible for the system to be run on physically different machines (the web server, the application server and the database server).

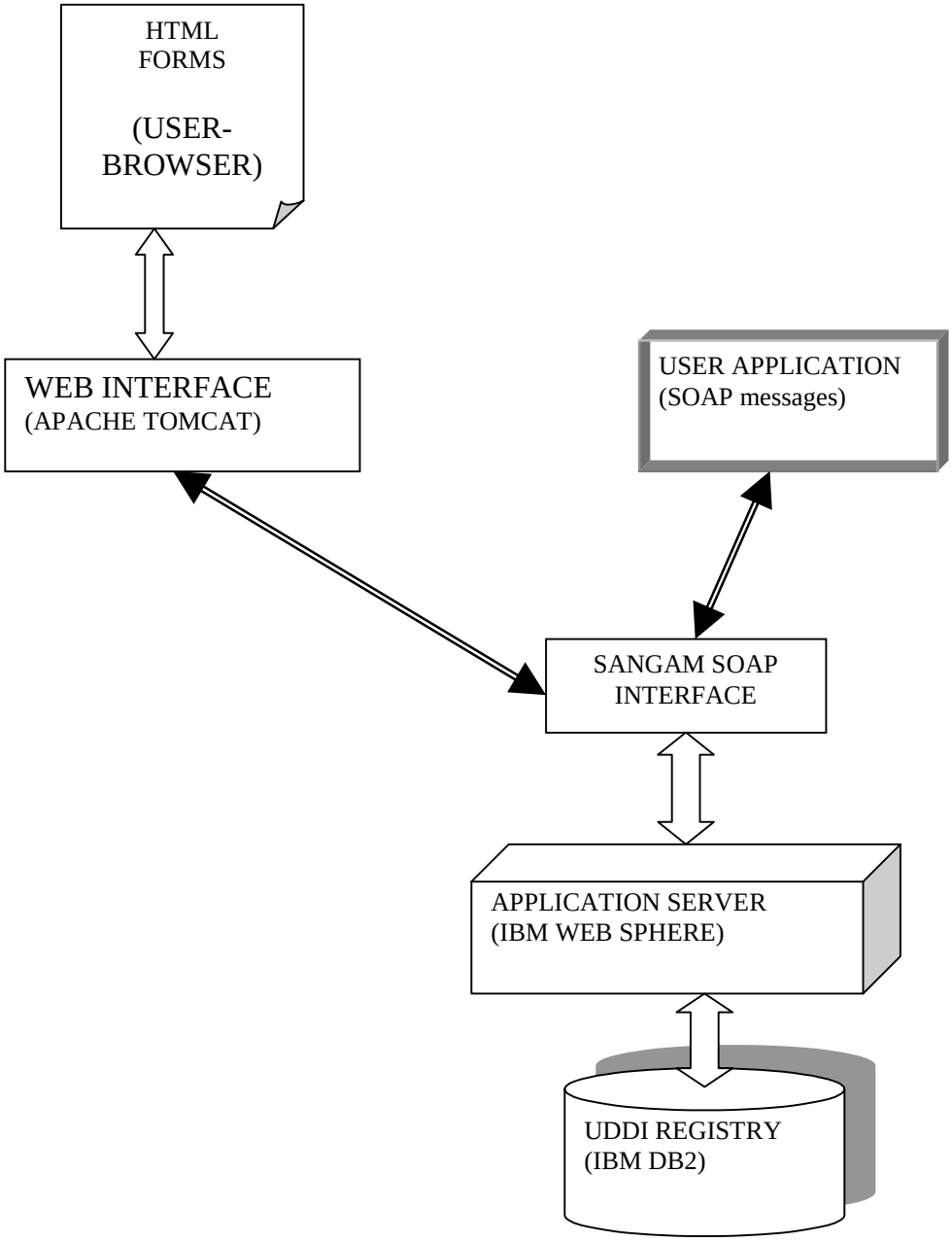

Figure 5-1. System implementation of local broker.

Implementation Scenarios and Screenshots

The Broker can be interacted using either SOAP messages or using the Web based Interface. The SOAP based messages are suitable for programmatically querying

Sangam without any human intervention. Sangam API gives programmers a Sangam Client to make SOAP based requests to any Sangam Local Broker. This section describes a scenario of the Sangam broker with messages and web screen shots.

1. *Service Provider Registration.* The Service provider can provide information about his business and register in the system. The business category information has to be included during registration. The existing categories can be browsed before registration to decide which categories fit the business of the service provider.

The SOAP message used for this purpose is “putServiceProvider”. It is given in the listing below:

```
<?xml version = '1.0'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <putServiceProvider>
            <businessName>forExample</businessName>
            <businessDesc>examples need not be the ample.</businessDesc>
      <Category>Retail Trade</Category>
      <Category>Retail Trade</Category>
    </putServiceProvider>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5-2. Message to publish service provider

It can be seen that the SOAP messages are very simple and make it possible for simple transfer of messages. Fig.5-3 shows the screen shot of the corresponding web-interface (which internally uses the above SOAP message).

Figure 5-3. Web-interface to publish service provider

2. *Find Service.* Once the business category and the service template are identified, the application can be coded at design time to use a specific service template. At run time broker finds a matching service for every request. This is done using the "findService" message.

The structure of the response received from broker for a request to "findService" is given below in Fig 5-4. This shows the description needed to access the service using any of the protocols. An application can parse this response message and interact with the service provider directly instead of using a web-interface.

```
<?xml version = '1.0'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <SOAP-ENV:Body>
<serviceImplementationList>
        <serviceImplementation>
                <serviceName>WebServiceExample</serviceName>
                <serviceDesc></serviceDesc>
                <serviceAccess>
                  <Access> http://demohost:8080/soap/servlet/rpcrouter</Access>
                  <Access> https://host:8080/soap/servlet/rpcrouter</Access>
                 <Access> mailto:aswaran@cise.ufl.edu</Access>
                </serviceAccess>
        </serviceImplementation>
</serviceImplementationList></SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5-4. Response message for service discovery

The screen shot from the web interface is given in Fig.5-5 is a scenario for service discovery. The example shown here has discovered a service implementation of the requested service. This service discovered has three ways of access from the same service provider. Usually the same e-service might have many ways of access to make it interoperable with many applications.

In this chapter, we have seen simple access scenarios of the Sangam broker and its web interface to discover services. We present our conclusion of the research and future work in the next chapter.

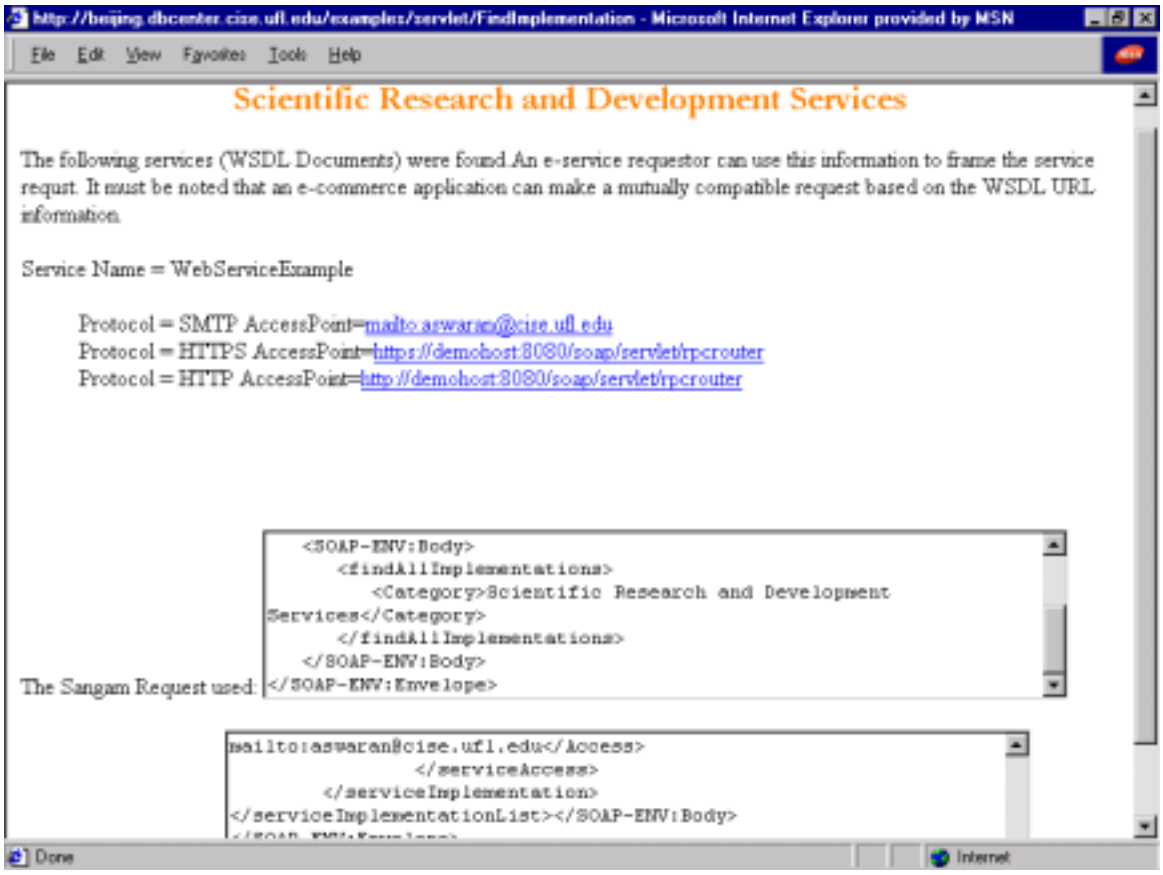

Figure 5-5. Screen shot of web interface for “findService”

CHAPTER 6
CONCLUSION

"All good things must come to an end."

A requirement to have *Universal Interop Protocols* is envisioned for interoperability of various systems using e-services. The systems, which want to use e-services, may range from a small device to a pervasive environment. This thesis has investigated on dynamic discovery and binding of e-services.

It has proposed the Sangam Architecture and its community protocols as the solution for brokering of P2P (peer-to-peer) and ubiquitous e-services. Sangam Protocols are built using well known technologies like HTTP and XML, which are the technologies used for Internetworking and Interoperability respectively. Also, Sangam uses WSDL and UDDI to describe e-services and from single unit of a broker. The single units of local brokers join to make a virtual community, which are scalable and can be used for brokering of ubiquitous e-services.

The solution proposed here is simple and is designed based on the significant concepts of various existing technologies like Jini Federation, e-Speak Communities, UDDI repository etc., The advantages of the Sangam solution are its very large scalability, interoperability and its simplicity (for implementation). It is also based on standards technologies and the existing Internet Protocol Stack, which makes it even more worthwhile and suitable for any possible extension in the standard stack in the future.

In this thesis we have surveyed the exiting protocols for service discovery and the emerging standards for e-services. We have also proposed our architecture and the implementation of the Local broker used in the architecture. The different messages used in Sangam are also presented.

Contributions

The contributions this thesis makes to the field of Internet and distributed computing are as follows:

- Architecture and protocols for Universal Interoperability.
- Dynamic discovery and binding of e-services which allow Machine-to-Machine business between the end points.
- A survey of exiting and emerging technologies.
- An implementation case of Sangam broker.

Future Work

We need to build massive real world application systems that use Sangam. The protocols need to be used in pervasive systems and mobile devices. Sangam can be proposed to IETF or other standard bodies as the Universal Protocol for e-services.

There are many areas in which the solution can be enhanced. Currently the broker does only basic brokering. Constraint based brokering mentioned in the thesis is an extension by embedding the constraints in the service description and the service query. Possible mechanisms to describe constraints and extracting the constraints from the service query to be used in CSP and CSR are to be studied. Another possible work to extend this thesis is regarding algorithms that can match the constraints for e-services.

The Superbroker protocols of the community and the community kernel of the broker are some of the work, which need further investigation.

LIST OF REFERENCES

[BAN 00] Guruduth Banavar, James Beck, Eugene Gluzberg, Jonathan Munson, Jeremy Sussman, and Debora Zukowski "*Challenges: An Application Model for Pervasive Computing"* Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000).

[BIE 01] Guy Bieber and Jeff Carpenter "*Introduction to Service-Oriented Programming"* Openwings,
URL = http://www.openwings.org.

[BOX 00] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Sathish Thatte and Dave Winer, "*Simple Object Access Protocol (SOAP)1.1*" W3C Note May 2000.
URL = http://www.w3.org/TR/SOAP/

[BOU 00] Toufic Boubez, Maryann Hondo, Chris Kurt. Jared Rodriguez and Daniel Rogers "UDDI Programmer's API 1.0"
URL = http://www.uddi.org/specification.html

[BRA 98] T.Bray, J.Paoli, and C.M.Sperberg-McQueen, "Extensible Markup Language (XML) 1.0", W3C Recommendation, February 10, 1998.
URL = http://www.w3.org/TR/REC-xml

[CAV 92] N.Caver and V.Lesser, "The Evoloution of Blackboard Control Architectures," CMPSCI Technical Report 92-71, October 1992.

[CHR 01] Erik Christensen, Francisco Curbera, Greg Meredith and Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1" W3C Note March, 2001
URL = http://www.w3.org/TR/wsdl

[CIC 00] Andrzej Cichocki, Abdelsalam (Sumi) Helal, Marek Rusinikiewicz and, Darrell Woelk. *Workflow and Process Automation: Concepts and Technology*. Kluwer Academic Publishers, 2000.

[COOL] HP Cool Town Web site. http://www.cooltown.hp.com/

[EDW 99] Keith W. Edwards, *Core Jini.* Prentice Hall, 1999.

[ESP] HP E-Speak Web site. http://www.e-speak.hp.com/

[GUT 99a] Erik Guttman, "*Service Location Protocol: Automatic Discovery of IP Network Serices"* IEEE Internet Computing, vol. 3, no.4, pp 71-80, 1999.

[GUT 99b] Erik Guttman, Perkins, C., Veizades, J., and Day, M., *"Service Location Protocol, Version 2,"* IETF, RFC 2608, June 1999.
URL = http://www.rfc-editor.org/rfc/rfc2608.txt

[HAM 98] Joachim Hammer, C.B Huang, Y.H Huang, Charnyote Pluempitiwiriyawej, Minsoo Lee, H. Li, L. Wang, Youzhong Liu, and Stanley.Y.W. Su, "The IDEAL Approach to Internet-Based Negotiation for E-Business. Proceedings of the 16th International Conferenece on Data Engineering.

[HEL 01a] Abdelsalam (Sumi) Helal, Mei Wang, Arun Jagatheesan and Raja Krithivasan, "Brokering Based Self Organizing E-Service Communities". Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems (ISADS) With an Emphasis on Electronic Commerce, March 26-28, 2001
URL = http://www.harris.cise.ufl.edu/projects/publications/isads2001.pdf

[HEL 01b] Abdelsalam (Sumi) Helal, Arun Jagatheesan and Mei Wang, "Service-Centric Brokering in Dynamic E-Business Agent Communities," Journal of Electronic Commerce Research (JECR), Baltzer Science Publishers.

[IBM 01] International Business Machines Corporation. "*Breaking through the boundaries: dynamic e-business"*
URL = http://www-4.ibm.com/software/solutions/webservices/pdf/boundaries.pdf

[JAT 99] JATLite web site, http://java.stanford.edu/, May 1999.

[J2E] Java2 Platform, Enterprise Edition Web site, http://java.sun.com/j2ee/white/index.html Sun Micro Systems.

[KRI 01] Raja Krithivasan "Biz-Builder: An e-Services Framework targeted for Internet Workflow". Master's Thesis. University of Florida

[MAR 99] Martin, G.L., Unruh, A., & Urban, S.D., " An Agent Infrastructure for Knowledge Discovery and Event Detection,", Oct 1999.
URL = http://www.mcc.com/projects/infosleuth/publications/,

[MCG 00] Robert E. McGrath "*Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing"* Presented at the Center for Excellence in Space Data and Information Science, NASA Goddard Space Flight Center. April 5, 2000.

[MEN 00a] Jie Meng, Abdelsalam (Sumi) Helal, and Stanley Su, "An Ad-Hoc Workflow Architecture Based on Mobile Agent and Rule-Based Processing," The special session on Software Agent-Oriented Workflows, Proceedings of the International Conference on Parallel and Distributed Computing Techniques and Applications, Las Vegas, Nevada, June 2000. pp. 245-251
URL = http://www.harris.cise.ufl.edu/projects/publications/adhocWF.pdf

[MEN 00b] David W. Mennie, *"An Architecture to Support Dynamic Composition of Service Components and it Applicability to Internet Security"* Masters Thesis, Carleton University, 2000.

[MIC 00] Microsoft Corporation, "*Universal Plug an Play Device Architecture"*, White Paper, Version 1.0, June 6, 2000.
URL = http://www.upnp.org/UpnPDevice_Architecture_1.0.htm

[MOC 87] Mockapetris, P., *"Domain Names-Implementation and Spcecification"*, IETF RFC 1035, October 1987.
URL= http://www.rfc-editor.org/rfc/rfc1035.txt

[NOD 99] Nodine, M.H., Bohrer, W., and Ngu, A., *"Semantic Brokering over Dynamic Heterogeneous Data Source in InfoSleuth,"* ICDE 1999: 358-365.

[SAH 01] Akhil Sahai and Vijay Machiraju "*Enabling Ubiquitous E-services Vision on the Internet"* E-Services Software Research Department, HP Laboratories.
URL= http://www.hpl.hp.com/org/stl/emd/pubs.html

[SU 87] Stanley Y.W. Su, Jozo Dujmovic, D.S. Batory, S.B. Navathe, and Richard Elnicki *"A Cost-Benefit Decision Model: Analysis, Comparision, and Selection of Data Management System"* ACM Transactions on Database Systems, Vol. 12, No.3, September 1987, Pages 472-520.

[UDD 00] UDDI website. http://www.uddi.org/. September,2000.

[WAH 97] Wahl,M., Howes, T., and Kille, S., "*Lightweight Directory Access Protocol (v3),"* IETF RFC 2251, December 1997.
URL= http://rfc-editor.org/rfc/rfc2251.txt

BIOGRAPHICAL SKETCH

Arun Swaran Jagatheesan received his Bachelor of Engineering degree in Computer Science and Engineering from Kumaraguru College of Technology, Coimbatore, India in 1998. He worked in Bangalore, India before the joining CISE graduate program at the University of Florida in 1999. He graduated in August 2001 with Master of Science degree. His research interests include Internet Computing, Pervasive Computing, Fuzzy Databases and Software Agents. He has co-authored a couple of publications for International Student Symposia in India. While at the University of Florida, he co-authored a few publications relating to Software Agent Communities and P2P e-Business.