OPEN SERVICE ARCHITECTURES FOR PEER DATABASE MANAGEMENT SYSTEMS

BY

TASMEIA YOUSAF

A THESIS SUBMITTED IN CONFORMITY WITH THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE,
SCHOOL OF COMPUTER SCIENCE,
AT CARLETON UNIVERSITY.

COPYRIGHT © 2006 BY TASMEIA YOUSAF.
ALL RIGHTS RESERVED.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-16482-2
Our file Notre référence
ISBN: 978-0-494-16482-2

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Open Service Architectures for Peer Database Management Systems

Master of Computer Science Thesis
Ottawa-Carleton Institute of Computer Science
Carleton University

by Tasmeia Yousaf
May 2006

Abstract

The need for data sharing across distributed and heterogeneous data sources is growing. Peer Database Management Systems (PDBMSs) offer one such data sharing approach, which favors a direct and dynamic node-to-node model of communication with no centralized control. Where Service Oriented Architectures (SOA) using Web services technologies allow users to leverage existing assets towards the goal of building new architectures and integrating existing systems that can be componentized. We propose two versions of Open Service Architecture for PDBMSs (OSAP). The first called OSAP I, hides all the JXTA services offered by a peer as private processes that can be used by other peers only through the use of the well-defined interface of a peer manager which is a web service. The second version, OSAP II, offers loose coupling of Web services, together with their decentralization. In OSAP II, the main services of the PDBMS are offered as Web services that are invoked via the network using a set of well-defined interfaces. This approach provides power and flexibility in terms of development and usage of the system. We have implemented both architectures within the Hyperion PDBMS framework. We provided detailed analysis of three the systems (Hyperion, SO/Hyp1, SO/Hyp2) in terms of architecture, characteristics and performance comparison.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgements

I am thankful to both of my supervisors Dr. Sivarama Dandamudi and Dr. Iluju Kiringa who gave me the opportunity to do research in the exiting and growing field of databases and Web services. They guided me to the way of innovation and novelty. Their valuable ideas and profound research experience kept me passionate and confident all the way of my thesis.

I also appreciate the help of Michele Amoretti, project supervisor of JXTA-SOAP, who helped me understanding the package and solved any problems I had with JXTA-SOAP in proficient and timely manner.

I am also grateful my colleagues working on the same project for their help and suggestions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Dedication

to my mother and father,

brothers Khurram Shahzad, Mubasher Shahzad

and sister Taooz

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Contents

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

Introduction

Peer-to-peer (P2P) computing has emerged as an architecture for networking that provides a direct and dynamic node-to-node communication model. Using a dedicated naming space, peers act as both clients and servers and take the responsibility of providing contents over the network. One of the main characteristics of P2P networks is that the peers are autonomous with respect to the control and structure of the network; peers may join or leave the network at any time. Peers join the network and create logical communication links called acquaintances with other peers to share and coordinate data.

The Hyperion project [27, 12] is investigating data management issues of P2P computing. These issues are centered on the development of Peer Data Management Systems (PDBMSs), which are DBMSs that manage databases, called peer databases, that are associated with each peer. The Hyperion project develops techniques for providing data sharing and data coordination with no central control, no global

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

schema, transient participation of peer databases, and constantly evolving coordination rules among peers. Each peer is supposed to be equipped with a peer database with its own schema and data. When peers become acquainted, logical metadata in the form of value correspondences, called mapping tables [22], that are necessary to allow data sharing are exchanged semi-automatically between the acquainted peers. The mapping tables help to produce mappings at the data level for bridging the semantic heterogeneities between the peers. We view PDBMs as a conventional DBMS augmented with a P2P interoperability layer. This layer implements the functionality required for PDBMSs to share and coordinate data without giving up their autonomy[22]. Interoperability is obtained through mapping tables that are created using metadata of the peers. A network of PDBMSs constitutes a peer database system (PDBS), similar to classical multidatabase systems [37]. However, unlike the latter, a PDBS is not set up at design time, but at run time.

Data management issues related to P2P computing have become an increasingly attractive research area lately. This research area is investigating architectural aspects of PDBMSs [40, 1]. Unfortunately, the rare architectural proposals made so far have not been compared in order to asses them. We believe that the reason lies in the divergences these architectures show in the underlying concepts used to formalize the heterogeneity of these systems. Two major directions exists concerning the handling of heterogeneity. On one hand, there is an architecture spearheaded by the Piazza system [40] which uses a rich mapping language, GLAV (Global-Local-As-View), to syntactically express the heterogeneity gap between peer databases.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

On the other hand, Hyperion uses mapping tables to semantically express the heterogeneity. Also, within these two major directions, there is no systematic study of possible architecture alternatives in order to asses their respective advantages and drawbacks.

1.1 Motivation and Objectives

There is an architecture for Hyperion [12] that is based on a logical architecture introduced in [1]. This existing architecture uses the notion of *service* in the sense of an encapsulated and well-interfaced module with a well-defined functionality. Moreover, it is implemented on top the JXTA P2P infrastructure [18]. However, this architecture defines peer services exclusively in the context of peer-to-peer networks, so the integration of a new service in the network requires it to be a pure P2P service. Even though the concept of services appears in the existing architecture of Hyperion, this concept does not meet the characteristics and requirements of a "Service Oriented Architecture (SOA)".

1.2 Motivating Example

Consider a system for airline-ticket reservations (This example is adapted from [1]). Peer databases belong to travel agencies and airlines. Acquaintances are established between affiliated travel agencies, between travel agencies and airlines, between partner airlines, and so on. Consider database schemas of two partner airlines as shown in Figure 1.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Schema for Alpha-Air Airline
AA_Passenger (pid, name)
AA_Flight (fno, date, dest, sold, cap)
AA_Ticket (pid, fno, meal)

Schema for Beta-Air Airline
BA_Fleet (aid, type, capacity)
BA_Passenger (pid, name)
BA_Flight (fno, date, to, sold, aid)
B_Reserve (pid, fno)

Figure 1.1: Database Schemas

Alpha-Air stores for each passenger, his/her identifier and name; for each flight it stores the flight number, date, destination city, number of tickets sold, and capacity of the flight. The information about the tickets sold is stored as the passenger identifier, flight number and the passenger meal preference. Beta-Air stores information about its fleet, namely, the identifier of each plane, as well as the type and capacity of the latter. Name and passenger identifier is stored for information about each passenger. For each flight the flight number, date, destination airport code, tickets sold, and identifier of the airplane are stored. For each reservation passenger identifier and corresponding flight number is stored. For simplicity it is assumed that flights originate in the city of Toronto for Alpha-Air and the Toronto airport for Beta-Air. In a service oriented peer-to-peer architecture, peers like Alpha-Air and Beta-Air should join the peer network and should be able to find other peers in the network. To offer services to other peers in the network, peers expose a well-defined interface; this includes detailed information about the peer, description of the data and list of services it offers. The services could include how to establish acquaintances, query

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

data, and subscribe to get notification and updates about the data, and so on. The acquaintance service associates two peers and translates their respective languages to each other so that both peers can share data. For example 'dest' attribute in the Alpha-Air is the destination city of a flight whereas 'to' attribute in Beta-Air is the name of the airport of the destination city. After the creation of the acquaintance between Alpha-Air and Beta-Air 'dest' can be translated to 'to' and vice versa. The acquaintance service should allow the capability of sending one peers' schema (say p1) to the other peer (say p2) to facilitate subsequent data sharing. So after the acquaintance is established peer p1 and p2 exchange their translated devices.

The query service allows peers to query acquainted peers. The query service simply exposes operations that allow to query a peer and return results. Similarly for any other services a peer exposes, these services are used via service calls and all the details are hidden from the user. A query can be executed locally at a peer or remotely by forwarding it to other peers; the details are not presented to the user. Furthermore should the implementation details of any service change, the user's application will not be affected. This is one of the characteristics of the SOA that it provides the ability to leverage existing assets, with the goal of building a new architecture by integrating new parts (which are componentized) into the system.

The main motivation for investigating the use of a Web service based architecture comes from the need for open service architectures that facilitate fine-grained data access in the P2P environments. Fulfilling this need would close a gap similar to one that the related field of grid computing has already filled.

Recently grid computing has emerged as a significant important field that is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

distinct from traditional distributed computing by focusing on resource sharing on large scale, and high performance orientation [7]. The need for open standards that define the interaction and promote interoperability between heterogeneous resources inspired the search for an Open Grid Services Architecture (OGSA), specified by the Open Grid Services Infrastructure working group of the Global Grid Forum (GGF) in June 2002. The project Open Grid Service Architecture - Data Access Integration (OGSA-DAI) provides extension to the OGSA [35]. The OGSA-DAI architecture promotes the construction of a middleware based on the Web Services Description Language (WSDL) specifications to support access and integration of data from separate heterogeneous data sources via the grid. An effort similar to OGSA-DAI is still missing for the P2P environments. Such an effort will require spelling out details of the involved architecture.

A second motivation is the perceived need for integrating schema and instance level data mappings in a common framework [22, 40]. Such an integration can only succeed if a clear defined interface between the two worlds is given. Such an interface would allow the sharing of data among both worlds. Interface based on Web services would considerably ease this task.

1.3 Contributions

We propose two versions of Open Service Architecture for Peer Database Management Systems (OSAP) to provide service-oriented architectures for peer database

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

systems, which inherit all the benefits of service oriented architecture. The contributions of this thesis are as follows.

- First, we propose two versions of the OSAP architecture. The first architecture, called OSAP I, hides all the JXTA services or functionalities offered by a peer in a PDBMS as private processes that are accessed only through a single entry point made of a peer manager. The latter is is a web service. The second architecture, OSAP II, offers a loose coupling of peers Web services, along with their decentralization. In this version, the main services of the PDBMS are offered as web services which are invoked by acquainted peers via a set of well-defined interfaces.

- Second, we implemented both proposed architectures, OSAP I and OSAP II within the Hyperion PDBMS framework and run experiments to compare their performance and ease of use with the existing, JXTA-based implementation of Hyperion.

Other than service orientation for the P2P architecture, the main goal and interests of the OSAP is to present architectures not for data integration, but for data coordination in a context where each peer database does not need to constrain itself in order to share data with acquainted peers.

The OSAP architecture is a consistent Service Oriented Architecture upgrade to P2P architecture, which preserves all the specifications and provisions of the PDBMS and takes advantages of the benefits inherent from SOA.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.4 Outline

The rest of the thesis is organized as follows. Chapter 2 presents background research of topics including interoperability of autonomous databases, peer databases and grid computing. In addition to this we also introduce the Service Oriented Architecture (SOA) in this chapter. In Chapter 3 we propose architectures for service oriented peer DBMS. Chapter 4 gives implementation details, and Chapter 5 presents detailed architectural and performance analysis. In Chapter 6 related work is presented, finally we conclude with Chapter 7, which offers a summary of the thesis and discussion of the future work.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

Background

2.1 Interoperability of Autonomous Databases

As the need of sharing data increased at steady pace database systems with shared access to heterogeneous data created by multiple autonomous applications in centralized environment was considered to be the solution. The idea was successful but users then required shared access to multiple autonomous databases. Multidatabase and federated database systems solve many problems arise from centralized data sharing. These systems make databases interoperable that is they are used without globally integrated schema, also they preserve the autonomy of each database supporting shared access [26].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.1.1 Taxonomy of Multidatabase and Federated Databases

A database system may be centralized or distributed. A centralized DBMS manages a single database on the same computer whereas a distributed DBMS manages multiple databases, that may reside or one or many computers. A Multidatabase System (MDBS) supports operations on multiple database systems (DBS), each DBS is managed by a separate DBMS that may either be a centralized or distributed. MDBS is further divided into homogeneous and heterogeneous categories. A MDBS is called homogeneous if all the DBMSs of the DBS are same otherwise it is called heterogeneous.

Different types of federated database systems with various architectures can be created by different levels of integration of DBSs and by different levels of federation services. The taxonomy shown in Figure 2.1 is considered in [25] that focuses on the autonomy dimension. In this taxonomy multidatabase system is classified in two types as non-federated and federated database systems. A non-federated database system is an integration of DBS that are not autonomous. It has a single level of management and does not distinguish local and nonlocal users.

A Federated Database System (FDBS) is a collection of cooperating and autonomous database systems. Each DBS has control over the data they manage, where they cooperate to allow different degrees of integration with no centralized control. FDBS is further classified as no integration and total integration. Based on the management and component integration issues FDBSs are categorized into loosely coupled and tightly coupled. An FDBS is loosely coupled if there is no control enforced by the federated system and administrators and it is the user's responsibility

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

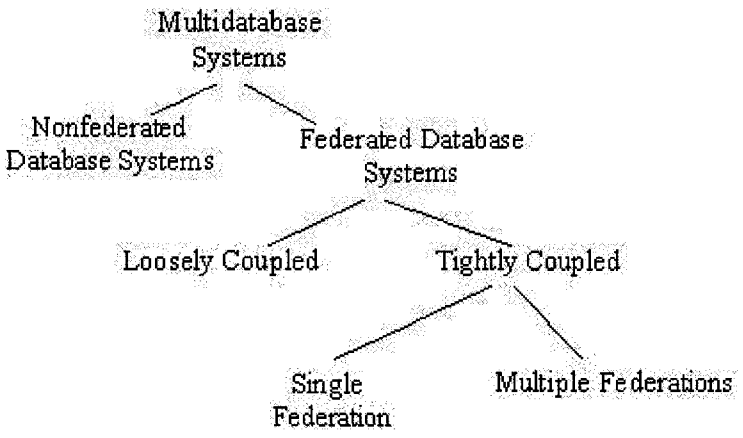


Figure 2.1: Taxonomy of multidatabase systems [25]

to create and manage the federation. This type of FDBS is also called interoperable database system. An FDBS is tightly coupled if it is responsibility of the federation and its administrators to create and maintain the federation [37].

2.1.2 Federated Databases

A Federated Database System (FDBS) is a collection of cooperating and autonomous component database systems (DBS). The software that manages and coordinates the manipulation of cooperating database systems is called Federated Database Management System (FDBMS). Component databases can participate in more than one federation and the DBMS of the component DBS can be centralized, or distributed DBMS or an FDBMS [26, 37].

A component DBS can perform local operations and participate in a federation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

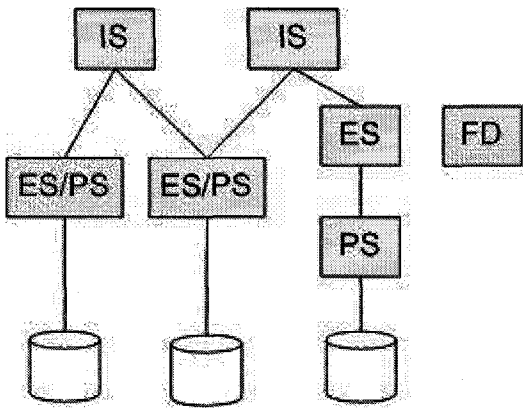


Figure 2.2: Federated database architecture. IS, import schema; ES, export schema; PS, private schema; FD, federal dictionary [10]

at the same time. The integration of component DBSs in FDBS can be managed by the users of the federation or by the FDBS administrator together with the administrators of the component DBSs. One of the significant aspects of the FDBS is that users or administrators can chose the amount of integration depending on their needs and desires to participate in the federation and share their databases.

Figure 2.2 presents federated database architecture. In this figure each database presents an export schema to the federation. The export schema can either be actual conceptual schema or a derived schema which hides some private data. Import schema defines data to be manipulated by federation users and it can be grouped from several export schemas. Each FDBS holds one federal dictionary which is a distinguished component whose information zone is the federation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

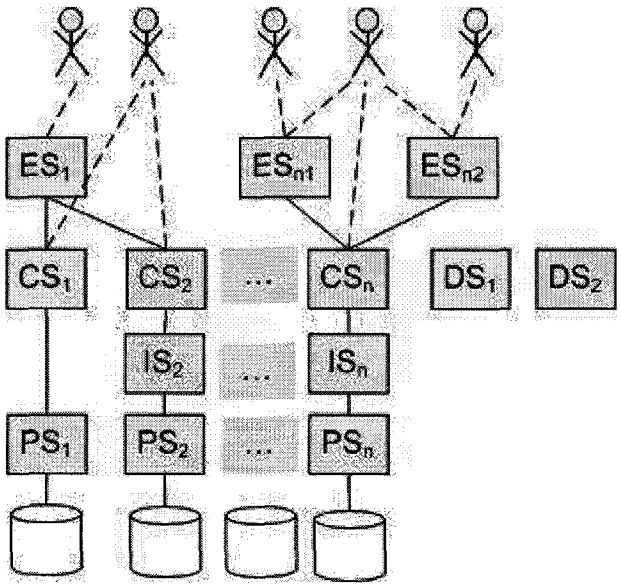


Figure 2.3: Multidatabase architecture. ES, external schema; CS, conceptual schema; DS, dependency schema; IS, internal logical schema; PS, physical schema. [10]

2.1.3 Multidatabases

Litwin argued in [25] that DBMSs should evolve towards MDBS, so they are able to respond to developments in information retrieval, accommodate existing databases, allow applications to be systematically designed in such a way that different data can resided in desired databases. Database autonomy can be preserved and they could become small and less complex [26].

Multidatabase systems are classified as open and closed database system. A closed MDBS manages multidatabases created under its control or managed by DBMSs with the same data model and language as the MDBS. In a closed MDBS the protocols for concurrency control, transaction processing and recovery must be presented to the DBMSs as applications. Sybase and Oracle V5 are examples of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

closed MDBS.

An open MDBS allows operations on multidatabases of other MDBSs or multidatabases managed by DBMSs with different data models and languages. In an open system multidatabase operations can only be processed if different system component having the same role are able to interoperate.

The MDBS architecture illustrated in Figure 2.3 [25] extends the ANSISPARCH architecture. It is divided into three levels; Internal Level, Conceptual Multidatabase Level and External Level. The Internal level contains existing databases managed by DBMSs with its physical internal schema. The Conceptual Multidatabase Level presents the DBMS to the next level that is multidatabase level. It is the conceptual schema (CSi) of the database willing to cooperate. The CSi may be the actual conceptual schema or local external schema. In the latter case, the actual conceptual schema is called an internal logical schema (IS). The conceptual schemas may support different data models and hide private data. The multidatabase level also includes the definition of dependencies between sub-collections of databases expressed as dependency schemas (DSi). The (DSi) allow database administrator to specify and enforce consistency between databases in the absence of global schema. In the External Level of the MDBS architecture external database schemas (ESi) are constructed. A multidatabase schema can also present a collection of databases as a single integrated database where an actual database may however enter different external schemas and may be manipulated locally.

As in the Figure 2.3 a user of MDBS can access multiple databases in two ways, either directly at the multidatabase level using the functions of the multidatabase

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

language or through an external view.

2.1.4 Similarities and differences of FDBS and MDBS

Comparing figures 2.2 and 2.3, the reference architectures of federated databases and multidatabases are similar. Import schema in the FDBS architecture corresponds to an external schema in multidatabase system. A private schema corresponds to the internal logical schema or the conceptual schema at the multidatabase level. An export schema can be considered equivalent to a conceptual schema of the multidatabase level. The FDBS architecture does not specify whether the user to manipulate the export schemas directly, separately or jointly. It also does not support interdatabase dependencies as defined in [26].

On the other hand, the multidatabase architecture does not consider dictionary equivalent to the federal dictionary as a basic feature. Also it does not include capabilities of interdatabase negotiations. Other than these aspects, the difference between two approaches is only in the terms used for similar concepts. The major notion in the federated approach is autonomy and cooperation in interdatabase sharing, where the multidatabase approach shares these goals but it also stresses on the concepts of multidatabase manipulation and interoperability. For a multidatabase to exist a multidatabase language is required. So, multidatabase is a federation of databases, joined through the existence of the multidatabase language that allows the declaration of interdatabase dependencies. A conceptual schema at the multidatabase level can be stated as export schema of the federated database [26].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2 Data sharing in Grid Computing

Gird computing has emerged as a significant important field that is distinct from traditional distributed computing by focusing on resource sharing on large scale, and high performance orientation. The actual problem that inspires the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [8]. Resource sharing is not only concerned with file exchange but direct access to computers, software, data and other resources that are required by a range of collaborative problem-solving and resource brokering strategies. So virtual organizations (VO) are groups of individual, institutions or resource providers who decide to share resources and behave as a single virtual organization [8].

A broad collection of heterogeneous resources that interact and behave in a well-known and consistent manner with each other comprises a grid. The need for open standards that define this interaction and promote interoperability between heterogeneous resources is the inspiration for the Open Grid Services Architecture (OGSA), specified by the Open Grid Services Infrastructure working group of the Global Grid Forum (GGF) in June 2002.

OGSA uses underlying concepts of Grid computing, which is defined as "a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities" [7]. Grid provides a platform that integrates and coordinates resources without centralized control, uses standard, open, general-purpose protocols and interfaces that address fundamental issues as authentication, authorization, resource discovery and access. It also provides various qualities of service such as response time, throughput, availability and security

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

among others.

The project Open Grid Service Architecture - Data Access Integration (OGSA-DAI) provides extension to the OGSA. The main goal of the OGSA-DAI is to produce specifications for generic data services that are based on a common design framework and consistent with the OGSA framework. By the OGSA-DAI interfaces, different, heterogeneous data resources can be accessed and controlled in a single logical resource. Furthermore OGSA-DAI components also hold the capability to be used as basic primitives in creating sophisticated higher-level services that offer the capabilities of data federation and distributed query processing within a Virtual Organization (VO) [32]. OGSA-DAI is the basis on which the Grid data services software Globus Toolkit is built [32]. Another significant goal of the OGSA-DAI is to guarantee that Grid model and OGSA standards accomplish all the requirements of data access and integration of large user populations and large data volumes. It also ensures that Grid data services meet the levels of service required including performance, scalability, resilience, availability and manageability evolution and distribution.

OGSA-DAI consists of building a middleware to support with access and integration of data from separate heterogeneous data sources via the grid. Using the concepts and technologies from the Grid and Web services, it defines Grid services to create, name, access, request service and discover other services. The OGSA-DAI defines interfaces using the Web Services Description Language (WSDL) specification that is used to create and compose distributed systems that have capabilities of lifetime management, change management, and notification. It provides location transparency and integration with underlying native platforms [32].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2.1 Requirements of OGSA-DAI

This section presents some fundamental requirements of the OGSA-DAI that assist it to accomplish its goals in effective and proficient manner. Grid does not account for any central controlling authority so in order for any one to use service, service properties has to be published via registries. Clients query registries to discover the appropriate services. A semantic Web/Grid is constructed to make use of metadata to describe, query and match the relevant properties.

Grid classifies the data of a client into many categories such as personal data, service data, ancillary data, collaboration data and primary structured data. Personal data is data collected by or about individual users, which is typically private. Service data is used to provide a data access and integration infrastructure, some examples of service data includes data describing Web services, defining authorization policies and so on. Ancillary data is the structured metadata, which is used to support bulk or structured data. The collaboration data is classified as the data that is collected to enable scientific information to be shared efficiently and accurately. The scientific and instrumental observations are recorded as primary structured data [32].

In addition to database operations, data transport is an important operation that must be supported by Grid data service. To facilitate an open-ended variety of data models, operations and transport mechanisms, operations are requested using a request document. It specifies a series of activities such as database operations defined using standard query languages and data delivery [32].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

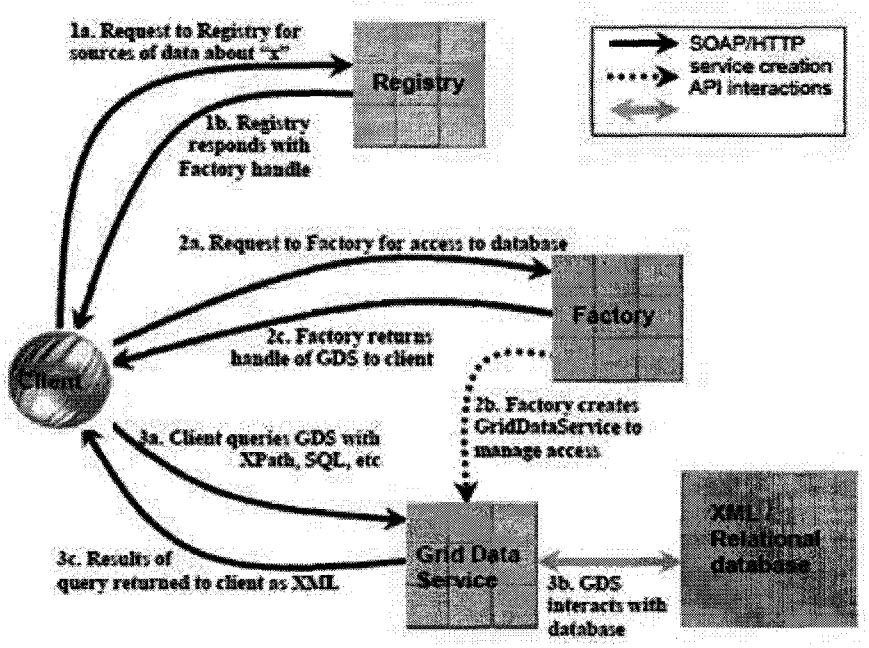


Figure 2.4: Illustration of OGSA-DAI architecture [10]

2.2.2 OGSA-DAI an Overview

This section presents the OGSA-DAI in detail and show how the requirements posed by the Grid architecture are met by the OGSA-DAI.

Grid Services

A Grid service is a Web service which is associated with a particular lifespan, exhibits a set of behaviors and agrees upon a set of interfaces through which a client may interact [10].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

OGSA-DAI

The OGSA-DAI does more than fulfilling objectives of the OGSA specifications by allowing data resources such as databases to be incorporated in an OGSA framework. The motivation behind OGSA-DAI can be listed as follows [32]:

- Integrate data sources and resources into an architecture that conforms to OGSA.
- Acquire information about data that may be distributed amongst several heterogeneous database environments.
- Locate data that is not known beforehand.
- Integrate different data models from distributed databases.
- Discover databases that hold the required data and be able to interpret that data.
- Access data through uniform interfaces.
- Integrate data from various data sources to obtain the required information.

The above requirements of OGSA-DAI yield some constraints that must be satisfied. These constraints include: (1) to provide a service for registration and discovery of databases with the required data, (2) support access to these databases and interaction with them, (3) provide control over the structure of the results in the interaction, (4) provide control over the method and location of delivery of results in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the database interaction, (5) ensure that the database interaction and data transport methods are independent of the type of database and the data model.

The logical view of OGSA-DAI is as a number of co-operating Grid services, some of which may act as proxies or gateways for the system that holds that data. A typical scenario of data access using the OGSA-DAI is that a client (an application program or portal) uses Data Registry to locate a Grid Data Service Factory (GDSF) service. This service has the capability of generating the required access and integration facilities, whereas Data Registry is an extension of the standard OGSA registries that also uses metadata about the contents, organization and operations of the data resources that may be reached. Information that is returned to the client allows it to choose the right GDSF and activate it using its Grid Service Handle (GSH). The GDSF produces required number of Grid Data Services (GDS) that provide the desired access to data resources. The client then uses the GDS to obtain required functionalities [32]. Figure 2.4 depicts the OGSA-DAI architecture.

Grid Data Service Factory

The Grid Data Service Factory (GDSF) is a particular service which offers a data resource and is able to create new Grid Data Services (GDS) through which clients can interact with this data resource. Furthermore, clients can obtain information about the data resource exposed by the GDSF and so decide whether the data resource meets their requirements. GDSF is a persistent service that lives as long as the distributed system wants to publish their data and provide its services [11].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Grid Data Service

The Grid Data Service (GDS) is a significant OGSA-DAI component. It assists in data access, data integration and data delivery functions. GDS supports document-oriented interface for data requests and response, GDS-Perform documents are used by clients to specify operations on the data resource, where the status of the operation, delivery locations as well as data is returned to clients via the GDS-Response document. Each GDS allows access to a single data resource, where the GDS is created by the Grid Data Service Factory (GDSF) which is responsible of managing access to that data resource. GDSF creates GDS for a given lifetime, after expiring of its lifespan GDS cannot be accessed any further, or it can explicitly be terminated via Destroy operation. Clients can access information about the state of the GDS, information about the data resource it provides access to, as well as operations that can be performed upon this particular GDS [10].

2.2.3 Limitations of Grids

As mentioned above, the concept of virtual organization refers to a set of individuals and/or institutions sharing data and computing resources by a variety of collaborative strategies. The goal of the Grid is to provide an appropriate infrastructure for virtual organizations, which is based on standardized services that implement well-established and largely supported models. But this kind of infrastructure conceals the complexity of heterogeneous and distributed data sources, and handles the dynamics of the underlying networking environment.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Grid-based integrated databases are basically loosely coupled database federations, which integrate heterogeneous sources, with the purpose of responding to business demands in a flexible manner. The integration logic is usually contained in specific applications to integration at the "functional" level. In fact, analysts build the integrated virtual schema based on views over the source schemas, and the integration semantics they implement is usually entangled in their internal code. As a consequence, a change in data sources such as adding a new node, or changing metadata would require reprogramming the applications. Researchers in the field of Semantic Grids state that as a risk in the implementation of many real interesting application scenarios, and their effort is intended at prevailing over this limitation. Indeed, Grid Databases suffer of a certain inflexibility, which limits the exploitation of Data Grids in many real situations [3].

2.3 Peer DBMS - (PDBMS)

Peer-to-peer (P2P) computing has emerged as an alternative networking approach to the traditional server-client networking architecture, which favors a direct and dynamic node-to-node model of communication with no centralized control. Peers in the network are autonomous with respect to the control and structure of the network, as peers may join or leave network at any time. When a peer joins the network, it can establish acquaintances with other peers, where acquainted peers can share and coordinate data.

The P2P layer of a PDBMS acts as interoperability layer, that implements the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

functionality required for peers to share and coordinate data without loosing peers autonomy. Services are supported in each peer by adding a conventional DBMS with a P2P layer that allows peers to use each other's data even in the presence of heterogeneous underlying databases. So the P2P layer plays the role that interoperability layers play in multidatabases or federated database systems [12].

2.3.1 Hyperion: A Peer DBMS

The Hyperion project [1] aims at providing data sharing and data coordination with no central control, no global schema, transient participation of peer databases, and constantly evolving coordination rules among peers. In the Hyperion peer database system each peer is a database with its own schema and data. Peers belong to interest groups where all peers in a group share a common interest. When peers become acquainted, logical metadata that is necessary to allow data sharing is exchanged semi-automatically. These metadata help to produce mappings at the data level as well as schema level, which helps to bridge semantic and schematic heterogeneities between the peers. Hyperion provides peer database system as a conventional DBMS amplified with a P2P interoperability layer. This layer implements the functionality required for peers to share and coordinate data without loosing their autonomy [27].

The P2P layer of the Hyperion PDMS consists of a P2P User Interface, a Peer Manager, a JXTA Acquaintance Service, a JXTA Query Service and a JXTA ECA Rules Service. The P2P **User Interface** is where local or remote queries are posed, and to specify distributed ECA rules describing the pattern of data coordination. The **Peer Manager** handles a set of JXTA services offered by a peer node, where

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a service is associated to distributed computation performed by the peer on behalf of its acquaintance. The **JXTA Acquaintance service** semi-automatically establishes and abolishes acquaintances thus inducing a logical peer-to-peer network. The **JXTA Query Service** contains the ability to execute local and remote queries. The **JXTA ECA Rules Service** manages and executes distributed ECA rules to enforce consistency policies and coordinate updates between peers [27].

2.3.2 JXTA

Peer database management systems are based on the P2P architecture, where any peer can interact with other peers and resources directly even when some of the peers and resources are behind firewalls.

JXTA is a well-known P2P architecture that provides a set of open protocols, which allow connected devices on the network to communicate and collaborate in a P2P manner. The goals of JXTA includes interoperability across different P2P systems and platform independence with multiple and diverse languages, systems and networks. "Project JXTA" [18] by Sun Microsystems provides a reference implementation of the JXTA architecture in Java.

The JXTA protocols standardize the way peers discover each other, organize themselves in groups, advertise and discover network resources, communicate with each other and monitor each other [19].

A JXTA pipe is a virtual communication channel that is used to send and receive messages between peers. Input pipes are used to receive input messages from other peers and output pipes are used to send messages to peers. A Message is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

an envelop that contains information and is transmitted using pipes. Messages are usually defined using XML format [19].

Peer databases are built using JXTA and traditional database management systems where peers join the network and advertise communication channel and a set of JXTA services. Peers discover JXTA services of other peers and use them; these services are JXTA services and peers are discovered using JXTA discovery mechanism.

2.4 Service Oriented Architecture (SOA)

The Service Oriented Architecture (SOA) provides architecture as well as a programming model. It enables software designers to design systems that provide services to other applications through publishable and discoverable interfaces where the services can be invoked via the network. Implementing an SOA using Web services technologies enable a new way of building applications within a more powerful, flexible programming model, where development and ownership costs can be reduced. SOA provides ability to leverage existing assets, where the goal is to build a new architecture and integrate existing system that can be componentized [38].

2.4.1 Web Services

Web service is defined as "a programmable application logic accessible using standard Internet protocols" [39]. Web services are accessed via Web protocols (e.g. HTTP) using well-known data format such as XML. In practice, Web services have emerged

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

as a prevailing means for integrating IT systems as assets. Businesses today use Web services for application integration, reusing existing IT assets and connecting to business partners and customers securely.

Service Oriented Architectures have grown over the past years to support high performance, scalability, reliability and availability. Applications have been designed as services so that they can be accessed through programmable interface, where clients have been accessing these services using distributed computing protocols such as DCOM, CORBA or RMI, but these protocols are tightly coupled and are effective for building a specific application and limit flexibility of the system. Also each of these distributed computing protocols are constrained by dependencies on vendor implementations, platforms or languages thus result in limiting interoperability [39]. On the other hand, Web services capture the characteristics of service oriented architecture and combine the latter with the Web. Since Web services support communication using standard protocols, they are completely independent of vendor, platform and language specific details. Therefore the Web services and Web combined eliminate the restriction of DCOM, CORBA or RMI.

Web services communicate with each other using XML (Extensible Markup Language), which is used to describe Web service's interface and message encoding. Standard formats and protocols are used to interpret XML messages; these XML-based technologies include SOAP, WSDL, and UDDI. The Simple Object Access Protocol (SOAP) is a standard communication protocol for Web services. The Web service Description Language (WSDL) presents a standard way to describe a Web service, whereas the Universal Description Discovery and Integration (UDDI) defines

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

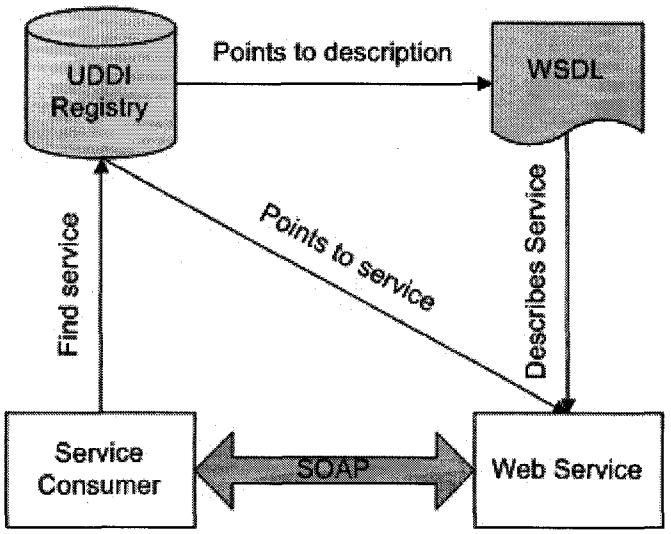


Figure 2.5: Interaction of Core Web Service technologies [39]

a standard to register and discover Web services [39].

Figure 2.5 shows at a high level how SOAP, WSDL and UDDI work together with Web services and a client (Web service consumer). The Web service is described using WSDL specification and registered in UDDI registry. The UDDI maintains pointers to WSDL and the service. When a service consumer wants to use the service, he/she first queries the UDDI registry to find a service that matches his needs and gets the WSDL of the service. Then the service consumer uses the WSDL specification to construct SOAP message to communicate with the service.

2.4.2 Service Oriented P2P

Major database and information systems companies have clearly recognized the need for either Grid or P2P data access. For example, IBM and Oracle are working with

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

National e-Science Center in the United Kingdom on the development of the OGSA-DAI framework. Also a framework for P2P data integration on the Grid is under consideration by IBM-Italy. However these efforts are centered around the Grid infrastructure. There seems therefore to be a clear need for a similar development for a service-oriented infrastructure that is purely centered around the concept of P2P, which though could offer a bridge to the Grid world.

2.4.3 JXTA-SOAP

JXTA-SOAP [19] is another project by Sun Microsystems for peer-to-peer networks; this project aims at providing SOAP communication over the JXTA P2P network. The design goal of this project is to wrap Web services around JXTA services, using JXTA for Web service discovery and message transport as well as SOAP encoding, faults, security etc over the JXTA network.

JXTA-SOAP is a significant addition to JXTA that allows Web services to be incorporated in the JXTA network, a Web service wrapped around a JXTA service serves client in same way as a regular Web service does. The significant advantage using JXTA-SOAP is that it will assist in the creation of a service oriented peer-to-peer network. Service oriented peer-to-peer network will inherit all the benefits from the Service Oriented Architecture. The following points provide detail about the benefits of service oriented P2P and limitations of conventional JXTA P2P networks.

- SOA enables systems to be designed that provide services to other applications through publishable and discoverable interfaces [38]. Conventional P2P using JXTA provides ability to publish and discover services using JXTA publishing

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and discovery mechanism. This is limited to publishing and discovery of JXTA services only.

- Implementing SOA using Web Services technology provides applications with a powerful, flexible programming model where development and ownership costs can be reduced [38].

- SOA provides ability to leverage existing assets, where the goal is to build a new architecture and integrate existing system that can be componentized [38]. Conventional P2P using JXTA provide services to be integrated but the services have to be published, discovered and called using JXTA mechanisms. So if a peer wants to use an existing service, it has to be changed carefully to conform to JXTA interfaces.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

Open Service Architecture for Peer DBMS

We propose an Open Service Architecture for Peer Database Management Systems (OSAP) to provide service-oriented architecture for peer database management systems, which inherits all the benefits of service oriented architecture. The P2P architecture is straightforward. As there is a clear need of either Grid or P2P data access or both, OSAP has to conform to latest technologies and standards.

Other than service orientation for the P2P architecture, the main goal and interests of the OSAP is to present architecture not for data integration, but also for data coordination in a context where each peer database does not need to constrain itself in order to share data with acquainted peers. The intuition behind it is to extend the concept of interoperability used in federated databases. In federated databases interoperability is setup at design time using a globally accessible interface. In peer

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

DBMSs, however interoperability is set up at run-time and access to the peers is strictly local, due to the independence of underlying peers.

The OSAP is a consistent Service Oriented Architecture (SOA) upgrade to peer-to-peer architecture, which preserves all the specifications and provisions of the PDMS and takes advantages of the benefits inherent from SOA.

We presented two different approaches for OSAP architecture. First of all OSAP I is simple and similar to the conventional PDBMS architecture. But OSAP I does not take full advantage of SOA, along with its limited flexibility, extensibility, and maintainability. We further tune the OSAP I architecture to propose OSAP II architecture that has better properties than OSAP I described later in the thesis.

3.1 Requirements

This section documents the requirements of the OSAP architecture and compares it with the requirements of data access and integration on Grids used in the OGSA-DAI framework; these requirements are posed due to the service orientation of the OSAP architecture . Furthermore we document non-functional requirements in this section.

Figure 3.1 shows comparison of OGSA-DAI with the OSAP architecture. The OGSA-DAI is composed of the Grid architecture and Web services; clients in the Grid are autonomous and publish their services using the Grid Data Registry. The PDMS also assumes no central authority and the peers in the network are autonomous, by joining or leaving the network at their discretion. Peers join peer network and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

OGSA-DAI	OSAP
Publish Web service	Publish Web service
Discover Web service	Discover Web service
Request Web service	Establish acquaintance, Request Web service
Query service	Establish acquaintance, Query service
Provide metadata	Provide mappings, coordination rules
Subscribe	Data coordination service
Notification service	Data coordination service

Figure 3.1: Requirements Comparision of OGSA-DAI and OSAP

publish their services as Web services.

Clients in the OGSA-DAI framework discover services published by other service providers using the Grid Data Registry, whereas peers in the PDMS join the peer network and discover services published by other peers. Acquaintance service is the most important service of peer architecture; it enables the mutual peer understanding.

After an OGSA-DAI service is discovered, it can be requested through the well defined interface that is published on the Grid Data Registry; similarly, in the OSAP architecture peers discover services of other peers and request these service using the service's well defined interface. The key difference in requesting a service in the OGSA-DAI and the OSAP is that in OSAP the requesting peer has to get acquainted with the servicing peer to get the service. The acquaintance service is a special service where this rule does not apply because peers have to get acquainted first in order to use further services.

In the OGSA-DAI, a query is processed through the Grid Data Service (GDS) which is created using the Grid Data Service Factory (GDSF), each GDS being

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

responsible for one data source. The response to the query may contain integrated data from several data sources, which is transparent to the user. Whereas queries in the PDMS are posed to the query service on the local peer, answers are retrieved by translating the query according to the schema of acquainted peers to pose query to them. Queries are requested using query service of acquainted peers.

The Grid Data Service provides a schema for the data source it is responsible for. This schema gives detailed information about the data source. The schema can be requested using the well-defined interface published by the Grid Service. In the PDMS, peers have full control on their data which they can share with their neighbors fully or partially, in other words some of the data can be made public whereas the other private. Using mapping table peers allow their acquaintances to pose query on them. Mapping tables are generated using data level mappings.

Services can be subscribed using the OGSA-DAI where any changes, updates or deletions in the data or in the subscribed service itself are notified to the client using the notification service. Using OSAP peers have full control on their data in the PDMS; data can be changed or updated at their desire, if there is a change in the data or its structure that is shared with other peers can result in inconsistency. Data coordination service provides ability to keep consistency of data sharing between.

3.1.1 Non-functional Requirements

Analyzing existing peer database management systems the non-functional requirements such as quality of service, security, scalability, availability and user friendliness and extensibility play significant role. Details of each of these requirements are given

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

below.

Quality of Service (QOS)

Acquaintance establishment and query execution must be done according to agreed upon QOS; this includes, but is not limited to providing proper information about peer and services. For query execution using correct acquaintance information to answer a query, queries may be answered by posing them to remote peers and returning the results in a reasonable time frame. Data coordination services are provided to peers whenever required according to the terms of coordination.

Security

Security is a significant requirement for any system; OSAP needs to provide security at the message level as well as at the transport level. At servicing the request, authentication is necessary; all the peers in the network can request to establish acquaintances, whereas only acquainted peers can query each other. Peers that are not acquainted should not be responded or responded in proper manner.

Scalability

A scalable peer database management system is important in terms of providing data sources of large number of peers and choices to query the acquainted peers increase. However management of a large scaled system is complex and requires more work and responsibility on the part of management team.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Availability

Peers can go online or offline at their own discretion, as well as they can become victim of natural disaster or crash. All of these states of peers should be considered under availability requirement of OSAP. The acquaintances between peers are consistent even they go offline; that means that peers rejoining the network can discover each other and resume their acquaintance. Peers that go offline after receiving unexecuted queries will execute these queries when they rejoin the network according to the peer availability policies.

User friendliness and Extensibility

User friendliness and extensibility of the system is a major requirement that must be satisfied in proficient manner. There may be different types of users with varied level of knowledge. All users at the same level of knowledge would have the system hiding low level details and presenting much higher level views of its functions. The OSAP system should present different levels of usage where user should be able to choose at which level of abstraction he/she may desire to work. Even though it is hard to know before hand all the needs of a user, the system should nevertheless be extensible enough that different components can be replaced and the OSAP framework could evolve over time. But this extensibility should not compromise the objectives and requirements of OSAP.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

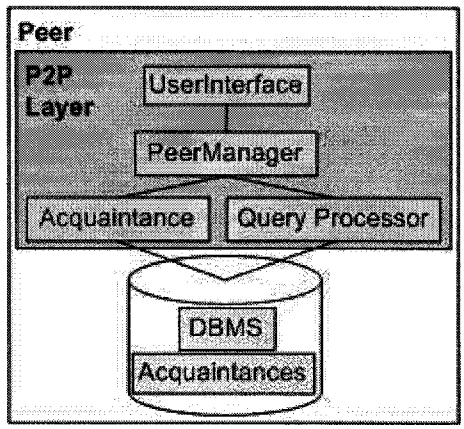


Figure 3.2: Peer Database Management System [27]

3.2 Conventional Peer DBMS

The Figure 3.2 shows a conventional Peer Database Systems Management System where different components such as acquaintance and query processor work together on top of local Database Management System [27]. This can be upgraded to be a Service Oriented Architecture. Below, we present two different alternatives for an OSAP architecture.

3.3 OSAP I

The Open Peer Service Architecture for Peer DBMS - I is one of the settings of service orientation on top of peer databases. Figure 3.3 depicts this architecture. In this setting the P2P layer has four components; a Peer Manager, a Dispatcher, an Acquaintance and a Query processor. The User Interface is an interface between user and the P2P layer. The Database Access helps in accessing different types of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

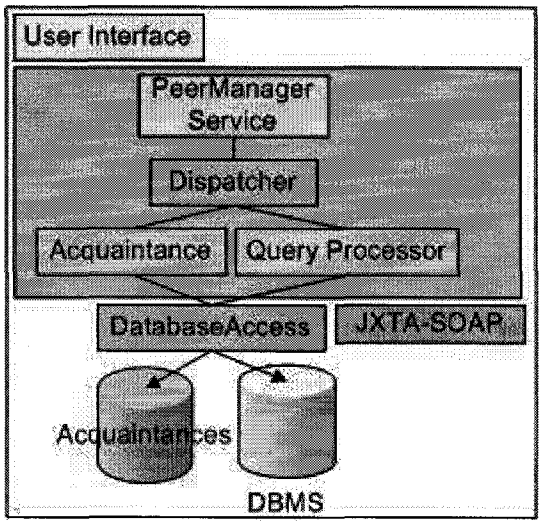


Figure 3.3: Open Service Architecture for Peer Database Management System I

data managed by the DBMS, including acquaintances. The Acquaintance and Query Processor are internal processes which are JXTA services in conventional PDMS.

3.3.1 Peer Manager Web Service

The Peer Manager in the OSAP I architecture is a Web service. It acts as gateway for receiving and sending responses. This is the only Web service in a peer and the communication among peers take place in the form of SOAP calls.

The Peer Manger publishes a well-defined interface that defines behavior of the Web service. The types of request and response manages at the Peer Manager include request acquaintance and response acquaintance, send and receive query execution request, and send and receive query results. When the Peer Manager receives a request, it passes it to the Dispatcher to forward that request to the appropriate internal process. After a request is processed by an internal process, the response

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is handed to the Dispatcher, which forwards the response to the Peer Manager Web service that sends it to the appropriate acquainted peer.

Peer Manager Service Description

In the OSAP I there is only one Web service that is PeerManager service. Peers establish acquaintance and share data using this service. In this section we present a possible WSDL definition for the PeerManager service. The operations in the WSDL are named as; **requestAcquaintance**, **responseAcquaintance**, **releaseAcquaintance**, **respRelease** and **sendMessage**. The service description of this Web service is given in the Appendix A.1

3.3.2 Dispatcher

The role of the Dispatcher is to pass request and response messages between the Peer Manager Web service and the internal processes. It receives messages from the Peer Manager and dispatch them to the appropriate internal processes.

3.3.3 Acquaintance

The Acquaintance process processes the acquaintance establishment. This component of the OSAP architecture acts as an inner process in the P2P layer; it is neither a Web service nor a peer service. According to the design of the OSAP I architecture, the Peer Manager Web service is a gateway for the peer to receive and send messages to acquainted peers. So the Acquaintance process interacts with the acquainted peers indirectly via Dispatcher and the Peer Manager Web service.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

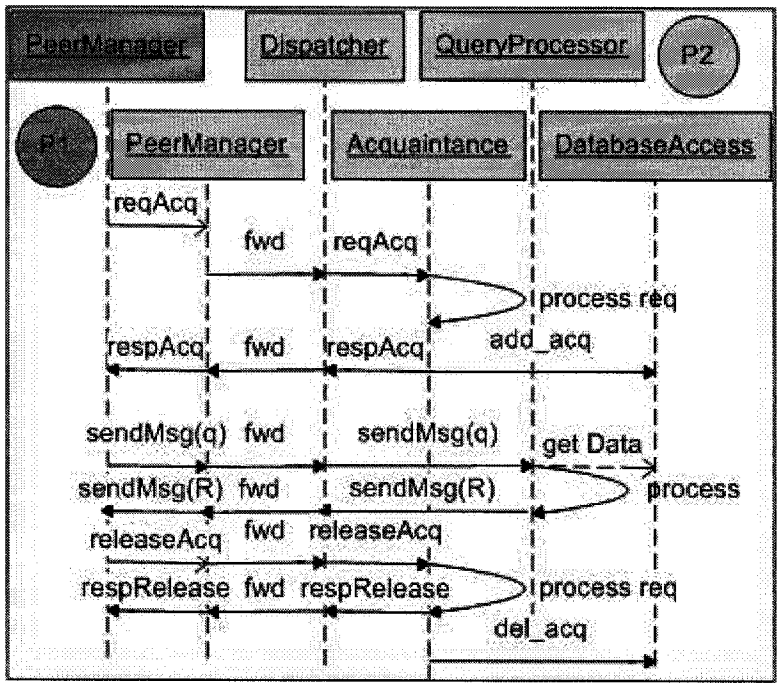


Figure 3.4: Sequence Diagram of OSAP I

3.3.4 Query Processor

The Query Processor is another internal process of the P2P layer of OSAP. Acquainted peers share data by posing queries to each other. Queries are received at a peer by the Peer Manager Web service where the query is dispatched to the Query Processor. Query results are sent to the Dispatcher which forwards them to the Peer Manager service to send to the acquainted peer.

3.3.5 Sequential View of OSAP I

Figure 3.4 provides a sequential view of the Open Service Architecture for a network of Peer Database Management System composed of two peers; p1 and p2. Peer p2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is composed of a PeerManager, a Dispatcher, an Acquaintance manager, a Query Processor, and a DatabaseAccess component, whereas for simplicity, only the PeerManager service is shown for p1.

It is assumed that the peers have already joined the network and p1 has discovered p2. The scenario starts with p1 sending a request to p2 to get acquainted. The peer p2 processes the request and sends a response back to peer p1 by dispatcher and PeerManager service. If p2 wants to establish the acquaintance then it adds the acquaintance information for p1 in the Acquaintances data source using Database Access. When p1 receives the response for acquaintance, it also adds the acquaintance information in the Acquaintances data source.

Since p1 and p2 are acquainted both peers can exchange data and query each other. When a query is posed at the p1 it first processes it locally, if more results are desired then the query is posed to the acquainted peer. The PeerManager of p2 receives the request for the query from p1 and sends it to the Dispatcher, which forwards it to its Query Processor which processes the query and sends the results back to p1 through the Dispatcher and the PeerManager.

To release an acquaintance, peer p1 sends releaseAcquaintance request to the peer p2. If there is an establishment of acquaintance between the two peers p2 deletes the acquaintance from the Acquaintance datasource and sends responseReleaseAcquaintance to p1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.3.6 OSAP I - Acquaintance Scenario

An Acquaintance is established in the OSAP I using the Peer Manager service, the Dispatcher and the Acquaintance processes. Figure 3.5 depicts this scenario. First of all we assume that both peers p1 and p2 have already joined the peer network, p1 discovers p2 and both peers know each other's description. In Figure 3.5 the scenario starts at the point where peer p1 requests p2 to establish acquaintance. The PeerManager of p2 forwards the request to the Dispatcher which dispatches the request to Acquaintance process. The Acquaintance process processes the acquaintance request. If peer p2 wants to establish acquaintance with p2, then it adds acquaintance information in the Acquaintances data source using Database Access. It also sends a response to p1 using Dispatcher and PeerManager service. When peer p1 receives the response it forwards it to the Acquaintance process. If p2 has acknowledged the request to establish acquaintance then p1 adds Acquaintance information in the Acquaintance data source using DataAccess.

3.3.7 OSAP I - Query Scenario

There are two types of queries in peer database management systems, namely local queries and remote queries. Query posed to a peer is remote if it is posed by some other peer in the network otherwise the query is local query. In this scenario depicted in Figure 3.6 it is assumed that both peers p1 and p2 are acquainted. Also the Database and Acquaintances repository are not shown in the figure. The Query is posed by a client to peer 1, this client can be the user of p1 or another peer requesting the query. The query arrives at Peer Manager of p1 as this is the only service exposed

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

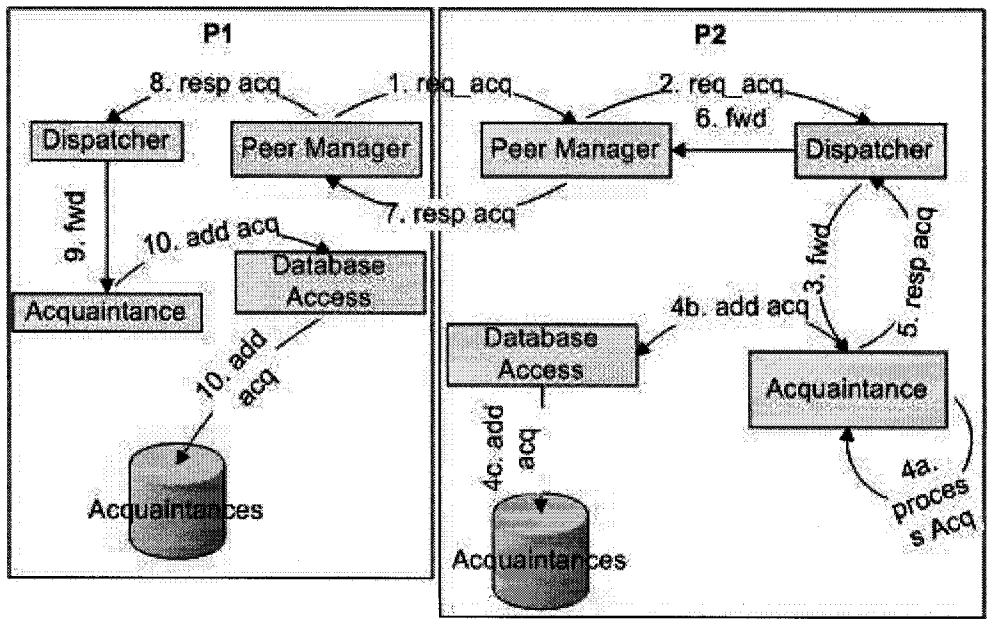


Figure 3.5: OSAP I Acquaintance Scenario

by the peer. The request is dispatched to the Query Processor using the Dispatcher. The details of the given query processing scenario are as follows.

- The Client poses a query at the Peer Manager of p1; the client can be user of peer 1 or another peer requesting query.
- The Peer Manager dispatches the query to the Query Processor using the Dispatcher.
- The Query Processor processes the query.
- In order to retrieve complete results, the Query Processor retrieves acquaintance information and translates the query to the acquainted peers. The Query can be translated and posed to more than one acquainted peer (but in this particular scenario only one acquainted peer is considered).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- The Query Processor sends translated queries to acquainted peers through the Peer Manager using the Dispatcher.
- When the query is received at p2, the Peer Manager forwards it to the Query Processor using the Dispatcher.
- The Query Processor processes the query, retrieves data from the database management system using the Database Access and sends results to the Peer Manager using the Dispatcher. Note that the query is already translated for p2 according to its database information, so p2 does not need to translate the query.
- The Peer Manager sends the results to p1, where it is forwarded to the Query Processor using the Dispatcher.
- After receiving results of query from acquainted peers the Query Processor sends them to the Peer Manager using the Dispatcher that forwards it to the appropriate client. To make the figure easy to understand, this last point in the scenario is omitted.

3.3.8 Benefits

The benefits of having only one Web service and one point of request and response for the P2P layer of OSAP is that this makes the design and implementation of the architecture simple. All the services offered by a peer in a conventional PDMS are hidden as private processes that can be used by other peers only through the use of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

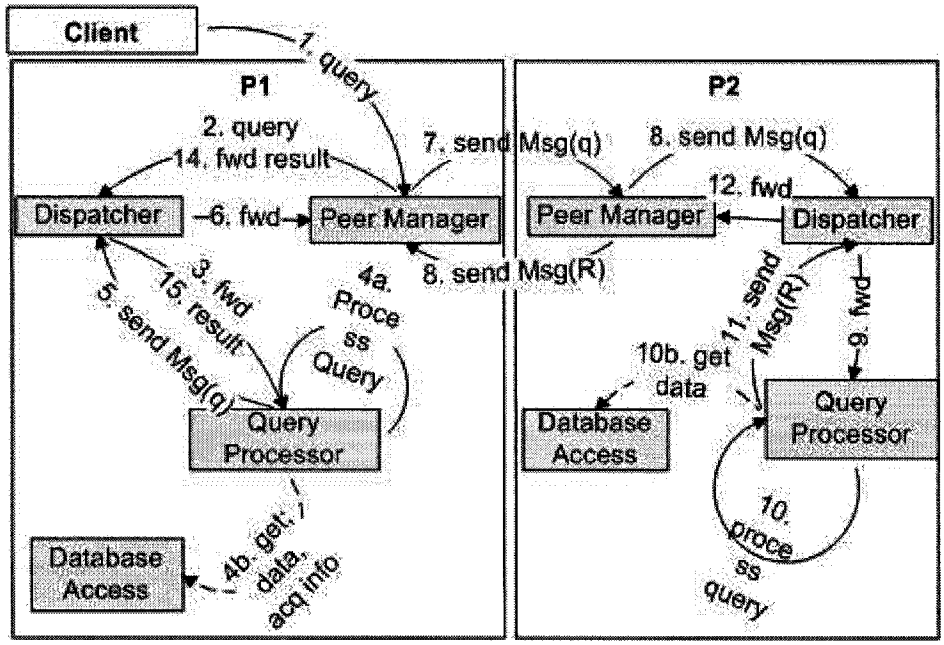


Figure 3.6: OSAP I Query Scenario

the well-defined interface of Peer Manager. The control and management of the peer is simple as there is only one point of access to the system that makes it simple to handle and easy to manage.

3.3.9 Drawbacks

The major drawback using this approach is that the architecture appears to be centralized towards the Peer Manager. Any type of communication between the peers has to go through so called the gate of Peer Manager, where the internal processes themselves have no control. The centralization in the OSAP loses the true spirit of a service oriented architecture; that is, the components that make the P2P layer of a service-oriented PDMS are tightly coupled together. The internal processes

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

such as the Acquaintance process and the Query Processor are related to the Peer Manager in a very tight relation. Moreover, that is the existence of the internal processes depends on the existence of the Peer Manager. Furthermore, to add a new process in the system, the definition of the Peer Manager has to be updated in order to act appropriately.

The centralized approach lacks fault tolerance and is prone to bottleneck. Any problem to Peer Manager would be propagated to the whole peer because the Peer Manager is the entry point for requests and exit point for responses.

3.4 OSAP II

OSAP II avoids the centralization and tight coupling that characterize the previous approach. Figure 3.7 depicts the OSAP II approach. OSAP II contains components such as the Acquaintance and Query Processor Web services in the P2P layer. To interact with the user, a User Interface is added on top of the P2P layer. To communicate with the DBMS and Acquaintances data source 'Database Access' is used. The User Interface and Database Access are neither peer services nor Web services but they help to create a service-oriented peer.

3.4.1 Acquaintance Web Service

The role of the Acquaintance Web service is to provide a description about a peer as well as the data and services that it offers. Before getting acquainted, peers use this information to gain knowledge about each other. Interested peers can send an

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

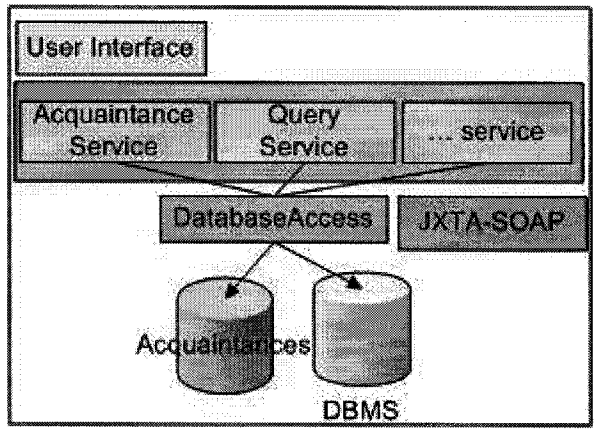


Figure 3.7: Open Service Architecture for Peer Database Management System II

invitation to other peers to get acquainted. Peers can request the description of other peers and its data sources if desired. In essence this service brings two peers to get to know each other before getting acquainted. If for some reason peers do not share similar interests or do not want to share data, then peers can just deny the acquaintance proposal. The acquaintance service is the only service that is exposed to the entire peer network. Since peers have independent data sources and vocabulary so to exchange or query data, they must understand each other's way of describing the data. At the high-level acquaintance service creates a logical communication link. When peers desire to release acquaintances between them they can do it by invoking the appropriate methods of the Web service.

Acquaintance Service Description

The acquaintance service is used to establish acquaintance between peers. In this section we present a possible WSDL definition for the Acquaintance service. The operations in the Acquaintance service WSDL are named as; **requestAcquaintance**,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

responseAcquaintance, **releaseAcquaintance**, and **respRelease**. The service description of the acquaintance Web service is given in the Appendix A.2

3.4.2 Query Web Service

Peers get acquainted to share and exchange data; the query service is a Web service that is exposed to acquainted peers. Request and response to the query are sent using SOAP calls. To pose a query from a peer p1 to another peer p2, Query Service(s) of both peers are used. The Query Web service controls the query execution and more than one query can be executed at a peer.

Query Service Description

The Query Service is described using two operations named as:
sendQuery, and **sendResult**.

Both operations takes an XML encoded string that contains names of the source and the destination peers, query id, and query or query result. When a peer p1 sends a query to peer p2, it invokes sendQuery operation of peer p2, whereas when the peer p2 sends results of the query it invokes sendResult operation of peer p1. Peer obtains acquaintance information from the Acquaintances data repository to check if the query or query result received is from the acquainted peer. The service description of this Web service is given in the Appendix A.3.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.4.3 Additional Web Services

In addition to acquaintance and query processing, further services can be added to a peer and exposed to acquainted peers. Any additional Web service will be a stand alone service similar to the acquaintance and query services.

3.4.4 Sequential View of OSAP II

Figure 3.8 provides a sequential view of the OSAP II architecture. The figure shows the Acquaintance and Query Processor services and the Acquaintance, Query Processor and Database Access of p2. This scenario starts with the assumption that both peers have already joined the peer network and discovered each other. Peer p1 sends a requestAcquaintance message to create an acquaintance with p2, peer p2 processes the acquaintance request and sends the response to p2. If p2 accepts the acquaintance then it adds the acquaintance information in the Acquaintances data source. When p1 receives the acquaintance response, it also adds the acquaintance information in the Acquaintances data source.

Acquainted peers query each others data. In this scenario, p1 queries p2 using the Query Processor Web service (the details of query processing is not shown in p1), after receiving query from p1 processes the query, retrieves data from database using Database Access, and sends results to Query Processor Web service of p1. To delete acquaintance, peer p1 sends releaseAcquaintance message to p2 which responds with respRelease message. Both peers update the acquaintance information.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

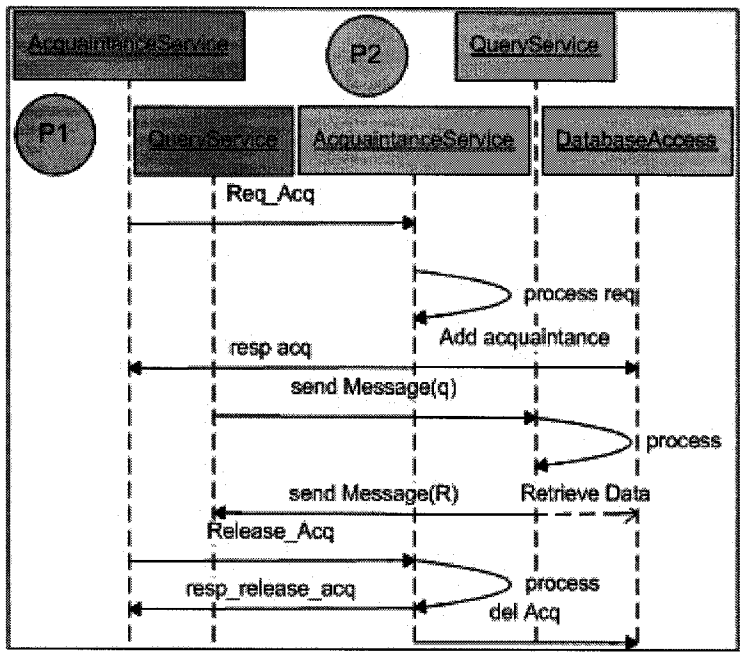


Figure 3.8: Sequence diagram of OSAP II

3.4.5 OSAP II - Acquaintance Scenario

Since peers have independent data sources and vocabulary so to exchange or query data they must understand each other's way of describing the data. The acquaintance Web service establishes an understanding between two peers so that they can send each other's data and be able to query it. Figure 3.9 explains in detail how an acquaintance is established between two peers.

3.4.6 OSAP II - Query Scenario

The OSAP II query scenario is given in Figure 3.10. We assume that both peers; p1 and p2 are acquainted, also Database and Acquaintances database is not shown in the figure. In this scenario a client requests a query from the Query Web Service of p1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

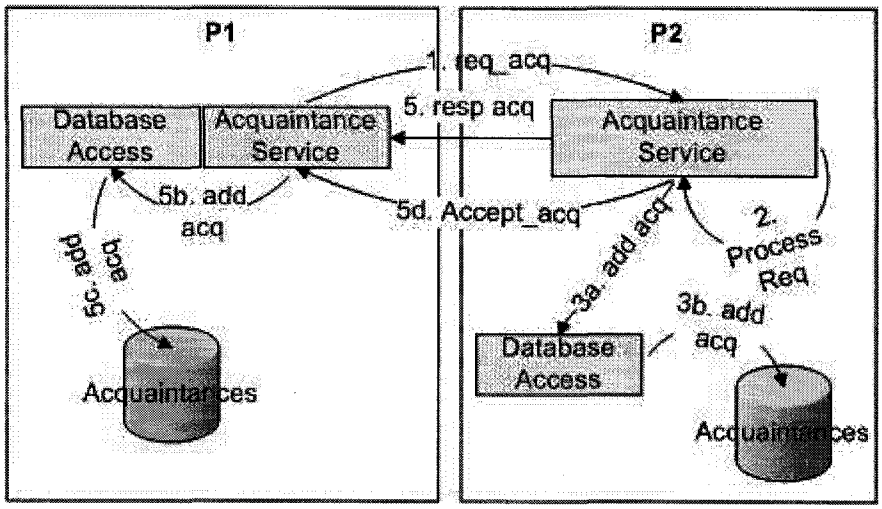


Figure 3.9: OSAP II Acquaintance Scenario

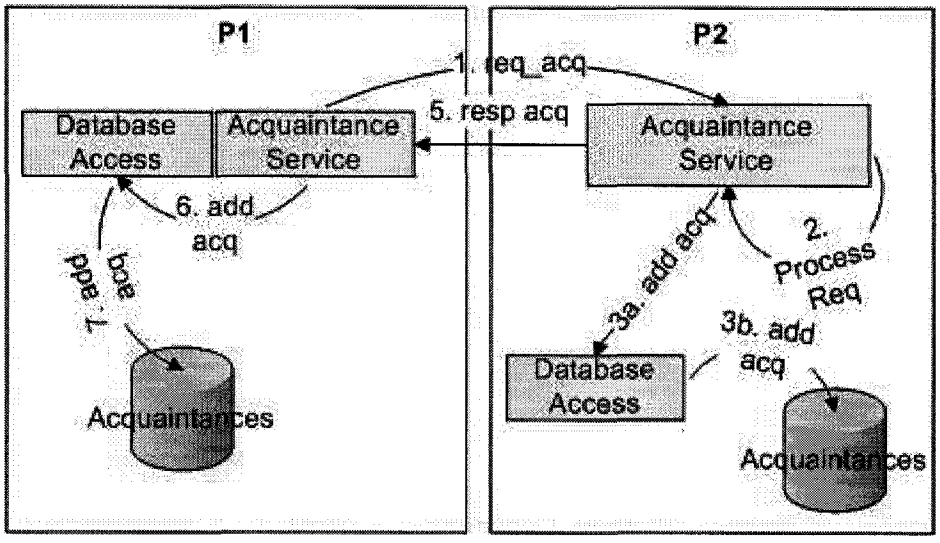


Figure 3.10: OSAP II Query Scenario

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

which processes the query locally, and retrieves essential data from the local database using the Database Access. To retrieve further answers, the query is translated using acquaintance information over the acquainted peers. The translated query is posed to the acquainted peer p2 using the Query Web Service. The Query Web Service processes the query and retrieves data from the database using the Database Access. The results of the query are sent to the Query Web Service of p1. After getting all the results from the acquainted peers, the Query Web Service sends results to the client. For simplicity we omit this step in the figure.

3.4.7 Significances

The two main advantages of this approach to OSAP I are loose coupling of Web services, together with their decentralization. In this approach the P2P layer is componentized into different Web services, where each service is offered as a Web service that is invoked via the network using a well-defined interface. This approach provides power and flexibility in terms of development and usage of the system. The control and management of Web services is distributed and each service is managed independently of other ones.

Additional Web services can be added to the system by updating the service list offered by the peer in the Acquaintance service and publishing a well-defined interface of the new service in the peer network.

Making a service out of each component such as Query Processor and the Acquaintance component yields a fault tolerant system since problems incurring in one service do not propagate to other services directly.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.4.8 Drawbacks

A drawback in this approach is existence of multiple points of request and response for a peer. This makes the control management of peer complex. Having multiple points of request and response OSAP II is more prone to security breaches than OSAP I.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

Implementation

In this chapter, we provide a reference implementation of the OSAP-I architecture for the Hyperion PDBMS. First the details of the conventional Hyperion PDBMS have been given in Section 4.1. Then in Section 4.2 we explain how message passing using JXTA-SOAP works. Then the details of the service oriented Hyperion (SO/Hyperion) SO/Hyp1 and SO/Hyp2 PDBMS is given in Section 4.3.

4.1 Hyperion Message Handling

Hyperion Peer Database Management System uses JXTA as underlying P2P architecture. JXTA is a well-known P2P architecture that provides a set of open protocols that allow connected devices on the network to communicate and collaborate in a P2P manner. Furthermore "Project JXTA" by Sun Microsystems provides a reference implementation of the JXTA architecture in Java.

In Hyperion PDBMS peers join the network to share and coordinate data. Peers

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

establish acquaintance with other peers and acquainted peers exchange and query each other's data. Peers publish JXTA services such as acquaintance service, query service, etc. for other peers to use. Acquainted peers send messages to each other using JXTA pipes, the message handling mechanism and how it is dispatched to appropriate service is presented in detail in the following sections.

Acquainted peers use pipes to send messages; input and output pipes are allocated by each peer for an acquaintance between them. For example if peer p1 is acquainted with p2; each peer allocates output and input pipes for the acquaintance.

JXTA allows pipe binding capabilities that means input and output pipes are associated with each other. When a peer p1 sends a message to p2; p2 receives message in its input pipe sent by p1 using its output pipe. Messages in Hyperion contain information about source (origin of the message) and destination peer, message identifier, message type, message handler and etc.

Messages arrive at input pipes are handled by the peer. In Hyperion PDBMS peers advertise different JXTA services for other peers so messages from acquainted peers arrive at input pipes are handled using dispatcher-like mechanism. For each acquaintance a dispatcher is created. The dispatcher implements PipeMsgListener interface that allows listening of events as message arrivals. The method pipeMsgEvent() is called any time a message is received over an acquaintance. Following steps are taken to dispatch a message:

- Check if destination of the message is correct.
- Get name of the service handler from the message.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

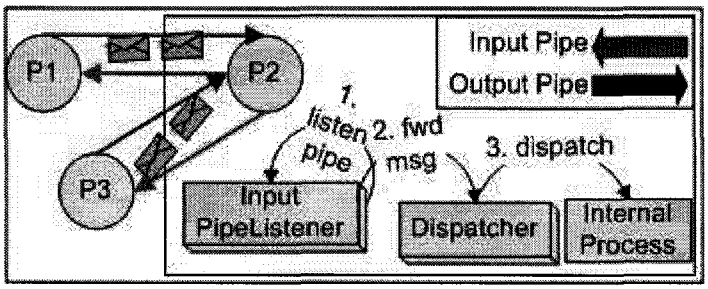


Figure 4.1: Hyperion PDBMS Message Handling

- Using service manager get handle of the service.
- If the message is a response from the acquainted peer add message to its response queue.
- Else the message is a request from the acquainted peer then add message to its request queue

The Figure 4.1 shows the details of how a message is received from the acquainted peer. The figure presents high level detail of arrival of message and its dispatching. In this figure peer p2 is presented in detail where the rest of peers use the same mechanism. Each pipe is associated with a listener as well as a dispatcher that dispatches the messages. Any time a message is arrived at the pipe, it is forwarded to dispatcher that dispatches the message to appropriate JXTA service.

4.1.1 Code Details of Message Handling

HyperionPeer

The HyperionPeer is the main class that represents the peer, this class as it starts joins the default JXTA platform and group. It advertises itself to the group and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

starts the HyperionServiceManager thread. It also discovers other peers in the same group, create new groups, join a specific group or leave it. The HyperionPeer provides methods to establish new acquaintances and keeps reference of the acquainted peers.

Acquaintance

Acquaintance is an interface that describes core behavior required to establish an acquaintance between two peers. This includes getting name and other information of the acquainted peers, send messages and release an acquaintance. The acquaintance between two peers is established using a request/response protocol. For example to establish acquaintance between two peers; p1 and p2, peer p1 sends an initial request to p2 to establish acquaintance. P2 receives the request and acknowledges it with a response message. P1 and p2 adds each other in their acquainted peers list.

AcquaintanceFactory

AcquaintanceFactory is an abstract class that uses factory and singleton design pattern. This class returns instance of the factory to create new acquaintances. The singleton factory object makes sure that a peer does not hold more than one factory objects to create acquaintances and only one object is responsible for that particular job. This class also provides abstract method definition to create acquaintance object of type 'Acquaintance'.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BiDiPipeAcquaintanceFactory

The BiDiPipeAcquaintanceFactory extends AcquaintanceFactory, it overrides the operation in the AcquaintanceFactory class to create acquaintance objects. The acquaintance objects created are of type BiDiPipeAcquaintance, since BiDiPipeAcquaintance is an Acquaintance.

BiDiPipeAcquaintance

BiDiPipeAcquaintance implements Acquaintance interface, so this class defines all the behaviors of an acquaintance. For each acquaintance object it stores information such as; peer group, reference to the HyperionPeer object which created the acquaintance, acquainted peer name, acquainted peer id and reference to a dispatcher as BiDiPipeDispatcher. In addition to this, BiDiPipeAcquaintance also creates a bidirectional pipe of the type JxtaBiDiPipe. The pipe is advertised in the peer group and a dispatcher is set to its listener. The sendMessage() operation inherited from Acquaintance interface in the BiDiPipeAcquaintance takes 'Message' object as a parameter that is located in the JXTA package. The message is sent to the acquainted peer simply by using the sendMessage operation of the bidirectional pipe which takes the Message object: *bidipipe.sendMessage(msg:Message).*

Dispatcher

The Dispatcher is an interface that extends PipeMsgListener class, it defines core behavior of dispatching a message as well as listening to the pipe inherited by the PipeMsgListener. The method definition to dispatch a message is given as: *public*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

void dispatchMessage(Message message).

DispatcherFactory

The DispatcherFactory is an abstract factory class that creates singleton object of the factory. It gives an abstract definition of the method to create dispatcher associated with an acquaintance: *public abstract Dispatcher createDispatcher(Acquaintance acq).*

BiDiPipeDispatcherFactory

The BiDiPipeDispatcherFactory extends DispatcherFactory class and provides the implementation of the createDispatcher method.

BiDiPipeDispatcher

The BiDiPipeDispatcher class implements the Dispatcher interface, since the Dispatcher extends the PipeMsgListener so this class provides implementation for the inherited method: *public void pipeMsgEvent(PipeMsgEvent event).*

As the BiDiPipeDispatcher object is a pipe listener it stores a reference to the acquaintance associated with that pipe. Each time a message is arrived at the pipe the dispatcher listens to the message and pipeMsgEvent() is invoked automatically. The pipeMsgEvent() method checks the message arrived and invoke dispatchMessage() method. The message arrived contains the information such as the type of message; request or response and the service associated with the request or response. If the message arrived is a service request from the acquainted peer, the dispatcher

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

adds the request to the request queue of the HyperionServiceManager. On the other hand if the message arrived is a response from the acquainted peer, the message is added to the service response queue of the HyperionServiceManager.

HyperionServiceManangerImpl

The HyperionServiceManangerImpl implements HyperionServiceMananger interface. It keeps track of the services running on the peer such as query service. The name 'service' is used in abstract manner as these services are neither Web services nor JXTA services. For example a query service is implemented as a thread which can be started and stopped any time desired by the user. The HyperionServiceManangerImpl manages several queues that stores information for services running on the peer, requests for the services and responses for the services. The HyperionServiceMananger extends Runnable interface and the HyperionServiceManangerImpl is a thread. This thread runs for the lifetime of the peer. Any time when there is a request or response in the request or response queues, it retrieves reference to the appropriate service and hands over the request or response.

4.2 Message-passing using JXTA-SOAP

This section gives details of message passing among peers using JXTA-SOAP. Other than JXTA services, JXTA-SOAP provides capability of publishing Web services in a peer network. JXTA-SOAP allows the user to publish a Web service in a similar way as it is published in Web servers such as Tomcat or BEA WebLogic Web servers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

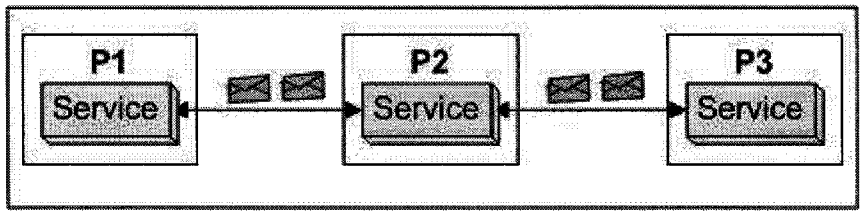


Figure 4.2: Message passing using JXTA-SOAP

The Web services are discovered by peers in the network, it contains a service description with it in the form of a WSDL specification. To request the service, pipe advertisements are located for the required service and a 'Call' object is created. Such an object takes several parameters such as the pipe advertisements, service descriptor, WSDL file, service name and port name. The operation name is passed to the 'Call' object and the call is invoked. JXTA-SOAP publishes JXTA services wrapped as Web services and SOAP calls can be made through JXTA pipes.

Figure 4.2 shows that peer P2 is acquainted with peer P1 and peer P3. It shows a high level the convenience of message passing and handling using JXTA-SOAP. Peers use each others' services without taking care of pipes explicitly.

4.3 A Reference Implementation

We provide a reference implementation of the OSAP I and OSAP II architectures for the Hyperion peer database system. We first give a brief summary of how the SO/Hyperion1 works, and then provide the detailed code description of both implementations.

The PeerManager Web service is published or advertised in the peer network

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

when the peer is started. The Web service is advertised using a unique service name in the peer group. This Web service contains a port name, a peer id, a peer group id as well as a service description. The Web service description contains definitions of public methods that the service implements. Using this description other peers can invoke methods of the service according to the method definitions. When the Web service is advertised, JXTA pipes are opened automatically. SOAP messages are transferred in the network using the JXTA pipes.

The unique service name for the PeerManager Web service is generated using the string "PeerManagerService" concatenated with the unique peer name. For example a peer named "UnitedAirlines" and "AirCanada" advertise their PeerManager Web services as "PeerManagerServiceUnitedAirlines" and "PeerManagerServiceAirCanada" respectively.

Peers establish acquaintance with other peers by discovering them and sending a request to establish an acquaintance. Acquaintance establishment is a simple request/resposne protocol. During the process of establishing an acquaintance, peers also discover Web services of the acquainted peers. Using the found advertisements, peers invoke the Web services of acquainted peers.

The PeerManagerService Web service is invoked and the message is passed as a parameter. Figure 4.3 shows details of how a Web service is invoked and how a message is dispatched to the internal processes. In this figure, peer p2 is acquainted with peer p1 and p3. A detailed picture of peer p2 is shown in the figure. The scenario starts where the PeerManagerService is invoked, the parameter of the method invoked gives the details of the message. The message is forwarded to the Dispatcher that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

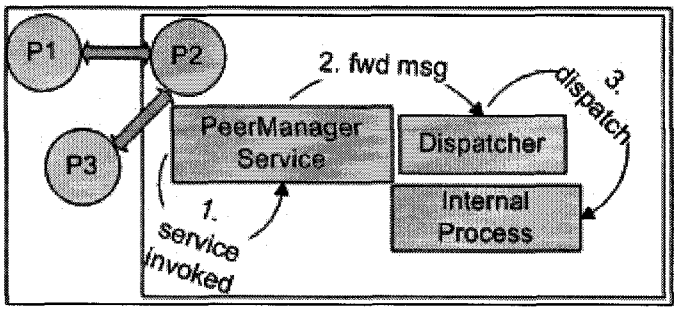


Figure 4.3: Service Oriented Hyperion PDBMS Service Invocation Scenario

dispatches it to the appropriate internal process.

4.3.1 Code Details of OSAP-I HYPERION

The following sections give the detailed code description of the SO/Hyperion peer database system that follows the specifications of the OSAP-I architecture. The PeerManagerService class in the SO/Hyperion maps directly to the PeerManagerService in the OSAP-I architecture. The Dispatcher interface defines the core behavior of dispatching messages in the peer which is implemented by the BiDiPipeDispatcher class and it maps directly to the Dispatcher component of the OSAP-I architecture. The functionalities of the Acquaintance component in the OSAP-I architecture are performed by the HyperionPeer class. The query processing component in the SO/Hyperion is same as the query processing component in the Hyperion peer database system. The query processing is taken as a black-box component or service by the HyperionServiceManager. The request and response messages are queued in the HyperionServiceManager by the dispatcher which hands over the messages to appropriate components. The internal query processing component in SO/Hyperion

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

maps to the Query Processor in the OSAP-I architecture.

The SO/Hyperion stores the acquaintances in the form of Acquaintance objects in the AcquaintanceData data-structure which is edited by the HyperionPeer class whenever an acquaintance is established. The translation of schemas of the acquainted peers is stored as files in each peer. Both the AcquaintanceData data-structure and the acquaintance files are mapped to the Acquaintances data source in the OSAP-I architecture. The database used in the SO/Hyperion is the MySQL database which simply maps to the Database in the OSAP-I architecture. Therefore the SO/Hyperion meets all the specifications of the OSAP-I architecture.

HyperionPeer

The HyperionPeer is the main class that represents a peer, this class as it starts joins the default JXTA platform and group. It advertises itself to the group and starts the HyperionServiceManager thread. This class publishes PeerManagerService Web service in the group that the peer belongs to. The Web service name is unique in the peer group and it is generated using the string 'PeerManagerService' concatenated with the unique peer name. The HyperionPeer class also contains the functionality for discovering other peers in the same group, for creating new groups, and for joining a specific group or leaving it. The HyperionPeer class provides methods to establish new acquaintances. Each time an acquaintance is created the reference of the acquainted peer is stored in a simple data-structure composed of AcquaintanceData and AcquaintanceDataFactory.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

AcquaintanceData

The AcquaintanceData class is a simple data-structure that stores reference to the acquaintance objects. The acquaintance objects can be added to AcquaintanceData or already contained acquaintance objects can be removed from the data-structure. The AcquaintanceData class is a singleton created by AcquaintanceDataFactory. The purpose to create the AcquaintanceData to be a singleton is that each peer contains only one instance of the acquaintance data-structure that is manipulated by multiple objects in the peer.

BiDiPipeAcquaintance

BiDiPipeAcquaintance implements Acquaintance interface, so this class defines the core behavior of an acquaintance. For each acquaintance object, it stores information such as peer group, reference to the HyperionPeer object which created the acquaintance, acquainted peer name, acquainted peer id, and reference to the dispatcher as BiDiPipeDispatcher. During the establishment of an acquaintance, the Web service of the acquainted peer is discovered using its unique name. So the acquaintance object stores the reference to the Web service of the acquainted peer which is used to invoke the service whenever needed. The explicit handling of pipes is not required in the presence of the Web services. Also there is no need of attaching a separate dispatcher for every acquaintance object, as there are no pipes so the pipe listener is not required as well.

To send a message to acquainted peers the Web service of the acquainted peer is invoked using the 'Call' object as: *call.invoke(new Object[] { message })*.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Dispatcher

The Dispatcher interface defines the core behavior of dispatching a message. The method definition to dispatch a message is given as: *public void dispatchMessage(Acquaintance acq, Message message).* The dispatcher dispatches a message according to the acquaintance object passed and the Dispatcher class is not a pipe listener in the SO/Hyperion.

DispatcherFactory

The DispatcherFactory is an abstract factory class that creates singleton object of the factory. It gives an abstract definition of a method that creates a dispatcher as: *public abstract Dispatcher createDispatcher().*

BiDiPipeDispatcherFactory

The BiDiPipeDispatcherFactory extends the DispatcherFactory class and provides the implementation of the *createDispatcher()* method. The main difference in the Hyperion and the SO/Hyperion implementation is that the dispatcher created by the BiDiPipeDispatcherFactory in the SO/Hyperion is a singleton object and is not associated with a particular acquaintance. The reason for this is that the dispatcher object is not a pipe listener and the only purpose of this object is to dispatch a message. So there is no need of having a separate dispatcher for each acquaintance if the same functionality can be provided by one object.

The advantage of having a singleton dispatcher is that it will reduce the overhead of k objects if a peer p1 is acquainted with k peers where $k \geq 0$. The singleton

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

dispatcher object will not become a bottleneck in the peer system. To see this consider a scenario where a peer p1 is acquainted with k peers. If all of those k peers invoke the Web service of the peer p1 to request a query at the same time then the requests will be queued and one request at a time will be processed by the Web service. It will not decrease response time if the peer p1 has a separate dispatcher for each acquaintance, because the requests are processed one at a time. So having one dispatcher in the system will not create a bottleneck.

On the other hand, in the implementation of the Hyperion system there is a separate pipe for each acquaintance which is associated with a separate dispatcher. One might argue that this is better than having a Web service where all the messages arrive at the Web service using Remote Procedure Call (RPC), and the Web service is prone to bottleneck. However, this is not the case, even if there are more pipes to receive messages from acquainted peers, a single peer is executed on one processor where the execution takes place sequentially. There is no parallelism in the dispatching of messages in the Hyperion approach. This means that the Hyperion system, having separate pipes and dispatcher objects for each acquaintance, has no advantage over the SO/Hyperion. Section 5.3 gives details about the performance comparison between the Hyperion and SO/Hyperion.

The SO/Hyperion is more scalable than the Hyperion system because it reduces the overhead of the dispatcher objects as well as the pipe management to the degree of the number of acquainted peers in the whole peer network.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

BiDiPipeDispatcher

The BiDiPipeDispatcher class implements the Dispatcher interface. It follows the singleton design pattern therefore the dispatcher is not associated with any particular acquaintance. The BiDiPipeDispatcher class provides implementation of the dispatchMessage() method as: *public void dispatchMessage(Acquaintance acqn, Message message).* The message parameter contains the information such as the type of message such as request or response and the internal process associated with the request or response. If the message that arrived is a service request from the acquainted peer, the dispatcher adds the request to the request queue of the HyperionServiceManager. On the other hand if the message that arrived is a response from the acquainted peer, the message is added to the service response queue of the HyperionServiceManager.

PeerManagerService

The PeerManagerService is a simple class that is published as a Web service in the peer network. It provides implementation of the sendMessage() method according to the WSDL definition published in the peer network. The PeerManagerService holds a service descriptor of the type ServiceDescriptor. The descriptor is initialized by the name of the service class, service name published, service description string, peer group ID, peer group name, and peer group description. The service descriptor is used by the HyperionPeer class to initialize the Web service.

The sendMessage() operation of the PeerManagerService Web service takes the Message object as a parameter. Among other information the message contains the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

acquainted peer name, so that the PeerManagerService accesses the Acquaintance-Data data-structure to retrieve the object for that acquaintance. After retrieving the acquaintance object, the dispatcher is called and the message is dispatched as: *acq.getDispatcher().dispatchMessage(acq, message).*

4.3.2 Code Details of OSAP-II HYPERION

The following sections give the detailed code description of the SO/Hyperion peer database system that follows the specifications of the OSAP-II architecture. The AcquaintanceService class in this SO/Hyperion maps directly to the Acquaintance-Service in the OSAP-II architecture. The QueryService maps to the QueryService in the OSAP-II architecture.

Similar to the previous implementation Hyperion stores the acquaintances in the form of Acquaintance objects in the AcquaintanceData data-structure which is edited by the HyperionPeer class whenever an acquaintance is established. The database used in the SO/Hyperion is the MySQL database which simply maps to the Database in the OSAP-II architecture. Therefore the SO/Hyperion meets all the requirements and specifications of the OSAP-II architecture.

HyperionPeer

The HyperionPeer is the main class that represents a peer, this class as it starts joins the default JXTA platform and group. It advertises itself to the group and publishes AcquaintanceService Web service in the group that the peer belongs to. The HyperionPeer class also contains the functionality for discovering other peers in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the same group, for creating new groups, and for joining a specific group or leaving it.

AcquaintanceData

The AcquaintanceData class is a simple data-structure that stores reference to the acquaintance objects. The acquaintance objects can be added to AcquaintanceData or already contained acquaintance objects can be removed from the data-structure. The AcquaintanceData class is a singleton created by AcquaintanceDataFactory. The purpose to create the AcquaintanceData to be a singleton is that each peer contains only one instance of the acquaintance data-structure that is manipulated by multiple objects.

BiDiPipeAcquaintance

BiDiPipeAcquaintance implements Acquaintance interface, so this class defines the core behavior of an acquaintance. For each acquaintance object, it stores information such as peer group, reference to the HyperionPeer object which created the acquaintance, acquainted peer name, acquainted peer id, and reference to the dispatcher as BiDiPipeDispatcher. When there is a message to send to acquainted peer, the Web service required by the message is discovered and the service is invoked.

The explicit handling of pipes is not required in the presence of the Web services. Also there is no need of dispatcher in the peer.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

AcquaintanceService

The AcquaintanceService is a class that is published as a Web service in the peer network. It provides implementation of the requestAcquaintance, responseAcquaintance, releaseAcquaintance, and respRelease methods according to the WSDL definition published in the peer network. The AcquaintanceService holds a service descriptor of the type ServiceDescriptor. The descriptor is initialized by the name of the service class, service name published, service description string, peer group ID, peer group name, and peer group description. The service descriptor is used by the HyperionPeer class to initialize the Web service.

QueryService

The QueryService is published on demand when there is a query request by the peer. QueryService controls the query processing of the peer where no dispatcher is required. Peers interact with each other via the QueryService to send and receive queries and query results.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

Performance and Architectural Analysis

The performance analysis of the proposed architectures is based on three dimensions. First of all, we perform an architectural analysis for the conventional PDBMS architecture versus OSAP I and OSAP II architectures. Second, we give a detailed comparison analysis of the characteristics of each architecture. Third we present the performance of each one of the architectures provided; we conduct experiments to compare according to the performance metrics.

5.1 Architectural analysis

We proposed two approaches to service oriented peer DBMS architecture, the first approach is very simple and similar to the conventional peer DBMS architecture whereas the second approach uses Service Oriented Architecture to define peer DBMS.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.1.1 Conventional PDBMS vs. OSAP I

The OSAP I architecture is very simple and similar to the conventional PDBMS architecture. The main difference in these two architectures is that the former provides a Web service and SOAP communication over P2P network. Rather than having a PeerManager process in the conventional PDBMS, OSAP I provides PeerManagerService as Web service. Peers communicate with each other using the Web service via SOAP message calls.

5.1.2 Conventional PDBMS vs. OSAP II

The OSAP II architecture takes full advantage of the Service Oriented Architecture. The conventional PDBMS architecture uses a three-tier architecture with a presentation layer, a business logic layer and a database access layer. OSAP II architecture uses the first and last layer of the three-tier architecture with embedding of SOA into the second layer.

SO Layer in OSAP II

The Service Oriented (SO) layer in the OSAP II provides more than a set of technologies; it is an application architecture within which all functions are defined as independent services with well-defined invokable interfaces [4]. The services are independent services that are publishable, discoverable, and can be invoked over the network using the open standards. Some examples of services in the PDBMS are Acquaintance service, Query service, Data-coordination service, etc. respecting the properties of PDBMS Acquaintance service is provided to all peers in the network

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

whereas the rest of the services are provided to the acquainted peers only. Each peer may offer a variety of independent services that act as black boxes.

The idea of the service-oriented layer in the OSAP II will enhance interoperability due to use of open standards by the services. In terms of technology Java has contributed platform-neutral programming, and XML has contributed self-describing as well as platform-neutral, data. Now, Web services have removed another difficulty by allowing the interconnection of applications in an object-model neutral way. By using an XML-based messaging scheme, Java applications can invoke DCOM (Distributed Common Object Model) based or CORBA (Common Object Request Broker Architecture) compliant applications. The best part of this is that the invoking application does not need to have an idea where the transaction will run, what language it is running in, or what route the message may take [4].

5.2 Characteristics of the Proposed Architectures

As mentioned above the main difference between the OSAP I architecture and the conventional PDBMS lies in the usage of Web service for the PeerManager, whereas the main difference between the latter and OSAP II is that they share the presentation and data access layer of the three-tier architecture. This section characterizes the benefits obtained using the SOA in the OSAP II architecture.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2.1 Maintainability

According to the definition in [17] maintainability is "the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment". SOA provides the ability to have independent Web services through which peers interact with each other directly, this allows maintenance of the system distributed in terms of the services. The services can be improved and optimized independently from each other provided loose coupling from the service orientation.

5.2.2 Flexibility

Flexibility is defined in [17] as "the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed". The OSAP II architecture is very flexible and it inherits this property from the SOA. Peers can incorporate different types of Web services into the system which was not possible with the conventional PDBMS because to integrate a service in the conventional PDBMS, the service has to be published and used using JXTA mechanisms.

5.2.3 Reusability

Reusability is defined in [17] as "the degree to which a software module or other work product can be used in more than one computing program or software system". The idea behind the SOA is to create services from the existing software components

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and allow and increase reusability of such components and services. The OSAP II architecture inherits this important property from the SOA to integrate existing software services in the peer system or services from the peer system in other software.

5.2.4 Scalability

The given definition of scalability in [17] is "the ease with which a system or component can be modified to fit the problem area". Data sharing and coordination is the most important motivation for PDBMS. There is no limit defined on the number of peers in the peer network and number of entities with which a peer can share and coordinate data with. The OSAP II architecture provides scalability and response time better than the traditional PDBMS (See experiments results).

5.2.5 Fault Tolerance

The OSAP II architecture will provide fault tolerance better than the tradition peer DBMSs. In traditional PDBMSs PeerManager is the gateway for the system through which requests and responses arrive to the peer. This centralized approach lacks fault tolerance and is prone to a bottleneck effect. Problems in the PeerManager would be propagated to the whole peer because PeerManager is the entry point for requests and exit point for responses. OSAP II architecture is more fault tolerant than the OSAP I in terms of providing services. Unless there is no logical connection between the Web services if one of the service is not working it will not impact the other service from running. For example if acquaintance Web service of a peer stops functioning, the peer still can receive query and send results to acquainted peers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.3 Performance Analysis

In this section we present performance comparison of three systems, conventional Hyperion (Hyp), implementation of OSAP I architecture in Hyperion framework (SO/Hyp1) and the implementation of OSAP II architecture in Hyperion framework (SO/Hyp2). Same experiments are designed and performed on each of the three systems to compare their performances. The performance measurements used in the experiments are the time to execute queries on peers with different positions in the network, number of acquaintances, system load and scalability. Query execution process for Hyperion system has been provided in [12], and all three implementations use the same query processor. Experiments are performed on IBM computers equipped with Windows XP operating systems, 3.0 GHz CPU speed Pentium 4 and 756MB of RAM. The query execution time in the graphs is average time of running the same experiment five times.

The experiments performed are using the peer configuration shown in Figure 5.1, of airline example used in section 1.1. The names of airline peers are United Airlines (UA), Air Canada (AC), Air France (AF), Dutch Airlines (KLM), Lufthansa (LH) and Alitalia Flights (AZ). In this figure a connection between two peers shows an acquaintance between them. Figure 5.2 gives the schema of all the six peers. Attributes *fno* denotes flight number, *date* and *deptime* denotes date and departure time of the flights. Attribute *arrtime* represents the arrival time of flights.

Figure 5.3 gives an example of database instance of airlines UA and LH and the mapping table between UA and LH that is used to exchange data between both peers during query processing. The table (c) is the mapping table that is used to translate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

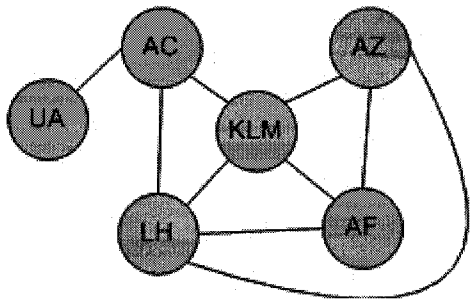


Figure 5.1: The P2P Network

LH(Fno, Date, Time, Dest)
UA(Flight, Dt, Tm, To)
AC(fno, date, deptime, dest)
AF(fno, date, deptime, dest, arrtime)
AZ(fno, date, deptime, dest, arrtime)
KLM(fno, date, deptime, dest, arrtime)

Figure 5.2: Database Schema of Peers

the flight number (FNo) of LH to flight number (Flight) of UA and vice versa. The mapping table (d) maps the date of LH directly to the date of UA. And the mapping table (e) translates the destination (dest) of LH table to the destination (To) of UA table and vice versa.

5.3.1 Experiment 1

The first experiment uses the P2P network topology as in Figure 5.1. In this experiment we executed a single query from each of the peers one after the other to record query execution time. The purpose of this experiment is to detect query execution time of each peer with its position in the network along with the number of acquaintances for both Hyperion and SO/Hyperion.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Fno	Date	Time	Dest
LH401	11/05	10:00	S.F.
LH541	11/15	00:30	N.Y.
LH690	11/21	12:00	L.A.
LH691	11/22	12:00	L.A.

(a) Table LH

Flight	Dt	Tm	To
UA120	11/15	10:30	JFK
UA134	11/15	11:00	LAX
UA135	11/21	12:00	ONT
UA141	11/22	07:00	ORD

(b) Table UA

Fno		Flight
LH541		UA120
LH690		UA135

(c) Mapping table fno2flight

Date		Dt
X		X

(d) Mapping table date2dt

Dest		To
N.Y.		JFK
N.Y.		LGA
N.Y.		EWR
L.A.		LAX
L.A.		ONT

(e) Mapping table dest2to

Figure 5.3: Database instances and Mapping tables

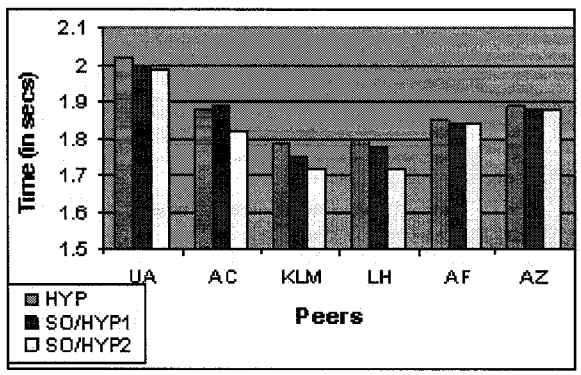


Figure 5.4: Experiment 1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

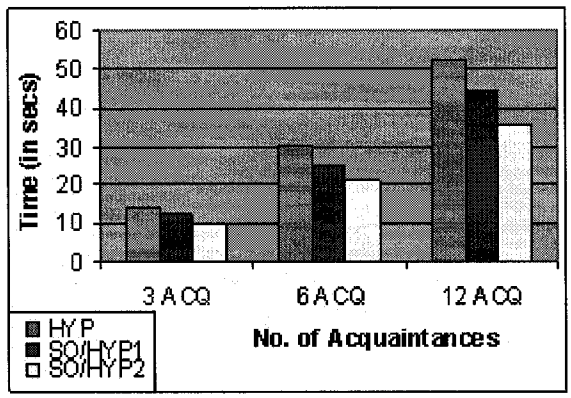


Figure 5.5: Experiment 2

The experiment results are shown in Figure 5.4. It is shown in the graph that the number of acquaintances of the peer affects time to execute a query. For example peer UA has only one acquaintance with peer AC, and a query started at peer UA will go through the network only through the peer AC. UA peer has the maximum amount of time it takes to execute the query. Whereas peer KLM has the maximum number of acquaintances and the time to execute a query from KLM is the minimum.

The experiment is performed for the three systems Hyperion, SO/Hyp1, and SO/Hyp2. The graph in Figure 5.4 shows that there is no considerable difference in the query execution time in the three systems. The reason is that the SO/Hyp1, and the SO/Hyp2 is expected to perform better when there is high number of acquaintances and queries at a peer. In a situation where there is up to four acquaintances for each peer and a single query execution, there is no speedup and performance.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

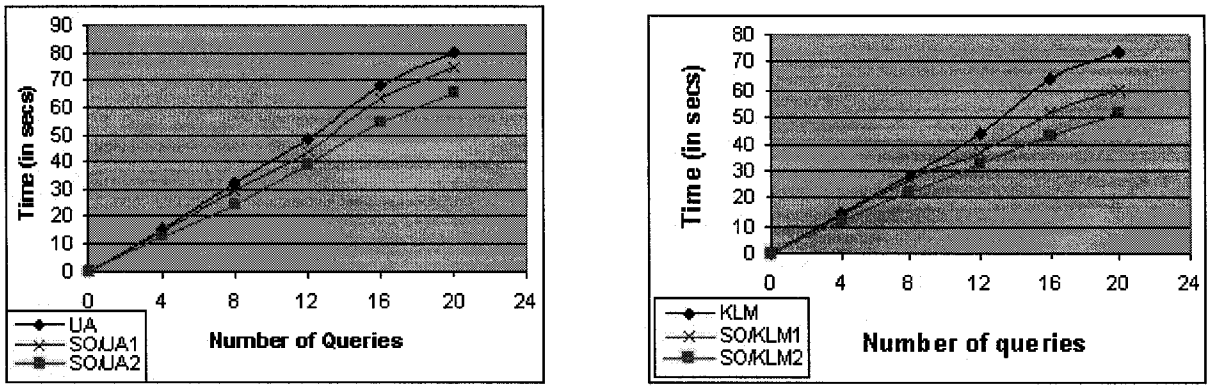
5.3.2 Experiment 2

In this experiment three cases of the same peer (say p1) are considered, where p1 is acquainted with 3 peers, 6 peers and 12 peers. The acquainted peers of p1 are also acquainted with each other or other peers in the network. Other than peer p1 on average peers are acquainted with 3 other peers. In each case all the acquainted peers of p1 send a query at the same time, so the peer p1 receives 3, 6 and 12 queries at the same from the acquainted peers. The time is recorded for the total query execution time of 3, 6 and 12 queries at peer p1. The graph in Figure 5.5 shows that in the first case where the peer p1 is acquainted with three peers there is slight difference in query execution time in Hyperion, SO/Hyp1, and the SO/Hyp2 systems. The difference increases as the number of acquaintances for peer p1 increases from 3 to 6 and 12. The reason for this performance achievement is that the Hyperion system uses pipe listeners for each acquaintance, also a separate dispatcher is associated with each pipe of the acquaintance. Whereas this overhead is not present in the SO/Hyp1 system. The SO/Hyp2 eliminates need of dispatcher with independent query Web service which provides more performance. So when there is high load in the peer slight overhead can make much difference in the overall performance.

5.3.3 Experiment 3

Another experiment is performed to see how peers behave when the system is fully loaded with queries from every peer in the network. In this experiment six peer network is used as shown in Figure 5.1. Every peer starts four queries per second at the same time which are propagated in the network. The graphs for peer UA and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Experiment 3.1

(b) Experiment 3.2

Figure 5.6: Experiment 3

KLM are shown in Figure 5.6. UA and KLM peers are chosen because UA only has one acquaintance whereas KLM has the maximum number of acquaintances in the network. So these two peers show different behaviors.

The graph in Figure 5.6 shows the results for peer UA for the three systems under considerations. Time is recorded every time four queries are executed in the peer. The graph shows that time to execute queries for both curves is very close to each other from 4 to 12 queries. But the starts varying from 12 to 20 queries. The reason for this variance is that SO/Hyperion systems (SO/Hyp1 and SO/Hyp2) tend to do better than Hyperion when the system load is high.

The second graph in the Figure 5.6 shows the results for KLM peer. The timing results for KLM peer are better than UA peer, the performance achievement is more from 8 to 20 queries than the UA peer. KLM peer has the maximum number of acquaintances and it acts as a hub in the network. So the KLM peer finishes query execution before UA peer as it has only one link to AC peer.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

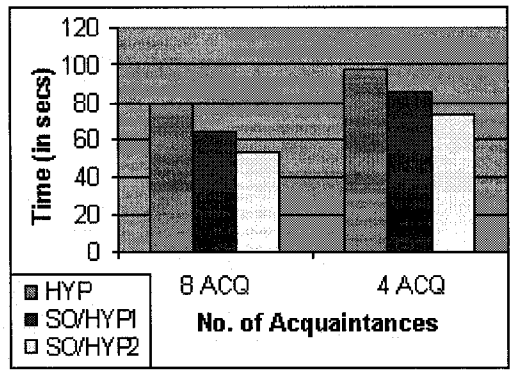


Figure 5.7: Experiment 4

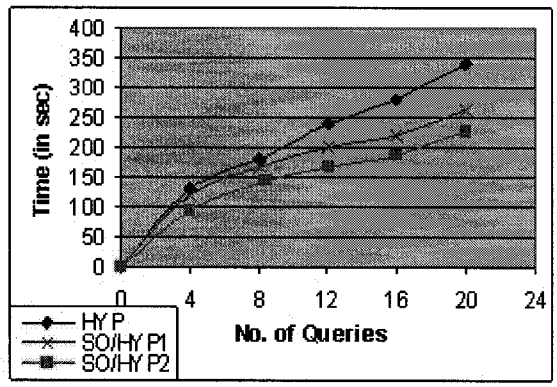
5.3.4 Experiment 4

The last two experiments are performed on a large system of size 80 peers. Two types of peers are considered in this experiment, one with four acquaintances and the other with eight acquaintances. In this system 10 peers have four acquaintances, other 10 peers are acquainted with eight peers and the rest of the peers are acquainted with 6 peers on average. The time in the Figure 5.7 is the average query execution time from each peer with least number of acquaintances and maximum number of acquaintances. The result is consistent with previous experiments that is peers with more acquaintances have less query execution time than the peers with less number of acquaintances.

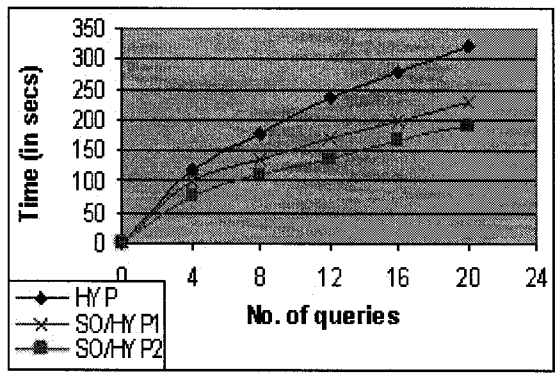
5.3.5 Experiment 5

The last experiment is performed to see how peers behave when the system is fully loaded with queries from every peer in a large network. Every peer starts four queries per second at the same time which are propagated in the network. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



(a) Experiment 5.1



(b) Experiment 5.2

Figure 5.8: Experiment 5

average query execution time of each 10 peers (acquainted with 4 peers) and other 10 peers (acquainted with 8 peers) is given in the graphs.

In the Figure 5.8 the average query execution time is given of peers acquainted with four other peers in the network for Hyperion and SO/Hyperion systems.

The graph shows that the performance achievement of SO/Hyp2 is much better when the system load is high from 12 to 20 queries. There is high variance in the timing between the Hyperion and SO/Hyperion systems when the peers have high number of queries (see Figure 5.8). The results show that in a large peer network the performance achievement of the SO/Hyperions over the Hyperion system is greater than in smaller network. This is because in high system load SO/Hyperions prevent the overhead incurred in the Hyperion system as described above.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 6

Related Work

Several research directions have emerged with respect to data management issues related to P2P computing [1, 13, 22, 27, 40, 33, 36, 30]. In [1], a suitable data model for relational peer database applications is proposed. This work provides a logical architecture, which has an underlying logical data model. It was the first paper to introduce the view that peers are local relational databases which establish acquaintances with each other to build the P2P network topology. Each acquaintance is characterized both by a mapping (the paper did not say which) between the peer databases involved and by a first order theory defining the semantic dependencies between the peer databases. The vision in [13] is about the problem of selecting views to materialize and then distributing these views and original data among the peers. However, contrary to [1], the authors did not discuss any architecture alternatives in their papers.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

PeerDB [33], the first implementation of a PDBMS, is a P2P data sharing system built on top of a generic and self-configurable P2P framework. It is a PDBMS that supports fine-grained and content-based searching, integrates mobile agents, and provides a frontend for searching schemaless data. The Piazza project [40] has adapted the popular Global-As-View (GAV) and Local-As-View (LAV) schema mapping techniques ([24]) to the P2P context to allow distributed query processing across heterogeneous schemas. In Piazza, each peer database schema is a GAV/LAV view of the remaining P2P network of database schemas. Query processing now reduces to a special case of answering queries using views. Though they provide architectures for their respective systems, the authors of [33] and [40] did not discuss nor experiment with any architecture alternatives yet.

The work reported in [3] describes a framework for providing a P2P-oriented database access and integration on the Grid infrastructure using the OGSA-DAI standard. We are much interested in offering a framework, not for data integration, but for data coordination in a context where each peer database does not need to constrain itself in order to share data with acquainted peers.

Major database and information systems companies have clearly identified the need for either Grid or P2P data access. For example, IBM and Oracle are working with National e-Science Center in the United Kingdom on the development of the OGSA-DAI framework. Also, IBM-Italy teamed up with the University of Rome to develop a framework for P2P data integration on the Grid [3]. However, these efforts are centered around the Grid infrastructure. There seems therefore to be a clear need for a similar development for a service-oriented infrastructure that is purely centered

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

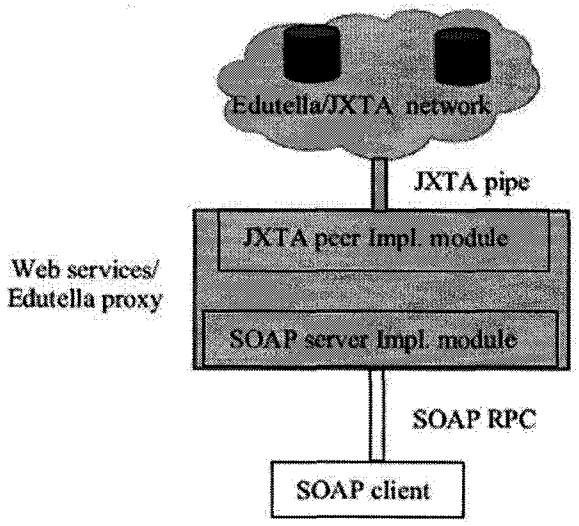


Figure 6.1: Exposing existing Edutella/JXTA P2P services as Web Services [30]

around the concept of P2P, which though could offer a bridge to the Grid world.

6.1 Edutella Peer-to-Peer Network

The goal of Edutella project [36, 30] is to design and implement a schema-based P2P infrastructure for Semantic Web. It relies on the W3C metadata standards, RDF and RDF schema to describe distributed resources and use JXTA framework for P2P. RDF and RDFS are used to denote for resources on the Web and provide way by which computer systems can exchange data. The main building blocks of Edutella project are schema language, query language and a network topology. RDFS is used to represent schema based on classes, properties and property constraints to define vocabulary used to describe resources. As expressive query formalism is required than a simple key and keyword-based queries, a rule language based on Datalog

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(or Horn logic in general) is used for Edutella, RDF-QEL (Resource Description Framework-Query Execution Language). Edutella peers are connected to network using a wrapper-based architecture, where the wrapper is responsible for translating local queries into the Edutella common query model. Edutella takes the step to organize peers in a setting so that the network search is efficient. Super-peer network topology, clustering and routing using indices are the techniques considered for making the query execution efficient [36].

Edutella other than using JXTA to provide peer services, it also investigates the interaction between Edutella/JXTA and Web services to exchange distributed functionalities between the two platforms. Two scenarios have been considered to achieve this first; expose existing Edutella/JXTA P2P services as Web services and second; integrate Web service enabled content providers into Edutella JXTA. Edutella uses proxies to achieve the service layer interaction between Edutella/JXTA and Web services [30].

Edutella/JXTA and Web services are different in three ways as; first Edutella/JXTA and Web services use different entity identification systems as well as searching and locating entity functionalities. Second Edutella/JXTA and Web services use different transport and message protocols to communicate with service provider and client. In P2P setting clients communicate with each other using JXTA protocols through JXTA pipes, but Web service use HTTP for transport. Third Edutella/JXTA and Web services use different formats to describe distributed services, a JXTA service is advertised, discovered and used using JXTA protocols. Whereas a Web service is described, published, discovered and requested using WSDL, UDDI and SOAP. Fourth

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

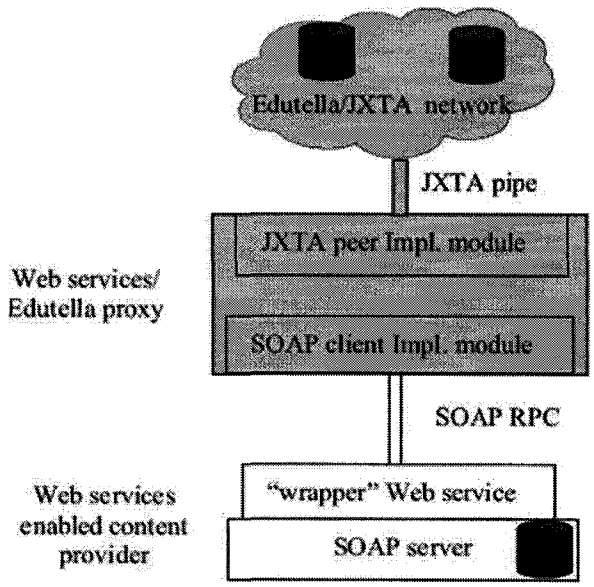


Figure 6.2: Integrating Web Services enabled content providers into Edutella/JXTA [30]

the security models of both Edutella/JXTA and Web services are incompatible with each other [30]. To overcome the challenges described above Edutella used Web service/Edutella proxies, which is an Edutella/JXTA peer that also contains a SOAP implementation responsible for mediating the SOAP RPC between Edutella/JXTA and Web services.

The first scenario is to expose existing Edutella/JXTA P2P services as Web services is implemented by providing Edutella/JXTA proxy for each JXTA service that is to be provided as Web service. JXTA services are implemented as before, that can be advertised, discovered and used using JXTA pipes. This JXTA service can be exposed to Web service clients using Web service/Edutella proxy which is a usual JXTA peer that and has ability to search, locate and interact with the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Edutella query service. Besides this the Web services/Edutella proxy also runs a SOAP server responsible for exposing the functionality of the Edutella query service. Web service/Edutella proxy is described by the WSDL version of the JXTA service advertisement so requests from the client to SOAP server are translated by the proxy and forwarded to JXTA query service. This is depicted in Figure 6.1 in detail.

The second scenario is to integrate Web service enabled content providers into Edutella JXTA. The Edutella/JXTA P2P network has the ability to only contain the content providers that are JXTA peers, so to accommodate heterogeneous content providers it adopts a wrapper-like architecture that constraint the content providers to strictly follow the content provider integration protocol called Edutella Common Data Model (ECDM). This scenario is also implemented using Web services/Edutella proxy that is a JXTA peer holding the implementation of JXTA as well as implementation module for running SOAP server. This proxy is a wrapper Web service having contents to share with the Edutella P2P network. The request to this Web service is RDF-QEL query and response as SOAP RPC message. After receiving a query it has to translate it to the local query of the underlying content repository, and then transform the local query result into the RDF-QEL query result. The implementation detail is depicted in the Figure 6.2.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 7

Conclusion and Future work

7.1 Conclusion

The work reported here presented two versions of an Open Service Architecture for PDBMSs. Such architectures are motivated by the need for closing a gap similar to one that the related field of grid computing has already filled with its OGSA-DAI architecture for data access on the grid; it is also motivated by the need for future integration of schema and instance level data mappings in a common framework. The first version, OSAP I, confines all the services offered by a peer in a PDBMS in private processes that other peers can access only through a peer manager Web service. The second version, OSAP II, allows loose coupling and decentralization of services. The OSAP II version offers the main services of the PDBMS as web services that are invoked via the network using well-defined interfaces. We have implemented both OSAP I and OSAP II architectures within the Hyperion PDBMS

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

framework and ran experiments. We compared both architectures on the grounds of extensibility, flexibility, ease of use and performance efficiency.

7.2 Future Work

In the future, we plan to extend the proposed architectures and refine the basic services offered such as Acquaintance and Query Web services.

In terms of Acquaintance Web service, we plan to bring closer the two different directions that exists concerning the handling of heterogeneity by clearly defining the interface for data sharing. On one hand there is the Piazza system [40] which uses a rich mapping language, GLAV (Global-Local-As-View), to syntactically express the heterogeneity gap between peer databases. On the other hand, Hyperion uses mapping tables to semantically express the heterogeneity. The data access for the Query Web service will also be modified if changes takes place in the Acquaintance service, but this modification will not change the way how Query Web service is defined and used by the acquainted peers.

We also plan to integrate more services into the framework such as data coordination and notification services.

Other than refining the OSAP architectures we also plan to have a well defined interface to publish the peer Web services as well as use the Grid Services of the OGSA-DAI middleware. This would fulfill the gap between the two growing data sharing approaches such as P2P database systems and Grid systems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Bibliography

[1] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, and J. Mylopulos. Data Management for Peer-to-Peer Computing: A vision. In *WebDB*, 2002.

[2] M. Boyd, S. Kittivoravitkul, C. Lazanitis, P. McBrien, N. Rizopoulos. AutoMed: A BAV Data Integration System for Heterogeneous Data Sources. In *CAiSE*, 2004

[3] D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, and G. Vetere Hyper: A Framework for Peer-to-Peer Data Integration on Grids. In Proc. of *ICSNW* 2004.

[4] K. Channabasavaiah, K. Holley, E. Tuggl. IBM Software Group. Migrating to a service-oriented architecture. 16 Dec 2003

[5] R. Domenig, K.R. Dittrich. Query Exploratiweness for Integrated Search in Heterogeneous Data Sources. In *CAiSE*, 2002

[6] A. Elmagarmid and M. Rusinkiewicz and A. Sheth. Management of Heterogeneous and Autonomous Database Systems. Morgan Kaufmann Publishers, 1999.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[7] I. Foster and C. Kesselman The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 2004

[8] I. Foster, C. Kesselman, J. M. Nick and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.

[9] I. Foster, C. Kesselman and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations.

[10] Grid Data Service. World Wide Web URL: *http://www.ogsadai.org.uk/docs/R4.0/doc/GDS.html.*

[11] Grid Data Service Factory. World Wide Web URL: *http://www.ogsadai.org.uk/docs/R4.0/doc/GDSF.html.*

[12] P. Gianolli, M. Garzetti, L. Jiang, A. Kementsietsidis, I. Kiringa, M. Masud, R. Miller, J. Mylopoulos Data Sharing in the Hyperion Peer Database System In Proc. of *VLDB* 2005.

[13] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? In *Proceedings of the First Workshop on Databases and the Web*, 2001.

[14] Gnutella. World Wide Web URL: *http://www.gnutelliums.com*

[15] F. Giunchiglia and I. Zaihrayeu. Making Peer Databases Interact-A vision for an Architecture Supporting Data Coordination. In *CIA*, 2002.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[16] Hyperion Project. World Wide Web URL: *http://www.cs.toronto.edu/db/hyperion/.*

[17] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

[18] JXTA. World Wide Web URL: *http://www.jxta.org.*

[19] JXTA-SOAP. World Wide Web URL: *http://www.soap.jxta.org.*

[20] Kazaa. World Wide Web URL: *http://www.kazaa.com*

[21] A. Kementsietsidis and M. Arenas. Data Sharing Through Query Translation in Autonomous Sources. In *VLDB*, pages 468-479, 2004.

[22] A. Kementsietsidis, M. Arenas, and R.J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *SIGMOD*, 2003.

[23] V. Kantere, I. Kiringa, and J. Mylopoulos. Coordinating Peer Databases Using ECA Rules. In *P2P DBIS*, pages 108-122, September 2003.

[24] M. Lenzerini. Data integration: a theoretical perspective. In *PODS*, 2002.

[25] LITWIN, W., et al. 1982. SIRIUS Systems for Ditstributed Data Management, H. J. Schneider, Ed. North-Holland Publishing, The Netherlands, pp. 311-366.

[26] W. Litwin, L. Mark, N. Roussopoulos Interoperability of Multiple Autonomous Databases *ACM Ccompting Surveys* Vol. 22, No. 3, September 1990.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[27] M. Arenas, V. Kantere, A. Kementsietsidis, and I. Kiringa. The hyperion project: From Data Integration to Data Coordination. In *ACM SIGMOD RECORD*, 2003.

[28] Napster. World Wide Web URL: *http://www.napster.com*

[29] W. S. Ng, B. C. Ooi, K. L. Tan, and A. Y. Zhou. PeerDB:A P2P-Based System for Distributed Data Sharing. In *Data Engineering*, 2003.

[30] W. Nejdl, W. Siberski, and M. Sintek. Interacting the Edutella/JXTA Peer-to-Peer Network with Web Services. In Proc. of *SAINT* 2004.

[31] OGSA. World Wide Web URL: *http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf.*

[32] OGSA-DAI Overview. World Wide Web URL: *http://www.ogsadai.org.uk/docs/R4.0/doc/DAIOverview.html.*

[33] B.C. Ooi, Y. Shu, and K.L. Tan. Relational data sharing in peer-based data management systems. *SIGMOD Record*, 32(3):59–64, 2003.

[34] M.T. Ozsu and P. Valduriez. Principles of Distributed Database Systems. Prentice Hall, Upper Saddle River, 2nd Edition, 1999.

[35] N. Paton, M. Atkinson, V. Dialani, D. Pearson, T. Storey, and P. Watson. Data Access and Integration Services on the Grid. Tech ReportUKeS-2002-03, UK National e-Science Center, 2002.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[36] C. Qu and W. Nejdl. Design Issues and Challenges for RDF- and Schema-Based Peer-to-Peer Systems. In *SIGMOD* Records (2003).

[37] A. P. Sheth, J. A. Larson Federated Database Systems for Managing Distributed, Heterogenous, and Autonomous Databases *ACM Ccompting Surveys* Vol. 22, No. 3, September 1990.

[38] Service Oriented Architecture. World Wide Web URL: *http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html.*

[39] Systinet Corporation. Web Services: A Practical Introduction to SOAP Web Services. Oct 2003.

[40] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, and X. Dong. The Piazza Peer Data Management Project. In *ICDE*, 2003.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix A

Web Service Definition Languages

A.1 Peer Manager Service Description

```
<?xml version="1.0" encoding="UTF-8"?> <wsdl:definitions
targetNamespace="http://DefaultNamespace"
 xmlns:impl="http://DefaultNamespace"
 xmlns:intf="http://DefaultNamespace"
 xmlns:apachesoap="http://xml.apache.org/xml-soap"
 xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
   <wsdl:message name="requestAcquaintanceRequest">
```


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="responseAcquaintanceRequest">
    <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="sendMessageRequest">
    <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="releaseAcquaintanceRequest">
    <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="respReleaseRequest">
    <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>
<wsdl:portType name="PeerManagerService">
    <wsdl:operation
    name="requestAcquaintance" parameterOrder="in0">
        <wsdl:input name="requestAcquaintanceRequest"
        message="impl:requestAcquaintanceRequest"/>
    </wsdl:operation>
 <wsdl:operation name="responseAcquaintance" parameterOrder="in0">
        <wsdl:input name="responseAcquaintanceRequest"
        message="impl:responseAcquaintanceRequest"/>
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    </wsdl:operation>
  <wsdl:operation name="sendMessage" parameterOrder="in0">
      <wsdl:input name="sendMessageRequest"
      message="impl:sendMessageRequest"/>
    </wsdl:operation>

  <wsdl:operation name="releaseAcquaintance" parameterOrder="in0">
      <wsdl:input name="releaseAcquaintanceRequest"
      message="impl:releaseAcquaintanceRequest"/>
    </wsdl:operation>
      <wsdl:input name="respReleaseRequest"
      message="impl:respReleaseRequest"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="PeerManagerServiceSoapBinding"
type="impl:PeerManagerService">
    <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="requestAcquaintance">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="requestAcquaintanceRequest">
        <wsdlsoap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://DefaultNamespace"/>
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        </wsdl:input>
     </wsdl:operation>
    <wsdl:operation name="responseAcquaintance">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="responseAcquaintanceRequest">
           <wsdlsoap:body use="encoded"
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
           namespace="http://DefaultNamespace"/>
        </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="sendMessage">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="sendMessageRequest">
           <wsdlsoap:body use="encoded"
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
           namespace="http://DefaultNamespace"/>
        </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="releaseAcquaintance">
        <wsdlsoap:operation soapAction=""/>

        <wsdl:input name="releaseAcquaintanceRequest">
           <wsdlsoap:body use="encoded"
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
            namespace="http://DefaultNamespace"/>
        </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="respReleaseAcquaintanceRequest">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="respReleaseAcquaintanceRequest">
         <wsdlsoap:body use="encoded"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="http://DefaultNamespace"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="PeerManagerServiceService">
     <wsdl:port
     name="PeerManagerService"
     binding="impl:PeerManagerServiceSoapBinding">
        <wsdlsoap:address location=
        "http://localhost:8080/axis/services/PeerManagerService"/>
     </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A.2 Acquaintance Service Description

```
<?xml version="1.0" encoding="UTF-8"?> <wsdl:definitions
targetNamespace="http://DefaultNamespace"
xmlns:impl="http://DefaultNamespace"
xmlns:intf="http://DefaultNamespace"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
   <wsdl:message name="requestAcquaintanceRequest">
      <wsdl:part name="in0" type="xsd:string"/>
   </wsdl:message>
   <wsdl:message name="responseAcquaintanceRequest">
      <wsdl:part name="in0" type="xsd:string"/>
   </wsdl:message>
   <wsdl:message name="releaseAcquaintanceRequest">
      <wsdl:part name="in0" type="xsd:string"/>
   </wsdl:message>
   <wsdl:message name="respReleaseAcquaintanceRequest">
      <wsdl:part name="in0" type="xsd:string"/>
   </wsdl:message>
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
<wsdl:portType name="AcquaintanceService">
   <wsdl:operation name="requestAcquaintance" parameterOrder="in0">
      <wsdl:input name="requestAcquaintanceRequest"
      message="impl:requestAcquaintanceRequest"/>
   </wsdl:operation>
 <wsdl:operation name="responseAcquaintance" parameterOrder="in0">
      <wsdl:input name="responseAcquaintanceRequest"
      message="impl:responseAcquaintanceRequest"/>
   </wsdl:operation>
 <wsdl:operation name="releaseAcquaintance" parameterOrder="in0">
      <wsdl:input name="releaseAcquaintanceRequest"
      message="impl:releaseAcquaintanceRequest"/>
   </wsdl:operation>
 <wsdl:operation name="respReleaseAcquaintance" parameterOrder="in0">
      <wsdl:input name="respReleaseAcquaintanceRequest"
      message="impl:respReleaseAcquaintanceRequest"/>

   </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="AcquaintanceServiceSoapBinding"
type="impl:AcquaintanceService">
   <wsdlsoap:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="requestAcquaintance">
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="requestAcquaintanceRequest">
            <wsdlsoap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace"/>
        </wsdl:input>
        <wsdl:output name="requestAcquaintanceResponse">
            <wsdlsoap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="responseAcquaintance">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="responseAcquaintanceRequest">
            <wsdlsoap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace"/>
        </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="releaseAcquaintance">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="releaseAcquaintanceRequest">
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
            <wsdlsoap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace"/>
        </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="respReleaseAcquaintance">
        <wsdlsoap:operation soapAction=""/>

        <wsdl:input name="respReleaseAcquaintanceRequest">
            <wsdlsoap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace"/>
        </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="AcquaintanceServiceService">
     <wsdl:port name="AcquaintanceService"
     binding="impl:AcquaintanceServiceSoapBinding">
        <wsdlsoap:address location=
        "http://localhost:8080/axis/services/AcquaintanceService"/>
     </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A.3 Query Service Description

```
<?xml version="1.0" encoding="UTF-8"?> <wsdl:definitions
targetNamespace="http://DefaultNamespace"
xmlns:impl="http://DefaultNamespace"
xmlns:intf="http://DefaultNamespace"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
   <wsdl:message name="sendQueryRequest">
      <wsdl:part name="in0" type="xsd:string"/>
   </wsdl:message>
   <wsdl:message name="sendResultRequest">
      <wsdl:part name="in0" type="xsd:string"/>
   </wsdl:message>
   <wsdl:portType name="QueryService">
      <wsdl:operation name="sendQuery" parameterOrder="in0">
         <wsdl:input name="sendQueryRequest"
         message="impl:sendQueryRequest"/>
      </wsdl:operation>
      <wsdl:operation name="sendResult" parameterOrder="in0">
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
        <wsdl:input name="sendResultRequest"
        message="impl:sendResultRequest"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="QueryServiceSoapBinding"
type="impl:QueryService">
    <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="sendQuery">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="sendQueryRequest">
            <wsdlsoap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace"/>
        </wsdl:input>
    </wsdl:operation>

    <wsdl:operation name="sendResult">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="sendResultRequest">
            <wsdlsoap:body use="encoded"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://DefaultNamespace"/>
        </wsdl:input>
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="QueryServiceService">
      <wsdl:port name="QueryService"
      binding="impl:QueryServiceSoapBinding">
         <wsdlsoap:address
         location="http://localhost:8080/axis/services/QueryService"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.