UPPSALA
UNIVERSITET

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 755

Querying Data Providing Web Services

MANIVASAKAN SABESAN

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2010

ISSN 1651-6214
ISBN 978-91-554-7852-0
urn:nbn:se:uu:diva-128928

Dissertation presented at Uppsala University to be publicly examined in Room 1211, Building 1, Polacksbacken, Lägerhyddsvägen 2, Uppsala, Friday, October 8, 2010 at 13:15 for the degree of Doctor of Philosophy. The examination will be conducted in English.

Abstract
Sabesan, M. 2010. Querying Data Providing Web Services. Acta Universitatis Upsaliensis. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 755. 37 pp. Uppsala. ISBN 978-91-554-7852-0.

Web services are often used for search computing where data is retrieved from servers providing information of different kinds. Such *data providing web services* return a set of objects for a given set of parameters without any side effects. There is need to enable general and scalable search capabilities of data from data providing web services, which is the topic of this Thesis.

The Web Service MEDiator (WSMED) system automatically provides relational views of any data providing web service operations by reading the WSDL documents describing them. These views can be queried with SQL. Without any knowledge of the costs of executing specific web service operations the WSMED query processor automatically and adaptively finds an optimized parallel execution plan calling queried data providing web services.

For scalable execution of queries to data providing web services, an algebra operator *PAP* adaptively parallelizes calls in execution plans to web service operations until no significant performance improvement is measured, based on monitoring the flow from web service operations without any cost knowledge or extensive memory usage.

To comply with the Everything as a Service (XaaS) paradigm WSMED itself is implemented as a web service that provides web service operations to query and combine data from data providing web services. A web based demonstration of the WSMED web service provides general SQL queries to any data providing web service operations from a browser.

WSMED assumes that all queried data sources are available as web services. To make any data providing system into a data providing web service WSMED includes a subsystem, the *web service generator,* which generates and deploys the web service operations to access a data source. The WSMED web service itself is generated by the web service generator.

Keywords: views of web service operations, web service queries, adaptive parallelization, query optimization

Manivasakan Sabesan, Department of Information Technology, Computing Science, Box 337, Uppsala University, SE-75105 Uppsala, Sweden

ISSN 1651-6214
ISBN 978-91-554-7852-0
urn:nbn:se:uu:diva-128928 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-128928)

என் பெற்றோருக்கு இந்நூல் சமர்ப்பணம்

To my parents

List of Papers

This Thesis is based on the following papers, which are referred to in the text by their Roman numerals.

I Manivasakan Sabesan, and Tore Risch, Web Service Mediation Through Multi-level Views, International Workshop on Web Information Systems Modeling (WISM 2007), *In Proc. Workshops and Doctoral Consortium*, tapir academic press , pp 755-766, 2007.

II Manivasakan Sabesan, and Tore Risch , Adaptive Parallelization of Queries over Dependent Web Service Calls, 1st IEEE Workshop on Information & Software as Services(WISS 2009), *In Proc. 25th International Conference on Data Engineering (ICDE2009)*,IEEE Computer Society, pp 1725-1732, 2009.

III Manivasakan Sabesan, and Tore Risch, Adaptive Parallelization of Queries to Data Providing Web Service Operations, *submitted for conference publication,* 2010.

IV Manivasakan Sabesan, Tore Risch, and Feng Luan, Automated Web Service Query Service, *accepted for publication in International Journal of Web and Grid Services (IJWGS),* Inderscience, Volume 6, Number 4, 2010.

Reprints of papers I, II, and IV were made with permission from the respective publishers.

Other Related Publications

V Manivasakan Sabesan, Tore Risch, and Gihan Wikramanayake, Querying Mediated Web Services , *In Proc. 8th International Information Technology Conference (IITC 2006)*, Infotel Lanka Society Ltd, pp 39-44, 2006.

VI Manivasakan Sabesan, Querying Mediated Web Services, *Thesis for the degree of Licentiate of Philosophy in Computer Science with specialization in Database Technology*, Department of Information Technology, Uppsala University, 2007.

VII Manivasakan Sabesan, and Tore Risch, Web Service Query Service, *In Proc. 11th International Conference on Information Integration and Web-based Applications & Services (iiWAS2009)*, ACM and Austrian Computer Society, pp 692-697, 2009.

VIII Manivasakan Sabesan, and Tore Risch, Adaptive Parallelization of Queries Calling Dependent Data Providing Web Services, *In Divyakant Agrawal, K. Selcuk Candan and Wen-Syan Li (Editors): New Frontiers in Information and Software as Service,* Lecture Notes in Business Information Processing (LNBIP) series, Springer-Verlag, 2010.

Contents

Abbreviations

AMOS	Active Mediators Object System
CDM	Common Data Model
DBMS	Data Base Management System
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
PAP	Parameterized Adaptive Parallelization
RDBMS	Relational Database Management System
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
UDDI	Universal Description Discovery and Integration
URL	Uniform Resource Locator
WQL	WSMED Query Language
WSDL	Web Services Description Language
WSMED	Web Service MEDiator
XaaS	Everything as a Service
XML	Extensible Markup Language

1. Introduction

The growth of the Internet and the emergence of XML for data interchange in a loosely coupled way have increased the importance of web services [7] incorporating standards such as SOAP [18], WSDL [9], and XML Schema [42]. Web services support an application infrastructure by defining a set of *operations* that can be invoked over the communication network. Web service operations are self contained using meta-data to describe data types of their arguments and results, i.e. their signatures, using the Web Service Description Language, WSDL. Thus web services provide a general infrastructure for remote calls to predefined operations.

Web services are often used for retrieving data from servers providing information of different kinds. A *data providing web service operation* returns collections of objects for a given set of arguments without any side effects. This is known as a form of search computing [8]. However, data providing web service operations don't provide general query language or view capabilities to search and join data from one or several data providing web services, which is the topic of this Thesis.

As an example, consider a query to find information about places in some of the US states along with their zip codes and weather forecasts. Four different data providing web service operations can be used for answering this query. First the *GetAllStates* operation from the web service *GeoPlaces* [10] is called to retrieve the desired states. The *GetInfoByState* operation by *USZip* [36] returns the zip codes for a given US State. The *GetPlacesInside* operation by *Zipcodes* [11] retrieves the places located within a given zip code area. The *GetCityForecastByZip* operation by *CYDNE* [12] returns weather forecast information for a given zip code.

A mediator [39] is a system that allows data from different data sources to be combined and queried. In our setting a mediator enables queries joining data from different data providing web service operations.

In this work it is investigated how to build a general system for scalable querying of data providing web service operations. The development of a web service based mediator prototype called *WSMED (Web Service MEDiator)* is expected to provide insights into a number of research questions:

1. To what extent can web service standards, such as WSDL and SOAP, be utilized by a mediator to query data providing web service operations efficiently and scalable?

2. How can views of data providing web service operations for a high level query language such as SQL be automatically generated based on WSDL descriptions?
3. How can query optimization and rewrite techniques be used to provide efficient and scalable search from different data providing web services?
4. How can the query optimizer speed up general queries calling web service operations without knowing their costs?
5. How can data sources that are not accessible via web services be simply transformed into data providing web service operations, making them queryable by a web service mediator?
6. How can the *Everything as a Service* (XaaS) paradigm [33] be used for querying data providing web services? That is, can a web service mediator be provided as a web service and be used in a browser without any additional software installations and hardware setups?

To answer the research questions we have developed and evaluated the *WSMED* prototype, which enables high level and scalable queries over any data providing web services.

WSMED can access dynamically any web service operation by retrieving its WSDL document. WSMED contains a generic web service database for representing descriptions of any WSDL document. This database is used to dynamically construct the web service operation calls required to process a query. This provides the answer to research question **one**.

A web service operation is presented by WSMED as an SQL view. SQL queries can be expressed in terms of these views. For a given web service WSMED automatically generates such views for all its web service operations based on its WSDL definition. The views are generated using the internal *WSMED query language* (WQL), which has support for the web service data types. The automatic generation of SQL views provides the answer to research question **two**.

Web service operations are usually parameterized where input parameters have to be bound before they are called. Two web service operation calls in a query are *dependent* if one of them requires as input an output from the other one, otherwise they are *independent*. In the above example, the web service operations *GetPlacesInside* and *GetCityForecastByZip* are dependent on *GetInfoByState* but independent of each other. A challenge here is to develop methods to optimize queries containing both dependent and independent web service calls. In general such optimization depends on some unknown web service properties. Those properties are not explicitly available and depend on the network and runtime environments when and where the queries are executed. In such scenarios it is very difficult to base execution strategies on a static cost model, as is done in relational databases.

To improve the response time without a cost model, WSMED uses an approach to automatically parallelize the web service calls at run time while

keeping the dependencies among them. For each web service operation call in a query the WSMED query optimizer generates a parameterized sub-plan, called a *plan function,* which encapsulates the web service operation call and makes data transformations such as nesting, flattening, filtering, data conversions, and calls to other plan functions. WSMED will decompose the query plan to guarantee that dependent web service operations are called with proper parameter bindings.

The query performance is often improved by setting up several parameterized web service calls in parallel rather than to call the operations in sequence for different parameters. In WSMED multi-level parallel execution plans are automatically generated as process trees where different plan functions are called in parallel in different processes, called *query processes*. For adaptive parallelization of queries with web service operation calls, the algebra operator *PAP* (Parameterized Adaptive Parallelization) is implemented. *PAP* dynamically modifies a parallel plan by local monitoring of plan function calls without any cost knowledge.

The adaptive parallelization of queries calling data providing web service operations provides the answer research questions **three** and **four**.

WSMED assumes that queried data sources are available as web services. To implement a new data providing web service for a data source requires development of software to access the data source from web service operations, defining a WSDL document to describe the interface, and deploying the interface code. To simplify the implementation of data providing web services WSMED includes a subsystem, the *web service generator,* which generates and deploys the web service operations to access a data source. The programmer first defines data source interface functions to access the data source as queries by developing a wrapper in the extensible wrapper/mediator system Amos II [32]. Once the interface functions are defined the WSMED web service generator automatically generates the corresponding web service operations and dynamically deploys them without restarting the web server. The signature of each so generated web service operation is defined in an automatically generated WSDL document based on the signatures of the interface functions. The WSDL document completely describes the web service interfaces of the deployed operations. Each operation calls the interface function and sends back the result as a collection. Interface functions have been defined for many different kinds of data sources [1], e.g. relational DBMSs, semantic web data, topic maps, and CAD servers.

Automatic generation and deployment of web services for wrapped data providing systems provides an answer to research question **five**.

WSMED itself is available as a general web service to process queries over other web services, known as the *WSMED web service*. It provides web service operations to handle user sessions, import WSDL documents for web services to query, user authentications for accessed web service operations,

inspecting the schema for the generated SQL views, and executing queries over the views. The WSMED web service is generated by the web service generator. The automatically generated WSDL document *wsmed.wsdl* [41] describes the interface of the WSMED web service operations. The functionality of WSMED is demonstrated through a publicly accessible web based demonstration [40]. A JavaScript program enables the user to query any data providing web service by calling the WSMED web service operations directly from a browser without downloading any software. This shows that the WSMED web service adheres to the XaaS paradigm and provides an answer to research question **six**.

The reminder of this Thesis is organized in the following way: Section two introduces the technical background on which the research work is based. Section three explains how the papers I-VIII contribute to answering the research questions. Finally, Section four concludes and indicates future directions.

2. Background

This chapter presents the technical background of the major enabling technologies for mediating and querying web services. It briefly covers database management systems and the core technologies involved with web services.

2.1. Database Management Systems

A software system that allows creating and manipulating huge amounts of data in a structured way is known as a *Database Management System (DBMS)* [14]. A *database* is defined as the group of data managed by a DBMS. A DBMS facilitates the following:

- It allows the users to create a database and specify its structures as a *database schema* through a *Data Definition Language (DDL)*.
- It permits the users to insert, delete, update and query data from a data base through a *Data Manipulation Language (DML)*.
- It provides a security system to support multilevel authentication control.
- It preserves the consistency of data through an integrity system.
- It provides transaction and recovery control to restore the database to a previous consistent state after hardware and software failures.

To describe the data requirements of an organization in a readily understandable way by the users, a higher-level description language for schemas is required: that is known as the *data model* for the DBMS. DBMSs use different kind of data models. The most common data model is the relational data model where data is represented as tables. Central in the relational data model is the provision of a *high level query language* for efficient database search using declarative queries. The most common relational query language is the *Structured Query Language (SQL)* [14]. SQL is used in this Thesis work for querying data providing web services rather than data stored in tables.

A relational *view* is virtual relation (i.e. table) defined through a query expression. A view is not physically stored in the database but can be queried as other relations. It is sometimes possible to modify views by an insertion, deletion, or update, so called *updatable views*. In this Thesis

relational views are defined that search data from data providing web service operations.

The *Entity-Relationship (ER) model* is a graphical data model for abstract representation of database schemas. During the database design process, the database schema is represented in the ER model and then converted to the data model of the DBMS, e.g. the relational model.

In a functional data model [34] data is represented using typed functions rather than tables. This Thesis work uses the functional DBMS Amos II [32] to internally represent web service meta-data and views over web service operations.

Query processing

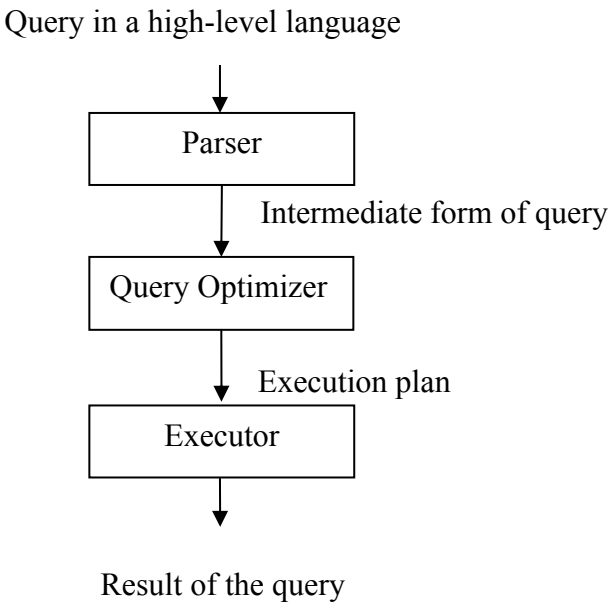

Figure 1 Query processor

Query processing (Figure 1) is the process of efficiently executing declarative queries over large databases. It transforms a declarative query into an *execution plan,* which is a program that specifies in details how the data is retrieved. The *query processor* is the group of components of a DBMS responsible for query processing. It has the following components:

- The *parser* ensures that the query syntax follows the grammar of the query language. It transforms the query into an internal intermediate form, usually a logical calculus expression.

- The *query optimizer* translates the parsed query into an execution plan, which is a program to retrieve data. The query execution plan is a program with DBMS-specific evaluation primitives such as scan operators, selection operators, various index scan operators, several join algorithms, sort operators, and a duplicate elimination operator. A query typically has many feasible execution plans, and choosing an efficient plan is named *query optimization*, which is performed by the query optimizer. The traditional query optimization is based on cost-based optimization [17]. It considers all likely execution plans and estimates the cost of each of the plans based on the number of disk blocks read, central processing unit (CPU) usage, and communication cost. Meta-data provides cost metrics. Based on this the cheapest execution plan is chosen. Typically heuristics are applied to transform the execution plan to reduce the optimization cost.
- The *executor* interprets the execution plan to produce the query result.

In this Thesis work query optimization techniques are developed for generating efficient execution plans that contain calls to web service operations.

Adaptive Query Processing

The traditional cost-based optimization strategies often expose limitations and have bad performance when the execution costs cannot be estimated precisely enough. In particular, it is not always possible to get the precise statistics about derived data collections. Furthermore, the statistics are sometimes unreliable due to dynamically changing data at runtime and work load characteristics. Therefore, adaptive query processing (AQP) techniques [13] have been developed for query optimization while the query is executing. AQP utilizes runtime feedback and modifies the query execution plan on the fly. To increase the opportunities of adaptation, special dynamic execution plan operators are introduced, such as Symmetric Hash Join [29] and Eddies [3].

In this Thesis work techniques are introduced for run time adaptive parallelization of execution plans that call expensive functions such as web service operation.

Distributed and Parallel databases

In distributed databases [30], data management is distributed over many processing nodes that are interconnected via a network. The data distribution is not visible to the end user. The database administrator provides data distribution hints to the distributed DBMS. Distributed DBMSs effectively manage distributed databases by query optimization and reliable data

management. Distributed query optimization is the process of generating an efficient execution plan for the processing of a query to a distributed database system. In this Thesis queries over distributed data providing web service operations are optimized.

Parallel DBMSs [30] is a kind of a distributed database system that runs on a cluster of processing nodes to achieve better performance through parallel execution of operators. In contrast to distributed database managements systems, data distribution is not visible to the database administrator in parallel DBMSs. Cost-based approaches, such as two-phase query optimization [19], is used in parallel database management systems to speed up queries. This Thesis work adaptively parallelizes queries calling distributed web service operations without any cost model.

2.2 Mediators

Mediators [39] are software modules used to query heterogeneous data sources. A mediator represents a virtual view or composition of views that integrate data from different data sources. Mediators don't store any data themselves and this contrasts mediation from the data warehouse [16] approach where all data is uploaded from data sources to a database. Instead, as shown in Figure 2, mediators make use of interfaces called *wrappers* to retrieve data dynamically from the data sources.

Views play a prominent role in mediation. Since the diverse sources represent the same information differently from the mediator schema, a mediator must include view definitions describing how to map the source schema into the mediator's schema. Further, the views must be able to join and convert conflicting and overlapping data from different data sources. The views are defined by means of a common data model (CDM).

The system interpreting the mediator modules is known as the *mediator engine*. The mediator engine interprets queries expressed in terms of the CDM. Performance and scalability over the amounts of data retrieved are important design aspects of mediator engines.

A *wrapper* is a software module that facilitates query processing and translation of data from a particular external data source. When a query is given to the mediator engine, it constructs the appropriate sub queries to send to the wrappers. A wrapper accepts queries from the mediator engine and translates them so they can be answered by the underlying data source. Then it returns back the result to the mediator engine. The mediator engine collects data from several wrapped data sources and post-processes them before sending back the result of the query to the user.

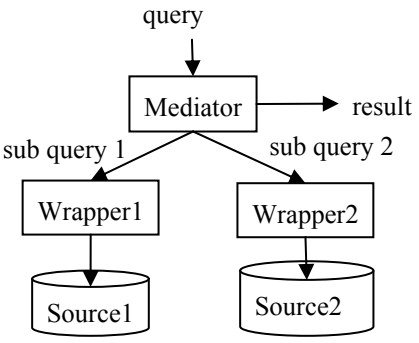

Figure 2 Mediation architecture

There are several systems such as Garlic [35], Information manifold [23], and TSIMMIS [15] using mediators for data integration from heterogeneous data sources.

This Thesis work extends the Amos II mediator engine [32] to process data from wrapped web service operations.

Capability based optimization in mediators

Wrapped data sources often limit certain attributes as inputs and produce values of other attributes as outputs, but have no general query capabilities. We say that such sources have *limited capabilities*. For example, web service operations can be seen as data sources with limited capabilities.

Capability-based query optimization [25] [43] is tailored to generate feasible plans accessing data sources with limited capabilities. Cost measures can be used to choose among the feasible plans. Source capabilities are represented and examined during the query optimization mainly in two ways:

- *Rule-based checking*: This approach is implemented in mediator systems such as Garlic [35], Information Manifold [23], and TSIMMIS [24] to match the source capabilities. Source capabilities are represented as capability records [23] or by some special description language such as Relational Query Description Language (RQDL) [37]. Complex rules are applied to find the suitable sources. During the query optimization phase rewrite rules are applied for efficient query execution.
- *Binding patterns:* Source capabilities are represented by a set of *adornments* known as *binding patterns* [16]. Matching sources are selected by analyzing the binding patterns. For example, the web query optimization system [44] and Amos II [32] utilize binding patterns to represent source capabilities. Adornments are attached with each

attribute of a data source. It is represented by an alphabet with specific meaning:

I *f (free)* - the value of the attribute need not to be specified
II *b(bound)* - the value of the attribute must be specified
III *c[L] (choice from a list L)* - the value of the attribute must be specified from the values in the list L.
IV *o[L] (optional, from the list L)* - the value of the attribute is optional, and if a value is specified it could be chosen from the list L.

f, b, and *c[L]* are the common adornments used to address the capabilities of sources that can be accessible via web services. *o[L]* is common when accessing web forms.

This Thesis work use binding patterns for defining capability limited view over web service operations.

Estimating cost metrics in the mediation environment is often difficult as the data sources are independent from the mediator. For example, with data accessible via web services the data retrieval time can vary due to congestion on the communication network or that the server providing service is highly loaded by several requests for data. Long-term observation or continuous monitoring of services [20] and adaptive query processing strategies can alleviate this. This Thesis work uses adaptive parallelization to dynamically optimize queries calling web service without using cost metrics of web service operations.

2.3 Web Services

Web services provide a message exchanging framework for applications by defining a set of *operations* that can be invoked over the communication network. Each web service operation defines a specific action performed. Web services incorporate standards such as SOAP [18], WSDL [9], XML Schema [42], HTTP [21] and UDDI [6]. A web service is described using the WSDL language. A WSDL description uses XML-Schema to describe data types of the arguments and results of operations. WSDL descriptions are published in a UDDI directory, which is a central place that holds set of web service descriptions. Any one can find required web service descriptions by querying the UDDI directory. A SOAP message is used to invoke a web service operation call by packing all the necessary details in a standard format. HTTP may be used to transfer the SOAP message to invoke a web service and return the result back.

The layered web service architecture is illustrated in Figure 3. The *discovery* layer acts as a centralized repository of web services. By querying this repository one can find a required web service based on their

descriptions. The open standard technologies UDDI and WS-Inspection [5] is used at this layer for how to publish, categorize, and search for services based.

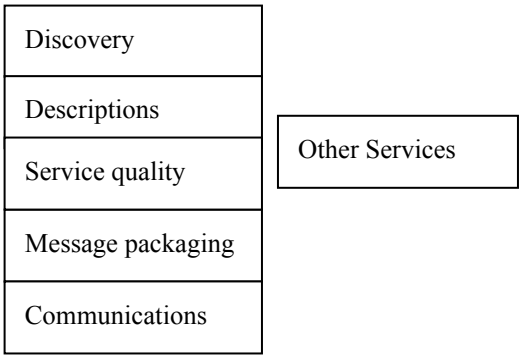

Figure 3 Web service architecture

The *descriptions* layer deals with how to represent service behavior, capabilities, and requirements in machine readable form. WSDL is used to define the functional capabilities of a service in terms of operations, service interfaces, and message types. Also it supplements deployment information such as network addresses, transport protocols, and encoding formats of the message transmission.

The *communications* layer carries the data over the network for the application. Data is converted into an internal format by the *message packaging* layer. SOAP provides a standard way for such message packaging. Then the packed message will be transported by the communications layer using internet technologies including HTTP, SMTP [26] and FTP [28].

The *service quality* layer addresses protocols that ensure the quality of the service such as security, reliable messaging, transactions, management etc. The WS-policy framework [4] declares the service quality requirements and their capabilities to enable service quality policies of web services to be attached to the different parts of a WSDL definition. Security policies for authentication, data integrity, and data confidentiality are standardized by OASIS as WS-Security policy [22]. The web service management task force [38] is tailoring the standards for web service management that involves with monitoring, controlling, and reporting of service qualities and usage.

Other service layers represent the protocols used for various purposes such as composing services to create new applications. For example, BPEL4WS [2] provides a workflow oriented composition model well suited for business applications.

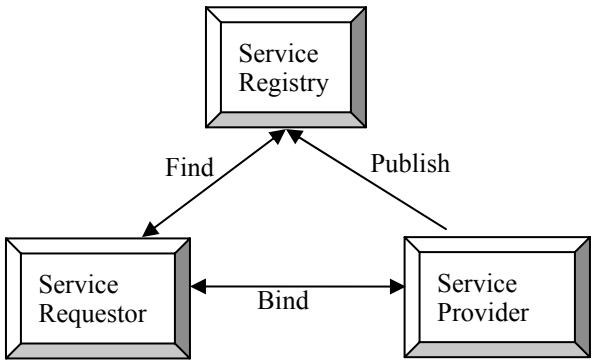

Figure 4 Service-oriented architecture

Figure 4 illustrates the interrelationship of SOAP, WSDL and UDDI in a service oriented environment. The *service provider* is responsible for generating and deploying a service. It publishes a service description using WSDL in a *service registry, UDDI*. The UDDI advertises the service and allows a *service requestor* to send queries to the registry to find a service either by name, category, identifier, or a supported specification. Once the service is found, the service requestor receives the information about the location of its WSDL document. Then the service requestor creates a SOAP message in accordance with service descriptions of the WSDL document and sends it over the network to the service provider to use the service. The *bind* operation embodies the relationship between the service requestor and the service provider.

Web Services Description Language

The functional description of a web service is defined by the XML based Web Services Description Language (WSDL). A WSDL document describes:

1. *What a service does*: The operations provided by the service and the data needed to invoke them.
2. *How a service is accessed*: Details of the data formats and protocols necessary to access the service's operations.
3. *Where a service is located*: Details of the protocol-specific network address, such as a URL.

A WSDL document defines *services* as set of network endpoints, called *ports***.** In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions. *Messages* define abstract

descriptions of the data being exchanged. *Port types* are abstract collections of *operations*. An operation defines the description of an action supported by the service. A protocol such as SOAP, HTTP, and data type specifications for a particular port type represent a *binding* for a web service operation. A *port* is defined by associating a network address with a binding. XMLSchema is used to describe message formats. WSDL allows user defined type definitions known as *extensibility elements*.

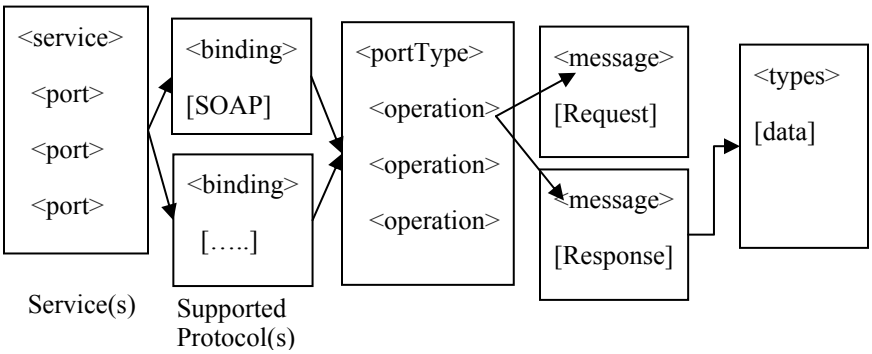

Figure 5 Document structure of WSDL

Figure 5 illustrates a simple WSDL document structure. Each service has several ports to define where it is located. In turn each port is attached to one or more bindings that describe how a web service is accessed. Each binding is attached to a *portType* having a set of operations to answer what a service is does. Request and response messages are associated with each operation to indicate the input and output of an operation.

In this Thesis work web service operations' meta-data are imported from the WSDL documents that describe the operations. Those meta-data are used to automatically define SQL views over web service operations.

SOAP

SOAP is an XML based lightweight, platform independent protocol for information exchange in a distributed environment. SOAP is used not only with HTTP but also used in combination with other protocols such as SMTP and TCP [27]. The simplicity and extensibility are the major design goals of SOAP.

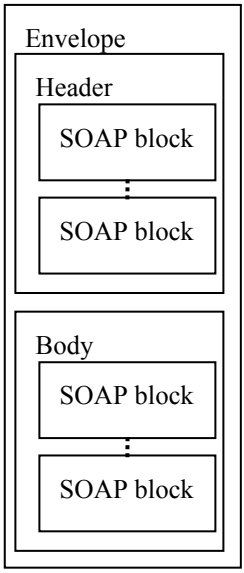

Figure 6 SOAP Message

A SOAP message (Figure 6) is made up of three elements:

1. The *SOAP Envelope* is a top element that encapsulates the other two elements representing the message.
2. The optional *SOAP header* provides a generic mechanism for adding additional features to the message such as routing and delivery setting, authentication assertions, and transaction contexts.
3. The *SOAP body* contains the actual message to be delivered and processed.

In addition to the above components a *fault* block could appear with in the body whenever there is an error to be reported to the sender of the SOAP message. The SOAP block denotes a single computational unit of data by the processor of a message.

In this Thesis work the query processor constructs SOAP calls to web service operations using the imported WSDL meta-data.

2.4 Active Mediators Object System (Amos II)

Our prototype system WSMED is based on the existing mediator engine Amos II [32]. Amos II has a functional data model as CDM. The functional query language, *AmosQL*, is the primary query language. Wrappers can be

defined to make heterogeneous data sources queryable. A wrapper performs [31] the following:

- *Schema importation* translates a sources' schema into a form compatible with the CDM of Amos II.
- *Query translation* converts AmosQL queries into API calls or query expressions executable by a source.
- *Statistics computation* estimates costs and selectivities for the calls to retrieve data from sources.
- *Proxy OID generation* constructs proxy object identifiers to describe the data from sources.

The basic concepts of the Amos II data model are *objects*, *types*, and *functions*. It is used as the CDM for the mediation and it is an extension of the Daplex [32] [34] functional data model.

Objects model all the entities in the database. Amos II has *system objects* and *user-defined objects*. Objects are represented in two ways, as *literal* or *surrogates*. Surrogates represent the real world entities such as vehicles, persons, etc; and have associated OIDs. They can be explicitly created and deleted by the users. The OIDs are maintained by the system. Literal objects are self-described system-maintained objects and do not have any explicit OIDs. For example numbers and strings. There are also *collections* of other objects: *bags*, *vectors*, and *records*. A *bag* represents unordered sets with duplicates while *vectors* denote the order-preserved collections. Vectors are accessed by the notation *v[i]* where *v* is a variable holding a vector, and *i* is the index of an element in a vector. Records are useful to manage data retrieved through web services as they often handle nested structures. Records access uses the notation *s[k]*, where *s* is a variable holding a record, and *k* is the name of an attribute in a record. Thus records are indexed by arbitrary keys while vectors are indexed by numbers only. Literals are automatically deleted by a garbage collector when they are no longer referenced.

Types: Objects are classified into *types* and each object is an *instance* of one or more types. The *extent* of a type represents the set of all instances of the type. Types are ordered into a multiple inheritances type hierarchy. A type is defined and stored in the internal database of the system with system function *create type*. For example:

```
create type Vehicle;

create type Truck under Vehicle;
```

Functions represent properties of objects, computations over objects, relationships between objects, and are used as primitives in queries and views. A function contains two parts: a *signature* and an *implementation*. The signature defines the types and names of the arguments and the result of

a function. For example, the signature modeling the attribute *color* of the type *Vehicle* would have the signature:

```
colour(Vehicle) → Charstring
```

The *implementation* defines the mapping of a function to compute results for given arguments. Further, Amos II can inversely compute arguments values of a function if the expected result value is known. The inverse usage of functions is crucial to specify general queries with function calls over the database. For example:

```
select vehichlenumber (v)
from   Vehicle v
where  colour (v) ='blue';
```

Functions can be classified according to their implementations as:

- *Stored* functions are used to represent the properties of objects stored in an Amos II database, similar to tables in a relational database.
- *Derived* functions are defined as queries in terms of other Amos II functions. They are side-effect free and they are precompiled and optimized as soon as they are defined. The queries are expressed in AmosQL, using has an SQL-like select statement for defining derived functions. Derived functions correspond to views in relational databases.
- *Foreign* functions enable low-level interfaces for wrapping external systems. For example, in this Thesis a general mechanism to call any web service operation is implemented as a foreign function named *cwo*.
- *Multi-directional functions* enable to associate several implementations of inverses for a given function. This defines functional views having different implementations depending on the actual binding pattern of its parameters. For example, a view over web services may be implemented using several web service operations as in Paper I where different operations are called depending on what parameters are known.

3. Summary of the Papers

This section summarizes how Paper I - VIII contribute to answering the research questions proposed. Paper I - IV are the main contributions.

3.1 Paper I

Paper I presents the overall architecture of WSMED and the general capabilities of WSMED for querying data accessible via web service operations. After the system has imported meta-data by reading WSDL documents for the operations to query, the user can manually define views that extract data from the results of web service operations calls. The views can be queried using SQL. In Paper I the views are manually specified as a set of declarative queries that access web service operations differently depending on what view attributes are known in a query. To enable semantic optimization of queries over the views based on automatic query transformations the user can specify key attributes of a view as a semantic enrichment. We evaluated the effectiveness of such enrichments over multi-level views of publicly available web service operations and showed that the key constraint enrichment substantially improves query performance. Paper I answers research question **one** and partially answers research questions **two, three,** and **four.** However, the optimization is based on semantic enrichments that have to be manually defined by the view definer.

3.2 Paper II

Paper II describes and evaluates strategies for adaptive parallelization of web service calls based on automatically generated SQL views of web service operations. Each generated view encapsulates a data providing web service operation for given parameters and emits the result as a flattened stream of tuples. SQL queries can be made over these views with the restriction that the input attributes must be known in the query. When joining such views it is often the case that in the execution plan the output of one web service call is the input for another, etc. The challenge addressed in Paper II is to develop methods to speed up such dependent calls by parallelization. Since web service calls incur high-latency and message set-up costs, a naïve

approach making the calls sequentially is time consuming and parallel invocations of the web service calls should improve the speed. Our approach automatically parallelizes the web service calls by starting separate query processes, each managing a plan function for different parameter values. For a given query, the query processes are automatically arranged in a multi-level process tree where plan functions are called in parallel. The parallel plan is defined in terms of an algebra operator, *First Finished Apply in Parallel* (FF_APPLYP), to ship in parallel to other query processes the same plan function for different parameters. By using FF_APPLYP we first investigated ways to set up different process trees manually. We concluded from our experiments that the best performing query execution plan is an almost balanced bushy tree. To automatically achieve the optimal process tree we modified FF_APPLYP to an operator *Adaptive First Finished Apply in Parallel* (AFF_APPLYP) that adapts the process tree locally in each query process until optimized performance is achieved. AFF_APPLYP starts with a binary process tree. During execution each query process in the tree makes local decisions to expand or shrink its process sub-tree by comparing the average time to process each incoming tuple. The query execution time obtained with AFF_APPLYP is shown to be close to the best time achieved by manually built query process trees. Paper II answered research questions **one** and **two** and partially answered research questions **three** and **four**.

3.3 Paper III

In general queries calling data providing web service operations may have both dependent and independent calls. Paper III generalizes the adaptive strategy presented in Paper II to handle both independent and dependent web service operation calls. The adaptive operator PAP speeds up queries with independent web service operation calls by calling in parallel the plan functions encapsulating each independent call. Dependent web service calls are handled by adaptive parallelization of sequences of PAP calls. This is shown to substantially improve the query performance without any cost knowledge or extensive memory usage compared to other strategies. Paper III answers the research questions **one, two, three,** and **four** by providing a generalized approach to query both dependent and independent data providing web service operations. The performance of PAP is evaluated using publicly available web services.

3.4 Paper IV

Paper IV describes the overall functionality of the WSMED system. This includes the WSMED query processor, the WSMED web service to query

any data providing web service operations, the web based demonstration of WSMED, and the web service generator.

The generation and deployment of web services for data providing systems answers research question **five**.

The web based demonstration of WSMED allows making SQL queries combining data from any data providing web services. This answers research question **six**.

3.5 Paper V

Paper V provides some preliminary work for Paper I. The WSMED architecture and a proposed method to manually define SQL views over web service operations are outlined.

3.6 Licentiate Thesis (Paper VI)

The Licentiate Thesis outlines some of the research questions, presents the technical background on which the research work is based, and proposes the WSMED architecture. Paper I and V are based on the Licentiate Thesis.

3.7 Paper VII

Paper VII describes the web based demonstration of WSMED that directly invokes WSMED web service operations from a web browser. This work is included and elaborated in Paper IV.

3.8 Book Chapter (Paper VIII)

The book chapter in Paper VIII is based on Paper I and II. It summarizes the WSMED architecture and the adaptive query processing strategies used.

4. Conclusions and Future Work

WSMED provides general database query capabilities over any data providing web service operations given their WSDL meta-data descriptions. For each data providing web service operation in a given WSDL document, WSMED automatically generates relational views by reading web service operations' WSDL descriptions. Such automatically generated relational views can be queried with SQL.

Without any cost knowledge the WSMED query processor automatically and adaptively finds an optimized parallel execution plan calling the queried data providing web service operations. The algebra operator *PAP* locally adapts the parallel plan until no significant performance improvement is measured, based on monitoring the flow from data providing web service operations. The operator handles queries where data providing web service operations are called both dependently and independently. A strategy using *PAP* is developed, which substantially improves the query performance without any cost knowledge or extensive memory usage compared to other strategies.

WSMED assumes that all queried data sources are available as web service operations. To make any data providing system into a web service WSMED includes a subsystem, the web service generator, which generates and deploys the web service operations to access a data source.

To comply with the XaaS paradigm WSMED itself is implemented as a web service that provides SQL query functionality to query and join any data providing web service operations. The WSMED web service is also generated by the web service generator. To enable search of any data providing web services from a browser without any need for installing software, the web based demonstration is written as a JavaScript program that directly calls the WSMED web service. In summary the contributions of the Thesis are:

1. The WSMED system architecture provides general SQL query capabilities over any data providing web services based on their WSDL documents.
2. To enable SQL queries to data providing web services, SQL views are automatically generated for any data providing web service operations by reading their WSDL documents.
3. To automatically parallelize queries to data providing web service, an algorithm is implemented to transform a non parallel plan into a parallel

plan by introducing the adaptive operator *PAP* that encapsulates plan functions calling data providing web service operations.

4. To automatically and adaptively optimize a parallel plan, the operator *PAP* adapts an initial parallel query process tree by locally monitoring result flows from each child query process until satisfactory performance is obtained. The adaptive query parallelization does not need any static cost model.
5. To generate data providing web service interfaces to any data providing system a web service generator automatically generates web service operations for wrapped data sources defined as interface functions. The generated web service operations are dynamically deployed without restarting a web server.
6. To comply with the XaaS paradigm, the WSMED web service is provided to query any data providing web services. It can be used directly from a browser without any software installations. The WSMED web service operations are generated by the web service generator.

All performance measurements were made with publicly available web service operations. A possible future work is to develop a benchmark to simulate the parallel web service calls for controlled experiments.

WSMED presently handle relational views that calls data providing web services operations without any side effects. Updatable relational views over web services is a subject for future work.

5. Summary in Swedish

Sökning bland datagenererande web services

Den kraftigt ökande tillgången till internetbaserade informationssystem har skapat ett behov att utveckla *web services* [7], dvs. system och standarder för att utbyta information mellan internetbaserade program. Medan s.k. *webbtjänster* gör det möjligt att utbyta information mellan människor och webbaserade program i vanliga webbläsare, tillhandahåller *web services* en infrastruktur för informationsutbyte mellan olika webbaserade program. För web services har man utvecklat ett antal standarder som SOAP[18], WSDL [9] och XML Schema [42]. Web services tillhandahåller verktyg för programutvecklare att definiera *operationer* (eng. operations) som är programmeringsgränssnitt för att anropa andra program via Internet. Dessa web service-operationer (WSO) är självbeskrivande i den meningen att information om hur de anropas och hur data som skall överföras skall se ut (s.k. meta-data) beskrivs för varje WSO m.h.a ett speciellt språk som heter *Web Service Description Language, WSDL*. WSDL-beskrivningarna läggs upp på Internet som maskinläsbara dokument. Genom att läsa WSDL-dokumentet för en web service har ett program all information som behövs för att kunna anropa de WSOer som beskrivs i dokumentet.

Web services används ofta för att hämta data från servrar som tillhandahåller information av olika slag. En *datagenerande WSO* returnerar datamängder för givna sökparametrar utan att ha sidoeffekter som ändrar data på servern. Sådana tjänster är en form av *sökbearbetning* (search computing) [8]. Andra typer av web services utför någon åtgärd, t.ex. gör en banktransaktion eller startar en maskin.

Ämnet för denna avhandling är att undersöka hur *frågespråk* kan göra det möjligt att effektivt söka bland olika datagenererande WSOer. Ett frågespråk är ett kraftfullt högnivåspråk för att söka bland data. T.ex. är frågespråket SQL standardspråk för sökning i konventionella databaser. I avhandlingen används SQL för att söka bland data från olika datagenererande WSOer i stället för från en konventionell databas. För att utföra motsvarande sökningar utan frågespråk programmerat i ett konventionellt programmeringsspråk måste man för varje fråga utveckla ett specialiserat program som implementerar en detaljerad strategi för hur sökningen bland datagenererande WSOer skall gå till.

Som ett exempel, antag att vi vill ställa en fråga som returnerar information om namngivna platser i några av USAs delstater, t.ex. deras postnummer och väderprognoser. Fyra olika datagenererande WSOer kan användas för att besvara frågan. Först kan operationen *GetAllStates* från web servicen *GeoPlaces* [10] anropas för att finna allmän information om delstater i USA. Sedan kan operationen *GetInfoByState* från web servicen *USZip* [36] anropas för att finna alla postnummer i en given delstat. Operationen *GetPlacesInside* från *Zipcodes* [11] returnerar alla platser inom ett postnummerområde. Slutligen kan operationen *GetCityForecastByZip* från *CYDNE* [12] anropas för att få väderprognosen för ett givet postnummer.

Ytterligare teknik som används i avhandlingsarbetet är mediatortekniken [39]. En mediator är ett system för att utföra frågor som kombinerar data från många olika datakällor. I detta arbete avses med en mediator ett system som gör det möjligt att m.h.a. ett frågespråk specificera frågor som kombinerar data från olika datagenererande WSOer.

I avhandlingen undersöks hur man kan bygga ett generellt system för skalbara frågor över datagenererande WSOer. Ansatsen är att utveckla ett prototypsystem med benämningen *WSMED (Web Service MEDiator)* för att ge svar på ett antal forskningshypoteser:

1. I vilken utsträckning kan standarder för web services som WSDL och SOAP utnyttjas av en web service mediator för att effektivt och skalbart utföra frågor till datagenererande WSOer?
2. Hur kan man, baserat på WSDL-beskrivningar automatiskt generera vyer över datagenererande WSOer för ett högnivåfrågespråk som SQL?
3. Hur kan optimerings- och transformationstekniker för databasfrågor användas för att tillhandahålla effektiv och skalbar sökning bland data från olika datagenererande WSOer?
4. Hur kan en frågeoptimerare snabba upp sökning från datagenererande WSOer utan att innehålla kunskap om hur kostsamma operationerna är?
5. Hur kan datakällor som inte är tillgängliga som web services på ett enkelt sätt transformeras till datagenererande WSOer för att göra det möjligt att ställa frågor till dem från en web service mediator?
6. Hur kan paradigmen ”*allt som en service*” (XaaS) [33] tillämpas för att ställa frågor mot datagenererande WSOer? Det vill säga, kan en web service mediator implementeras i form av en web service som anropas från en godtycklig webbläsare utan att kräva att användaren först installerar speciell programvara i sin dator?

För att besvara ovanstående forskningsfrågor har WSMED-prototypen utvecklats och utvärderats och har nu förmågan att skalbart utföra frågor över datagenererande WSOer.

WSMED kan dynamiskt anropa en godtycklig WSO genom att läsa dess WSDL-dokument. WSDL-dokumenten lagras i WSMED i en generell *web service databas* som kan representera beskrivningar av godtyckliga WSDL-dokument. Databasen används för att dynamiskt konstruera anrop till de WSOer som behövs för att utföra en fråga. Detta ger svar på forskningsfråga **ett**.

En WSO presenteras av WSMED som en tabell (vy) i SQL. SQL frågor kan ställas över dessa vyer. För en given web service genererar WSMED automatiskt SQL vyer för alla dess WSOer genom att läsa WSDL dokumentet. SQL vyn för en WSO definieras i termer av ett internt frågespråk som heter *WQL (WSMED Query Language)* och kan hantera de datatyper som behövs för att anropa WSOer. Den automatiska genereringen av SQL-vyer besvarar forskningsfråga **två**.

WSOer är normalt parametriserade i den meningen att de kräver att in-parametrar har kända värden för att de skall kunna anropas. Två WSO-anrop i en fråga är *beroende* om det ena kräver in-parametrar som produceras i resultatet av ett annat WSO-anrop, i annat fall är de *oberoende*. I exemplet ovan är *GetPlacesInside* and *GetCityForecastByZip* WSO-anrop som beror på *GetInfoByState* men som är oberoende av varandra. En utmaning är här att utveckla metoder att automatiskt optimera frågor som innehåller både beroende och oberoende WSO-anrop. Generellt är sådan optimering beroende av olika egenskaper hos WSO-anropen. Dessa egenskaper är i allmänhet inte tillgängliga och beror på olika nätverks- och datoregenskaper när och var frågorna körs. I sådana fall är det mycket svårt att basera optimeringen på en statisk kostnadsmodell av de olika ingående kostnaderna, vilket är den teknik för frågeoptimering som tillämpas i traditionella databaser.

För att optimera frågorna utan en kostnadsmodell av underliggande WSOer använder WSMED en ansats där WSO-anropen dynamiskt parallelliseras vid frågetillfället med hänsyn tagen till beroenden mellan olika WSO-anrop i en fråga. Ofta förbättras prestanda dramatiskt genom att systemet ser till att WSOer anropas parallellt i stället för att anropa dem efter varandra. WSMED genererar automatiskt parallella sökprogram, *exekveringsplaner*, som anropas i ett träd av kommunicerande processer, ett *processträd*, där olika exekveringsplaner anropas parallellt. Under körning optimeras och ändras processträdet dynamiskt genom att systemet mäter tiden att utföra delplaner utan kännedom om kostnaden att anropa underliggande WSOer. I avhandlingen visas att denna dynamiska frågeoptimering ger stora prestandaförbättringar och detta resultat besvarar forskningsfrågorna **tre** och **fyra**.

WSMED antar att de datakällor som anropas är definierade som WSOer. Att skapa en ny datagenerarande web service för en datakälla kräver normalt en del programmeringsarbete, t.ex. för att implementera WSOer, definiera WSDL-dokument och att driftsätta web servicen på nätet. För att på ett

enkelt sätt göra ett dataproducerande system tillgängligt som datagenererande WSOer innehåller WSMED en *web service-generator* som skapar och driftsätter WSOer. Programmeraren måste först definiera ett gränssnitt mot datakällan i mediatorsystemet Amos II [32]. Därefter generar systemet automatiskt motsvarande WSOer och gör dem omedelbart tillgängliga på nätet. Samtidigt genererar system ett WSDL-dokument som beskriver genererade WSOer. Denna automatiska generering och driftsättning av WSOer ger ett svar på forskningsfråga **fem**.

WSMED-systemet självt är tillgängligt som en web service som kan utföra frågor till andra datagenererande web services. Denna *WSMED web servic*e innehåller WSOer för att sätta upp sessioner, importera WSDL-dokument för de web services som man vill söka i, inspektera de SQL-vyer som generats, ställa frågor mot SQL-vyerna och autentisera användaren. WSMED web servicen har genererats automatiskt m.h.a. web service-generatorn. WSMEDs funktionalitet demonstreras genom ett webbaserat användargränssnitt som är tillgängligt från en godtycklig webbläsare. Ingen programvara behöver då installeras eftersom gränssnittet är implementerat som ett JavaScript-program som exekveras i webbläsaren och direkt anropar WSMED web servicen. Detta visar att WSMED uppfyller XaaS paradigmen vilket besvarar forskningsfråga **sex**.

6. Acknowledgements

First and foremost I would like to thank my supervisor Professor Tore Risch for supervising me. I'm deeply appreciating his willingness to assist me in writing papers and Thesis by providing valuable suggestions and fruitful comments. I am very grateful to him to sharing his precious knowledge with me and being always ready to discuss the new directions and the research problems. My second supervisor Professor G.N.Wikramanayake is supporting me by his constructive advices and guidance and I appreciate his assistance. Dr.S.Mahesan and Dr.S.Kanaganathan are my first Computer Science teachers and emboldened me as a research student in Computer Science. I would like to thank them for their rewarding guidance and assistance.

I also wish to thank all Sri Lankan Sida split PhD program management committee members and Sida coordinator for Uppsala University for their great support all the time.

I offer my sincere gratitude to the administrative authorities of Department of Computer Science and Faculty of Science, University of Jaffna for their enormous support.

I am in debt to all present and past UDBL group members for helping and sharing with me difficulties and happiness. I am also like to thank all my fellow Sri Lankans for their friendship and support.

Ulrika Andersson and all the others at the Department of Information Technology, Uppsala University who have helped me immensely need mentioning.

I'm grateful to my wife, Sutha and my daughters Sruthy and Sharana for their generous support and patience.

I have great pleasure to dedicate this Thesis to my parents, Manivasakan and Saroginidevi, who have always encouraged and supported me to study.

This work was supported by the Swedish International Development and Cooperation (Sida), and the Swedish Foundation for Strategic Research under contract RIT08-0041.

Bibliography

[1] AmosII wrappers, http://user.it.uu.se/~udbl/amos/wrappers.html

[2] T.Andrews et al., Business Process Execution Language for Web Services, Version 1.1, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf, 2003

[3] R.Avnur and J.M.Hellerstein, Eddies: Continuously Adaptive Query Processing, *Proc. 2000 ACM SIGMOD International Conference on Management of Data*, pp 261-272, 2000

[4] S.Bajaj et al., Web Services Policy Framework (WSPolicy), *http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polfram/ws-policy-2006-03-01.pdf*, 2006

[5] K.Ballinger, P.Brittenham, A.Malhotra, W.A. Nagy, and S.Pharies, Web Services Inspection Language (WS-Inspection), *ftp://www6.software.ibm.com/software/developer/library/ws-wsilspec.pdf*, 2001

[6] T.Bellwood et al, UDDI Version 3.0.2, UDDI Spec Technical Committee Draft, *http://uddi.org/pubs/uddi_v3.htm#_Toc85907967*, 2004

[7] D.Booth, H.Haas, F.McCabe, E.Newcomer, M.Champion, C.Ferris, and D.Orchard, Web Services Architecture,W3C Working Group Note, *http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/* , 2004

[8] S.Ceri, Search Computing. *Proc. International Conference on Data Engineering*, IEEE Computer Society, pp. 1- 3, 2009

[9] E.Christensen, F.Curbera, G.Meredith, and S. Weerawarana, Web services description language (WSDL) 1.1., W3C Recommendation, *http://www.w3.org/TR/wsdl*, 2001

[10] codeBump, GeoPlaces web service *http://codebump.com/services /PlaceLookup.asmx*

[11] codeBump, Zipcodes web service *http://codebump.com/services /ZipCodeLookup.asmx*

[12] CYDNE, *http://ws.cdyne.com/WeatherWS/Weather.asmx?WSDL*

[13] A.Deshpande, Z.G:Ives and V.Raman, Adaptive Query Processing, *Foundations and Trends in Databases*, 2007

[14] R.Elamasri, and S.M.Navathe, *Fundamentals of Database Systems*, 4th Edition, ISBN 0-321-20448-4, Pearson Education, pp 855-856, 2004

[15] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A.Rajaraman, Y. Sagiv, J.D. Ullman, V. Vassalos, and J.Widom, The TSIMMIS Approach to Mediation: Data Models and Languages, *Journal of Intelligent Information Systems*, 8(2), pp 117-132, 1997

[16] H.Garcia-Molina, J.D Ullman, and J.Widom, *Database Systems: The Complete Book,* ISBN 0-13-098043-9, Prentice Hall, pp 1047-1069, 2002

[17] G.Graefe, Query evaluation techniques for large databases**,** *ACM Computing Surveys (CSUR)***,** 25(2), pp 73-169, 1993

[18] M.Gudgin, M.Hadley, N.Mendelsohn, J.Moreau, and H.Frystyk Nielsen, SOAP Version 1.2 Part 1: Messaging Framework,W3C Recommendation, *http://www.w3.org/TR/soap12-part1/* ,2003
[19] W. Hasan, *Optimization of SQL queries for Parallel Machines*, Springer-Verlag,1997
[20] Z.He, B.S.Lee, and R.Snapp, Self-Tuning Cost Modeling of User-Defined Functions in an Object-Relational DBMS, *ACM Transactions on Database Systems*, 30(3), pp 812-853, 2005
[21] Hypertext Transfer Protocol, W3C Architecture domain, *http://www.w3.org/Protocols/*
[22] K.Lawrence, C.Kaler, A.Nadalin, M.Gudgin, A.Barbir, and H.Granqvist, WS-SecurityPolicy v1.0, OASIS Working Draft, *http://www.oasis-open.org/committees/download.php/15979/oasis-wssx-ws-securitypolicy-1.0.pdf*, 2005
[23] A.Y.Levy et al., Querying Heterogeneous Information Sources Using Source Descriptions, *Proc. of 22nd Very Large Data Bases Conference(VLDB 96)*, pp 251-262 , 1996
[24] C.Li et al., Capability Based Mediation in TSIMMIS, *Proc. 1998 ACM SIGMOD International Conference on Management of Data,* 1998, pp 564-566
[25] Y.Papakonstantinou, A.Gupta, and L.Haas, Capabilities-base query rewriting in mediator systems, *Proc. Conference on Parallel and Distributed Information Systems*, pp 170-183, 1996
[26] J.Postel, SIMPLE MAIL TRANSFER PROTOCOL, RFC 821, *http://www.ietf.org/rfc/rfc0821.txt*, 1982
[27] J.Postel, Transmission Control Protocol, *http://www.ietf.org/rfc/rfc793.txt,* 1981
[28] J. Postel, and J. Reynolds, FILE TRANSFER PROTOCOL (FTP), *http://tools.ietf.org/html/rfc959*, 1985
[29] L.Raschid and S.Y.W.Su, A Parallel Processing Strategy for Evaluating Recursive Queries, *Proc. 12th Very Large Data Bases Conference(VLDB '86)*, pp 412-419, 1986
[30] T.Risch, Distributed Architecture, in L.Liu and M.Tamer Özsu (eds.): *Encyclopedia of Database Systems*, 2(1), Springer, pp 875-879, 2009
[31] T.Risch and V.Josifovski, Distributed Data Integration by Object-Oriented Mediator Servers, *Concurrency and Computation: Practice and Experience J.*, 13(11), John Wiley & Sons, pp 933-953, 2001
[32] T.Risch, V.Josifovski, and T.Katchaounov, Functional Data Integration in a Distributed Mediator System, in P.Gray, L.Kerschberg, P.King, and A.Poulovassilis (eds.): *Functional Approach to Data Management - Modeling, Analyzing and Integrating Heterogeneous Data,* Springer, pp 211-238, 2003
[33] S.Robison, The Next Wave: Everything as a Service, *http://www.hp.com/hpinfo/execteam/articles/robison/08eaas.html*
[34] D. Shipman, The Functional Data Model and the Data Language DAPLEX, *ACM Transactions on Database Systems*, 6(1), pp 140-173, 1981
[35] M. Tork-Roth, and P. Schwarz, Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources, *Proc. 23rd Very Large Data Bases Conference(VLDB 1997)*, pp 266-275, 1997
[36] USZip, *http://www.webservicex.net/uszip.asmx*
[37] V.Vassalos, and Y.Papakonstantinou, Describing and Using Query Capabilities of Heterogeneous Sources, *Proc. 23rd Very Large Data Bases Conference(VLDB 97)*, pp 256-265*,* 1997
[38] Web Services Management Work by the Web Services Architecture Working Group, *http://www.w3.org/2002/ws/arch/4/management/*

[39] G. Wiederhold, Mediators in the Architecture of Future Information Systems, *IEEE Computer*, 25(3), pp 38-49, 1992
[40] WSMED Demo, *http://udbl2.it.uu.se/WSMED/wsmed.html*
[41] WSMED WSDL, *http://udbl2.it.uu.se/WSMED/wsmed.wsdl*
[42] XML Schema, *http://www.w3.org/standards/xml/schema*
[43] R.Yerneni, C.Li, H.Garcia-Molina, and J.D:Ullman, Computing capabilities of mediators, *Proc. 1999 ACM SIGMOD International Conference on Management of Data*, pp 443-454, 1999
[44] V.Zadorozhny, L.Raschid, M.E.Vidal, T.Urban, and L.Bright, Efficient Evaluation of Queries in a Mediator for WebSources, *Proc. 2002 ACM SIGMOD International Conference on Management of Data,* pp 85-96, 2002

Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 755

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-128928

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2010