

DISSERTATION

Distributed Heterogeneous Web Data Sources Integration

DeXIN Approach

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften unter der Leitung von

Univ.-Prof. Dr. Reinhard Pichler

Abteilung für Datenbanken und Artificial Intelligence

Institut für Informationssysteme

eingereicht an der Technische Universität Wien
Fakultät für Informatik

von

Muhammad Intizar Ali

0627742

Favoritenstrasse 9-11/184-2, A-1040 Wien

Diese Dissertation haben begutachtet:

Univ.-Prof. Dr. Reinhard Pichler

Univ.-Prof. Dr. Uwe Zdun

Wien, 14.08.2011

Muhammad Intizar Ali

DISSERTATION

Distributed Heterogeneous Web Data Sources Integration

DeXIN Approach

submitted in partial fulfillment of the requirements for the degree of
Doctor of Technical Sciences under supervision of

Univ.-Prof. Dr. Reinhard Pichler

Database and Artificial Intelligence Group

Institute for Information Systems

at Vienna University of Technology
Faculty of Informatics

by

Muhammad Intizar Ali

0627742

Favoritenstrasse 9-11/184-2, A-1040 Wien

This Thesis has been reviewed by:

Univ.-Prof. Dr. Reinhard Pichler

Univ.-Prof. Dr. Uwe Zdun

Wien, 14.08.2011

Muhammad Intizar Ali

Erklärung zur Verfassung der Arbeit

Muhammad Intizar Ali
Favoritenstrasse 9-11/184-2, A-1040 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)

(Unterschrift Verfasserin)

Abstract

In modern business enterprises, it is frequent to develop an integrated application to provide uniform access to multiple existing information systems running internally or externally of the enterprise. Data integration is a pervasive challenge faced in these applications that need to query across multiple autonomous and heterogeneous data sources. Integrating such diverse information systems becomes a challenging task particularly when different applications use different data formats and query languages which are not compatible with each other.

With the growing popularity of web technologies and availability of the huge amount of data on the web, the requirements for data integration has changed from the traditional database integration approaches. The large scale of web data sources has not only led to high levels of distribution, heterogeneity, different data formats and query languages. Additionally, the data is also associated with data concerns like privacy, licensing, pricing, quality of data, etc. Hence, the data integration tools not only have to provide the optimal solution to mitigate the heterogeneity in data formats and query languages. In addition, also the various data concerns should be preserved when data is published and utilized. Moreover, data service selection and data selection should be based on these data concerns.

The goal of this thesis is to provide better means to easily and dynamically integrate distributed heterogeneous web data sources (particularly XML and RDF data sources) in such a way that the user can easily build data integration applications while assuring all the data concerns associated with the data.

The main topic of this work is devoted to the distributed heterogeneous data integration for web data sources. In order to deal with the challenge of XML and RDF data integration, we propose “*DeXIN (Distributed extended XQuery for heterogeneous Data INtegration)*”, an extensible framework for distributed query processing over heterogeneous, distributed and autonomous data sources. DeXIN considers one data format as the basis (the so-called “aggregation model”) and extends the corresponding query language to executing queries over heterogeneous data sources in their respective query languages. We come up with an extension of XQuery which covers the full SPARQL language and supports the decentralized execution of both XQuery and SPARQL in a single query.

For the assurance of the data concerns associated with the published data over the web, we introduce a “*Data Concerns Aware Querying System*”. A data concerns aware querying system incorporates several data concerns into a query language, thus enabling data services integration systems to handle data concerns associated with the data services. Our concerns aware querying system extends the XQuery language to make it concerns aware, with the introduction of special keywords for mentioning data concerns within the query.

In the last part of this thesis, we design a mashups tool on top of DeXIN. We propose a query based aggregation of multiple heterogeneous data sources by combining powerful querying features of XQuery and SPARQL with an easy interface of a mashup tool for data sources in XML and RDF. Our mashup editor allows for automatic generation of mashups with an easy to use visual interface. For the dynamic integration of heterogeneous web data sources we utilize the concept of data mashups, which uses the extension of XQuery proposed in DeXIN.

Kurzfassung

In modernen Wirtschaftsunternehmen wird häufig eine integrierte Anwendung entwickelt, um einheitlichen Zugriff auf mehrere bestehende Informationssysteme zu bieten, die innerhalb oder außerhalb des Unternehmens laufen. Datenintegration ist eine tiefgreifende Herausforderung dieser Anwendungen, da Abfragen über mehrere autonome und heterogene Datenquellen reichen. Die Integration solcher unterschiedlicher Informationssysteme ist eine anspruchsvolle Aufgabe, insbesondere wenn verschiedene Anwendungen unterschiedliche Datenformate und Abfragesprachen verwenden, die nicht untereinander kompatibel sind.

Mit der wachsenden Popularität von Web 2.0-Technologien und der Verfügbarkeit riesiger Mengen an Daten im Web, haben sich die Anforderungen für die Datenintegration, im Vergleich zu traditionellen Ansätzen der Datenbankintegration, geändert. Der große Umfang an Web-Datenquellen hat nicht nur zu einem hohen Maß an Verteilung, Heterogenität, sowie unterschiedlichen Datenformaten und Abfragesprachen geführt, sondern darüber hinaus sind die Daten auch mit zusätzlichen Dateneigenschaften verbunden, wie zum Beispiel Datenschutz, Lizenzierung, Kosten, Qualität der Daten, etc. Daher müssen die Datenintegration-Tools nicht nur einen optimalen Weg zur Verfügung stellen, um die Heterogenität der Datenformate und Abfragesprachen zu reduzieren, sondern darüber hinaus sollten auch die verschiedenen zusätzlichen Dateneigenschaften beibehalten werden, wenn die Daten veröffentlicht oder genutzt werden. Weiters sollte die Auswahl der Datendienste und die Selektion der Daten diese Dateneigenschaften berücksichtigen.

Das Ziel dieser Dissertation ist es, bessere Mittel bereitzustellen zur einfachen und dynamischen Integration von verteilten heterogenen Web-Datenquellen (insbesondere XML und RDF-Datenquellen), in einer Weise, die es dem Benutzer vereinfachen, Datenintegrationsapplikationen zu erstellen, während gleichzeitig alle Dateneigenschaften mit den damit verbundenen Daten sichergestellt werden.

Das Hauptthema dieser Arbeit ist der verteilten heterogenen Datenintegration für Web-Datenquellen gewidmet. Um die Herausforderung der XML und RDF-Datenintegration zu bewältigen, schlagen wir "DeXIN (Distributed extended XQuery for heterogeneous Data Integration)", ein erweiterbares Framework für die verteilte Verarbeitung von Abfragen über heterogene, verteilte und autonome Datenquellen vor. DeXIN verwendet ein Datenformat als Grundlage (das sogenannte "aggregation model") und erweitert die entsprechende Abfragesprache, um Abfragen über heterogene Datenquellen in ihren jeweiligen Abfragesprachen durchzuführen. Wir stellen eine Erweiterung von XQuery vor, welche die volle SPARQL Sprache abdeckt und die dezentrale Ausführung von XQuery als auch SPARQL in einer einzigen Abfrage unterstützt.

Für die Sicherstellung der Dateneigenschaften, die mit den veröffentlichten Daten im Web verbunden sind, führen wir ein “Data Concerns Aware Query System” ein. Dieses System vereinigt mehrere Dateneigenschaften in eine Abfragesprache, wodurch es Datenservice-Integrationssystemen erlaubt wird, Dateneigenschaften, die mit den Datendiensten verbunden sind, zu behandeln. Unser “Data Concerns Aware Query System” erweitert die XQuery-Sprache, um Dateneigenschaften zu berücksichtigen. Dafür werden spezielle Schlüsselwörter eingeführt, um Dateneigenschaften innerhalb der Abfrage auszudrücken.

Im letzten Teil dieser Arbeit entwerfen wir ein Mashup-Tool, welches auf DeXIN aufbaut. Wir präsentieren eine Abfrage-basierte Aggregation von mehreren heterogenen Datenquellen durch die Kombination von vielseitigen Abfrage-Features von XQuery und SPARQL mit einer intuitiven Benutzerschnittstelle eines Mashup-Tools für Datenquellen in XML und RDF. Unser Mashup-Editor ermöglicht die automatische Generierung von Mashups mit einer einfach zu bedienenden visuellen Schnittstelle. Wir nutzen das Konzept der Daten-Mashups, um dynamisch heterogene Web-Datenquellen zu integrieren, indem wir die in DeXIN vorgeschlagene Erweiterung von XQuery benutzen.

Dedication

I dedicate my thesis to my beloved homeland

PAKISTAN

for

giving me a unique recognition with her rich culture and beautiful traditions.

Acknowledgements

I would like to express my deepest gratitude and admiration to my supervisor, Prof. Dr. Reinhard Pichler for his excellent guidance, patience and his encouragement in helping me to accomplish this research. He introduced me to new dimensions of knowledge, with his guidelines, encouragements and valuable suggestions. I really wish to express my appreciation to him for his full cooperation, kindness, patience, thoughtfulness and continuous support that made this work a reality.

My sincere thanks to Prof. Dr. Schahram Dustdar, who provided me an opportunity to improve my research work in collaboration with the Distributed Systems Group at the Institute of Information Systems, Vienna University of Technology, Austria.

I am very thankful to Dr. Hong-Linh Truong for his great cooperation, guidance and contributions of his ideas to streamline my research work. Without his valuable discussions and objective criticism I would not have been able to complete this task.

I am grateful to Prof. Dr. Uwe Zdun for his precious time to proof read the manuscript and be my external supervisor for this thesis.

I am thankful to Higher Education Commission of Pakistan for providing me the opportunity to study in Austria and provide full financial support for my studies in Austria.

I acknowledge the financial support from the Vienna Science and Technology Fund (WWTF) for providing me partial funding from the project ICT08-032.

I am thankful to Stefan Rümmele for translating the abstract of my thesis into German language.

And finally I admit that I completed my PhD due to the encouragements of my loving parents, family and friends who always pray for my success.

Vienna, August 2011
Muhammad Intizar Ali

Contents

Abstract	iii
Kurzfassung	v
Dedication	vii
Acknowledgements	ix
List of Figures	xvi
List of Tables	xviii
1 Introduction	1
1.1 The Problem of Data Integration	1
1.2 Motivation and Challenges	3
1.2.1 Distributed Heterogeneous Data Integration	3
1.2.2 Data Concerns for Data Services Integration	4
1.2.3 Database Oriented Mashups	4
1.3 Contributions	5
1.3.1 DeXIN: <i>D</i> istributed <i>e</i> xtended <i>X</i> Query for Data <i>I</i> Ntegration	5
1.3.2 Concerns Aware Querying	5
1.3.3 Data Mashups for Data Services Integration	6
1.4 Structure of the Thesis	6
2 Foundations and Related Work	9
2.1 Data Integration: Theoretical Background	9
2.2 Data Models	10
2.2.1 Relational Data Model	10
2.2.1.1 Relational Data	10
2.2.1.2 Relational Schema	11
2.2.1.3 Structured Query Language	12
2.2.2 Semi-structured Data Model	12
2.2.2.1 XML Data Structure	14
2.2.2.2 DTD/XML Schema	14
	xi

2.2.2.3	XQuery	14
2.2.3	Semantic Data Model	16
2.2.3.1	Resource Description Format	16
2.2.3.2	RDF Vocabulary Description Language	17
2.2.3.3	Ontology Web Language	18
2.2.3.4	SPARQL	18
2.3	Data Transformation (XML ↔ RDF)	19
2.3.1	GRDDL	20
2.3.2	Embedding SPARQL into XQuery/XSLT	21
2.3.3	XSPARQL	22
2.4	Distributed Query Processing (SPARQL and XQuery)	23
2.4.1	DXQ	23
2.4.2	XRPC	25
2.4.3	DARQ	25
2.5	Discussion	26
3	DeXIN	29
3.1	Motivation	30
3.2	Application Scenario	30
3.3	Approaches for Heterogeneous Data Sources Integration	32
3.4	XML and RDF Data Sources Integration	33
3.4.1	Heterogeneity	33
3.4.2	Distribution	35
3.5	DeXIN: Distributed <i>extended</i> XQuery for Data <i>I</i> Ntegration	37
3.5.1	Architectural Overview	38
3.5.2	Query Evaluation Process	39
3.5.2.1	Parser	39
3.5.2.2	Query Decomposer	39
3.5.2.3	Metadata Manager	39
3.5.2.4	Optimizer	40
3.5.2.5	Executor	41
3.5.2.6	Result Reconstruction	41
3.5.2.7	Query Rewriter	41
3.6	XQuery extension to SPARQL	41
3.6.1	Syntax	42
3.6.2	Query Decomposition and Processing	43
3.7	Implementation and Experimental Analysis	44
3.7.1	Experimental Application: Web Services Management	46
3.7.2	Performance Analysis	47
3.7.2.1	Data Distribution over the Testbed	47
3.7.2.2	Experiments	47
3.8	DeXIN Demo	48
3.8.1	DeXIN as a Data Integration Web Service	48

3.8.2	Searching Available Data Services	49
3.8.3	Registration of the Data Sources	50
3.8.4	Data Source Statistics and Schema Information	50
3.8.5	Query Execution	51
3.9	Discussion	51
3.9.1	Data Transformation V/S Results Transformation	51
3.9.2	Data Shipping V/S Query Results Shipping	52
3.9.3	Summary	52
4	Data Concerns Aware Querying	53
4.1	Motivation	54
4.2	Data Services	55
4.2.1	Roles of Data Services	57
4.2.1.1	Data Provider	57
4.2.1.2	Service Provider	57
4.2.1.3	Consumer	58
4.2.2	Data Services Pricing Models	58
4.2.2.1	Volume Based Model	58
4.2.2.2	Data Type Based Model	58
4.2.3	Data Services: Advantages	58
4.2.4	Data Services: Challenges and Issues	59
4.3	Data Concerns	60
4.3.1	Data Quality	61
4.3.1.1	Timeliness	61
4.3.1.2	Accuracy	61
4.3.1.3	Completeness	62
4.3.1.4	Availability	62
4.3.2	Licensing	62
4.3.2.1	Usage Permission	62
4.3.2.2	Data Location	62
4.3.2.3	Usage Fee	63
4.3.2.4	Law Enforcement	63
4.3.3	Quality of Data Service	63
4.3.3.1	Performance	63
4.3.3.2	Reliability	63
4.3.3.3	Trustworthiness	63
4.3.3.4	Service Location	63
4.4	Data Concerns Aware Integration	64
4.5	Models of Data Concerns	64
4.5.1	Model (a): Querying with Data Concerns	65
4.5.2	Model (b): Query with Data Concerns with Centralized Repository	66
4.5.3	Model (c): Concerns Aware Querying	66
4.5.4	Model (d): Concerns Aware Querying with Dynamic Discovery	67

4.6	Concerns Aware Querying	67
4.6.1	Data Concerns Collection	67
4.6.1.1	Service Level Concerns	69
4.6.1.2	Data Level Concerns	69
4.6.2	Query Processing	69
4.6.3	Concerns Aware XQuery	70
4.7	Implementation and Evaluation	71
4.7.1	Implementation	71
4.7.2	Experimental Application: Distributed Extended XQuery for Data Integration (DeXIN)	72
4.7.3	Evaluation	72
4.8	Discussion	73
5	Data Integration Using Data Mashups	75
5.1	Motivation	76
5.2	Mashups	76
5.2.1	Architectural Layers of the Mashups	78
5.2.1.1	Data Layer	79
5.2.1.2	Process Layer	79
5.2.1.3	Presentation Layer	79
5.2.2	Types of Mashups	79
5.2.2.1	Consumer Mashups	79
5.2.2.2	Business Mashups	80
5.2.2.3	Data Mashups	80
5.2.3	Existing Mashups Tools	80
5.2.3.1	Yahoo Pipes	80
5.2.3.2	IBM Mashup Center	81
5.2.3.3	Intel Mash Maker	81
5.2.3.4	Microsoft Popfly	82
5.3	Data Integration using Data Mashups	82
5.3.1	Challenges	82
5.3.2	Database Oriented Mashups	84
5.3.2.1	Deri Pipes	84
5.3.2.2	MashQL	84
5.4	DeXIN as a Mashup Tool	85
5.4.1	Architectural Overview	86
5.4.2	Data Source Registration	87
5.4.3	Data Source Selection	87
5.4.4	Mashup Editor	88
5.4.5	Query Results	88
5.5	Discussion	88
6	Conclusion	91
6.1	Summary of the Contributions	91

6.1.1	Heterogeneous Web Data Sources Integration	91
6.1.2	DeXIN as a Data Integration Tool	92
6.1.3	Data Concern Aware Querying for Data Services Integration	92
6.1.4	Data Services Integration Using Data Mashups	93
6.2	Future Work	93
6.2.1	Extend DeXIN to Further Query Languages	93
6.2.2	Optimize the Distributed Query Processing	93
6.2.3	Bring more Data Concerns into Consideration	94
6.2.4	Incorporate DeXIN to Other Applications	94
6.2.5	Efficient Resource Allocation based on Data Concerns	94
6.3	Closing Remarks	95
A	List of Publications	97
	Bibliography	99

List of Figures

2.1	A Sample XML Document	13
2.2	A Sample XML Schema Document Excerpt	15
2.3	A Sample XQuery	15
2.4	RDF Triple	16
2.5	A Sample RDF document	17
2.6	RDF Schema Vocabulary Document	18
2.7	A Sample SPARQL Query	19
2.8	SPARQL Query Results in XML/RDF Format	20
2.9	Embedded SPARQL inside XQuery	21
2.10	DXQ Grammar Extensions	24
2.11	A DNS Resolver in DXQ	24
2.12	A Sample XRPC Query	26
3.1	Data Sources Related to a Web Service	31
3.2	Data Integration Approaches on Different Architectural Levels	32
3.3	Typical Data Integration Application Issues	34
3.4	Distributed Query Processing In Peer-2-Peer Environment	36
3.5	Traditional Query Technique for Web Data Sources	37
3.6	Architectural Overview of the DeXIN Framework	38
3.7	Query Evaluation Process	40
3.8	Extended XQuery	41
3.9	An Example extended XQuery for DeXIN	43
3.10	Query Processing in DeXIN	44
3.11	Result after Executing Query	45
3.12	Execution Time Comparison DeXIN Vs Naive Centralized	46
3.13	Execution Time Comparison (Varying Selectivity Factor)	48
3.14	Execution Time Comparison (Varying Number of Data Sources)	49
3.15	Data Source Registration Form	50
3.16	User Interface of DeXIN	51
4.1	Conceptual Architecture of a Web Service	56
4.2	Conceptual Architecture of the Data Services	57
4.3	Activities in Data Concerns Aware Data Services System	65

4.4	Data Concerns Aware Querying Models	66
4.5	Data Concerns Tree	68
4.6	A Sample Concerns Aware XQuery	72
5.1	Mashups Architecture	77
5.2	Mashups Components Layers	78
5.3	The Damia Mashup Editor	83
5.4	DeXIN Mashup Architectural Layers	85
5.5	DeXIN Mashup Editor	86
5.6	A Sample Query in extended XQuery Format – Generated from the Mashup Visual Interface	87

List of Tables

2.1	A Sample Relational Database	11
2.2	A Sample SQL Query	13
2.3	Schematic View of XSPARQL	23
3.1	SPARQLQuery Function	42
3.2	XML Data Sources	45
3.3	RDF Data Sources	45
4.1	Some Data Concerns Associated with Data Services	61
4.2	Data Sources with Varying Size and Data Concerns	73
5.1	Comparison of the Features of Various Mashup Tools	82

Introduction

1.1 The Problem of Data Integration

The problem of data integration is among the oldest research problems in the database community and it emerged shortly after the introduction of database systems [50]. Data integration has been defined as the problem of providing unified and transparent access to a collection of data stored in multiple, autonomous, and heterogeneous data sources [51]. Integration of multiple information systems generally aims at combining selected systems so that they form a new unified view and give users the illusion of interacting with one single information system. Users are provided with a homogeneous logical view of data that is physically distributed over heterogeneous data sources. The reason for integration is twofold: First, given a set of existing information systems, an integrated view can be created to facilitate information access through a single information access point. Second, given a certain information need, data from different complementing information systems is combined to gain a more comprehensive platform to satisfy the need.

In modern business enterprises, it is frequent to develop an *integrated application* to provide uniform access to multiple existing information systems running internally or externally of the enterprise. Data Integration is a pervasive challenge faced in these applications that need to query across *multiple* autonomous and heterogeneous data sources [40].

Integrating such diverse information systems becomes a challenging task particularly when different applications use different data formats and query languages which are not compatible with each other. There are many real world scenarios where data integration is very crucial. For example, in large enterprises, the integration of a multitude of existing data sources within the enterprise, merge of similar companies; in scientific projects, where data sets are being produced independently by multiple researchers and it is needed to combine research results from different repositories; for better cooperation among government agencies, each with their own data sources [40].

The problem of data integration has been long studied and investigated over the period of last three decades. Many approaches of data integration have been proposed, designed and

implemented during these years.

- **Federated Database Systems (FDBS)** A federated database system (FDBS) is a collection of cooperating database systems that are autonomous and possibly heterogeneous [71]. FDBS provide a uniform data access by logically integrating data from multiple autonomous database management systems (DBMS). FDBS is a full-fledged DBMS which logically integrates all components of contributing databases. A global schema is defined to integrate data from all databases having their individual schema. It allows global query execution, global transactions, and global access control over all participating databases from a single point.
- **Data Warehousing (DWH)** Data warehouses use data storage approach, where copies of data from multiple data sources are stored and integrated in a single database, called data warehouse [42]. From several data sources, database systems or online transaction processing systems (OLTP) data is extracted, pre-processed, transformed and loaded (ETL) into the data warehouse. Data warehouses are mainly designed for reporting and analysis. Data is locally stored in the data warehouse which is periodically updated by data from the original data sources.
- **Mediators/Schema Mapping** Mediators are software components that contain a global query processor which sends subqueries to local data sources, executes them independently and combines local query results [81]. A mediated logical schema is designed which acts as a middle layer between the application and data sources. The application can query the global schema independent of the original structure of data stored in local data source. Several data integration approaches have been proposed using mediator [33, 77, 86].
- **Peer-to-peer (P2P)** Peer-to-peer data integration is a decentralized approach to integrate multiple, distributed and autonomous data sources (peers) [39]. P2P architecture was proposed after the inspiration from peer-to-peer file sharing systems [36]. Peer-to-peer integration systems do not require any global schema and any peer can contribute and share its data in a decentralized environment.
- **Service Oriented Data Integration** A service-oriented architecture (SOA) offers an ideal framework for implementing a common data integration approach across the enterprise. In SOA, data integration technology takes advantage of the level of abstraction that enables its components and services to be wrapped and reused without extensive hand-coding [41].
- **Semantic Data Integration** The semantic web extends the current web in such a way that, the information is given with “well-defined meaning” also called “semantics”, to better enabling computers and people to work in cooperation [49]. Inspired by the semantic web several data integration approaches have been proposed using the data representation defined by the semantic web [11, 17].
- **Web 2.0 or 3.0/ Mashups** One of the goals of the web 2.0 and 3.0 technologies is to facilitate integration of available web sources with simple development of *situational applications* for information sharing, reuse and integration. These technologies offer various

data integration approaches for the creation of situational applications for information or service integration (e.g. mashups and cloud computing) [46, 79]. Mashups allow the user to aggregate multiple existing services or sources to create a new service/application for a new purpose [46]. Mashups are mainly focused on integration of services using API's or web feeds (e.g. RSS and Atom feeds). Recently data mashups have been studied and analyzed for using mashup tools with respect to the data integration aspect [54].

1.2 Motivation and Challenges

1.2.1 Distributed Heterogeneous Data Integration

The advent of the internet and world wide web has changed the requirements and specifications of data integration from the requirements as it used to be many years ago. In the web environment, rich, diverse sources of heterogeneous and distributed data are ubiquitous. A huge amount of data is available on the internet in structured, semi structured and unstructured format. In recent years, there has been an enormous boost in semantic web technologies and web services. Web applications thus have to deal with huge amounts of data which are normally scattered over various data sources using various languages. Hence, these applications are facing two major challenges, namely (i) *how to integrate heterogeneous data* and (ii) *how to deal with rapidly growing and continuously changing distributed data sources*.

The most important languages for specifying data on the web are, on the one hand, the Extensible Markup Language (XML) [15] and, on the other hand, the Resource Description Framework (RDF) [9] and Ontology Web Language (OWL) [58]. XML is a very popular format to store and integrate a rapidly increasing amount of semi structured data on the web while the semantic web builds on data represented in RDF and OWL, which is optimized for data interlinking and merging.

There are several approaches for dealing with *heterogeneous* data consisting of XML, RDF and OWL: The most common approach is to transform all data sources into a single format [6,7] and apply a single query language to this data. Another approach to deal with heterogeneity is query re-writing which poses queries of different query languages to the data which is left in the original format, thus avoiding transformation of the whole data sources [8]. A major drawback of the transformation based approaches is that the transformation of data from one language into the other is a tedious and error prone process. Indeed, an RDF graph can be represented by more than one XML tree structure, so it is not clear how to formulate XQuery queries against it. On the other hand, XML lacks semantic information; so converting XML to RDF results in incomplete information with a number of blank nodes in the RDF graph.

There is an intense need for web applications to handle queries over heterogeneous, autonomous, and distributed data sources. Hence it becomes an issue of utmost importance, that “*how to integrate these diverse, distributed and heterogeneous data sources?*”, particularly in a single query avoiding any data transformation.

1.2.2 Data Concerns for Data Services Integration

More and more data providers are reaping the benefits of web 2.0 technology and provide their data on the web either through web services, API's (REST/SOAP), or data services – also referred to as *Data as a Service* (DaaS) [25, 37]. Data services combine the strength of database systems and query languages with the benefits of service oriented architecture.

Data services are increasingly used for data integration. Many tools and techniques are available to dynamically compose, integrate and execute different data services or sources [64]. These tools help to create situational applications by composing existing data services. The data thus published and processed is often associated with data concerns like privacy, licensing, pricing, quality of data, etc. Hence, data integration tools not only have to mitigate the heterogeneity in data formats and query languages. In addition, also the various data concerns should be preserved when data is published and utilized. Moreover, data service selection and data selection should be based on these data concerns. There is a clear need for an explicit system that (semi)automatically selects the most appropriate data service as well as data items for each user according to various data concerns.

Some data concerns like data quality, privacy, and quality of service (QoS) have long been studied in their respective domains of databases, data mining and web services. However, data services are different. Recently, [75] the importance of distinguishing data services from web services has been recognized. However a systematic integration of data concerns awareness into data services is still missing to date. Moreover, previous approaches of dealing with data concerns like privacy do not directly integrate the data concerns awareness into the query language, even though this would be very important for enabling the querying system to select the best suited data source for various parts of a given query.

1.2.3 Database Oriented Mashups

Mashups are the web applications that aggregate functionality, presentation, and/or contents from existing sources to create a new application. Mashups consume the available data from third parties. Contents are usually generated either using web feeds or an application programming interfaces (API's). All the contents are combined either on client side using client-side scripts or on server-side using any available server-side technology. Mashups are different from traditional web applications because they are usually dynamically created to serve a very specific and short lived task. Several mashups editors have been launched to encourage people to build new applications using the massive amount of publicly available contents.

However, the limitation of existing mashups editors is that they focus only on web feeds or API's. These web feeds can represent simple information but lack the capability to represent/query data items provided by querying interfaces or data services [78]. The development of data mashups to deal with complex data structures requires strong programming skills, hence diminishing the mashups feature of easiness for novice user.

Existing data mashups tools cannot deal with structural and semantic diversities of heterogeneous data sources. Recently, the importance of using data mashups for the data integration using database oriented mashups has been realized [78]. Inspired by Yahoo pipes, MashQL [43] and Deri Pipes [34] are few attempts to generate semantic queries from data mashups, but up to

our knowledge there is no existing data mashups tool which enhances the mashups capability to formulate queries over web data sources using their respective query languages and at the same time deals with the heterogeneity of the data sources.

1.3 Contributions

In this section the main contributions of this thesis are represented alongside the references to the publications which resulted from the research activities. The main contributions of this thesis are categorized into three underlying categories.

1.3.1 DeXIN: Distributed extended XQuery for Data INtegration

The main topic of our research was distributed heterogeneous data integration. In order to deal with the previously mentioned challenges of XML and RDF data integration, we present DeXIN [3] (Distributed extended XQuery for heterogeneous Data INtegration), an extensible framework for distributed query processing over heterogeneous, distributed and autonomous data sources. DeXIN considers one data format as the basis (the so-called “aggregation model”) and extends the corresponding query language to executing queries over heterogeneous data sources in their respective query languages.

We come up with an extension of XQuery which covers the full SPARQL language and supports the decentralized execution of both XQuery and SPARQL in a single query. We have implemented XML as aggregation model and XQuery as the corresponding language, into which the full SPARQL language is integrated. However, our framework is very flexible and could be easily extended to further data formats (e.g., relational data to be queried with SQL) or changed to another aggregation model (e.g., RDF/OWL rather than XML). DeXIN decomposes a user query into subqueries (in our case, XQuery or SPARQL) which are shipped to their respective data sources. These queries are executed at remote locations. The query results are then transformed back into the aggregation model format (for converting the results of a SPARQL query to XML, we adhere to the W3C Proposed Recommendation [9]) and combined to the overall result of the user query. It is important to note that in contrast to the transformation based approaches only the results are transformed to a common format.

We have implemented DeXIN and carried out experiments, which document the good performance and reduced network traffic achieved with our approach [4].

1.3.2 Concerns Aware Querying

Another topic of our research was to incorporate data concerns into query languages for data integration applications. We design a new concerns aware querying system which takes data concerns into account and data concerns are directly integrated into the XQuery language [5]. We achieved the main goal of this research activity by designing a querying system (i) which can take arbitrary data concerns into account, (ii) which integrates the data concerns awareness into the query language, and (iii) which automatically selects the appropriate data sources depending on current context, user requirements and data concerns.

We lay the foundations of our study by identifying the actors involved in data services and their specific concerns. In contrast to common web services, data services have normally three actors involved, namely consumer, service provider, and also a data provider. We present four possible models of data concerns aware querying. We discuss the characteristics and virtues of each model and select the best suited one for our system. We come up with our data concerns aware querying system. We describe how metadata is organized and stored by this system and how data concerns awareness is integrated directly into the XQuery language.

1.3.3 Data Mashups for Data Services Integration

Finally, new possible technologies and approaches for data integration have been investigated, in particular to provide an easy access for novice users to benefit from previous mentioned achievements of concerns aware querying for the distributed heterogeneous data sources integration.

We utilize the concept of data mashups and use it to dynamically integrate heterogeneous web data sources by using the XQuery extension proposed in DeXIN. All the available data sources over the internet are considered as a huge database and each data source is considered as a table. Data mashups can generate queries in extended XQuery syntax and can be executed over any available data source contributing to the mashup generation.

We propose a query based aggregation of multiple heterogeneous data sources by combining powerful querying features of XQuery with an easy interface of mashups tools. The novelty of our tool is that it augments the powerful features of database querying to the data mashups tools. It provides an easy to use interface of mashup editor to generate complex queries visually for the integration of a multitude of distributed, autonomous, and heterogeneous data sources.

1.4 Structure of the Thesis

This thesis is structured as follows:

Chapter 2 discusses the state of the art in the field of data integration. A broad overview of database models and their query languages is given which is proceeded by a brief description of the related work of distributed XQuery (DXQ) and distributed SPARQL (DARQ). We conclude this chapter by relating our novel work with the existing work on heterogeneous data sources integration.

Chapter 3 In this chapter we present DeXIN (**D**istributed **e**xtended **X**Query for heterogeneous **D**ata **I**ntegration) – an extensible framework for distributed query processing over heterogeneous, distributed and autonomous data sources. DeXIN integrates multiple, heterogeneous, highly distributed and rapidly changing web data sources in different formats, e.g. XML, RDF and relational data.

Chapter 4 The data published over the web is often associated with data concerns like privacy, licensing, pricing, quality of data, etc. In this chapter we focus on the importance of ensuring data concerns for the data integration systems. A new concerns aware querying system is proposed for data services integration to ensure that data consumers utilize the

data in the right way and are bound to the rules and regulations defined by the data owner and service provider. It supports automatic service selection based on data concerns and perfectly fits into dynamic data integration applications.

Chapter 5 Mashups use third parties data sources and aggregate them to build a new web application mostly using RSS and atom feeds. We utilize the concept of data mashups and use it to dynamically integrate heterogeneous web data sources by using the XQuery extension proposed in DeXIN. By using our approach data mashups can be strengthen for querying and integrating complex data structures while using a graphical and easy to use interface of mashups for the novice users.

Chapter 6 Finally, the thesis is concluded with possible research direction for future work in Chapter 6.

Foundations and Related Work

In this chapter the state of the art in the field of the data integration is presented together with a brief description of the main works and concepts of the heterogeneous data integration, data transformation and distributed query processing. We begin with the theoretical background of data integration and give a brief overview of different data models, their underlying data structures and query languages. We discuss existing related work of distributed query processing and data transformation of XML and RDF data sources. We justify our novel approach of integrating heterogeneous distributed data sources by relating our work with the state of the art. We conclude this chapter by a brief discussion on existing data integration approaches for web data sources and comparing them with our contributions.

2.1 Data Integration: Theoretical Background

Data integration is the problem of combining data residing at different sources and providing the user with a unified view of these data sources [38]. [51] defined Data Integration systems formally as a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where

- \mathcal{G} is the global schema, expressed in a language $\mathcal{L}_{\mathcal{G}}$ over an alphabet $\mathcal{A}_{\mathcal{G}}$. The alphabet comprises a symbol for each element of \mathcal{G} (i.e., relation if \mathcal{G} is relational, class if \mathcal{G} is object-oriented, etc.).
- \mathcal{S} is the source schema, expressed in a language $\mathcal{L}_{\mathcal{S}}$ over an alphabet $\mathcal{A}_{\mathcal{S}}$. The alphabet $\mathcal{A}_{\mathcal{S}}$ includes a symbol for each element of the sources.
- \mathcal{M} is the mapping between \mathcal{G} and \mathcal{S} , constituted by a set of assertions of the forms

$$\begin{aligned} \mathcal{Q}_{\mathcal{S}} &\rightsquigarrow \mathcal{Q}_{\mathcal{G}}, \\ \mathcal{Q}_{\mathcal{G}} &\rightsquigarrow \mathcal{Q}_{\mathcal{S}} \end{aligned}$$

where Q_S and Q_G are two queries of the same arity, respectively over the source schema \mathcal{S} , and over the global schema \mathcal{G} . Queries Q_S are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ over the alphabet $\mathcal{A}_{\mathcal{S}}$, and queries Q_G are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ over the alphabet $\mathcal{A}_{\mathcal{G}}$. Intuitively, an assertion $Q_S \rightsquigarrow Q_G$, specifies that the concept represented by the query Q_S over the sources corresponds to the concept in the global schema represented by the query Q_G (similarly for an assertion of type $Q_G \rightsquigarrow Q_S$).

2.2 Data Models

A data model is the foundation of any database systems. It describes the structure and organization of the data, constraints defined over data and rules for manipulation of data. A data model comprises of

- a data structure,
- a set of integrity constraints, and
- operations associated with the data structures.

Different database models have been defined to meet different requirements. Each model is appropriate to a particular type of problem or requirement. In this section we discuss three popular data models namely (i) Relational Data Model (ii) Semi-structured Data Model and (iii) Semantic Data Model. We briefly elaborate the underlying data structure of each data model, schema definition and query languages.

2.2.1 Relational Data Model

2.2.1.1 Relational Data

The relational model for data management was first presented by E. F. Codd in 1970 [21]. This paper proposed the relational model for data management and laid the theoretical foundation of relational databases. This model is based on first order predicate logic and describes a database as a collection of predicates over a finite set of predicates variables, describing constraints on the possible values and combination of the values. The term relational model refers to the broad class of database models that have relations as the data structure and that incorporate some or all of the query capabilities, update capabilities and integrity constraints.

The relational model is based on simple and easy to understand data structures known as “relations”. The simplicity of relations makes databases accessible to a broad range of users and provides an excellent framework for the first generation of theoretical research into the properties of databases [1].

A relational data structure is a collection of tables or relations which have the following features.

- A relation is a collection of rows and tuples.
- A tuple is a collection of columns and attributes.

Movies	Title	Director	Actor
	The Trouble with Harry	Hitchcock	Gwenn
	The Trouble with Harry	Hitchcock	Forsythe
	The Trouble with Harry	Hitchcock	MacLaine
	The Trouble with Harry	Hitchcock	Hitchcock

	Cries and Whispers	Bergman	Andersson
	Cries and Whispers	Bergman	Sylwan
	Cries and Whispers	Bergman	Thulin
	Cries and Whispers	Bergman	Ullman
Location	Theater	Address	Phone Number
	Gaumont Opera	31 bd. des Italiens	47 42 60 33
	Saint Andre des Arts	30 rue Saint Andre des Arts	43 26 48 18
	Le Champo	51 rue des Ecoles	43 54 51 60

	Georges V 1	44 av. des Champs Elysees	45 62 41 46
	Les 7 Montparnassiens	98 bd. du Montparnasse	43 20 32 20
Pariscope	Theater	Title	Schedule
	Gaumont Opera	Cries and Whispers	20:30
	Saint Andre des Arts	The Trouble with Harry	20:15
	Georges V	Cries and Whispers	22:15

	Les 7 Montparnassiens	Cries and Whispers	20:45

Table 2.1: A Sample Relational Database [1].

- A domain is a pool of values from which the actual attribute values are taken.

Table 2.1 shows a sample cinema relational database. Data is represented as tables which consists of a group of rows, where each row represents a specific object with uniform structure.

2.2.1.2 Relational Schema

A relational schema is a formal way to specify the structure and organization of the relational data. Formally, a schema is a set of formulas that describe the constraints imposed on the data structure. In [1], a database schema is defined as a nonempty finite set \mathcal{R} of relation names. This can be written as:

$$\mathcal{R} = \{ \mathcal{R}_1[\mathcal{U}_1], \dots, \mathcal{R}_n[\mathcal{U}_n] \}$$

where \mathcal{R} is the relation schema over finite set of relations \mathcal{R}_1 to \mathcal{R}_n , while \mathcal{U} is finite number of attributes in a relation.

For example, the database schema for the database shown in Table 2.1 is defined as

CINEMA = Movies, Location, Pariscope

where

Movies = Title, Director, Actor

Location= Theater, Address, Phone Number

Pariscopes= Theater, Title, Schedule.

2.2.1.3 Structured Query Language

Structured Query Language also known as SQL (pronounced SEQUEL) is a standard language for accessing the relational database management systems (RDBMS) [26]. SQL is both data manipulation language (DML) and data definition language (DDL). Being DML, it can execute queries, retrieve, insert, update and delete data records in a database while being DDL it can create new databases, new tables/relations, stored procedures and views in a database. SQL can also be used to set permissions, constraints and restrictions.

Although SQL is an ANSI standard, there are many different versions and extensions of SQL language. However basic SQL statements which are supported by all versions of the language include:

- CREATE : to create a new data structure.
- SELECT : to select one or more rows from a table.
- INSERT : to add one or more rows into a table.
- DELETE : to remove one or more rows from a table.
- UPDATE : to update the values of columns in a row.
- DROP : to delete a data structure.

Table 2.2 shows a sample SQL SELECT query executed over a relational database.

2.2.2 Semi-structured Data Model

The semi-structured data model is a very popular data model for the internet applications. In the semi-structured data model, the information is usually modeled as a graph/tree with labels. The semi-structured data is self-describing and the information associated with a schema is contained within the data. The semi-structure data model is very popular for data sources such as the web, which can be treated as a database but cannot be constrained by a schema. The flexible data format is also desired by applications which need to exchange data on different platforms and data structures.

```

SELECT books.title, authors.name
FROM
    books, authors
WHERE
    books.id = authors.boookid
AND
    books.publishingdate > '01/01/2006'

```

Table 2.2: A Sample SQL Query

```

<?xml version="1.0" encoding="UTF-8"?>
<institute >
    <name>Information Systems </name>
    <groups>
        <group id='184/2'>
            <gname>Database & Artificial Intelligence Group</gname>
            <professors>
                <professor id='18421'>
                    <fname>Georg</fname>
                    <lname>Gottlob </lname>
                    <address></address>
                    <hpage></hpage>
                    <phone></phone>
                    <email></email>
                </professor>
                <professor id='18422'>
                    <fname>Reinhard </fname>
                    <lname>Pichler </lname>
                    <address></address>
                    <hpage></hpage>
                    <phone></phone>
                    <email></email>
                </professor>
            </professors>
        </group>
    </groups>
</institute >

```

Figure 2.1: A Sample XML Document

2.2.2.1 XML Data Structure

The Extensible Markup Language (XML) is a W3C standard to describe semi-structure data. It is a simple, very flexible text format derived from SGML (ISO 8879) [16], originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the web and elsewhere [15].

The Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. XML was created so that richly structured documents could be used over the web. The design goals of XML emphasize simplicity, generality, and usability over the internet.

XML provides a generic syntax used to mark up data with human readable tags. There are no predefined tags for the XML elements, but developers can define their own tags and structure of the document, further it is flexible and extensible according to the application requirements.

The information in an XML document is stored in the hierarchy of elements that have names, optional attributes, and optional contents. The content of an element may consist of text, nested elements, or some mixture of these. The contents of an XML document has an intrinsic order, and it is important to preserve this order [19].

The W3C standard for XML [15] describes that an XML document is well formed if

- it contains one or more elements;
- there is exactly one element, called the root;
- for all other elements, the elements delimited by start- and end-tags should be nested properly within each other.

Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures. Many technologies e.g. web services are taking advantage of XML and nowadays XML is de facto data exchange language over the web. Many XML based languages have been developed including RSS, Atom, XHTML and SOAP etc. Figure 2.1 shows a sample XML document.

2.2.2.2 DTD/XML Schema

Even though XML documents are self describing and may come without any schema or constraints, but still schema definitions may be necessary for defining XML document structure and to impose constraints and restrictions which cannot be imposed by XML itself. W3C XML Schema is an XML-related standard that contains powerful and expressive methods to describe an XML related standard [80]. XML Schema expresses shared vocabularies and allows machines to carry out rules made by people. They provide means for defining the structure, contents and semantics of XML documents. The constraints can define valid elements sequences and nesting, correct data type of element text or attribute values. Figure 2.2 shows a sample XML schema document excerpt for the sample XML document shown in Figure 2.1.

2.2.2.3 XQuery

XQuery is a standardized language for querying XML documents [18]. It uses the structure of XML and can express queries across all kinds of XML data whether physically stored in XML or

```

<?xml version="1.0" encoding="utf-16"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="institute" />
  <xsd:complexType name="institute">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="groups" />
    </xsd:sequence>
  </xsd:complexType>
  .....
  <xsd:complexType name="professor">
    <xsd:sequence>
      <xsd:element name="fname" type="xsd:string" />
      <xsd:element name="lname" type="xsd:string" />
      <xsd:element name="address" type="xsd:string" />
      <xsd:element name="hpage" type="xsd:string" />
      <xsd:element name="phone" type="xsd:string" />
      <xsd:element name="email" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:int" />
  </xsd:complexType>
</xsd:schema>

```

Figure 2.2: A Sample XML Schema Document Excerpt

```

let $auction := doc("auction.xml")
return let :=
$auction/site/closed_auctions/closed_auction return
let := $auction/site/regions/europe/item
for $p in $auction/site/people/person
let $a := for $t in $ca
  where $p/@id = $t/buyer/@person
  return
  let $n := for $t2 in $ei
    where $t/itemref/@item = $t2/@id return $t2
  return <item>{$n/name/text()}</item>
return <person name="{ $p/name/text()}">{ $a }</person>

```

Figure 2.3: A Sample XQuery

viewed as XML. It helps in combining documents, databases, and web pages. It is very widely implemented and nowadays XQuery is a de facto language for querying XML documents. It is powerful, easy to learn and designed to be a language in which queries are concise and easily understood. It is also flexible enough to query a broad spectrum of XML information sources, including both databases and documents.

XQuery is replacing proprietary middle-ware languages and other web application's development languages. XQuery is replacing complex Java or C++ programs with a few lines of code. XQuery is simpler to work with and easier to maintain than many other alternatives [26, 28].

2.2.3 Semantic Data Model

The semantic data model brings 'semantics or meanings' into the data model, which enables machines to interpret the semantics of instances without any prior knowledge of a meta model. The most popular form of semantic data model is the semantic web, which is a web of data to provide semantics of the information available over world wide web [49]. The semantic web provides a common format for the integration and combination of the data from diverse data sources. It also provides syntax to record how the data is related to the real world objects. The web of data enables new applications which allow its users to query one data sources and then navigate/crawl to an unbounded number of related data sources till the required information is met.

2.2.3.1 Resource Description Format

The Resource Description Format (RDF) is a framework for representing information in the web. It is a W3C standard for describing and interchanging metadata using XML [9]. It provides an abstract syntax to define semantics and links within the data objects and subjects. The design goal of RDF is intended to provide a simple data model having formal semantics and inference using an extensible URI-based vocabulary over XML syntax. It is XML based and uses XML schema data types which makes RDF also an extensible data format.

Figure 2.4 shows a simple RDF triple graph structure. The nodes of an RDF graph are its subjects and objects. A directed arc is a predicate which defines the relationship between subject and object. The direction of arc always points towards object. The structure of any RDF expression is in the form of triples, each consisting of a subject, a predicate and an object, where

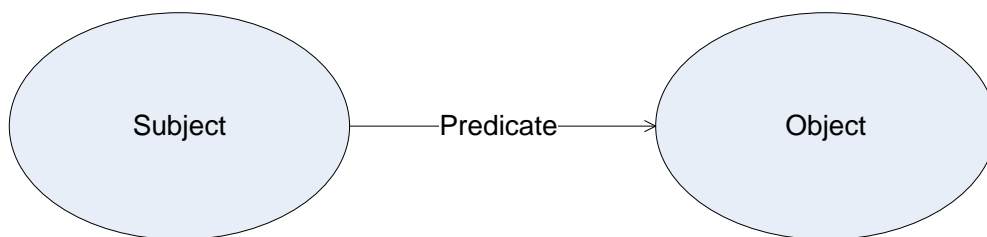


Figure 2.4: RDF Triple

```

<rdf:Description rdf:about=
    "http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about=
    "http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:fullName>Dave Beckett </ex:fullName>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about=
    "http://www.w3.org/TR/rdf-syntax-grammar">
  <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
</rdf:Description>

```

Figure 2.5: A Sample RDF document [9]

subject and predicate are resources or URI-s of the web, while objects can be either URI or a literal. RDF is a general model for such triples which has various machine readable formats e.g. RDF/XML, Turtle and N3 etc. A set of triples is called RDF graph. RDF triples form a directed labeled graph. Figure 2.5 shows a sample RDF/XML document.

2.2.3.2 RDF Vocabulary Description Language

The RDF Vocabulary Description Language also known as RDF Schema (RDFS) specifies the terms and conditions to define RDF statements. It defines which terms we can use, what kind of restrictions apply to these terms and what kind of extra relations are there between resources and objects? RDFS defines resources, classes and relationships among them. RDFS formalized the notion of classes, their instances and subclasses to provide general hierarchies. Resources can be divided into groups called classes while members of classes are known as instances of the class. The `rdf:type` property can be used to state that a resource is an instance of a class.

```

<rdf:Description rdf:ID="Novel">
  <rdf:type rdf:resource="http://www.w3.org
    /2000/01/rdf-schema#Class"/>
</rdf:Description>

```

Figure 2.6: RDF Schema Vocabulary Document

2.2.3.3 Ontology Web Language

The OWL Web Ontology Language is intended to provide a language to describe the classes and relations between them that are inherent in web documents and applications. OWL formalizes a domain to define both classes and individuals including their properties. It also facilitates reasoning about classes and individuals according to the degree of permission by formal semantics of the OWL language.

The OWL Web Ontology Language is designed for use by applications that need to process the contents of information instead of just presenting information to humans. It supports richer expressiveness than RDF schema, hence considered as an extension of RDF Schema. OWL facilitates greater machine interpretability of web contents as compare to XML, RDF, and RDF Schema (RDF-S). Because of its rich expressiveness it provides additional vocabulary along with formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full [58].

All the sub languages are categorized based on their expressiveness.

- OWL-Lite is the syntactically simplest of the all three sub languages. It supports users primarily needing a classification hierarchy and simple constraints. OWL Lite has lower complexity and provides a quick support for simple tools.
- OWL-DL introduces the addition of description logic in OWL-Lite which provides more expressiveness. OWL DL includes almost all OWL language constructs with some minor restrictions. OWL-DL retains computational completeness and decidability.
- OWL Full is for the users who want maximum expressiveness and syntactic freedom of RDF. There is no guarantee of being computationally complete and decidable with the RDF full.

2.2.3.4 SPARQL

The SPARQL Protocol and RDF Query Language (SPARQL) is a W3C recommended query language for querying RDF data [66]. SPARQL queries can contain triple patterns, conjunctions, disjunctions and optional patterns which allow us query the semantic web data. It can explore data by querying unknown relationships, perform complex joins in a single query and transform RDF data from one vocabulary to another.

A SPARQL query consists of

- Prefix declaration: It specifies an abbreviation for the URIs.


```

PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x ns:price ?price .
          FILTER (?price < 30.5)
          ?x dc:title ?title . }

```

Figure 2.7: A Sample SPARQL Query [66]

- Data set definition: An RDF graph which will be queried.
- Result clause: Defines the results of the query.
- Query pattern: Specifies the query which describes what information is required.
- Query modifiers: Ordering, arranging or pattern of the query results.

SPARQL queries can be executed against RDF data consisting of RDF graphs. There are several generic SPARQL end points available which allow users to query any web accessible RDF data. The results of a SPARQL query can be returned in multiple formats according to the user requirements e.g. XML, RDF, JSON and HTML etc.

A SPARQL query can have any of the following four forms.

- SELECT: Returns matching patterns of all, or subset of the bounded variables in a query.
- CONSTRUCT: Returns an RDF graph constructed by replacing variables in a set of triple templates.
- ASK: Returns a boolean value either true or false indicating whether a matching query pattern is available or not.
- DESCRIBE: Returns an RDF graph that describes the resources found.

Figure 2.7 shows a sample SPARQL select query with numeric values filter. All the variables in SPARQL query are mentioned using “?” before each variable name. While Figure 2.8 shows result of the SPARQL query in RDF/XML format.

2.3 Data Transformation (XML ↔ RDF)

Data transformation is one of the techniques to integrate data of different formats. In Section 2.2 we discussed the three most popular data formats. It is a matter of fact that there exist various data formats and their number is at continuous increase. Each data format is designed to fulfill particular requirements. For example some data models are designed to provide rich semantics while others are designed to fit in the lightweight and simple applications.

```

<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="title"/>
    <variable name="price"/>
  </head>

  <results>
    <result>
      <binding name="title"> The Semantic Web </binding>
      <binding name="price"> 10.5 </binding>
    </result>
  </results>
</sparql>

```

Figure 2.8: SPARQL Query Results in XML/RDF Format

Each data format fits into its own domain and it can easily be predicted that in the future the emergence of further data models will continue. To provide an integrated view over multiple heterogeneous distributed data sources is an utmost important requirement for applications dealing with various data formats. Below we discuss data transformation techniques for the data integration. For the sake of simplicity and sticking to the scope of the work of the thesis, we discuss data transformation techniques just for transformation between the XML and RDF data models.

2.3.1 GRDDL

GRDDL is a mechanism for **G**leaning **R**esource **D**escriptions from **D**ialects of **L**anguages. It is a W3C recommendation and is developed to “glean” RDF data from XML documents and XHTML pages [22]. It provides a set of mechanism and transformation algorithms for RDF content generation from XML and XHTML. XSLT [47] is widely used to obtain RDF data from XML, but GRDDL was intended to allow an other implementation using a standard transform library.

The data transformation mechanism is very simple for both XHTML and XML documents. GRDDL provides a number of ways for transformation to be associated with the contents, each of which is appropriate in different situation. Usually information related to transformation algorithms is included in individual documents using a reference to the transformation document. Below we discuss the XML transformation into RDF using GRDDL mechanism. We exclude the XHTML document transformation into RDF because of being out of the scope of this thesis. GRDDL provides use case scenarios of extracting RDF data from XML documents [32]. The GRDDL specification introduces markup for declaring that an XML document includes gleanable data and for linking to an algorithm, typically represented in XSLT, for gleaned the RDF

```

(1)    declare namespace foaf="http://xmlns.com/foaf/0.1/";
(2)    <results>{ for ($n, $m) in
(3)        SELECT ?name ?mbox
(4)        WHERE { ?x foaf:name ?name .
(5)                ?x foaf:mbox ?mbox .
(6)        FILTER regex(str(?mbox), "@work.example") }
(7)    return <result><name>{$n}</name><mbox>{$m}
(8)    </mbox></result>}</results>

```

Figure 2.9: Embedded SPARQL inside XQuery

data from the document.

The markup includes a namespace-qualified attribute for use in general-purpose XML documents and a profile-qualified link relationship for use in valid XHTML documents. The GRDDL mechanism also allows an XML namespace document (or XHTML profile document) to declare that every document associated with that namespace (or profile) includes gleanable data and for linking to an algorithm for gleanable data.

Just like a profile transformation, an XML namespace can have a transformation associated with it. This allows entire XML dialects (for instance, XML or Atom) to provide meaningful RDF. An XML document simply points to a namespace and when fetched, this namespace points to a namespace transformation.

This also allows very large amounts of the existing XML data in the web to become RDF/XML with a minimal effort from the namespace author. Once a document has been transformed, there is an RDF representation of that data. This output is generally put into a database and queried via SPARQL.

2.3.2 Embedding SPARQL into XQuery/XSLT

Another approach to provide uniform access over RDF and XML data is to embed a query language of one format into the query language of other format. [35] uses this approach and embedded SPARQL queries inside XQuery/XSLT. All SPARQL queries are embedded into XQuery/XSLT and automatically transformed into pure XQuery/XSLT queries to be posed against pure XML data after transformation of RDF data into XML. This embedding enables users to benefit from graph and tree language constructs of both SPARQL and XQuery. The authors defined a formal SPARQL algebra to transform a SPARQL query into an operator tree of SPARQL algebra. The operator tree is later translated into XQuery/XSLT.

Figure 2.9 shows an example XQuery with SPARQL embedded inside XQuery. The embedded SPARQL returns the names and email addresses of those people, the email addresses of which contain “@work.example”. The result of a SPARQL query is a bag (also called multiset), i.e. its unordered elements can appear more than once. The proposed extension of XQuery iterates through all environments of the result of S, where S is a SPARQL SELECT subquery,. In each iteration the values of bound variables of the current environment are mapped to the XQuery

variables which can be afterwards used in normal XQuery/XSLT instructions. SPARQL ASK queries return a boolean value, such that an embedding of ASK subqueries in boolean expressions of XQuery is natural.

The work presented in [35] is not an exclusive work on SPARQL embedded into XQuery. There exists some other similar work with slightly different approach also available in the literature. Contrary to embedding SPARQL into XQuery/XSLT, there are also some efforts to embed XPATH and XQuery into SPARQL queries [13, 27].

2.3.3 XSPARQL

Despite the availability of GRDDL transformation sets the translating between XML and RDF is a tedious and error prone task. In [2], a new query language based approach is used to transform XML into RDF and vice versa. XSPARQL is a query language combining XQuery and SPARQL for transformations between RDF and XML. XSPARQL subsumes XQuery and most of SPARQL (excluding ASK and DESCRIBE). Conceptually, XSPARQL is a simple merge of SPARQL components into XQuery. XQuery is a native Query language in XSPARQL and all XQuery queries are also considered as XSPARQL queries.

Table 2.3 gives a schematic view of the XSPARQL query. In order to execute SPARQL queries inside the body of XQuery, the XQuery FLWOR expression is slightly modified which they called FLWOR' expression. Concerning semantics, XSPARQL equally builds on top of its constituent languages. They extended the formal semantics of XQuery by additional rules which reduce each XSPARQL query to XQuery expressions; the resulting FLWORs operate on the answers of SPARQL queries in the SPARQL XML result format. All XSPARQL queries are rewritten in XQuery standard format while SPARQL queries are executed over SPARQL endpoints and results are returned in RDF/XML.

Following are the main features of XSPARQL.

- In the body XSPARQL allows SPARQL-style F'DWM blocks alternatively to XQuery's FLWO blocks. The new F' clause of the form for varlist is very similar to XQuery's native for clause, but instead of allowing a single variable (which is assigned to the results of an XPath expression), the new clause supports a white space separated list of variables (varlist). Each variable in varlist is then assigned the value resulting from evaluating a SPARQL query of the form:

select varlist DWM.

- In the head XSPARQL allows to create RDF graphs directly using construct statements (C) alternatively to XQuery's native return (R).
- Different forms of nesting are allowed, such as subqueries constructing RDF graphs appearing in let assignments or within SPARQL style from clauses, or value construction within SPARQL-style construct clauses.

Prolog:	P	declare namespace prefix=“ <i>namespace-URI</i> ” or prefix prefix: <namespace-URI>	
Body:	F L W O	for var [at posVar] in FLOWR’ expression let var := <i>FLWOR</i> ’ expression where <i>FLWOR</i> ’ expression order by <i>FLWOR</i> ’ expression	or
	F’ D W M	for varlist [at posVar] from / from named (<dataset-URI> or <i>FLWOR</i> ’ expr.) where pattern order by expression limit integer > 0 offset integer > 0	
Head:	C	construct template (with nested <i>FLWOR</i> ’ expressions)	or
	R	return <i>XML+ nested FLWOR’ expressions</i>	

Table 2.3: Schematic View of XSPARQL [2]

2.4 Distributed Query Processing (SPARQL and XQuery)

Both SPARQL and XQuery are query languages for web data sources. In the internet era, a huge amount of data is available over the web. This data is highly distributed because of the distributed nature of the internet. SPARQL and XQuery are designed based on the principle of fetching and querying data locally. It works fine with small files and data structures but fetching huge web data sources can easily become a bottleneck for distributed query execution. In this section we discuss different distributed XQuery and SPARQL query execution techniques.

2.4.1 DXQ

Distributed XQuery (DXQ) is an extension of XQuery to support the effective and efficient development of distributed XML applications [30]. The declarative nature of XQuery, combined with its support for XML processing, makes it well suited for rapid development of complex distributed systems, in particular web services, peer-to-peer applications, and distributed resource-management (DRM) applications.

A DXQ server is a complete query processor for XML, and can act as a client as well as a server. It exports a module written in DXQ, and accepts arbitrary DXQ queries that can invoke the server’s exported functions and even cause the server to query other servers. For example, a DNS resolver works by making a series of database queries, starting at a root name server, and following delegations until reaching a name server that has the final answer (a hostname’s IP address). This series of queries is naturally expressed as a single DXQ query, where following a delegation corresponds to a join of remote databases, computed at the resolver (the client). In contrast, a multicast can be expressed as a DXQ query that causes the server to forward messages to all child servers in the multicast tree, on behalf of the original client.

```

Interface ::= interface namespace NCName =
                URILiteral ; InterfaceProlog
Module ::= module namespace NCName = URILiteral
                (implements URILiteral)?; ModuleProlog
InterfaceImport ::= import interface namespace
                NCName = URILiteral
Expr ::=
| let server NCName implement NCName at Expr return Expr
| from server NCName return Expr
| at server NCName do Expr

```

Figure 2.10: DXQ Grammar Extensions

```

1. import interface Server = ‘‘http://www.dns.org/Server’’;
2. import module U = ‘‘DNSUtility’’;
3. declare function Resolver:lookup($x,$n) {
4. <rr>{
5. let server S implement Server at $x return
6. let $rr := from server S return S:resources()
7. return
8. $rr/a[@host=$n],
9. (for $ns in $rr/ns, $a in $rr/a
10. where $ns/@serv=$a/@host
11. and fn:not($ns/@dom = ‘.’))
12. and U:hostname-1t($ns/@dom,$n)
13. return Resolver:lookup($a/@addr, $n)/a)
14. }</rr>
15. };

```

Figure 2.11: A DNS Resolver in DXQ

Figure 2.11 contains a client program that implements the DNS resolver algorithm. It imports the DNS server interface on Line 1, which associates the namespace prefix *Server* with the module interface in <http://www.dns.org/Server>.

The language extension is small, but provides a powerful mechanism for distributed XML programming. For example, the core DNS resolver algorithm in Figure 2.11 is implemented by only ten-line, recursive lookup function.

2.4.2 XRPC

XRPC is an extension of XQuery to execute distributed XQuery queries over heterogeneous data sources [85]. It utilizes the concept of remote procedure calls to enhance XQuery functionality which executes XQuery for loop over multiple destinations using remote procedures calls.

XQuery is designed over the data shipping model for querying XML documents. When a user query is posted against a remote data source, the built in function of XQuery called `fn:doc()` fetches the entire XML document where it is queried locally. The amount of structured and semi-structured data in the form of XML is increasing rapidly and soon transporting the entire XML document will become a bottleneck for query execution over large XML data sets.

The goal of XRPC is to execute distributed XQuery over multiple XQuery processors at different locations in such a way that all the contributing XQuery processors can jointly execute a single distributed XQuery. This also raises the issue of network communication. In order to cope with this issue XRPC uses SOAP over HTTP for network communications. SOAP also brings an additional advantage of web services and Service Oriented Architecture (SOA) integration as XQuery data sources.

Another feature of XRPC is bulk RPC which allows to send multiple applications of the same functions (with different parameters) in a single request/response network interaction. Network traffic is biggest time factor in most of the distributed queries, bulk RPC helps to reduce network traffic by sending similar requests in bulk.

Remote function application take the following XQuery syntax:

```
execute at Expr FunApp ( ParamList )
```

where *Expr* is an `xs:string` expression of XQuery that specifies the URI of the peer on which *FunApp* is to be executed. The function to be applied can be either built-in or user-defined. For user-defined functions, it is only restricted to functions defined in an XQuery Module. A small extension to the network protocol allows functions defined inside the query to be executed over XRPC.

Figure 2.12 shows a sample distributed XQuery following XRPC syntax. An XQuery module `film.xq` is stored at `x.example.org`, the module defines a function called `filmsByActor()`. This function is executed over a remote peer using XRPC function “execute at”. It can also execute the same function on multiple remote peers.

2.4.3 DARQ

Integrated access to multiple distributed and autonomous data sources is a key challenge for many web applications. Semantic web popularity resulted in great amount of RDF data dis-

```

import module namespace f="films"
  at "http://x.example.org/film.xq";
<films> {
  for $actor in ("Julie Andrews", "Sean Connery")
  let $dst :=
    "xrpc://y.example.org"
  return
    execute at {$dst} {f:filmsByActor($actor)}}
</films>

```

Figure 2.12: A Sample XRPC Query

tributed all over the internet. As a reaction to this challenge, SPARQL, the current W3C Proposed Recommendation for an RDF query language, supports querying of multiple RDF graphs. Still the W3C recommendation for SPARQL [66] lacks the complete query federation over multiple distributed sources. This makes writing distributed SPARQL queries a hard and lengthy task. Furthermore, current implementations of SPARQL load all RDF graphs mentioned in a query to the local machine. This usually incurs a large overhead in network traffic, and sometimes is simply impossible for technical or legal reasons.

To overcome these problems a distributed SPARQL query called DARQ was proposed to provide an engine for federated SPARQL queries [67]. DARQ provides transparent query access to multiple SPARQL services, i.e., it gives the user the impression to query one single RDF graph despite the real data being distributed on the web. A service description language enables the query engine to decompose a query into sub-queries, each of which can be answered by an individual service. DARQ also uses query rewriting and cost-based query optimization to speed-up query execution. DARQ is available under GPL License at <http://darq.sf.net/>.

DARQ extends SPARQL query engine ARQ (included in Jena [44]) by adding a new query planning algorithm and a modified query execution engine. The work on DARQ includes a service description language and a basic query optimization algorithm. DARQ provides a single query interface (same as ARQ [45]), leaving the details of federation to the query engine.

2.5 Discussion

Query languages are designed while keeping in view a particular data model and its querying requirements. Each data model and its corresponding query language has its own domain and best serves the specific tasks for which it is designed. Some applications require an integrated view of the distributed heterogeneous data which have different data models. But sometimes it is not possible to use one query language to query the data represented in another data model.

We discussed in detail each of these data models and their query languages: SQL and the relational model of data, XQuery and its associated XML data model, and SPARQL and its associated RDF data model. It gives a better understanding of the relative positioning and value of the three languages, particularly as they apply to the concepts of the web data sources.

SQL is arguably the most widely used query language in the information management community. It is based on the relational model of data, a data model designed in terms of collections (“relations”, “tables”) of tuples (“rows”) of data. SQL has been standardized for more than 20 years and has a great many implementations, many of them “industrial-strength” in terms of robustness, manageability, performance, scalability, etc.

XQuery is designed to operate on an abstract XML data model, one based on trees. XQuery 1.0 and XPath 2.0 is the result of several years of research and development. XQuery, with its underlying data model, is a very powerful declarative, functional language for querying and transforming XML.

RDF is a “framework for representing information in the web”, usually envisioned as collections of triples (3-tuples), while SPARQL is designed for querying information expressed in RDF.

Different data models and their corresponding query languages exist to meet specific needs of different applications. Furthermore, new data models are emerging to fulfill the modern data application requirements. Restricting all the data applications to one data model is practically impossible because each data model and its query language has its own place in the ecosystem and it fulfills specific requirements which cannot be served by any other data model.

In order to deal with the integration of various data models different techniques are proposed which include data transformation, query rewriting, mediators etc. We discussed in Section 2.3 some of the data transformation techniques for XML to RDF transformation.

Another aspect of the web data is highly distributed data sources. Integrating such data sources which can be located physically apart and may or may not be known to each other before the integration process starts is a challenging task for web data applications. We discussed several distributed query processing techniques for XML and RDF data sources in Section 2.4. All these discussions lead to the conclusion that it is a matter of fact that there exist multiple heterogeneous data models and their query languages. Moreover, in future the emergence of further data models and their corresponding query languages cannot be excluded. The need of the time is to design the frameworks to build integrated applications which not only deal with the integration of heterogeneous distributed data sources, but also they are flexible enough to integrate further data models.

The main goal of this thesis is to provide an extensible framework for parallel query execution over distributed, heterogeneous and autonomous large data sources. The distribution factor cannot be ignored in web data integration so we proposed the data integration of distributed XML, RDF and OWL data sources in such a way that parallel queries can be executed in a distributed environment. Users of the integrated applications should get an illusion of interacting with a single data source despite the physical distribution of the data sources. In order to deal with the heterogeneity there should be no need to transform large data sources into a common format.

The related work of other contributions of this thesis namely data concerns and mashups is discussed in their corresponding chapters.

CHAPTER 3

DeXIN

In this chapter we present DeXIN (**D**istributed **e**xtended **X**Query for Data **I**ntegration). DeXIN is an extensible framework for providing integrated access over heterogeneous, autonomous, and distributed web data sources, which can be utilized for data integration in modern web applications and service oriented architecture. DeXIN extends the XQuery language by supporting SPARQL queries inside XQuery, thus facilitating the query of data modeled in XML, RDF, and OWL. DeXIN facilitates data integration in a distributed web and service oriented environment by avoiding the transfer of large amounts of data to a central server for centralized data integration and avoids the transformation of a huge amount of data into a common format for integrated access.

The work presented in this as well as in the subsequent chapters was conducted in the context of the SODI (Service Oriented Data Integration) project¹, where we aim that nowadays data integration is concerned with problems that arise from combining heterogeneous data sources and exchanging data between them, particularly web data sources. There has been considerable progress in this field over the past years. Nevertheless, current approaches to data integration do not yield satisfactory results in many realistic scenarios such as distributed heterogeneous web data sources. Particularly when the data sources are unknown at the design time. For such applications users need not just the actual data but also meta-information on these data which allows them to assess the usefulness of the data and to dynamically process and combine them with data from other sources.

This chapter focuses on our proposed framework DeXIN and describes the basic architecture of the framework. A detailed description of the query evaluation process is presented in this chapter. The work presented in this chapter has been published in [3]. We conclude this chapter with the experimental results as a proof of concept together with the DeXIN tool. The demo of the DeXIN tool is given in [4].

¹<http://www.dbai.tuwien.ac.at/proj/InfInt/>

3.1 Motivation

In the web environment, rich, diverse sources of heterogeneous and distributed data are ubiquitous. In fact, even the information characterizing a single entity - like, for example, the information related to a web service - is normally scattered over various data sources using various languages such as XML, RDF, and OWL. In recent years, there has been an enormous boost in semantic web technologies and web services. Web applications thus have to deal with huge amounts of data which are normally scattered over various data sources using various languages.

Hence, there is a strong need for web applications to handle queries over heterogeneous, autonomous, and distributed data sources. However, existing techniques do not provide sufficient support for this task. These applications are facing many major challenges, more importantly

- how to integrate *heterogeneous* data,
- how to integrate *highly distributed* data sources,
- how to deal with *rapidly growing* and *continuously changing* distributed data sources, and
- how to *dynamically select* and integrate a previously *unknown* data source.

The most important languages for specifying data on the web are, on the one hand, the Extensible Markup Language (XML) [15] and, on the other hand, the Resource Description Framework (RDF) [9] and Ontology Web Language (OWL) [58].

The XML is a very simple and flexible language which is a W3C standard to describe semi-structure data. XML is a very popular format to store and integrate a rapidly increasing amount of semi-structured data on the web and is a de facto language for data integration and data exchange. While the semantic web builds on data represented in RDF and OWL, which is optimized for data interlinking and merging. RDF provides an abstract syntax to present semantics and links of the data.

There exists a wide gap between these data structures, since RDF data (with or without the use of OWL) has domain structure (the concepts and the relationships between concepts) while XML data has document structure (the hierarchy of elements). Also the query languages for these data formats are different. For XML data, XQuery [18] has become the query language of choice, while SPARQL [66] is usually used to query RDF/OWL data.

It would clearly be useful to enable the reuse of RDF data in an XML world and vice versa. Many web applications have to find a way of querying and processing data represented in XML and RDF/OWL simultaneously.

3.2 Application Scenario

DeXIN can be profitably applied in any web environment where large amounts of heterogeneous, distributed data have to be queried and processed. A typical scenario where such a requirement arises is in the area of web service management.

The number of web services available for different applications is increasing day by day. In order to assist the service consumer in finding the desired service with the desired properties,

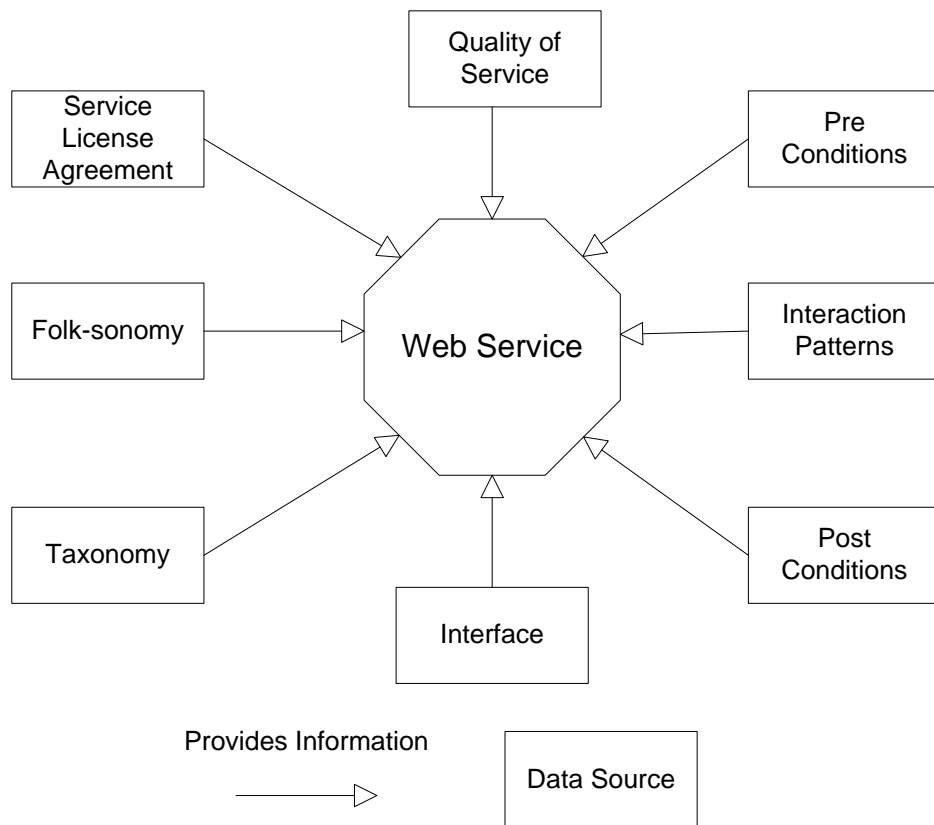


Figure 3.1: Data Sources Related to a Web Service [74]

several web service management systems have been developed. The Service Evolution Management Framework (SEMF)² [74] is one of these efforts to manage web services and their related data sources. SEMF describes an information model for integrating the available information for a web service, keeping track of evolutionary changes of web services and providing means of complex analysis of web services. SEMF facilitates the selection of the best web service from a pool of available web services for a given task.

Each web service is associated with different attributes which effect the quality of service. Figure 3.1 gives an impression of the diversity of data related to a web service. This data is normally scattered over various data sources using various languages such XML, RDF, and OWL. However, currently available systems do not treat these heterogeneous, distributed data sources in a satisfactory manner. What is urgently needed is a system which supports different query languages for different data formats, which operates on the data sources as they are without any transformations, and which uses decentralized query processing whenever this is possible. Moreover, this system should be flexible and allow an easy extension to further data formats. In

²We acknowledge the assistance of Martin Treiber (Distributed Systems Group ,Vienna University of Technology) for providing access to SEMF data.

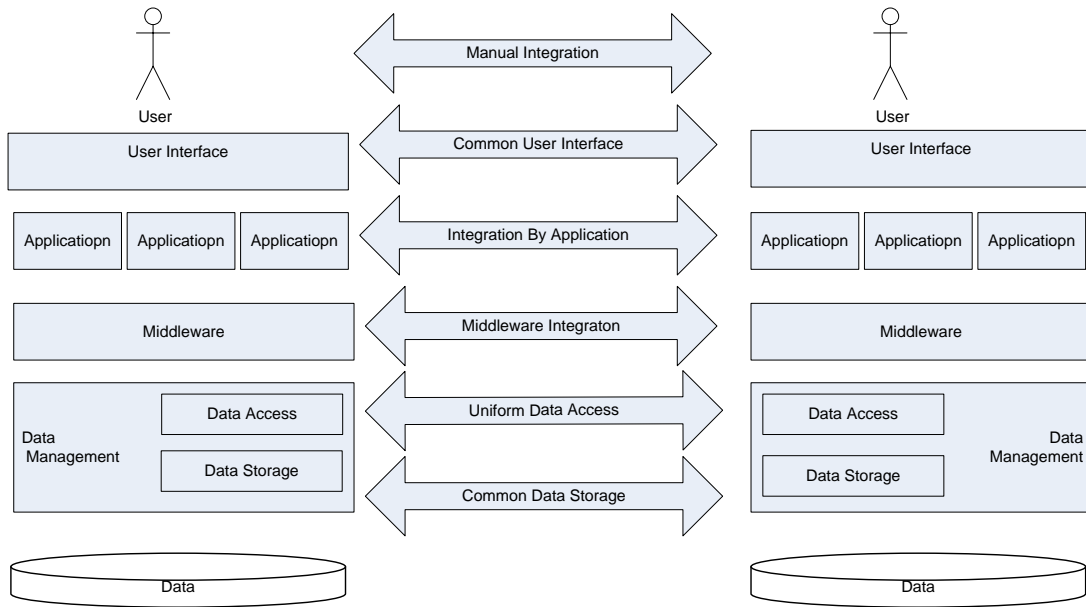


Figure 3.2: Data Integration Approaches on Different Architectural Levels [87].

fact, this is precisely the functionality provided by DeXIN.

3.3 Approaches for Heterogeneous Data Sources Integration

Heterogeneity of data has been a major issue since the evolution of different data types and different query languages have been designed for each data format. Each query language has its own role and importance to query its particular data sources. There are several ways to address the integration of heterogeneous data sources proposed by the scientific community over the last three decades.

Data integration approaches can be described using the layered architecture as shown in Figure 3.2. The problem of integration can be addressed on any architectural layer depending on the application requirements or constraints. Following are the principle approaches for the integration of heterogeneous data sources described in [87].

- **Manual Integration:** Users directly interact with all the relevant information and manually integrate selected data. Users need to have detailed knowledge of the location of data, its presentation and semantics.
- **Common User Interface:** A common user interface is provided which gives a uniform look and feel while data is still separately presented and has to be manually integrated by the user.

- **Integration by Application:** A data integration application accesses various data sources and returns integrated results to the user.
- **Integration by Middleware:** A middleware is introduced between data presentation and application, thus relieving the applications from the integration process.
- **Uniform Data Access:** A logical integration of the data is accomplished at the data access level. Data is still physically separated and distributed but a uniform data access illusion is provided with a unified global view.
- **Common Data Storage:** Data is physically integrated by transferring data to a new data storage or by transforming all data stores into one common format.

3.4 XML and RDF Data Sources Integration

With the advent of the semantic technologies RDF and OWL data formats are getting popularity and SPARQL is W3C recommendation as query language for semantic data. Some efforts have been done on transforming XML into RDF and vice versa, but currently transformation is a tedious and erroneous task with the help of available languages and tools for the transformation. There is strong need for the applications for effective and optimal integration of both XML and RDF data sources, particularly in a single query.

Figure 3.3 gives an idea of the issues which modern data integration applications face in a distributed environment. Contrary to legacy applications modern data integration applications are created on demand. Such applications has no prior knowledge of the data sources and all integrated sources are dynamically composed to give an integrated view to the user. Nowadays modern enterprises desire to have integrated applications which could easily deal with the issues of:

- heterogeneous data formats,
- highly distributed data sources,
- dynamic discovery, addition and deletion of a data source, and
- data provider's and data consumer's demands and concerns.

Below we discuss the issues of heterogeneity and distribution of data sources while remaining issues will be discussed in the subsequent chapter.

3.4.1 Heterogeneity

There are several approaches for dealing with *heterogeneous* data consisting of XML, RDF and OWL: The most common approach is to transform all data sources into a single format [22,35] and apply a single query language to this data. GRDDL provides a set of mechanism and algorithms for transformation of XML data into RDF. The information related to transformation

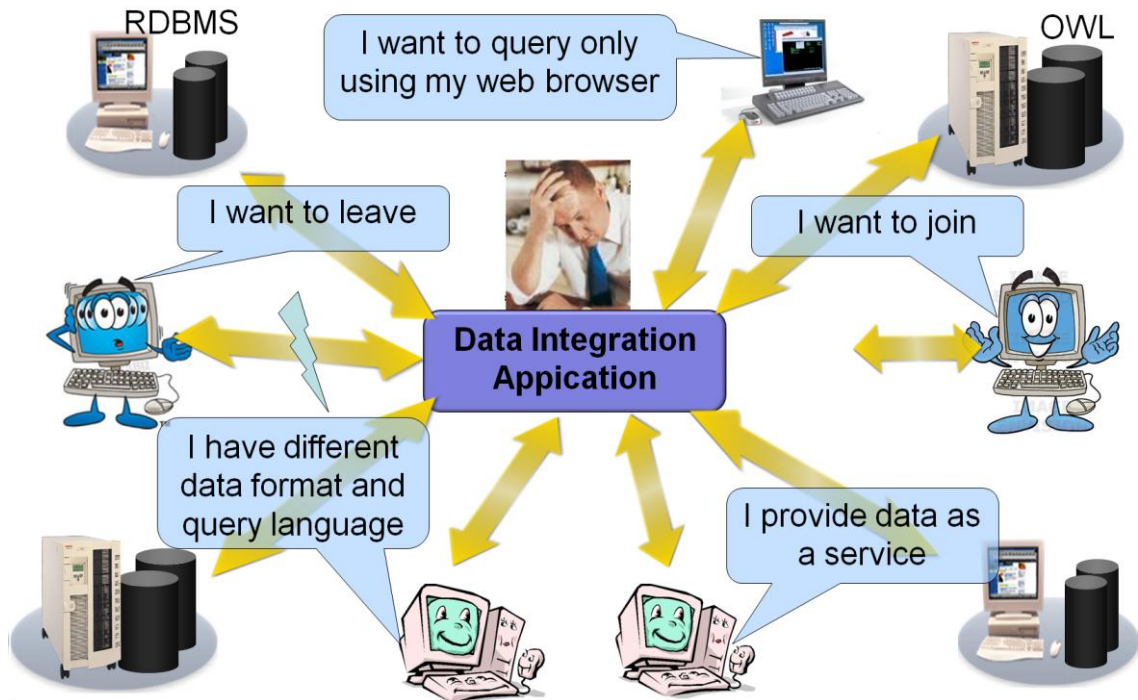


Figure 3.3: Typical Data Integration Application Issues

algorithm is embedded into individual documents which helps to generate RDF contents from the XML data sources.

According to GRDDL, all data sources can be transformed into a single language and the data can then be queried by using a single query language. A major drawback of the transformation-based approaches is that the transformation of data from one language into the other is a tedious and error prone process. Indeed, an RDF graph can be represented by more than one XML tree structure, so it is not clear how to formulate XQuery queries against it. On the other hand, XML lacks semantic information; so converting XML to RDF results in incomplete information with a number of blank nodes in the RDF graph. Moreover, many native XML and RDF data storage systems are now available to tackle rapidly increasing data sizes. We expect in the near future that many online RDF/XML sources will not be accessible as RDF/XML files, but rather via data stores that provide a standard querying interface. In such scenarios it will become practically impossible to transform the whole data sources from one format into another format at runtime.

Another approach to deal with heterogeneity is query re-writing which poses queries of different query languages to the data which is left in the original format, thus avoiding transformation of the whole data sources [2].

In [2], a new query language is designed which allows the formulation of queries on data

in different formats. The system automatically generates subqueries in SPARQL and XQuery which are posed to the corresponding data sources in their native format – without the need of data transformation. A major drawback of this approach is that the user has to learn a new query language even though powerful, standardized languages like XQuery and SPARQL exist. In some situation invention of new query language by combining two existing query languages limits language functionalities because usually such combination is applied on some subset of both queries which make it impossible to compile and execute any standard query in any of the parent query languages. Moreover, this approach is not easily extended if data in further formats has to be accessed like, for instance, querying relational data with SQL.

Similar to query Re-writing another approach is to embed one query language into another query language. One query language is used as main or host query language while other query language is embedded inside main query language as a subquery. For the XML and RDF data integration both types of proposals exists in literature, (i) embedding SPARQL into XQuery, and (ii) embedding XQuery into SPARQL.

The approach proposed in [35] is also based on transforming the heterogeneous data into a single language, namely XML. However, for querying the data, the embedding of SPARQL into XQuery is supported. The embedded SPARQL is then automatically translated into XQuery by the system. Providing a full-fledged translation from SPARQL to XQuery is a demanding task and, indeed, only a subset of the SPARQL language is supported in [35]. Yet another approach is presented in [2], where a new query language is designed which allows the formulation of queries on data in different formats. The system automatically generates sub queries in SPARQL and XQuery which are posed to the corresponding data sources in their native format – without the need of data transformation. A major drawback of this approach is that the user has to learn a new query language even though powerful, standardized languages like XQuery and SPARQL exist. Moreover, this approach is not easily extended if data in further formats has to be accessed like, for instance, querying relational data with SQL.

In great contrast to the above mentioned approaches, DeXIN does not apply any transformation to the data sources. Instead, subqueries in SPARQL (or any other language, to which DeXIN is extended in the future) are executed directly on the data sources as they are and only the result is converted. Moreover, in [35], only a subset of SPARQL is supported, while DeXIN allows full SPARQL inside XQuery.

In [2], a new query language XSPARQL was introduced (by merging XQuery and SPARQL) to query both XML and RDF/OWL data. In contrast to [2], our approach is based on standardized query languages (currently XQuery and SPARQL) rather than a newly invented language. Moreover, the aspect of data distribution is not treated in [2].

3.4.2 Distribution

For dealing with *distributed web data sources*, two major approaches for query processing exist:

- **centralized query processing:** which transfers the distributed data to the central location and processes the query there.
- **decentralized query processing** executes the queries at remote sites whenever this is possible.

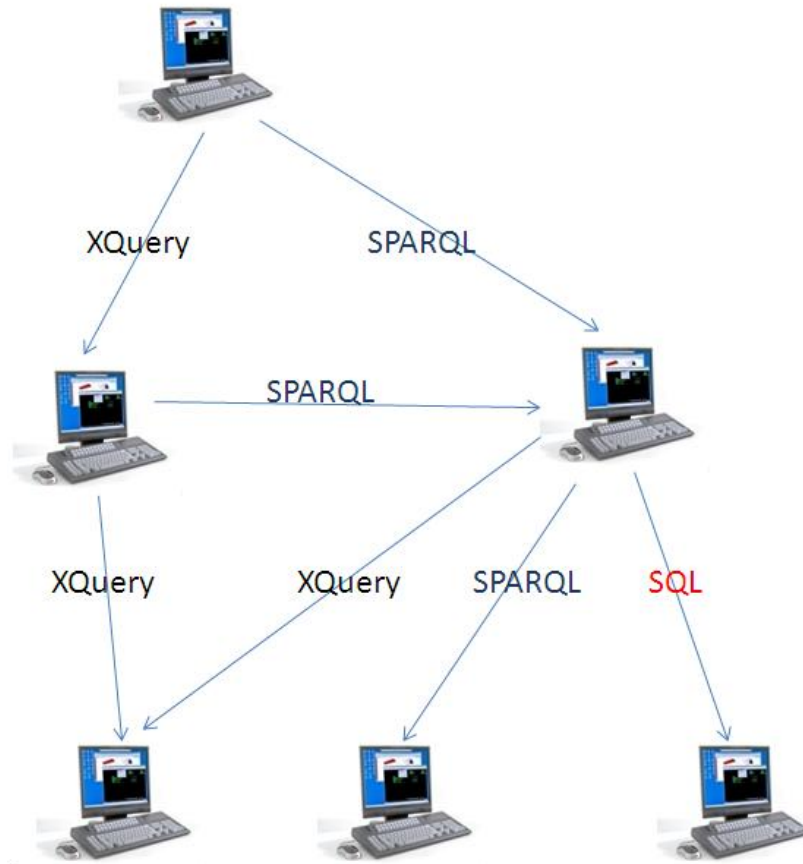


Figure 3.4: Distributed Query Processing In Peer-2-Peer Environment

With the centralized approach, the data transfer easily becomes the bottleneck of the query execution. Keeping replica on the central location is usually not feasible either, since we are dealing with autonomous and continually updating data sources. Hence, in general, decentralized query processing is clearly superior.

Recently DXQ [30] and XRPC [85] have been proposed for decentralized execution of XQuery and, likewise, DARQ [68] for SPARQL. However, to the best of our knowledge, a framework for decentralized query execution to facilitate data integration of heterogeneous web data sources is still missing.

DXQ [30], XRPC [85] and DARQ [68] are some efforts to execute distributed XQuery and distributed SPARQL separately on XML and RDF data. However, the integration of heterogeneous data sources and the formulation of queries with subqueries from different query languages (like SPARQL inside XQuery) are not addressed in those works.

Figure 3.4 depicts distributed query processing in a peer-2-Peer environment where the data sources are heterogeneous.

3.5 DeXIN: Distributed extended XQuery for Data INtegration

In this section we elaborate our novel framework for integrating distributed heterogeneous data sources. We propose DeXIN (**D**istributed **e**xtended **X**Query for heterogeneous Data **I**Ntegration), an extensible framework for distributed query processing over heterogeneous, distributed and autonomous data sources. DeXIN integrates multiple, heterogeneous, highly distributed and rapidly changing web data sources in different formats, e.g. XML, RDF and relational data. At the heart of DeXIN is an XQuery extension that allows users/applications to execute a single query against distributed, heterogeneous web data sources or data services. DeXIN considers one data format as the basis (the so-called “aggregation model”) and extends the corresponding query language to executing queries over heterogeneous data sources in their respective query languages.

Currently, we have only implemented XML as an aggregation model and XQuery as the corresponding language, into which the full SPARQL language is integrated. However, our framework is very flexible and could be easily extended to further data formats (e.g., relational data to be queried with SQL) or changed to another aggregation model (e.g., RDF/OWL rather than XML).

The main highlights of the features of the DeXIN are as follows.

- DeXIN is an extensible framework for parallel query execution over distributed, heterogeneous and autonomous large data sources.
- DeXIN provides extension of XQuery which covers the full SPARQL language and sup-

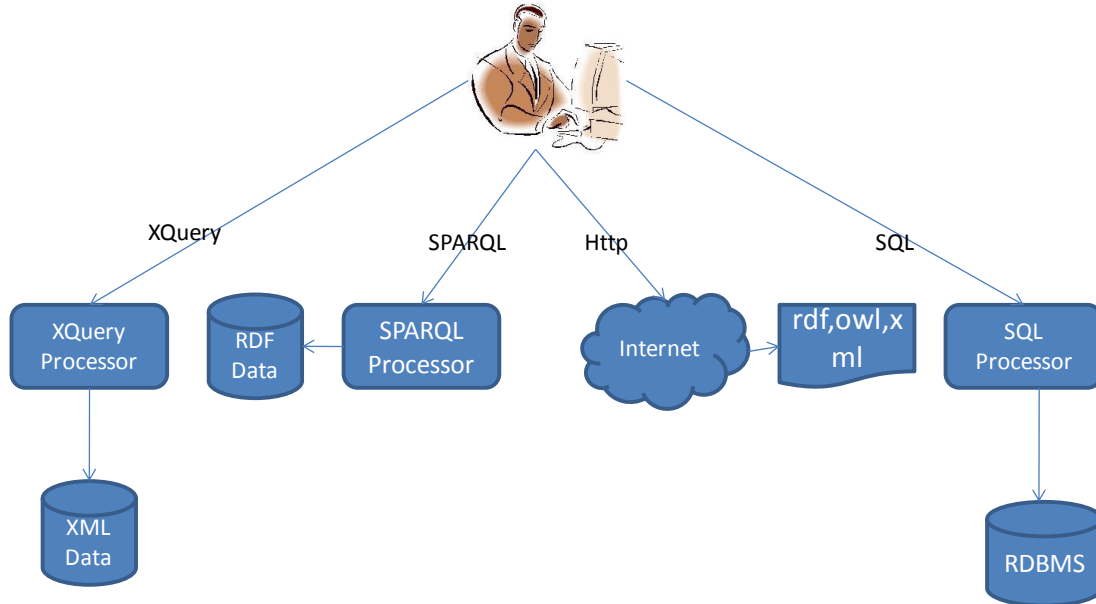


Figure 3.5: Traditional Query Technique for Web Data Sources

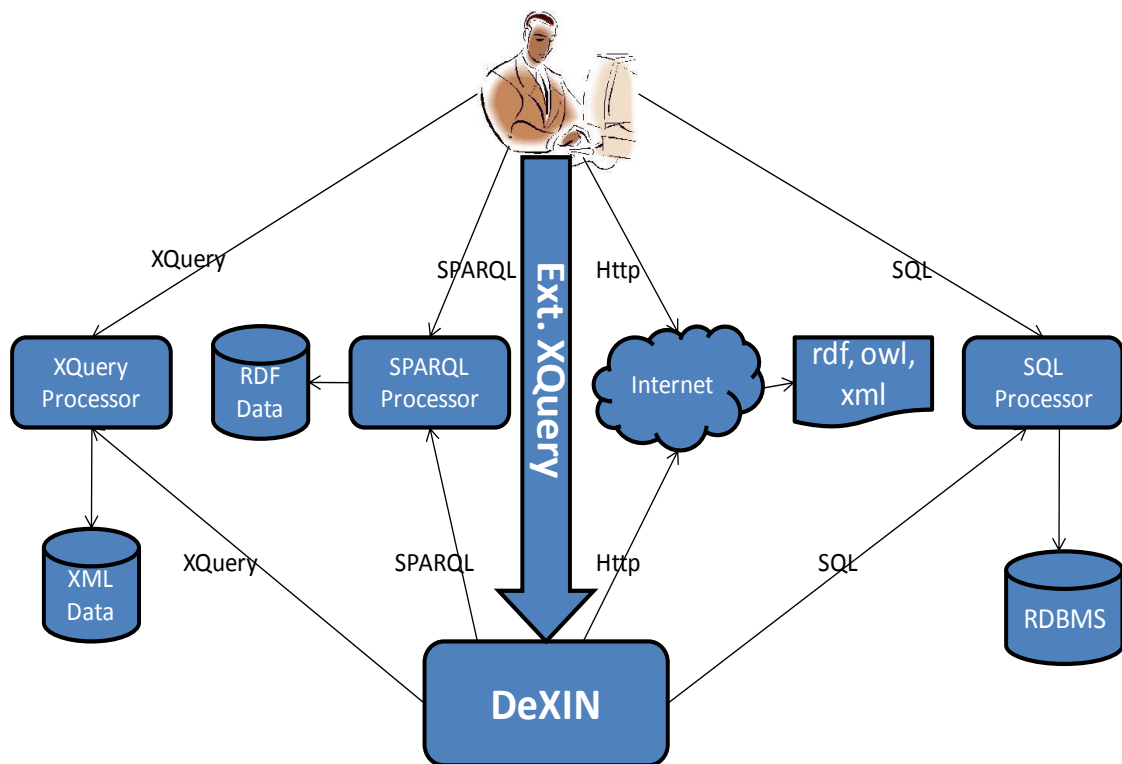


Figure 3.6: Architectural Overview of the DeXIN Framework

ports the decentralized execution both XQuery and SPARQL in a single query.

- DeXIN approach supports the data integration of XML, RDF and OWL data without the need of transforming large data sources into a common format.
- DeXIN is implemented as a web service to provide easy access using service oriented architecture.
- DeXIN can easily be integrated into existing web applications as a data integration tool.
- We carried out experiments, which document the good performance and reduced network traffic achieved with DeXIN approach.

3.5.1 Architectural Overview

An architectural overview of DeXIN is depicted in Figure 3.6. The main task of DeXIN is to provide an integrated access to different distributed, heterogeneous, autonomous data sources.

Figure 3.5 depicts the existing situation in most of the data integration applications for web data sources. Normally, the user would have to locate required data sources manually and then have to query each of these data sources separately. After the execution of the all the required

data sources, results are returned back to the user who has to integrate them locally to produce the desired results.

With the support of DeXIN, the user has a single entry point to access all these data sources as shown in Figure 3.6. By using our extension of XQuery, the user may still formulate subqueries to the various data sources in the appropriate query language. Currently, DeXIN supports XQuery to query XML data and SPARQL to query RDF/OWL data. However, the DeXIN framework is very flexible and we are planning to further extend this approach so as to cover also SQL queries on relational data. Note that not all data sources on the web provide an XQuery or SPARQL endpoint. Often, the user knows the URI of some (XML or RDF/OWL) data. In this case, DeXIN retrieves the requested document via this URI and executes the desired (XQuery or SPARQL) subquery locally on the site where DeXIN resides. DeXIN decomposes the user query, makes connections to data sources and sends subqueries to the specified data sources. If the execution fails, the user gets a meaningful error message. Otherwise, after successful execution of all subqueries, DeXIN transforms and integrates all intermediate results into a common data format (in our case, XML) and returns the overall result to the user. In total, the user thus issues *a single query* (in our extended XQuery language) and receives *a single result*. All the tedious work of decomposition, connection establishment, document retrieval, query execution, etc. is done behind the scene by DeXIN.

3.5.2 Query Evaluation Process

The query evaluation process in DeXIN is shown in Figure 3.7. The main components of the framework are briefly discussed below.

3.5.2.1 Parser

The Parser checks the syntax of the user query. If the user query is syntactically correct, the parser will generate the query tree and pass it on to the query decomposer. Otherwise it will return an error to the user.

3.5.2.2 Query Decomposer

The Query Decomposer decomposes the user query into atomic subqueries, which apply to a single data source each. The concrete data source is identified by means of the information available in the Metadata Manager (see below). Each of these atomic subqueries can then be executed on its respective data source by the Executor (see below).

3.5.2.3 Metadata Manager

All data sources supported by the system are registered by the Metadata Manager. For each data source, the Metadata Manager contains all the relevant information required by the Query Decomposer, the Optimizer or the Executor. Metadata also stores updated statistics e.g. response time, availability, reliability of data sources to support the Optimizer for query optimization.

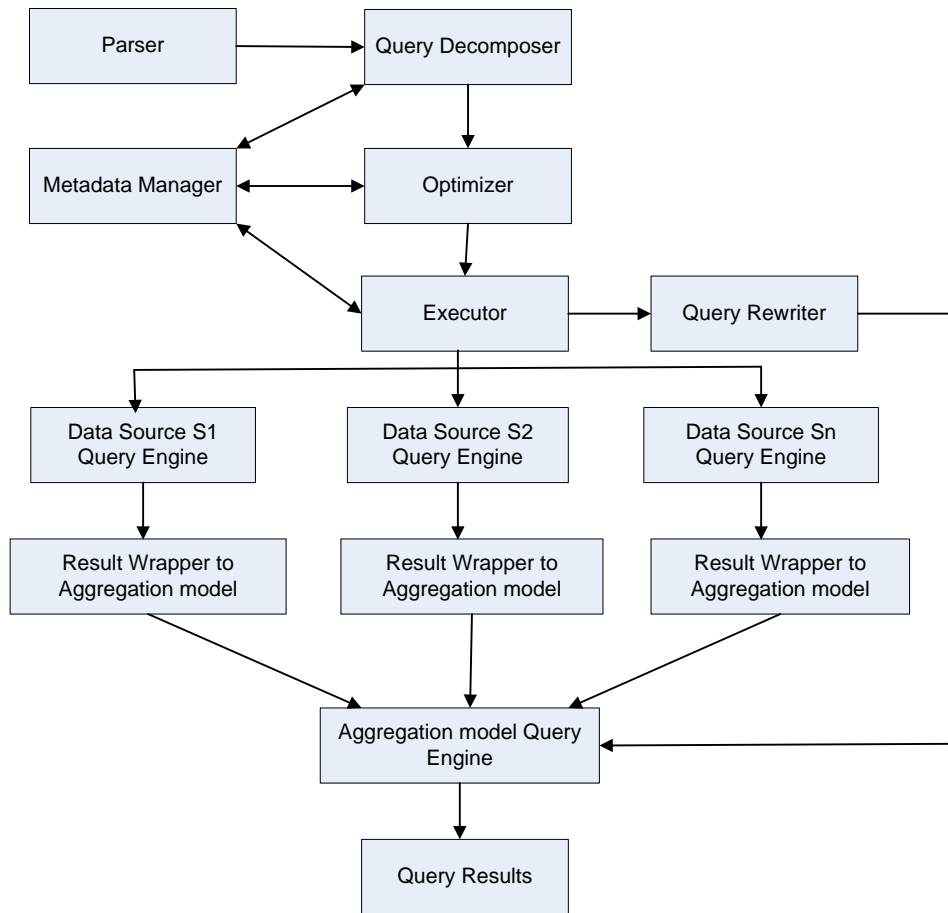


Figure 3.7: Query Evaluation Process

3.5.2.4 Optimizer

The optimizer searches for the best query execution plan based on static information available at the Metadata Manager. It should also perform some dynamic optimization to find variable dependencies in dependant or bind joins. Dependant or bind joins are basically nested loop joins where intermediate results from the outer relation are passed to the inner loop to be used as filter, which means that for each value of a variable in outer loop a new subquery is generated for execution at remote site. In such scenarios the optimizer will have to first look for all possible values of the variables in outer loop and ground the variables in sub query will all possible values, thus formulating a bundled query to ship at once to remote site. The current version of DeXIN has some known issues in the implementation of the Optimizer particularly for the dependant joins. Indeed, in one of our future agendas is to optimize the distributed query processing in DeXIN framework.

3.5.2.5 Executor

The Executor schedules the execution sequence of all the queries (in parallel or sequential). In particular, the Executor has to take care of any dependencies between subqueries. If a registered data source provides an XQuery or SPARQL endpoint, then the Executor establishes the connection with this data source, issues the desired subqueries and receives the result. If a registered data source only allows the retrieval of XML or RDF/OWL documents via the URI, then the Executor retrieves the desired documents and executes the subqueries locally on its own site.

Of course, the execution of a subqueries may fail, e.g., with source unreachable, access denied, syntax error, query timeout, etc. It is the responsibility of the Executor to handle all these exceptions. In particular, the Executor has to decide if a continuation makes sense or the execution is aborted with an error message to the user.

3.5.2.6 Result Reconstruction

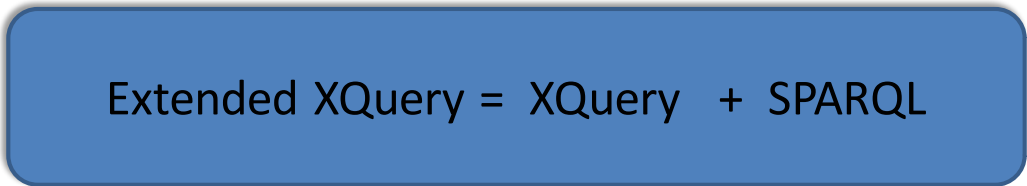
All the results received from distributed, heterogeneous and federated data sources are wrapped to the format of the aggregation model (in our case, XML). If all the results are wrapped successfully, this component will integrate the results and stores them in temporary files for further querying by the aggregation model query processor (in our case, an XQuery engine).

3.5.2.7 Query Rewriter

The Query Rewriter rewrites the user query in the extended query language (in our case, extended XQuery) into a single query on the aggregation model (in our case, this is a proper XQuery query which is executed over XML sources only). For this purpose, all subqueries referring to different data sources are replaced by a reference to the locally stored result of these subqueries. The overall result of the user query is then simply obtained by locally executing this rewritten query.

3.6 XQuery extension to SPARQL

In this section we present the extension of XQuery language to execute SPARQL queries inside XQuery. We call it “*Extended XQuery*” which extends XQuery functionality for SPARQL query execution.



Extended XQuery = XQuery + SPARQL

Figure 3.8: Extended XQuery

Method Name	Description
SPARQLQuery	Function can be used in XQuery wherever a reference to XML document can be mentioned
XML DOC	Return Type will be XML Document Tree
String sparqlQuery	First parameter is string containing SPARQL Query to execute
URI sourceURI	URI of the source data

Table 3.1: SPARQLQuery Function

3.6.1 Syntax

DeXIN is an extensible framework based on a multi-lingual and multi-database architecture to deal with various data formats and various query languages. It uses a distinguished data format as “aggregation model” together with an appropriate query language for data in this format. So far, we are using XML as aggregation model and XQuery as the corresponding query language. This aggregation model can then be extended to other data formats (like RDF/OWL) with other query languages (like SPARQL). In order to execute SPARQL queries inside XQuery, it suffices to introduce a new function called SPARQLQuery(). This function can be used anywhere in XQuery where a reference to an XML document may occur. This approach is very similar to the extension of SQL via the XMLQuery function in order to execute XQuery inside SQL (see [61]). The new function SPARQLQuery() is defined as follows:

XMLDOC SPARQLQuery(
String sparqlQuery,URI sourceURI)

The value returned by a call to this function is of type XMLDOC. The function SPARQLQuery() has two parameters: The first parameter is of type String and contains the SPARQL query that has to be executed. The second parameter is of type URI and either contains the URI or just the name of the data source on which the SPARQL query has to be executed. The name of the data source refers to an entry in the database of known data sources maintained by the Metadata Manager. If the indicated data source is reachable and the SPARQL query is successfully executed, then the result is wrapped into XML according to the W3C Proposed Recommendation [10].

To illustrate this concept, we revisit the motivating example of SEMF [74] discussed in Section 3.2. Suppose that a user wants to get information about available web services which have a license fee of less than one Euro per usage. Moreover, suppose that the user also needs information on the service license agreement and the quality of service before using this service in his/her application. Even this simple example may encounter the problem of heterogeneous data sources if, for example, the service license agreement information is available in XML format while the information about the quality of service is available in RDF format. A query in extended XQuery for retrieving the desired information is shown in Figure 3.9.


```

for
  $a in doc("http://SEMF/License.xml")/agreement,
  $b in SPARQLQuery(" SELECT ?title ?ExecutionTime
    WHERE {
?x <http://www.w3.org/2001/sub#title> ?title .
?x <http://www.w3.org/2001/sub#QoS> ?ExecutionTime "
      },http://SEMF/QoS.rdf)/result
WHERE
  $a/servicetitle = $b/title
AND $a/peruse/amount <= 1
RETURN
<Results>
  <Service>
    <ServiceTitle>{$a/title}</ServiceTitle>
    <Requirement>{$a/requirement}</Requirement>
    <ExecutionTime>{$b/ExecutionTime}</ExecutionTime>
  </Service>
</Results>

```

Figure 3.9: An Example extended XQuery for DeXIN

3.6.2 Query Decomposition and Processing

In this section we will have a closer look at the central steps for executing an extended XQuery query, namely the query decomposition and query processing.

The query tree returned by the Parser has to be traversed in order to search for all calls of the SPARQLQuery() function. Suppose that we have n such calls. For each call of this function, the Query Decomposer retrieves the SPARQL query q_i and the data source d_i on which the query q_i shall be executed. The result of this process is a list $\{(q_1, d_1), \dots, (q_n, d_n)\}$ of pairs consisting of a query and a source. The Executor then poses each query q_i against the data source d_i . If the execution of each query q_i was successful, its result is transferred to the site where DeXIN is located and converted into XML-format. The resulting XML-document r_i is then stored temporarily. Moreover, in the query tree received from the Parser, the call of the SPARQLQuery() function with query q_i and data source d_i is replaced by a reference to the XML-document r_i . The resulting query tree is a query tree of pure XQuery without any extensions. It can thus be executed locally by the XQuery engine used by DeXIN.

Figure 3.10 gives an overview of the query evaluation processing of the sample query shown in Figure 3.9. There are three peers involved in this query, Peer1 stores an XML file “license.xml”, Peer 2 stores an RDF file “QoS.rdf”, while Peer 3 has DeXIN installed on its system and executes the extended XQuery of Figure 3.9.

Once the query is decomposed by following the above mentioned process a list of pairs consisting of a query and a source is returned. DeXIN sends these pairs to the appropriate data sources, in this case, XQuery to Peer 1 and SPARQL to Peer 2. After the queries execution

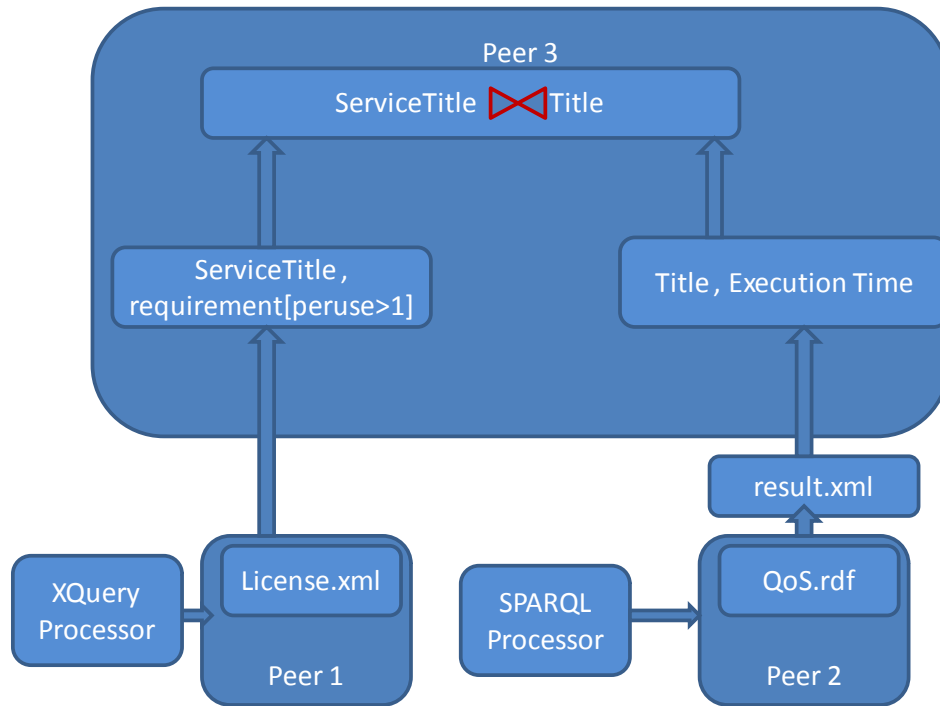


Figure 3.10: Query Processing in DeXIN

results are returned to Peer 3. SPARQL query results are first transformed into XML using W3C standard for SPARQL query results transformation into XML [10]. At Peer 3 DeXIN locally performs the aggregation of the results from multiple queries and return the aggregated results to the user. Figure 3.11 shows results of the sample query shown in Figure 3.9

Currently, DeXIN aggregates all the results locally once all the queries results have been executed successfully. One of the future goals on the DeXIN agenda is the incorporation of query optimization techniques (like semi-joins, a standard technique in distributed database systems [69]) into DeXIN.

3.7 Implementation and Experimental Analysis

DeXIN supports queries over distributed, heterogeneous and autonomous data sources. It can be easily plugged into applications which require such a facility. As a case study, we take the example of service management systems and show how DeXIN enhances service management software by providing this query facility over heterogeneous and distributed data sources.

TestBed. We set up a testbed which includes 3 computers (Intel(R) Core(TM)2 CPU, 1.86GHz, 2GB RAM) running SUSE Linux with kernel version 2.6. The machines are connected over a standard 100Mbit/S network connection. An open source native XML database eXist (release 1.2.4) is installed on each system to store XML data. Our prototype is imple-

```

<Results>
  <Service>
    <ServiceTitle>WISIRISFuzzySearch</ServiceTitle>
    <Requirement>
      <peruse>
        <payment>
          <amount currency='EUR'> 0.90 </amount>
          <taxpercent code='VAT'>20</taxpercent>
        </payment>
      </peruse>
    </Requirement>
    <ExecutionTime Unit='sec'>17</ExecutionTime>
  </Service>
  <Service>
    .....
  </Service>
  .....
</Result>

```

Figure 3.11: Result after Executing Query

Name	Description	Size
XS1	Articles	250MB
XS2	Proceedings	200MB
XS3	Books	50MB

Table 3.2: XML Data Sources

mented in Java. We utilize the eXist [60] XQuery processor to execute XQueries. The Jena Framework [44] (release 2.5.6) is used for storing the RDF data, and the ARQ query engine packaged within Jena is used to execute SPARQL queries.

Name	Description	No. of Tuples
RS1	Articles	7.6Million
RS2	Categories	6.4Milliom
RS3	Persons	0.6Million

Table 3.3: RDF Data Sources

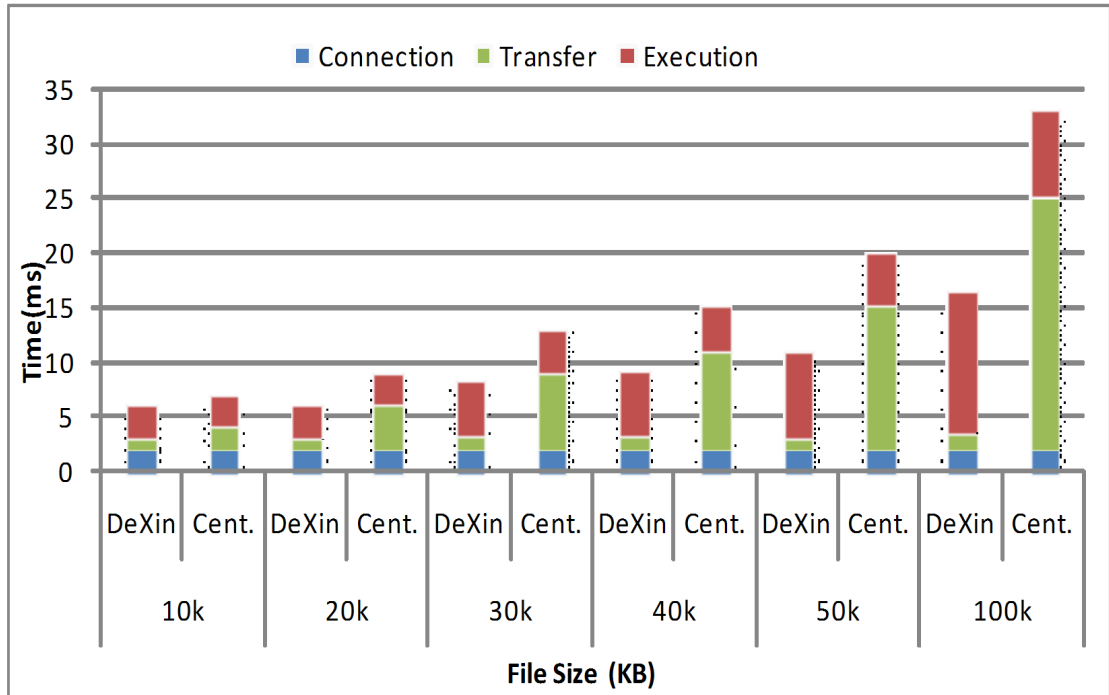


Figure 3.12: Execution Time Comparison DeXIN Vs Naive Centralized

3.7.1 Experimental Application: Web Services Management

One of the main motivations for developing this framework is to utilize it for service management systems like SEMF [74]. Being able to query distributed and heterogeneous data sources associated to web services is a major issue in these systems. SEMF stores and manages updated information about all the services listed in this framework. Recall the example use case given in Section 3.6: We consider a user who requests information about available web services which have a license fee of less than one Euro per usage. Moreover, the user needs information on the service license agreement and the quality of service. We assume that the service license agreement information is available in XML format while the information about the quality of service is available in RDF format. As we have seen in Section 3.6, our framework provides the user a convenient way of querying these distributed, heterogeneous data sources at the SEMF platform without worrying about the transformation, distribution and heterogeneity of the data sources involved by issuing the extended XQuery query of Figure 3.9 to SEMF. The result returned to the user is in XML format and may look like the XML file in Figure 3.11.

3.7.2 Performance Analysis

In order to analyze the performance of DeXIN, we have conducted tests with realistically large data. Since SEMF is only available as a prototype, the test data available in this context is too small for meaningful performance tests. We therefore chose to use DBPedia ³ and DBLP ⁴, which are commonly used for benchmarking.

3.7.2.1 Data Distribution over the Testbed

For the SPARQL query execution over RDF data, we use a subset of DBPedia, which contains RDF information extracted from Wikipedia. This data consists of about 31.5 million triples and is divided into three parts (Articles, Categories, Persons). The size of these parts is displayed in Table 3.3. The data is distributed over the testbed in such a way that the Articles, Categories, and Persons are stored on different machines. Moreover, we have further split these data sets into 10 data sources of varying size in order to formulate queries with subqueries for a bigger number of data sources. For the XQuery execution over XML data we used DBLP. DBLP is an online bibliography available in XML format, which lists more than 1 million articles. It contains more than 10 million elements and 2 million attributes. The average depth of the elements is 2.5. The XML data is also divided into three parts (Articles, Proceedings, Books), whose size is shown in Table 3.2. We distributed the XML data over the testbed such that the Articles, Proceedings, and Books are stored on different machines. As with the RDF data, we also subdivided each of the three parts of the XML data into several data sources of varying size.

3.7.2.2 Experiments

In the first scenario we consider a set of queries of different complexity varying from simple select-project queries to complex join queries. The queries use a different number of distributed sources and have different result sizes. The results shown are the average values over ten runs. The query execution time is subdivided as

$$\text{Total Time} = \text{Connection Time} + \text{Execution Time} + \text{Transfer Time}$$

Figure 3.12 presents the query execution time for a naive centralized approach compared with DeXIN. It turns out that the data transfer time is the main contributor to the query execution time in the distributed environment – which is not surprising according to the theory on distributed databases [65]. DeXIN reduces the amount of data transferred over the network by pushing the query execution to the local site, thus transferring only the query results. We observe that with increasing size of data sets, the gap in the query execution time between DeXIN and the naive centralized approach is widened.

In the second scenario we fix the size of data sources and execute queries with varying selectivity factor (i.e., the ratio of result size to data size) and compare the query execution time of DeXIN with the naive centralized approach. As was already observed in the previous

³ <http://dbpedia.org/>

⁴ <http://dblp.uni-trier.de/xml/>

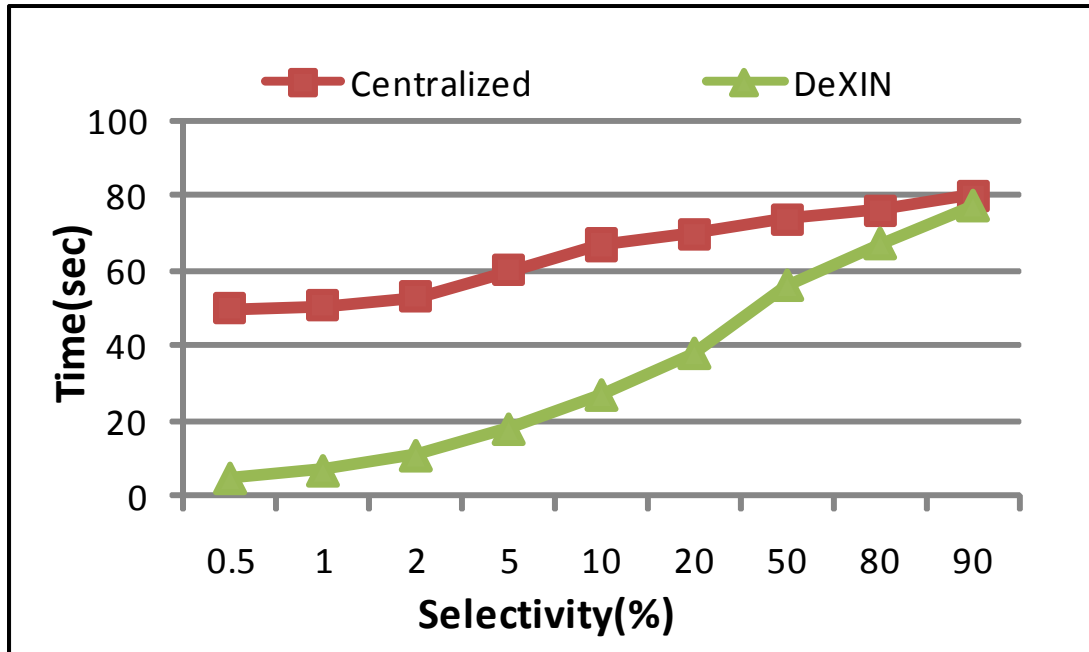


Figure 3.13: Execution Time Comparison (Varying Selectivity Factor)

scenario, the execution time is largely determined by the network transfer. Figure 3.13 further strengthens this conclusion and, moreover, shows that DeXIN gives a better execution time for queries with high selectivity. The results displayed in Figure 3.13 indicate that DeXIN is much stronger affected by varying the selectivity of queries than the centralized approach. DeXIN is superior to the centralized approach as long as the selectivity factor is less than 90% . Above, the two approaches are roughly equal.

In the third scenario, we observe the effect of the number of data sources on the query execution time. We executed several queries with varying number of sources used in each query. Figure 3.14 again compares the execution time of DeXIN with the execution time of a naive centralized approach. It turns out that as soon as the number of sources exceeds 2, DeXIN is clearly superior.

3.8 DeXIN Demo

3.8.1 DeXIN as a Data Integration Web Service

DeXIN is a RESTful data service which takes a single query in extended XQuery syntax as input, decomposes the query into sub-queries, executes each sub-query independently on its appropriate distributed data source at remote locations and outputs the integrated results from all data sources in XML format. Consider an example of a web application which need to provide

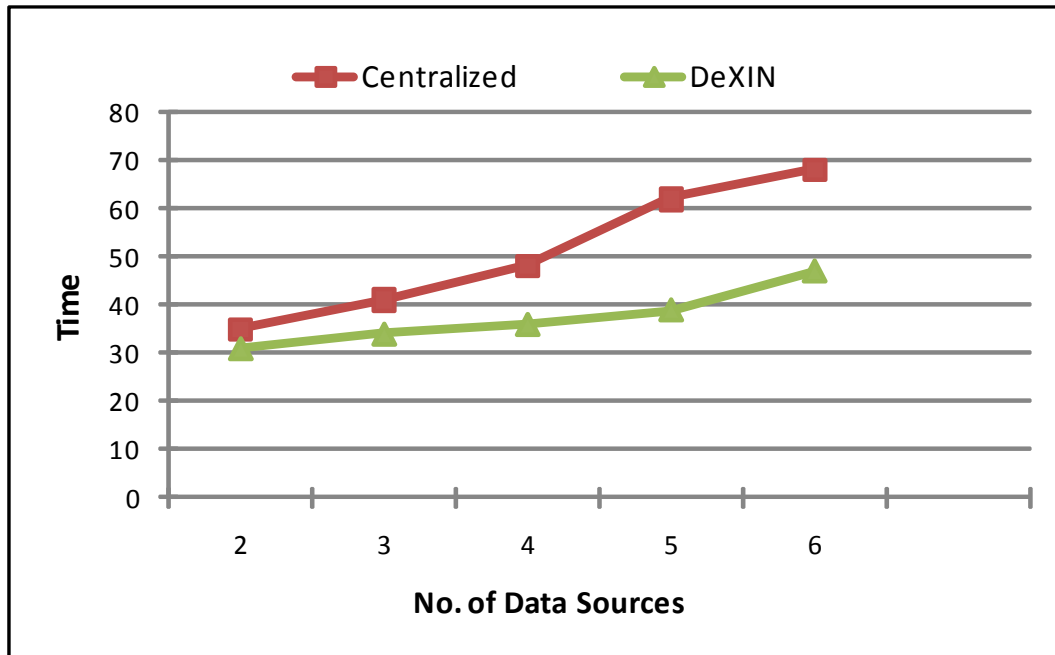
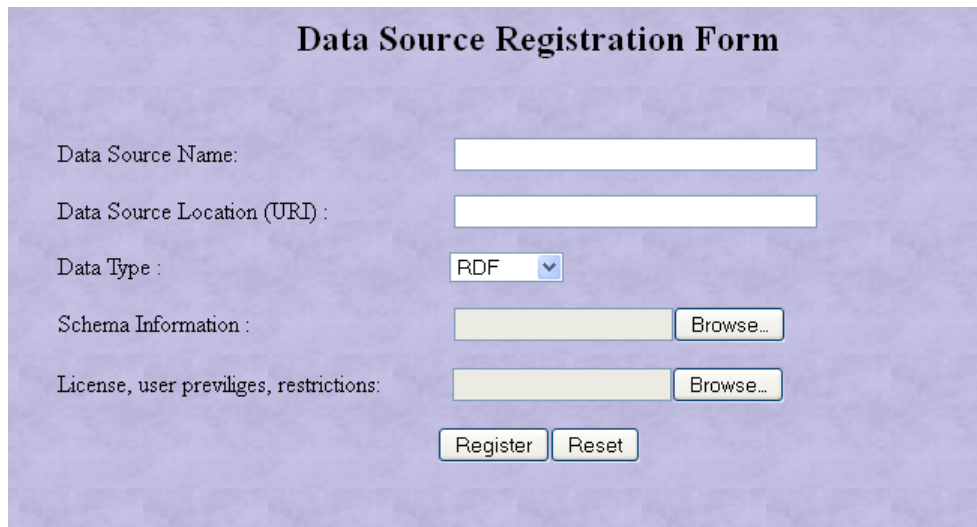


Figure 3.14: Execution Time Comparison (Varying Number of Data Sources)

the integrated access to the distributed and heterogeneous web data sources dynamically. Typical data integration approaches e.g. warehousing, mediation or ontology based, do not provide the desired results because they require some prior knowledge about the data sources. DeXIN can ideally serve such applications because it provides integrated access to heterogeneous distributed web data sources dynamically. Figure 3.16 shows the user interface of the DeXIN service. The user can write a query in extended XQuery format and gets the accumulated results of all the data sources. Currently, DeXIN supports two types of sub-queries inside XQuery namely, (i) SPARQL for RDF, OWL and (ii) SQL for relational data.

3.8.2 Searching Available Data Services

Many service providers have started to expose their data as a service by implementing the Resource Oriented Architecture (ROA). Some Database Management Systems (DBMS) also provide access to their data using Query Language + REST/SOAP. DeXIN can communicate with data service directories to find out the appropriate data service. The user can initiate a keyword search for the required data service, and all the available data services are listed by DeXIN to help the user to select an appropriate data service.



The image shows a web form titled "Data Source Registration Form" on a light purple background. The form contains the following fields and controls:

- Data Source Name:** A text input field.
- Data Source Location (URL) :** A text input field.
- Data Type :** A dropdown menu with "RDF" selected.
- Schema Information :** A text input field followed by a "Browse..." button.
- License, user privileges, restrictions:** A text input field followed by a "Browse..." button.
- Buttons:** "Register" and "Reset" buttons at the bottom center.

Figure 3.15: Data Source Registration Form

3.8.3 Registration of the Data Sources

The registration of data sources at DeXIN is not mandatory because DeXIN can interact with any data service at runtime, but providing some metadata about data sources by following the registration procedure makes querying simpler from the user's perspective. Figure 3.15 shows a data source registration form in DeXIN. Different Data Sources (e.g. RDF, XML, OWL or RDBMS) can be registered at DeXIN to get benefit from the integration facilities provided by DeXIN. Each data source must provide a unique name, should have one of the DeXIN supported data types, querying interface with connectivity facility and XML converter for query results. Data providers can provide additional information about schema, user privileges, license and legal issues to facilitate the users to interact with their data sources effectively. It is worth mentioning here that utilizing the concept of data services greatly eases the process of registration, because it uses standard HTTP protocol to interact with the data sources and XML for data transfer.

3.8.4 Data Source Statistics and Schema Information

DeXIN stores some metadata and statistics about registered data sources, which are helpful for the selection of the best available service from the user's perspective. The user can select any data service from the list of available data sources shown by DeXIN (see top right of Figure 3.16) and can see its statistics which are either stored in the DeXIN or retrieved by the DeXIN after communicating with the Service Management System.

If the data service provider provides some schema information about data sources, the user can click on "Show Schema", to see the schema information, which can be helpful for designing queries for that particular data source.

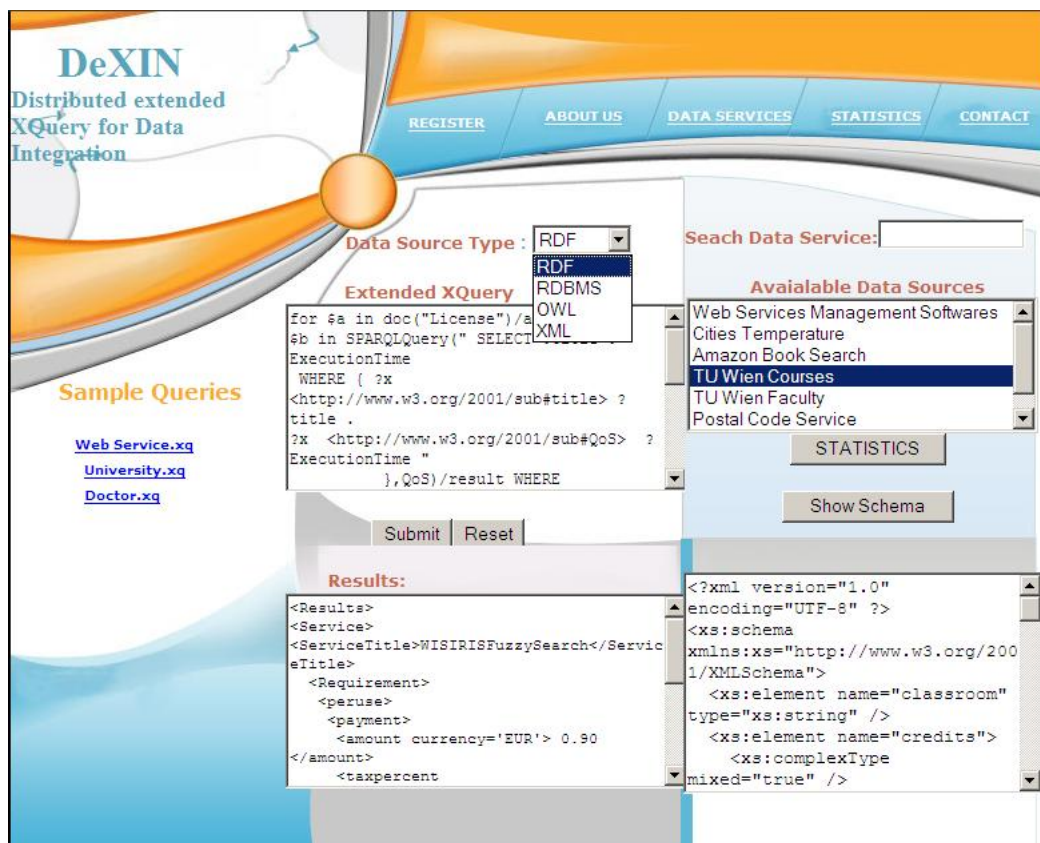


Figure 3.16: User Interface of DeXIN

3.8.5 Query Execution

Once the user submits a query to DeXIN in extended XQuery format, DeXIN (i) decomposes the query into multiple sub-queries for distributed, heterogeneous data sources (ii) connects with the data sources mentioned in query (iii) dispatches queries to their particular data source at remote locations (iv) displays integrated results of all the sub-queries into XML format.

3.9 Discussion

3.9.1 Data Transformation V/S Results Transformation

A common method for the integration of XML and RDF/OWL data is to transform the data into a single target format (e.g., only XML) and then query this data with a single query language (e.g., only XQuery). This approach is problematical for two reasons: Transforming the data requires access to the entire source data, which is of course not always the case due to copyright or other user restriction issues. Moreover, even if the entire source data is accessible, its transformation is very costly. In contrast, DeXIN uses a multi-database approach where the data sources reside

at the remote site in their original format and only the results of the subqueries are transformed into a common language (in our case, SPARQL results are thus converted to XML). Note that, in general, the result data is considerably smaller than the source data. The precise size of the result relative to the source is determined by the selectivity factor of the queries. A common assumption in query optimization is that every single selection reduces the data to 10% (see a standard database textbook like [69]).

3.9.2 Data Shipping V/S Query Results Shipping

In distributed query processing, the main cost factor is the network cost, i.e., most of the time is consumed by shipping the data from one site to another. With the help of new available technology and faster networks, the time of data transfer may be reduced. But still the time ratio between the system's Read/Write and the shipping cost is usually in the order of 1:10 (see [65]). With centralized query processing, the source data has to be shipped to a central location where the query is executed. In contrast, DeXIN applies decentralized query processing such that queries are executed on the remote data sources. Hence, only the results of subqueries have to be shipped. By the above considerations, the result data is usually much smaller than the source data. Hence, the data transfer cost is usually much lower with decentralized query processing. The potential of parallel execution of subqueries on different remote sites is another important advantage of the decentralized approach.

3.9.3 Summary

In this chapter, we have presented DeXIN – a novel framework for an integrated access to heterogeneous, distributed data sources. So far, our approach supports the data integration of XML and RDF/OWL data without the need of transforming large data sources into a common format. We have defined and implemented an extension of XQuery to provide full SPARQL support for subqueries. It is worth mentioning that the XQuery extension not only enhances XQuery capabilities to execute SPARQL queries, but SPARQL is also enhanced with XQuery capabilities e.g. result formatting in the return clause of XQuery etc.

We have demonstrated typical use cases of heterogeneous data integration which show that DeXIN is a simple but powerful tool to integrate rapidly changing heterogeneous data sources dynamically. DeXIN can be utilized by many applications where the data sources are unknown at design time, and it eases the integration process from the user's perspective by not requiring prior knowledge of data sources.

An important feature of our framework is its flexibility and extensibility. A major goal for future work on DeXIN is to extend the data integration to further data formats (in particular, relational data) and further query languages (in particular, SQL). We also want to extend the tests with DeXIN. So far, we have tested DeXIN with large data sets but on a small number of servers. In the future, when the web service management system SEMF [74] is eventually applied to realistically big scenarios, DeXIN will naturally be tested in an environment with a large-scale network.

Data Concerns Aware Querying

In this chapter, we introduce a “*Data Concerns Aware Querying System*”. A data concerns aware querying system incorporates several data concerns into a query language, thus enabling data services integration systems to handle data concerns associated with the data services. Our concerns aware querying system extends XQuery language to make it concerns aware, with the introduction of special keywords for mentioning data concerns within the query. A data concerns tree is attached with each data service which stores meta-data information of the concerns of all the stake holders of the data service and is also capable of querying data concerns dynamically. The query parser looks into the data concerns tree attached to the available data services and executes the query by selecting the most suitable data services for the particular user query.

More and more organizations provide their data on the web via data services – also referred to as Data as a Service (DaaS). Data services combine the strength of database systems and query languages on the one hand with the benefits of service-oriented architecture on the other hand. Data services are increasingly used for data integration. However, the data provided via data services is often associated with data concerns like privacy, licensing, pricing, quality of data, etc. Moreover, data service selection and data selection should be based on these data concerns. Current data integration systems using data services lack the ability to preserve data concerns while querying multiple services in an integrated environment.

The work presented in this chapter has been published in [5, 6], where we present our data concerns aware querying system. We lay the foundations of our study by identifying the actors involved in data services and their specific concerns. We design four possible models of data concerns aware querying. We discuss these models of data concerns aware querying and select the best suited one for our system. We describe a querying system where data concerns awareness is integrated directly into the XQuery language. We have implemented a concerns aware querying system by extending the XQuery language. We conclude this chapter with the implementation details and the experimental evaluation.

4.1 Motivation

More and more data providers are reaping the benefits of web 2.0 technology and provide their data on the web either through web services, API's (REST/SOAP), or data services [25, 37]. Data services combine the strength of database systems and query languages on the one hand with the benefits of service-oriented architecture on the other hand. Many big companies have started to publish their data via query interfaces (rather than simple forms in HTML) so that the data can be easily reused, composed and integrated with other data sources. Amazon Public Data Sets on AWS¹, Google Squared² and UN data API's³ are some prominent examples of publicly available data services.

Data services are increasingly used for data integration. Many tools and techniques are available to dynamically compose, integrate and execute different data services or sources [64]. These tools help to create situational applications by composing existing data services. The data thus published and processed is often associated with data concerns like privacy, licensing, pricing, quality of data, etc. Hence, data integration tools not only have to mitigate the heterogeneity in data formats and query languages. In addition, also the various data concerns should be preserved when data is published and utilized. Moreover, data service selection and data selection should be based on these data concerns. Consider for example, a meta-search query (a query that is posed against many data sources and selects the best possible integrated results among them). A user query will be executed on multiple data services registered at the integrated application. After integrating the results of all the data services, usually top- k results (where k is a constant value defined by the application) are returned. Now consider that different users have different priorities for the data selection, e.g., one user may be more interested in quality of data while another user is more concerned about the pricing. There is a clear need for an explicit system that (semi)automatically selects the most appropriate data service as well as data items for each user according to various data concerns.

Some data concerns like data quality, privacy, and quality of service (QoS) have long been studied in their respective domains of databases, data mining and web services. However, data services are different. Recently, the importance of distinguishing data services from web services has been recognized [75]. For instance, while licensing and quality of data are usually *static* for web services, they are dynamic for data services. Indeed, as the data gets "older", the licensing and the data quality (of which the up-to-dateness may be an important aspect) will most probably change. Moreover data concerns can be dynamically updated. Hence static information about usage permission or privacy cannot deal with the requirements of the dynamic integration application created by composing data services on the fly. Hence, new techniques are required to integrate data concerns into data services.

Among the various data concerns, *privacy* has received most attention. It has been studied in many different areas like data integration [12, 20], data mining [84], and web services [23, 48]. In [63] and similarly in [59], privacy has been studied in the context of data services. However a systematic integration of data concerns awareness into data services is still missing

¹<http://aws.amazon.com/publicdatasets/>

²<http://www.google.com/squared>

³<http://www.undata-api.org/>

to date. Moreover, previous approaches of dealing with data concerns like privacy do not directly integrate the data concerns awareness into the query language, even though this would be very important for enabling the querying system to select the best suited data source for various parts of a given query. Data integration tools not only have to mitigate the heterogeneity in data formats and query languages. In addition, the various data concerns should also be preserved when data is published and utilized.

Current data integration systems using data services lack the ability to preserve data concerns while querying multiple services in an integrated environment. We design a new querying system which takes data concerns into account. To this end we discuss several models of data concern aware querying and select the best suited one for our system. We describe a querying system where data concern awareness is integrated directly into the XQuery language. We also report on an implementation and experimental evaluation of this system. The goal of this work is to design a querying system

- which can take arbitrary data concerns into account,
- which integrates the data concerns awareness into the query language, and
- which automatically selects the appropriate data sources depending on current context, user requirements and data concerns.

4.2 Data Services

The concept of data services is based upon the service-oriented architecture (SOA), which includes standardized processes for accessing the data “where it resides” irrespective of the platform. Data services take advantage of service-oriented architecture to offer users a mediator for integrating information from database systems and other structured or non-structured data sources.

Data services provide a layer of software between physical distributed data sources and applications which want to access the data. The data is exposed to the customer via a virtual data model. It is the responsibility of the data service to connect to the back-end data sources via the available interfaces and to map the physical data schema to the virtual data model. The applications and services using the data service leverage this virtual data model to access the required information and the data service software handles the collection and distribution of the data as needed from the physical instances of the data.

Data services are different from traditional web services. In traditional web services after the discovery and binding of the web service only the requester and provider communicate for the execution of the service. Figure 4.1 gives a conceptual architecture of a web service. As shown in Figure 4.1, step 1 is the communication between the requester entity and the provider entity. The process of “becoming known” to each other can be initiated by any of the two parties involved in the communication. After the successful completion of step 2, the requester agent agrees to use the services of the provider agent. At step 3, a number of messages is exchanged between the requester and provider agents to agree on the semantics and description of the web

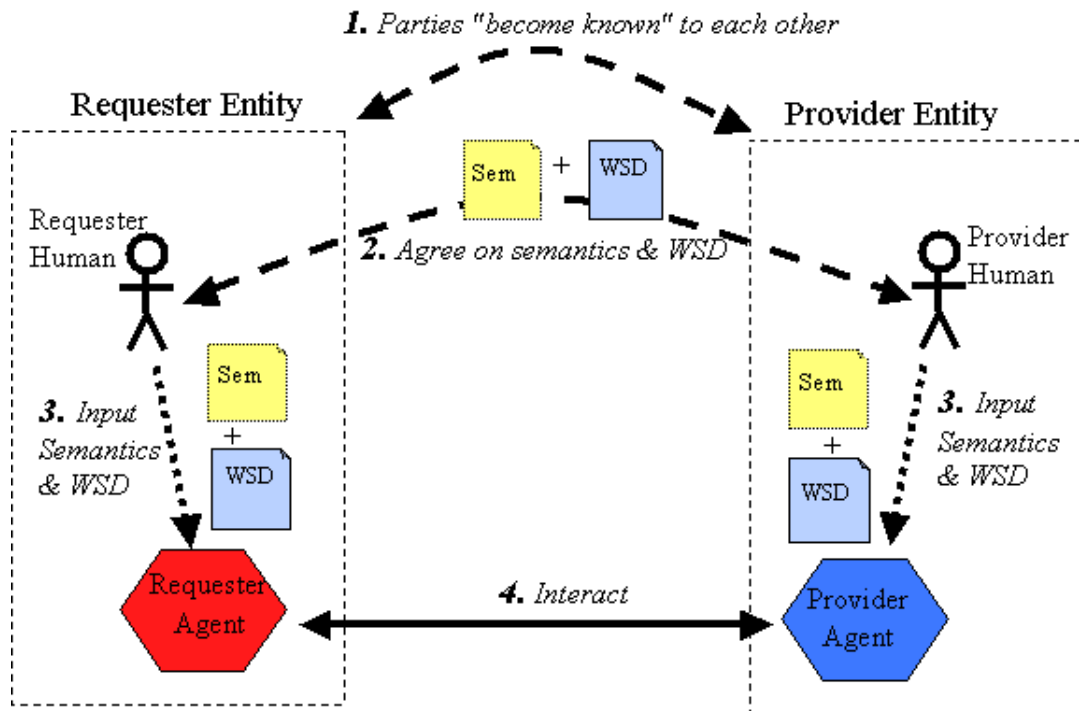


Figure 4.1: Conceptual Architecture of a Web Service [57]

service. Finally, at step 4 the requester and provider agents interact with each other to perform the required task by executing the web service.

In contrast to the web services, data services separate data provider and service provider as two entities. Figure 4.2 gives an overview of the conceptual architecture of the data services. We describe the details of the data services architecture in Section 4.2.1.

Data services represent a new market whose time has come. The technology exists already, and data services based businesses are emerging quickly. Businesses across sectors are beginning to see their data not only as fundamentally valuable, but also economically viable to distribute. The scale offered by an API strategy allows businesses to unlock the value of that data for their own revenue growth and their customer's benefit.

The traditional approach to the design of data services is that the service provider and consumer agree on how the service shall be used. The terms of this agreement are usually static and are not altered over time. Typically, data services in the form of integration server products have connectors or adaptors built to connect applications together. A number of them have added or will add new connectors for data service applications such as salesforce⁴.

⁴<http://salesforce.com>

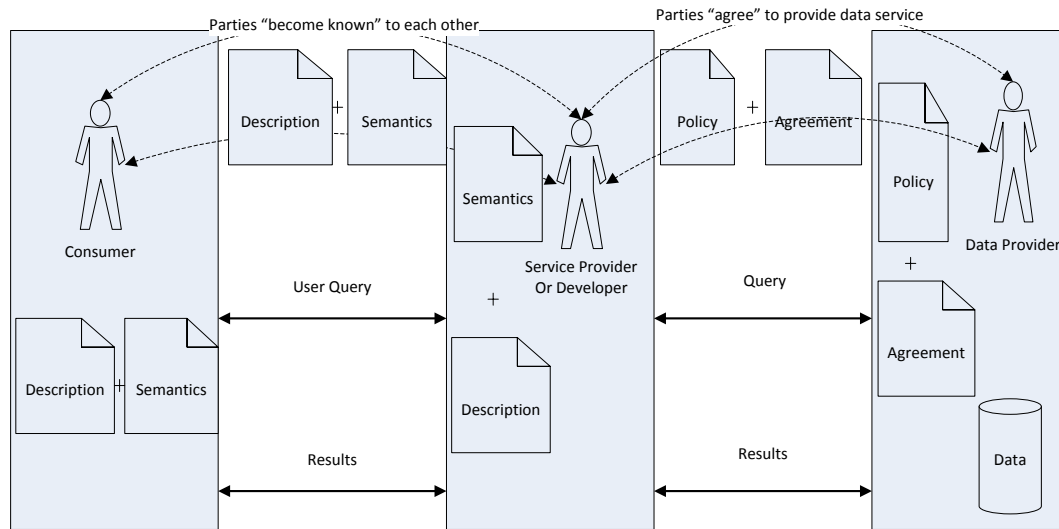


Figure 4.2: Conceptual Architecture of the Data Services

4.2.1 Roles of Data Services

As can be seen in Figure 4.2, there are three actors involved in a data service architecture, namely Data Provider (DP), Service Provider (SP), and Consumer. Below we discuss each of them briefly.

4.2.1.1 Data Provider

The data provider is responsible to provide the actual data for the service. The DP is not necessarily the owner of the data, since data can also be outsourced. But it is the responsibility of the DP to ensure the availability of the data for the usage by the data service. All the concerns related to the data must also be communicated to the service provider. Some data concerns for the data provider like the privacy and permission concerns can vary from data item level to the whole data source level.

4.2.1.2 Service Provider

The service provider entity is the person or organization that provides an appropriate agent to implement a particular service. Service provider is owner of the data service. After making the arrangement with the DP, the SP defines the functionality of the service. It is responsibility of the SP to make sure that all the concerns defined by the various data providers are preserved while the service is used.

4.2.1.3 Consumer

The consumer is a person or organization that wishes to make use of the data service. It will use a requester agent to exchange messages with the provider agent. In most cases, the requester agent interacts with the provider agent to exchange messages. The requester entity and provider entity agree on the service description (e.g. a WSDL document) and the semantics that will govern the interaction between the requester agent and the provider agent.

4.2.2 Data Services Pricing Models

Data services can be sub divided into two categories based on their pricing models. In this section we discuss these models in detail.

4.2.2.1 Volume Based Model

The volume based model considers the size of the data as a basic unit of cost calculation of a data service usage. The cost of using a data service depends on the amount and method of data fetched for usage by a particular application. There are two main volume-based pricing approaches:

- **Quantity-based Pricing:** With this approach companies are charged based on the amount of data they need to access. For instance, an organization requires 1000 calls per day or 100 calls per hour. This is the easiest method to calculate the cost of data service usage, because it fixes the number of API's/calls to access the data within a specified time.
- **Pay Per Call** Another model for quality based pricing is “pay per call”, where a call is a single request/response interaction with the API for data. This model is the most suitable for lower volume of data usage. Usually a fixed rate is agreed between data service provider and consumer for each call to the data service.

4.2.2.2 Data Type Based Model

The data type based model considers the data type and quality of the data as a basic the unit of cost calculation. This model separates the pricing tiers by data type or attribute and a consumer is charged based on the number and type of attributes accessed. An example is a mapping API that offers the geo-coordinates and zip codes of the neighborhoods in an urban area. Additional attributes could include school or post office locations, which are sold for an additional charge.

Data can be sliced and diced in many ways. Some complex pricing models combine data type based model with volume based model to create finer-grained pricing to better meet both the buyer's and seller's needs.

4.2.3 Data Services: Advantages

Data services provide several advantages over the existing data management models, a few of them include:

- **Dynamic Integration:** Data services provide a better way to create loosely coupled data integration applications. Each data service is independent and has a specific domain which enables data integration applications to dynamically locate and integrate data services into their system.
- **Low Maintenance:** Data providers can build the database with the help of data experts and outsource their data as data services. Companies can easily get rid of high maintenance cost and efforts by leaving all the data management tasks to the data providers.
- **Agility:** Data services provide agility and their customers can easily introduce new changes and can move quickly due to the simplicity of the data access. Because all the data management issues are handled by the data providers and the customers do not require any extensive knowledge of the underlying data. If customers require a slightly different data structure or they have location specific requirements, then the implementation is easy because the changes are minimal.
- **Cost Effectiveness:** Data services provide a cost effective and economic environment for companies which have to bear huge capital expense in the form of building their own data centers and their maintenance.
- **Data Quality:** Data providers gain expertise in data management because of their single domain restricted to data only. Agility of data services where any customer can easily move to other data services brings market competitiveness among the data providers. This competition results in better services and quality of the data. Data services also improve data quality by providing a single entry point to access the data controlled through the data services, which tends to improve data quality because there is a single point for updates.

4.2.4 Data Services: Challenges and Issues

Despite all the benefits and advantages of the data services described in the previous section, data services also face some critical issues and challenges which obstacle in the widespread use of the data services. Below we discuss a few of the challenges faced by data services.

- **Dealing with huge data sets:** Data services struggle to move large amounts of data sets. Despite all the technology improvements to provide high speed networks, it is still not easy to move multiple terabytes of data.
- **Privacy, Security and Trust:** Most of the organizations still want to keep a control over their data. Organizations are reluctant to adopt data services because of privacy, security and lack of trust.
- **Copyrights:** A common criticism specific to data services is that the consumer is really just “renting” the data, using it for a specific requirement. There is lack of complete implementation of copyrights laws and data re-usage permission.
- **Legal Issues:** International law and regulations are not designed up to the realities of the data services where an organization’s data can reside cross national borders.

- **Reliability:** Mostly data services provide high reliability and performance. But at the same time there is no guarantee for server failure or low connectivity and transmission.
- **Data Concerns:** The data published and processed using data services is often associated with data concerns like privacy, licensing, pricing, quality of data, etc. There is strong need of assuring the individual concerns of all the stakeholders of an integration application.

4.3 Data Concerns

Data services are gaining attention of the organizations because of their advantages over the legacy database systems. Data services provide access to the data from anywhere at anytime. It also reduces the cost of investment which makes it a big attraction for the organization looking for cost effective solution for the data management.

Data published using data services is often associated with various concerns. These concerns must be explicitly described in order to ensure that the data consumer can find and select the most appropriate data services and utilize the data in the right way while assuring all the concerns of data and service provider.

Service oriented architecture allows us to expose any data set or database as a web service. But data services have different requirements and concerns compared with traditional web services. There can be plenty of data concerns related to a data service. The importance of data concerns associated with data services have been realized recently and to tackle these issues, in [75], the authors give an overview of data concerns and discuss the different parties and their roles in data service creation and utilization. Data provided through data services is usually associated with many data concerns [75, 76]. These concerns must be precisely described, organized and stored in such a way that the service provider is capable of preserving these concerns while querying a data source.

For the sake of simplicity we have chosen a few of the most important concerns as shown in Table 4.1. Each concern can be categorized into a particular category depending on the type of the concern. We have categorized the chosen data concerns into three categories namely (1) Data Quality, (2) Quality of Service, and (3) Licensing. Every category of the data concerns has its scope which will be either service level or data level (for further details, see Section 4.6.1). For example in Table 4.1, data concerns belonging to the category of quality of service have service level scope while data concerns belonging to the categories of data quality and licensing have data level scope.

It is worth mentioning that each data concern (whether at service level or at data level) has an assigned value. Different algorithms are available for the calculation of the value of these parameters e.g. [8] for the calculation of the QoS parameter of a service and [82] for the calculation of the trustworthiness of a service. We assume that there are several data services management software packages (e.g. SEMF [74] for web services) which keep track of the values of data concerns belonging to any data service. For the sake of simplicity we populated the data concerns tree within the limited range between 1 to 10, where 10 is the highest value (see Section 4.6.1).

Below we discuss a few of the data concerns in detail.

Category	Scope	Data Concern	Description
Data Quality	Data Level	Timeliness	Defines the lifetime and freshness of the data
		Accuracy	Data correctness and consistency
		Completeness	Missing information in terms of null values etc.
		Availability	Defines possible access limitations
Quality of Service	Service Level	Performance	Defines performance of the data service, e.g. execution time, response time
		Reliability/availability	What is the failure probability and what is the recovery time in case of failure
		Dependability/Trust	Reputation, how trustful is the data service
		Service Location	Location of the service execution
License	Data Level	Usage Permission/Rights	How a data service can be used
		Data Location	Defines where the data resides
		Usage Fee	Defines the fee associated with the usage of the data or data service
		Law Enforcement	Defines laws which are used to deal with the use of the service

Table 4.1: Some Data Concerns Associated with Data Services

4.3.1 Data Quality

Data quality is one of the most important topics in database research community and it has same importance in data services as well. With the advent of data service or cloud computing technologies the focus of data quality has changed because of rapidly changing data sources, highly distributed and availability of the multiple resources to perform the same tasks. Quality of the data can be measured by many parameters. The importance of these parameters varies with the preference of the data consumer, service provider or data provider.

4.3.1.1 Timeliness

Timeliness defines the lifetime and freshness of the data. In the era of the internet, data changes are very frequent and quick. Importance of data quality is highly attached with timeliness of the data. Some companies provide different price models depending on the age of the data. For example, to better understand market trends of any product, eBay provides its users the historical data which is two days behind the current time.

4.3.1.2 Accuracy

Accuracy defines to which degree data is reliable. Does the data contain some possibilities of errors or is it 100% free of error? Different data consumers can have different preferences regarding the accuracy concerns. For example some consumers need data to predict future business decisions based on the analysis of the data, in this case the consumer will have high prefer-

ence for the highest level of accuracy. In contrast, some other consumers want to perform some surveys based on the data which do not require the highest degree of accuracy.

4.3.1.3 Completeness

Completeness describes whether the data has been missing values. The completeness of an individual data element defines whether a data element misses some data fields while the completeness of a data set defines whether a data set misses some data elements. Different data consumers can have different requirements regarding the degree of the completeness of a data service. Consider the example of a meta search engine (a search which is performed over multiple sources and aggregated results are returned) where a user wants to buy a product. For such a user the price attribute should have the highest degree of completeness in order to compare the prices provided by different vendors. In contrast, another user is just conducting a survey on the books categories and user's feedback. Such consumer can compromise on the degree of the completeness of the price attribute.

4.3.1.4 Availability

Availability describes the possible access limitations to access the data. A consumer can have concerns about the availability of the data, because some applications may require high response time. These applications cannot afford to have down time and require the highest degree of availability.

4.3.2 Licensing

Licensing is one of the most important issues of data services, because usually data is attached with many concerns regarding its usage after querying. Mostly, the service provider is concerned about the usage permission, copyrights and liability of using its data. Another issue for licensing is the data provider's concern for the lifetime of the data, e.g. will the query results be discarded after some specific time or can the consumer save the data for further processing?

4.3.2.1 Usage Permission

Describes how a data service can be used. This includes both data and service aspect of the usage permission. We focus only on the data level usage permission which includes (i) distribution, (ii) transfer, (iii) personal use, and (iv) commercial usage etc.

4.3.2.2 Data Location

One of the biggest hurdles in the adoption of data services is the data owner's concern about the location of the data. International laws are not matured enough to deal with the distribution of the data beyond the international borders. The data owner as well as the consumer want to know about the exact locality of the data sources before using them in order to avoid any legal issues.

4.3.2.3 Usage Fee

We have described data service pricing models in Section 4.2.2, each of them fulfilling different business requirements. Data consumers will require the complete knowledge of the usage fee and cost estimation before using a data service.

4.3.2.4 Law Enforcement

Different laws are applicable in different countries regarding the data storage, usage permission and legal issues. Data services provide access to the data where it resides from anywhere at anytime. These features of the data services bring many concerns regarding the law enforcement. For example if a data service is executed from Country A while data is physically located in Country B. Data consumers would require the knowledge of the law enforcement that which of the Country A or Country B law will be enforced on the execution of that particular data service.

4.3.3 Quality of Data Service

Despite its different nature from traditional web services, data services are still services and quality of a service is one of the important issues for the usage of the particular service.

4.3.3.1 Performance

Performance includes several parameters to describe the performance of a data service. The start time parameter shows how much time a data service takes from sending a request till the start of the service. Response time is calculated as the time difference between the first request and execution of the service. Execution time is the main parameter for the calculation of the data service performance which is the time taken for a data service to complete a data request.

4.3.3.2 Reliability

Reliability includes several parameters such as dependability, accessibility, downtime, recovery time etc.

4.3.3.3 Trustworthiness

Trust is also one of the biggest quality factors for the data services because usually data provider, service provider and data consumer discover each other dynamically and establish a contract without any prior knowledge of each other. This makes the trustworthiness parameter an important data concern for all the stakeholders of the data services. There are several methods to calculate the trustworthiness of a web service which can also be applied to data services.

4.3.3.4 Service Location

Similar to the data location data concerns service location is also one of the data concerns for the data service customers. The customers can be concerned regarding the execution location of

the service due to several factors like law enforcement, copyrights issues, and other legal issues etc.

4.4 Data Concerns Aware Integration

As mentioned above, there have been attempts to incorporate privacy concerns into data services [59,63]. Data source selection has been long discussed in the database and information retrieval community and different algorithms have been developed for optimal selection of the database [31]. Different frameworks and techniques are available for the best service selection in a web service environment [56]. In [14], the authors consider user preferences for data source selection while [53] used QoS attribute for the service selection. Selection of data services based on data concerns have not been considered so far. A query language extension method has been used to provide additional functionality for the integrated applications. A framework for data quality aware queries is presented by extending SQL query language [83] while privacy aware querying language has been designed for preserving privacy in distributed query evaluation [29]. To the best of our knowledge, there is no querying system available for data concerns aware querying to integrate data services.

However, the static nature of agreements on data concerns is a severe drawback of the existing technology – in particular in the dynamic data integration scenarios using mashups or cloud strategy. Moreover, several important aspects of data integration applications are not addressed in these solutions, such as (1) automatic updates, (2) changes of regulations, and (3) changes of the location of the data.

Figure 4.3 describes the activities in a data service system with data concerns. As an initial step, both DP and SP mutually agree to provide the facility of the data service. The data provider is required to provide accessibility to its data for the service operations. The DP should provide a mechanism to access its data and additionally provide data concerns associated with the data. The user can also contribute to the data concerns meta-data by providing her preferences. Such information can be easily managed using profiling techniques. The service provider describes the service operations to access the data sources. This access can be realized via a query language, REST/SOAP or a parameterized query for structured data. The SP has to make sure that all the concerns defined by the DP are preserved.

4.5 Models of Data Concerns

Figure 4.4 depicts four possible models for querying data services with concerns. As already discussed in Section 4.2.1 there are three actors/roles involved. For the sake of simplicity we assume that data provider and service provider have already agreed on the storage format of the data concerns: we assume that this information is stored in the form of an XML document whose schema will be described in Section 4.6.1.

The main difference between the models presented in Figure 4.4 is due to the way how the data concerns are associated with the queries that are posed against the data sources: The models in Figure 4.4(a) and (b) use the “*Querying With Concerns*” paradigm. In this case, the system sends a pair $\langle Q, C \rangle$ to the data source, where Q is the query for the data service and C is the

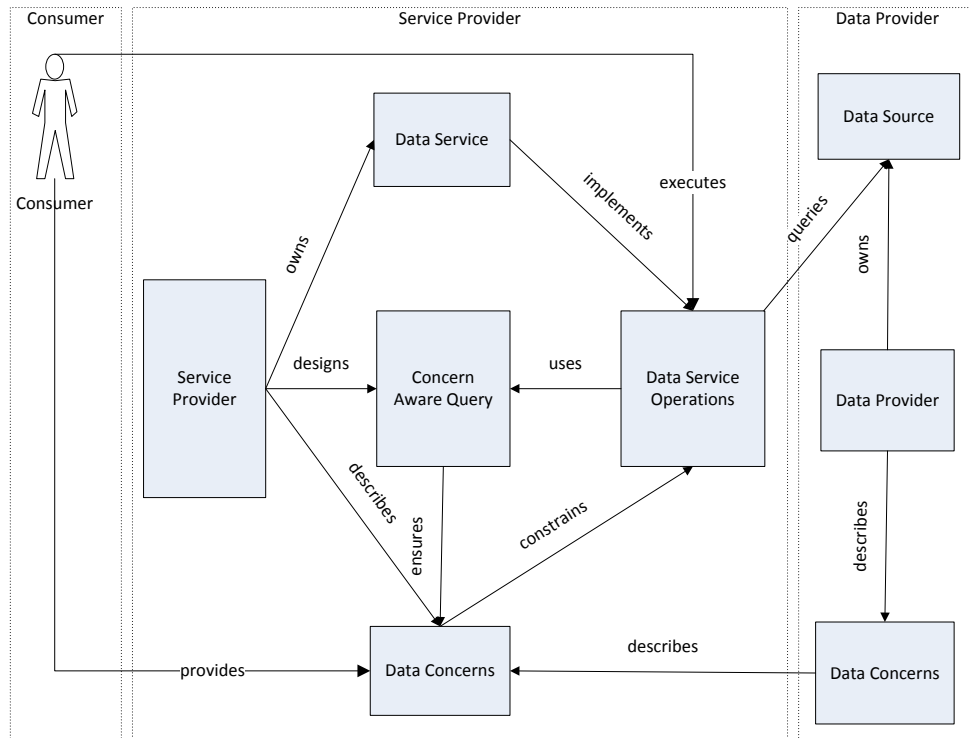


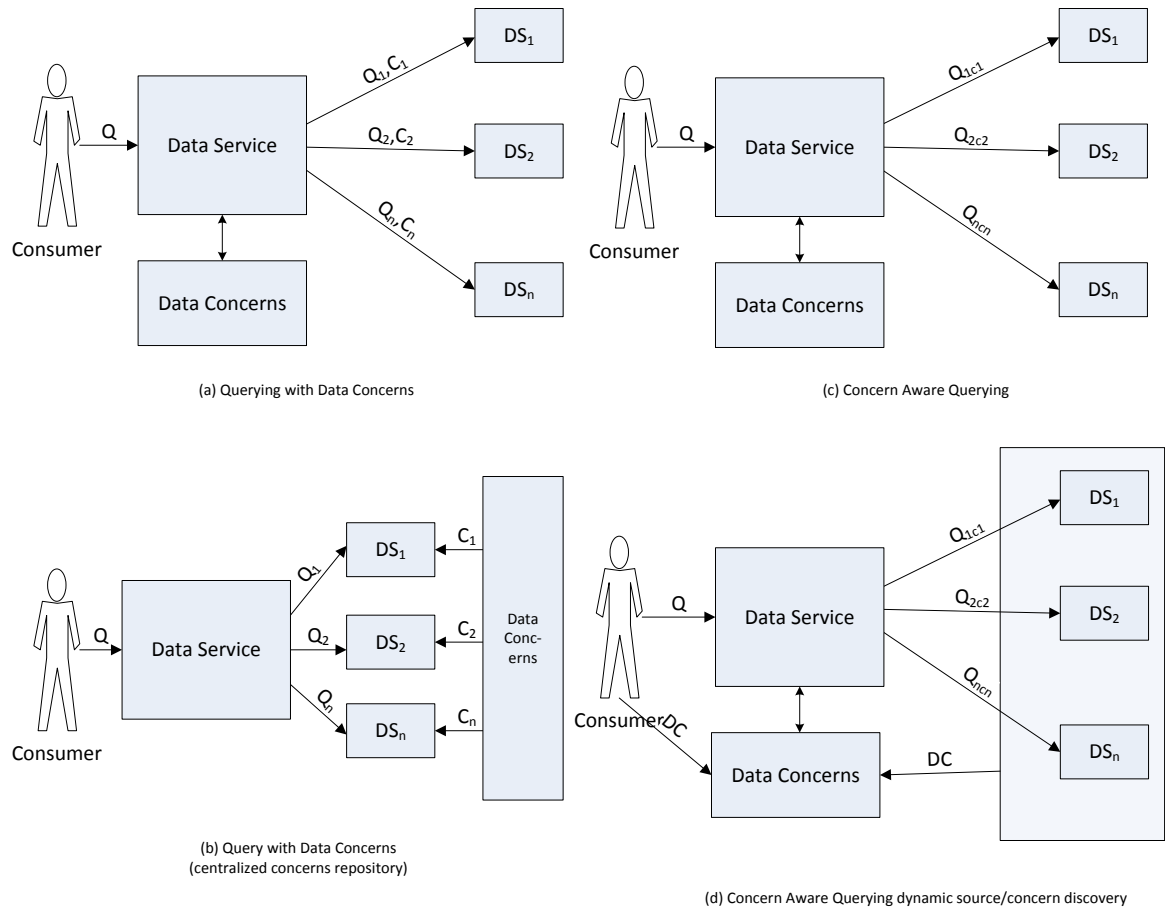
Figure 4.3: Activities in Data Concerns Aware Data Services System

collection of data concerns associated with the query. Each data service must be capable of preserving these concerns by using its own individual querying system. The models in Figure 4.4(c) and (d) are based on “*Concerns Aware Querying*”. The service provider has the concerns either stored locally or fetched dynamically and is capable of writing queries in such a way that they preserve all the data concerns associated with the particular query and its relevant data source.

Below we briefly discuss each of the models shown in Figure 4.4

4.5.1 Model (a): Querying with Data Concerns

Figure 4.4(a) shows the basic model of querying with data concerns. DP and SP have mutually agreed and have accumulated their data concerns. These data concerns are stored by the SP. A user query Q is subdivided into multiple sub-queries for multiple data sources. The SP must be capable of retrieving the data concerns meta-data of a particular data source and attaching it to the query. The data provider must be capable of taking care of the supplied concerns while executing the query and returning the results.

**Figure 4.4:** Data Concerns Aware Querying Models

4.5.2 Model (b): Query with Data Concerns with Centralized Repository

This model deals with the generic data concerns stored in a centralized repository. Such a model is best suited for an inter-organization data integration system, where data concerns are homogeneous for all of the data sources of the organization. As shown in Figure 4.4(b) this model is very similar to the existing data integration systems. The SP sends user queries to multiple data sources and assumes that the data sources are capable of preserving the data concerns by accessing a centralized repository of data concerns.

4.5.3 Model (c): Concerns Aware Querying

The concerns aware querying model of Figure 4.4(c) is a data service capable of re-writing user queries in such a way that all the concerns associated with a particular data service are

incorporated into the query itself. All the data sources and their concerns are already registered at the SP and the required meta-data information is locally stored by the SP. The SP divides a user query into multiple sub-queries in such a way that all the applicable data concerns (which are stored within meta-data file locally stored at the SP) are incorporated into these sub-queries.

4.5.4 Model (d): Concerns Aware Querying with Dynamic Discovery

Contrary to the legacy data integration systems for databases, data services usually have no prior knowledge of the schema or the data source. Mostly data sources are discovered dynamically and the most suitable data source has to be selected. Of course, storing meta-data information of all data sources at internet scale is out of the question. Figure 4.4(d) shows the model for concerns aware querying with dynamic discovery of data sources and their associated data concerns. The SP stores data concerns meta-data as in the concerns aware querying model, but at the same time it is capable of discovering data services dynamically and creating meta-data of data concerns by fetching them dynamically. These concerns are then applied to queries for a particular data source. This model also allows the consumer to provide her concerns and preferences within the query.

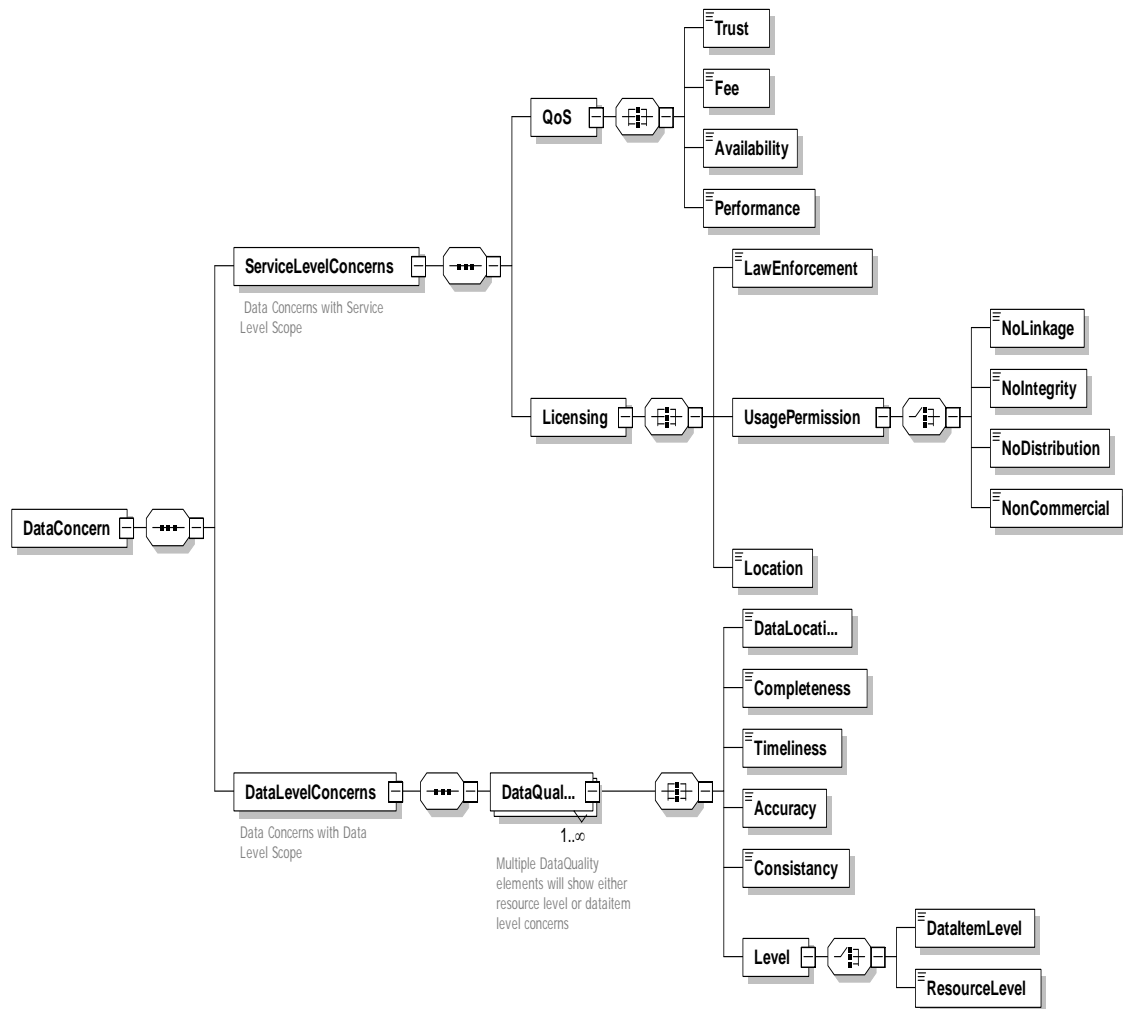
4.6 Concerns Aware Querying

For our system, we have chosen the concerns aware querying model as shown in Figure 4.4(d) since it is the most flexible and most powerful of the models discussed in the previous section. Our system stores the meta-data information of the concerns of all three stake holders of the data service and is capable of querying data concerns dynamically. All three actors of a data service have the possibility to contribute their concerns to the meta-data. Each data provider provides its data concerns associated with its data along with its data. Similarly each consumer has the option to add her preferences/concerns with her query. The service provider can add its concerns to form an aggregated data concerns meta-data which will be utilized by the query language to write concerns aware queries. A data concerns tree (dct) is generated from the available meta-data and is associated with each of the data sources.

We have extended the XQuery language to make it concerns aware, with the introduction of special keywords for mentioning data concerns within the query. The query parser looks into the data concerns trees attached to the available data sources and executes the query by selecting the most suitable data sources and data items for the particular user query. Below we discuss the concerns aware querying system in detail.

4.6.1 Data Concerns Collection

For storing the meta-data information of data concerns of a particular data source, we have decided to use XML because of its standardization and platform independence. Figure 4.6 shows the schema definition for the meta-data for the concerns aware querying. A data concerns tree will be generated by using the meta-data information available in the attached XML file. The schema shown in Figure 4.6 describes the data concerns as shown in Table 4.1. The data concerns collection file can be expanded to any number of concerns depending on the type and

**Figure 4.5:** Data Concerns Tree

requirement of the SP or DP. The data concerns tree can be subdivided into two categories based on the scope of the data concerns.

4.6.1.1 Service Level Concerns

Data concerns whose scope is the entire data service are called service level concerns. The data concerns tree of a data service contains exactly one value for any of the service level concerns. For instance, consider the performance concern of QoS category of a service. Clearly, a data service can have only one calculated value for its performance data concern.

4.6.1.2 Data Level Concerns

Data concerns whose scope is a single data item or a collection of data items are called data level concerns. Data level concerns can have multiple values in the data concerns tree attached to some data source. If a data source returns a node or a list of nodes of an XML document as a result of a user query (written in XQuery), then a data level concern can be described for each of the nodes in this result and for each element in the subtrees rooted at these nodes.

The data concerns tree is populated by the values of data concerns depending on the type of each data concern. There are two possible types of data concerns based on their value, namely (i) *boolean data concerns*, which can have either the value of true or false, e.g. usage permission for commercial purposes can either be allowed or restricted by the data provider, and (ii) *value based data concerns*, which can have any value within a specific range e.g. the performance concern of QoS of a data service or the completeness concern of a data item can have a specific value inside a pre-described range. Different algorithms are available for the calculation of QoS values of a data service or data quality measurements for a data item, which is outside the scope of this paper. We assume that after applying any of the available algorithms and techniques a fixed value (positive integer) is provided for the value based data concerns within a range between 1 and 10, where 10 is the highest level.

4.6.2 Query Processing

If concerns aware querying is used and a query contains data concerns, then the SP iterates through the data concerns trees of all available data sources. Using data concerns information provided in the query and the meta-data tree of data concerns, the data concerns aware querying system is able to select the most suitable data service for a particular query. Algorithm 1 shows the data source selection by assuring both types (i.e., boolean and value based) of service level concerns. A concerns aware query and a set of available data sources is provided as input. After evaluating the Algorithm 1, the most suitable data source or set of data sources (in case multiple data sources assure the desired concerns) is returned for data querying. Once the input is provided, the algorithm starts iterating through the available data sources. For each of the service level concerns mentioned inside a concerns aware query the system verifies whether that particular service level concern is also supported by the data source. If a particular service level concern is supported by a certain data source, as a next step the system compares the required value of service level concern mentioned inside the query with the value of service level concern

of the data source in its data concerns tree. If both conditions are met, a data source is included in the list of suitable data sources for that particular query. Similarly, DLCs are also evaluated using the same technique mentioned in Algorithm 1 for data selection to assure data level concerns.

inputs : Concerns Aware Query Q , $DS = \{ds_1, ds_2, \dots, ds_n\}$

output: $DS' \subseteq DS$

```

1   $DS' = \{\phi\}$ ;
2  for each  $ds_i \in DS$  do
3       $flag = true$ ;
4      for each  $Q.SLC_b$  do
5          if  $Q.SLC_b \neq ds.SLC_b$  then  $flag = false$ 
6          end
7      end
8      if  $flag == true$  then
9          for each  $Q.SLC_v$  do
10             if  $Q.SLC_v \geq ds.SLC_v$  then  $flag = false$ 
11             end
12         end
13     end
14     if  $flag == true$  then  $DS' \leftarrow ds_i$ 
15     end
16 end
17 Return  $DS'$ ;

```

Algorithm 4.1: Data Source Selection based on SLC

4.6.3 Concerns Aware XQuery

We have chosen XQuery because it is the de facto standard language for XML data and because of its capability to execute distributed queries over heterogeneous data sources [3]. Now consider a dynamic data sources integration system as described in Figure 4.4(d), where a user query can be executed on multiple data sources which perform the same task and one data concerns tree is attached to each data source. Current query languages for semi-structured data like XQuery or SPARQL need to provide the URI or location of the data source manually. In order to select the most appropriate data source from the available data sources which perform the same task, we have extended the XQuery syntax by providing additional keywords to make it concerns aware.

For service level concerns we introduce a new keyword “SLC”. A comma separated list of service level concerns inside square brackets will immediately follow the “doc” function of the XQuery language or wherever the URI for the location of data sources is provided. The syntax can be illustrated as

$doc("xmldoc.xml")$
 $SLC [ServiceLevelConcern1 \text{ op value}, \dots, ServiceLevelConcernN \text{ op value}]$

where N is a finite number of service level concerns in a concerns aware query (formulated in XQuery) and op can be any of the basic comparison operators.

For data level concerns we used the simple notation of attaching each data level concern with the variables defined within concerns aware XQuery. Variables having data level concerns will be written inside square brackets with one data level concern attached with the variable. The syntax can be illustrated as

[\$varA.DataLevelConcern op value]

where $\$varA$ can be any variable defined within XQuery and op can be any of the basic comparison operators.

4.7 Implementation and Evaluation

4.7.1 Implementation

We have implemented a data concerns aware XQuery tool to support the idea of concerns aware querying. To store XML data sources, we use the open source native XML database system eXist-db⁵ (release 1.4.0). Our concerns aware querying tool prototype is implemented in Java (JDK 6) on top of the XQuery processing facility provided by eXist-db. When a concerns aware query is submitted to the tool it implements the methods described above and selects the most suitable data sources for a particular query. Once the most suitable data sources have been selected, our tool excludes concerns aware querying clauses and sends standard XQuery to the selected data sources.

Figure 4.6 shows a sample query over the XMARK benchmark data of a web application for auctions, which return a list of person names and items bought by them within Europe. A comma separated list of all the service level concerns is described within square brackets using additional keywords defined for the service level concerns immediately after the URL/location of the data source mentioned in XQuery. In the above example two service level concerns are evaluated. The statement “performance > 7” will select only those data services which have a performance attribute value greater than 7 in their data concerns tree, while statement “CommercialUsagePermission=true” will make sure that data returned as a result from the query will have no restriction on the commercial usage of the data. For the purpose of simple illustration we allow using data level concerns related to particular data elements within an XQuery [QName.DLC] as part of concerns aware querying. For example the statement, “\$ei.completeness > 4” in the above example of Figure 4.6 will make sure that only those values of the variable “ei” are selected which have completeness data concern value greater than 4. Boolean data concerns can have either a true or false value. Basic comparison operators can be applied for the calculation of value based data concerns.

⁵<http://exist.sourceforge.net/>

```

let $auction :=
doc("auction.xml") SLC[performance > 7,
CommercialUsagePermission = true]
return let [$sca.timeliness] :=
$auction/site/closed_auctions/closed_auction return
let
    [$ei.completeness > 4] := $auction/site/regions/europe/item
for $p in $auction/site/people/person
let $a :=
    for $t in $sca
    where $p/@id = $t/buyer/@person
    return
        let $n := for $t2 in $ei
        where $t/itemref/@item = $t2/@id return $t2
        return <item>{$n/name/text()}</item>
return <person name="{ $p/name/text()}">{ $a }</person>

```

Figure 4.6: A Sample Concerns Aware XQuery

4.7.2 Experimental Application: Distributed Extended XQuery for Data Integration (DeXIN)

Our data concerns aware XQuery tool is built upon the DeXIN system [3, 4], which is a web based system to integrate data over heterogeneous data sources. DeXIN extends the XQuery language to support SPARQL queries inside XQuery, thus facilitating the integration of data modeled in XML, RDF, and OWL. DeXIN supports the data integration of XML and RDF data without the need of transforming large data sources into common format. It is a powerful tool for knowledgeable users or web applications to facilitate querying XML data and reasoning over semantic web data simultaneously.

To build our data concerns aware XQuery system, we have incorporated the data concerns awareness into DeXIN. Now DeXIN can integrate heterogeneous distributed data sources while preserving their individual data concerns. It is worth mentioning that by incorporating data concerns into the DeXIN system not only XQuery capabilities are enhanced for data concerns assurance but SPARQL is also enhanced with data concerns awareness using the DeXIN tool.

4.7.3 Evaluation

In order to evaluate the performance and concreteness of our concerns aware querying tool, we have conducted tests with realistically large data sets. As a proof of concept we have evaluated our system on XML benchmark data. We used the XMARK⁶ benchmark data set for experimental analysis. XMARK is a popular XML benchmark and models an internet auction application.

⁶<http://www.xml-benchmark.org/>

Data Source Name	File Size	No. of Copies	No. of SLC	No. of DLC
Auction1.xml	30 MB	20	3	5
Auction2.xml	70 MB	30	3	8
Auction3.xml	100 MB	10	3	4

Table 4.2: Data Sources with Varying Size and Data Concerns

We made three subsets of varying size of auction data provided by XMARK. Table 4.2 shows the details of the data services used for experimental analysis. We made further copies of the subset of the XMARK auction data and defined each as a data service. Each data service assures a varying number of service level concerns and data level concerns. The resulting data services were constructed with the same functionality but with different concerns.

Due to the unavailability of data services which support data concerns, we randomly generated data concerns tree meta-data for each data service and assigned different values to both service and data level concerns. To assure the distribution of the data services we set up a testbed which includes 3 computers (Intel(R) Core(TM)2 CPU, 1.86 Ghz, 2GB RAM), one running SUSE Linux with kernel version 2.6 while the other two running Windows XP. The machines were connected over a standard 100Mbit/S network connection. An open sources native XML database eXist is installed on each system to store XML data. We utilize the eXist XQuery processor to execute XQuery queries.

We used 20 different sample queries ⁷ provided with the benchmark and executed each of them with different data concern values. There was no reported failure in the concerns aware query execution and all the provided data concerns were assured, which proves the suitability of our tool and the potential for its incorporation into any data service integration application.

4.8 Discussion

In this chapter, we have designed a querying system which is capable of taking several kinds of data concerns into account. We have provided a basic model in which we concentrate on three concerns, namely data quality, quality of service, and licensing. However, our approach is generic in the sense that one can incorporate arbitrary data concerns. Indeed, one item on our agenda for future work will be to integrate further data concerns like pricing, data security, auditing model, etc. Another important goal for future work is the integration of our querying system into a powerful mash-up tool. So far, our querying system is designed to access data sources via XQuery. In the future, we want our system to access also data sources which expose their data via web services.

⁷<http://www.ins.cwi.nl/projects/xmark/Assets/xmlquery.txt>

Data Integration Using Data Mashups

Mashups are applications that aggregate functionality, presentation, and/or contents from existing sources to create a new application. Contents are usually generated either using web feeds or an application programming interface (API). Both approaches have limitations as web feeds do not provide powerful data models for complex data structures and lack powerful features of database systems. On the other hand, API's are usually limited to a specific application thus requiring different implementations for each of the sources used in the mashups.

In this chapter we propose a query based aggregation of multiple heterogeneous data sources by combining powerful querying features of XQuery and SPARQL with an easy interface of a mashup tool for data sources in XML and RDF. Our mashup editor allows for automatic generation of mashups with an easy to use visual interface. We utilize the concept of data mashups and use it to dynamically integrate heterogeneous web data sources by using the extension of XQuery proposed in DeXIN. All the available data sources over the internet are considered as a huge database and each data source is considered as a table. Data mashups can generate queries in extended XQuery syntax and can execute the sub-queries on any available data source contributing to the mashup. XML and RDF are the prevailing data formats for web data sources. To query these data sources, one can use XQuery and SPARQL – their respective query languages. The novelty of our tool is that it integrates the powerful features of database querying into a data mashup tool. It provides an easy to use interface of a mashup editor to generate complex queries visually for the integration of a multitude of distributed, autonomous, and heterogeneous data sources.

The work presented in this chapter has been published in [7], where we presented DeXIN as a mashup tool. We start this chapter by discussing different mashup types and their categorization based on frameworks. We briefly explain existing mashup tools and compare their features, advantages and drawback. We present DeXIN architectural view as a Mashup Editor and discuss its functionality. In the end, we show the implementation of the DeXIN tool as a mashup editor and explain how the DeXIN framework can be effectively utilized to benefit from combined features of (i) powerful query languages with (ii) a visual and easy to use interface of mashups.

5.1 Motivation

The amount of structured and semi-structured data available on the internet has been steadily increasing and many companies are now providing their data publicly accessible through API's, querying interfaces, RESTful web services, or data services [25]. The rapid growth of web 2.0 technologies has motivated many big companies to make their contents reusable for the creation of new applications using existing data. Many publicly accessible API's such as Google Maps¹, Amazon² and DBPedia³ are available for the users to generate their own new applications using their existing contents. A typical example of such a scenario is the combination of the list of hotels in a particular city with Google Maps to generate an interactive map of hotels or data collected from several news sites and merged together to provide a single access point to the user.

Mashups are web applications that consume the available data from third parties and combine/reuse them to build a new application. Mostly the contents are in the form of web feeds or API's. All the contents are combined either on client-side using client-side scripts or on server-side using some available server-side technology such as ASP, JSP, etc. Mashups are different from traditional web applications because they are usually dynamically created to serve a very specific and short lived task. Several mashup editors have been launched to encourage people to build new applications using the massive amount of publicly available contents. Yahoo Pipes⁴, Google Mashups⁵ and IBM Mashup Center⁶ are a few examples of the popular mashup editors.

However, the limitation of existing mashup editors is that they focus only on web feeds or API's. These web feeds can represent simple information but lack the capability to represent or query data items provided by querying interfaces or data services [78]. On the other hand, API's are usually limited to a specific application thus requiring different implementations for each of the sources used in the mashups. Currently, the development of data mashups to deal with complex data structures requires strong programming skills, therefore making the mashups generation process hard for the novice users.

5.2 Mashups

Mashups are web applications which are generated by the combination of the contents, presentation, and/or application functionality from existing web sources. Mashups typically integrate heterogeneous data sources available on the web, these sources are in the form of RSS or Atom feeds, various XML formats, or HTML. The aim of the mashups is to combine existing sources to create a new useful application or a service. Contents, functionality and presentation are combined using

- client side scripting e.g. Java script,

¹<http://maps.google.com>

²<http://www.amazon.com>

³<http://www.dbpedia.org>

⁴<http://pipes.yahoo.com/pipes>

⁵<http://code.google.com/gme>

⁶www.ibm.com/software/info/mashup-center/

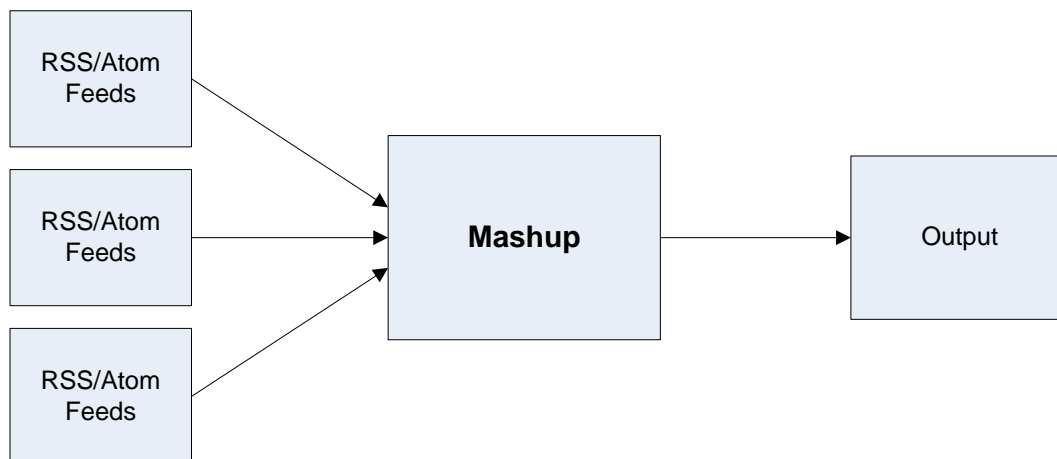


Figure 5.1: Mashups Architecture

- server side scripting e.g. JSP, PHP or Ruby,
- traditional programming languages such as Java or C#.

Figure 5.1 gives an overview of the abstract level of the mashups architecture. Mashups typically integrate heterogeneous web data sources available on the web in the form of RSS or Atom feeds, web services, contents from a third party, or widgets. Initially it was a hard task to combine these data sources using hand written programs, but nowadays the availability of mashups tools or mashups platforms has significantly improved the development process of the mashups. The most common tools for mashups generation are Yahoo Pipes⁷, Dapper⁸, or Intel Mash Maker⁹. These tools provide a graphical interface for the generation of the mashups which also enables unskilled web users to easily generate their own mashups.

In the last few years after the introduction of the mashups [46], mashups have become one of the hottest topic in the web applications areas. Many companies and organizations are rushing to provide mashups solutions. Different kinds of mashups combine several user interface components to build composite interfaces by simply integrating existing web data sources. Web services, data services, and API's have significantly simplified the access, reuse and integration of several components. This paradigm facilitates the development of "situational applications" - which are applications, where the developer is also the final user and those are created to serve a highly focused purpose on demand.

Despite rapidly increasing popularity of the mashups over the past few years, comprehensive mashups development tools and frameworks are still lacking. There are several tools which facilitate easy mashups generation using visual interfaces, but still in some situations, particularly for the creation of data mashups over complex data structures a significant amount of manual programming effort is required.

⁷<http://pipes.yahoo.com/pipes/>

⁸<http://www.dapper.net>

⁹<http://software.intel.com/MashMaker>

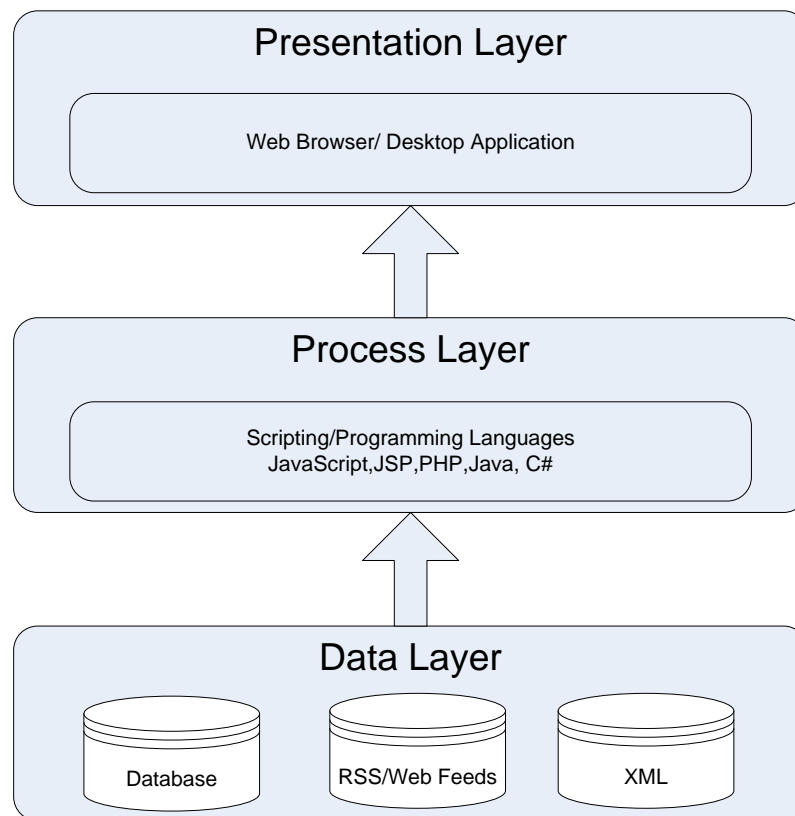


Figure 5.2: Mashups Components Layers

The term mashups is still vague and sometimes it is hard to distinguish between mashups and traditional integration applications. To clear this ambiguity it is important to define

- what mashups are?
- how they differ from traditional integration applications?
- what are different types of mashups and their frameworks?
- which are the existing tools for the mashups generation?

In this section, we discuss the architectural layers of a mashup. We elaborate different types of mashups, existing tools and compare their advantages and disadvantages.

5.2.1 Architectural Layers of the Mashups

Figure 5.2 gives an overview of the mashups components layers. Mashups applications consist of three major components, namely (i) Data layer, (ii) Process layer, and (iii) Presentation layer. All three components can be represented as a layered architecture and each component performs its role at its level. Below we discuss these components in detail.

5.2.1.1 Data Layer

The first and the core layer for the mashups applications is the data layer. This layer is responsible for the data integration. Data sources are usually accessible through different ways e.g REST, SOAP, data services, web services, API's etc. The first challenge for the data layer is to define the access pattern for the data residing in heterogeneous distributed data sources. It is also common in the web environment to have data sources with heterogeneous storage formats. It is the responsibility of this layer to define the rules to deal with the heterogeneity. To summarize, the data layer consists of all possible data manipulation operations (e.g. data retrieval, filtering, and transformation etc.) required to integrate different data sources.

5.2.1.2 Process Layer

The Process layer is also called the logic layer, this is the layer where the actual “mashup” of diverse data sources happens. All the application logic is programmed in this layer and this layer is responsible for the aggregation of the data sources. The process layer can reside either on the server side or on the client side depending on the type of the mashup. There are several languages and processes defined for the process layer in the software architecture as well as in the service oriented architecture. Recently, two languages like Bite [24] and Swashup [55] have been designed specifically for the composition of the mashups.

5.2.1.3 Presentation Layer

The presentation layer provides the interface to the user for the interaction with the mashup. This layer is also responsible to display the final results and the information to the user. Usually the results are presented in the form of a simple HTML page but they can also be presented in the form of complex web page structures using technologies like AJAX, Java Script etc. In the server side mashups the presentation layer is designed by the server which acts as a proxy between the mashup and its contributing services, while in the client-side mashups the presentation layer resides on the client side and performs the composition of the interfaces inside the client's browser using java script.

5.2.2 Types of Mashups

There are many types of mashups but mainly mashups are categorized into three types, (i) consumer mashups, (ii) business mashups, and (iii) data mashups. In this section we briefly describe each of these types and in the subsequent sections we focus on data mashups which are the main theme of this chapter.

5.2.2.1 Consumer Mashups

Consumer mashups are aimed at the general public. Consumer mashups, when offered with a web-based mashup facility can prove to be a very effective means for customer personalization of data/viewing. This is typically where the users use their web browser to combine and reformat the data according to their needs.

5.2.2.2 Business Mashups

Business mashups focus data into a single presentation and allow for collaborative action among businesses and developers. This works well for an agile development project, which requires collaboration between the developers and customer proxy for defining and implementing the business requirements. Business mashups differ from consumer mashups in the level of integration with business computing environments, security and access control features, governance, and the sophistication of the programming tools (mashup editors) used. Another difference between business mashups and consumer mashups is a growing trend of using business mashups in commercial software as a service offering.

5.2.2.3 Data Mashups

Data mashups combine similar types of media and information from multiple sources into a single representation. In contrast to the consumer and business mashups which mainly focus on the presentation layer, data mashups focus on the data transformation and integration techniques. Data sources for the data mashups can vary from spreadsheets to warehouses. Data mashups provide a cost-effective substitute for legacy ETL (extract, transform, and load) systems. Moreover, data mashups provide their users the freedom of dynamically applying the data transformation and integration techniques. An example of data mashup can be the combination of the news data from several data feeds from various news websites or combination of restaurants location data with google maps.

5.2.3 Existing Mashups Tools

There are several mashups tools available in the market. In this section we briefly describe the most popular mashups editors and discuss their virtues. After a brief introduction of the four mashups editors, we compare them based on different features. Table 5.1 gives a comparison of the different features of these tools. In addition to these four tools, there are several other mashup tools proposed and designed to target a specific category of users depending on their domain e.g. Damia [72], Apatar¹⁰, Exhibit¹¹, and Google Mashup Editor etc. are a few to mention. Figure 5.3 depicts the visual interface of the Damia Mashup Editor.

5.2.3.1 Yahoo Pipes

Yahoo Pipes is a web-based tool designed by Yahoo. It is a composition tool to aggregate and integrate content from diverse sources on the web. The contents for the mashup composition consists of web feeds, RSS, Atom or other services. These contents can be aggregated and filtered by applying several parameters e.g. join, filter, and sort etc.

Yahoo Pipes provide a visual interface for the generation of a mashup. A pipe consists of one or more modules, each performing a single task. The user can drag sources and operators on the canvas of the editor and connect them with pipes. To successfully operate with the data,

¹⁰apatar.com

¹¹www.simile-widgets.org/exhibit/

the user must have knowledge of the basic concepts of data, its types and metadata. The output of the mashups is typically a web feed or some other web components e.g. maps, which can be either accessed by a client as RSS or can be visualized on the Yahoo Map.

Yahoo Pipes played a vital role in the wide adaption of the mashups because its visual interface makes the process of the mashups generation a lot easier for the novice user as compared to writing complex program code, which requires skilled programmers. Yahoo does not reveal the source code of the created mashup, but provides a hosting facility to host the generated mashup on their server.

5.2.3.2 IBM Mashup Center

IBM Mashup Center¹² is an end to end mashup platform which is targeted for enterprise users. IBM Mashup Center is a shareware designed to provide an end to end mashup platform for the rapid creation, sharing, and discovery of reusable data. Users can mashup their data from various sources, transform data, and create widgets.

The mashup builder provided inside the IBM Mashup Center tool is responsible for assembling the mashups. The visual editor of the IBM Mashup Center is designed around the primary goal of making the mashup creation process fast and easy for the users of all skill levels. The user can easily build new web applications or mashups from existing sources by dragging and dropping widgets onto the page and then mash them up together.

In addition to mashing up data sources, the existing widgets can be further combined to mashups with a browser-based assembly tool. It is a visual flowchart tool to define an information flow between widgets. It is noteworthy, that in addition to the Mashup Center widgets, the Mashup Center can also utilize Google Gadgets, which are used in iGoogle¹³.

Once a mashup is assembled, it can be easily shared. Visual tools are provided that allow end users to define what users or groups can view or edit their various pages. Mashups can also be published to the catalog, where other users can easily reuse them.

5.2.3.3 Intel Mash Maker

Intel Mash maker is a web based tool designed by Intel. The main focus of the Mash Maker is integration of existing web pages which makes it slightly different from the other mashup tools. It allows editing, querying, and manipulating data over existing web pages.

Mash Maker is a combination of a web based user interface and a custom functional programming language. The web based browser interface allows users to create a mashup by browsing and combining different web pages. The final goal of the tool is to look for the possibility of some enhancements and if they are available for the visited web pages, then suggest them to the user for the creation of the mashup.

The custom programming language for Mash Maker can be accessed by the user using a user interface, which provides a programming by example facility in a spreadsheet style interface. The data is residing all the time on actual web pages and the user can extract and tag the desired data and apply functions to it. Mash Maker suggests some built-in functions for regular users

¹²www.ibm.com/software/info/mashup-center/

¹³<http://www.google.com/ig>

Features	Yahoo Pipes	Mashup Center	Mash Maker	Popfly
Input	Web feeds, RSS, web sources, limited XML	Web feeds, databases, local files, web pages	Web pages	Web pages and web feeds
Output	Feeds and interactions	Widgets	Enhanced web pages	Silverlight components
Editors	Visual	Visual, Programming by example, scripting	Programming by example, scripting	Visual, scripting
Abstraction Level	Intermediate	High, Intermediate, Low	High, Intermediate, Low	High, Intermediate, Low
Data Mapping	Semi-Automatic	Semi-Automatic	Manual	Manual, Automatic

Table 5.1: Comparison of the Features of Various Mashup Tools

while advanced users can also define new functions for Mash Maker. The output of the Mash Maker is an existing web page with added or enhanced data on it.

5.2.3.4 Microsoft Popfly

Popfly¹⁴ is a web-based mashup application designed by Microsoft. It allows the users to create a mashup by combining data from several data or media sources.

Popfly provides a visual flowchart editor built on Microsoft Silverlight technology. The user can generate mashups by combining small components created from web feeds or web pages. The components are connected in the editor either by using built-in components or users can also create them by themselves.

Popfly generates the result of a mashup as a Silverlight component. Popfly is much more about data visualization than data manipulation so the result of the mashup can be only visualized using the provided visualization tool.

5.3 Data Integration using Data Mashups

5.3.1 Challenges

The growing popularity of the mashups has also gained the attention of the database community and the idea of using data mashups for the data integration has emerged. Mashups are light weight applications which are usually generated to serve a short term purpose. Mashups are one of the remarkable successes of web 2.0 technologies which dynamically combine contents from multiple services and data sources.

However, mashups cannot fully serve the purpose of data integration, particularly when the integration of complex data structures is desired. Currently mashups contents are generated

¹⁴www.popfly.ms/

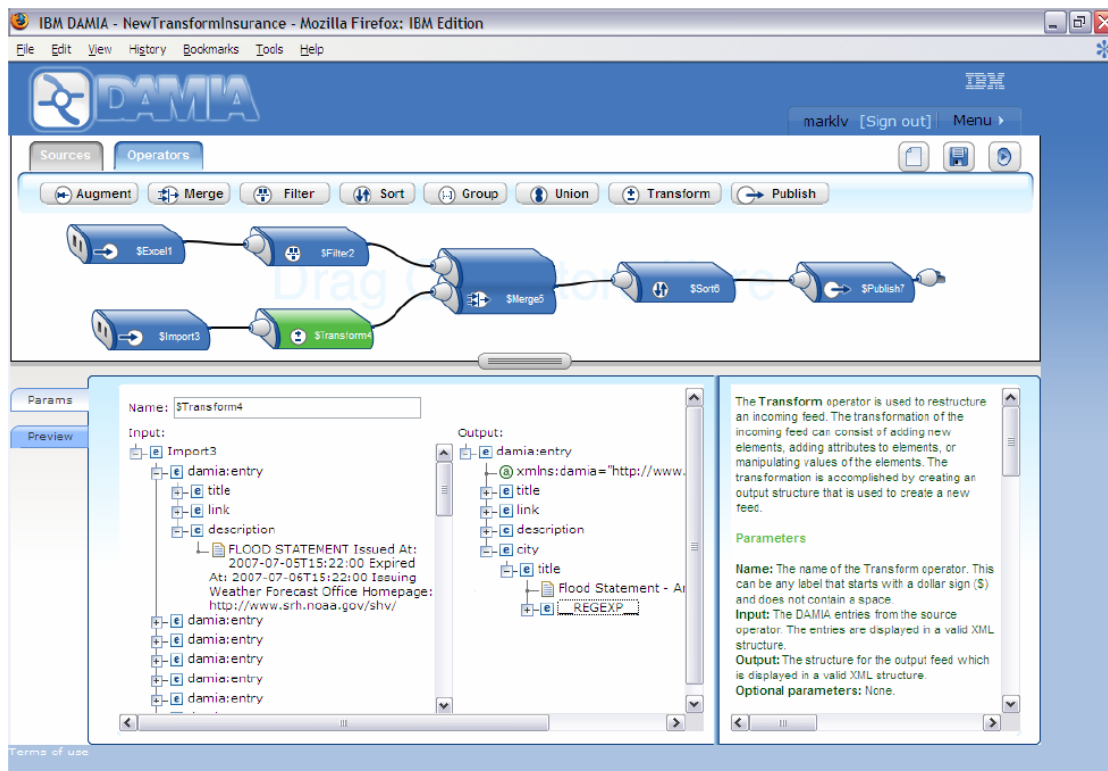


Figure 5.3: The Damia Mashup Editor [72]

either using web feeds or an application programming interface (API). Both approaches have limitations as web feeds do not provide powerful data models for complex data structures and lack powerful features of database systems, while API's are limited only to a specific application thus requiring different implementations for each of the sources used in the mashups.

In parallel to the continuous development of hypertext data on the web, there is also great increase in the amount of structured data accessible on the web through different API's or data services. There is an increasing amount of structured and semi structured data available over the web in a variety of data formats. DBLP data availability in XML format or DBpedia data availability in RDF format are a few examples. This trend of structured and semi structured data over the web is shifting the focus to the requirement of structured data retrieval from the web and its dynamic composition and integration. The only approach available to this requirement so far has been custom programming to perform a specific task. Unfortunately mashups are unable to deal with the complex data structures and are limited to the simple web contents or feeds.

Currently, the development of data mashups to deal with complex data structures requires strong programming skills, hence abolishing the mashups feature of easiness for novice users. Existing data mashup tools cannot deal with structural and semantic diversities of heterogeneous data sources. There are some visual editors to provide an easy interface for novice users without any programming skills. But the functionality of these editors is limited.

In order to properly utilize structured and semi-structured data over the web, there is a strong need for mashups tools which not only can deal with complex data structures but also provide an easy to use visual interface for data integration using mashups.

5.3.2 Database Oriented Mashups

The data layer in mashups is mainly concerned with accessing and integrating heterogeneous web data sources. These sources can provide structured, semi-structured or unstructured data. Recently, the importance of using data mashups for data integration using database oriented mashups has been realized [78]. A new mashup model for distributed data integration has been proposed by [62]. This model is a mixture of both client side and server side mashups techniques and provides a better way to deal with data mashups in distributed environment. A framework for data integration support for mashups has proposed by [73], a new scripting technique is proposed to support the development of more complex mashups incorporation dynamic data integration.

Inspired by Yahoo pipes, there are a few attempts such as MashQL [43] and Deri Pipes [34] to generate semantic queries from data mashups. However, to the best of our knowledge, there exists no data mashup tool which allows the user to formulate queries over web data sources using their respective query languages and at the same time deals with the heterogeneity of the data sources. Below we discuss two query based data mashup frameworks in detail.

5.3.2.1 Deri Pipes

Inspired by Yahoo's Pipes, Deri Pipes implement a generalization which can also deal with formats such as RDF (RDFa), Microformats and generic XML. Deri Pipes is an open source software, and as such it can be easily extended and applied in use cases where a local deployment is needed. The Deri Pipes provides a rich web GUI where pipes can be graphically edited, debugged and invoked. The Pipes execution engine is also available as a stand alone JAR, which is ideal for embedded use.

A semantic web pipe implements a predefined work flow that, given a set of RDF sources, processes them by means of special purpose operators. Unlike full-fledged workflow models, the semantic web pipes model is a simple construction kit that consists of linked operators for data processing. Each operator allows a set of unordered inputs in different formats as well as a list of optional ordered inputs, and exactly one output.

At the core of the Deri pipes is the XSPARQL language proposed in [2]. Pipes provide a visual interface for the data sources and once all the operators are implied using a GUI, a code is generated following the XSPARQL query language. This code is executed over the contributing data sources to fetch the results in the desired format, the output of a pipe is an HTTP-retrievable RDF model.

5.3.2.2 MashQL

MashQL generalizes the idea of mashups and regards the internet as a database. Each internet data source is seen as a table, and a mashup is seen as a query on these tables. MashQL is a

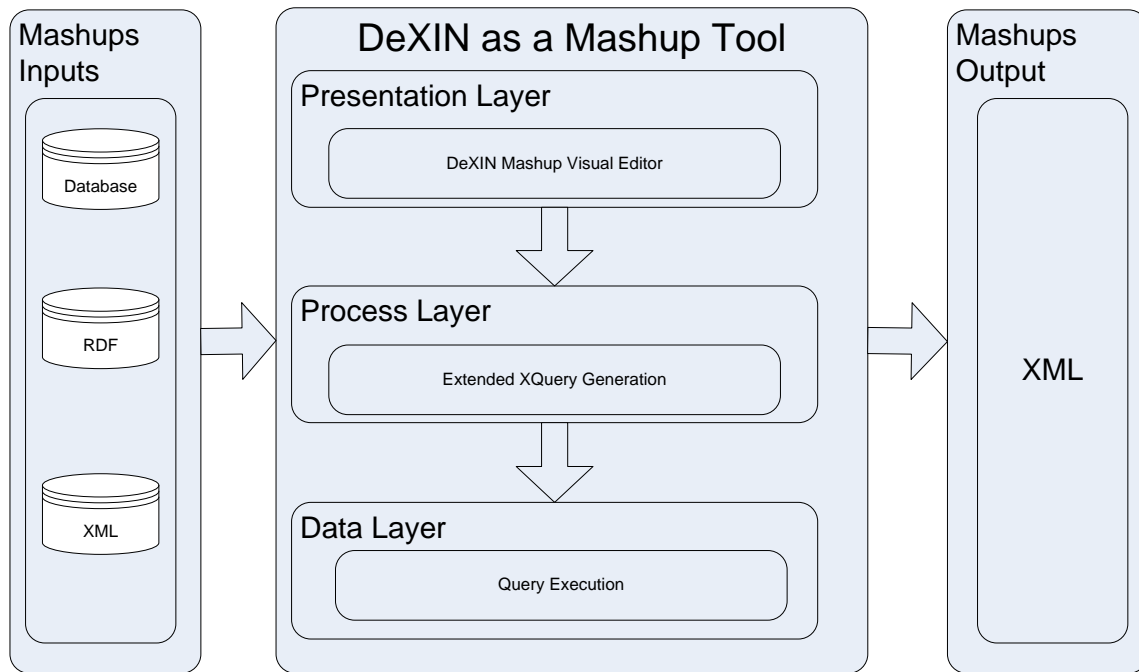


Figure 5.4: DeXIN Mashup Architectural Layers

query-by-diagram language topping SPARQL [43]. The goal of this language is to allow people to build data mashups diagrammatically. In the background, MashQL queries are translated into and executed as SPARQL queries. MashQL allows querying a data source without any prior understanding of the schema or the structure of this source. Users also do not need any knowledge about RDF/SPARQL to get started.

The web is full of various data sources in structured, semi-structured and unstructured format, while MashQL assumes that web data sources are represented in RDF, and SPARQL is the query language.

5.4 DeXIN as a Mashup Tool

We utilize the concept of data mashups and use it to dynamically integrate heterogeneous web data sources by using the extension of XQuery proposed in Chapter 3. All the available data sources over the internet are considered as a huge database and each data source is considered as a table. Data mashups can generate queries in extended XQuery syntax and can execute the sub-queries on any available data source contributing to the mashup. XML and RDF are the prevailing data formats for web data sources. To query these data sources, one can use XQuery and SPARQL – their respective query languages. The novelty of our tool is that it integrates the powerful features of database querying into a data mashup tool. It provides an easy to use interface of a mashup editor to generate complex queries visually for the integration of a multitude of distributed, autonomous, and heterogeneous data sources.

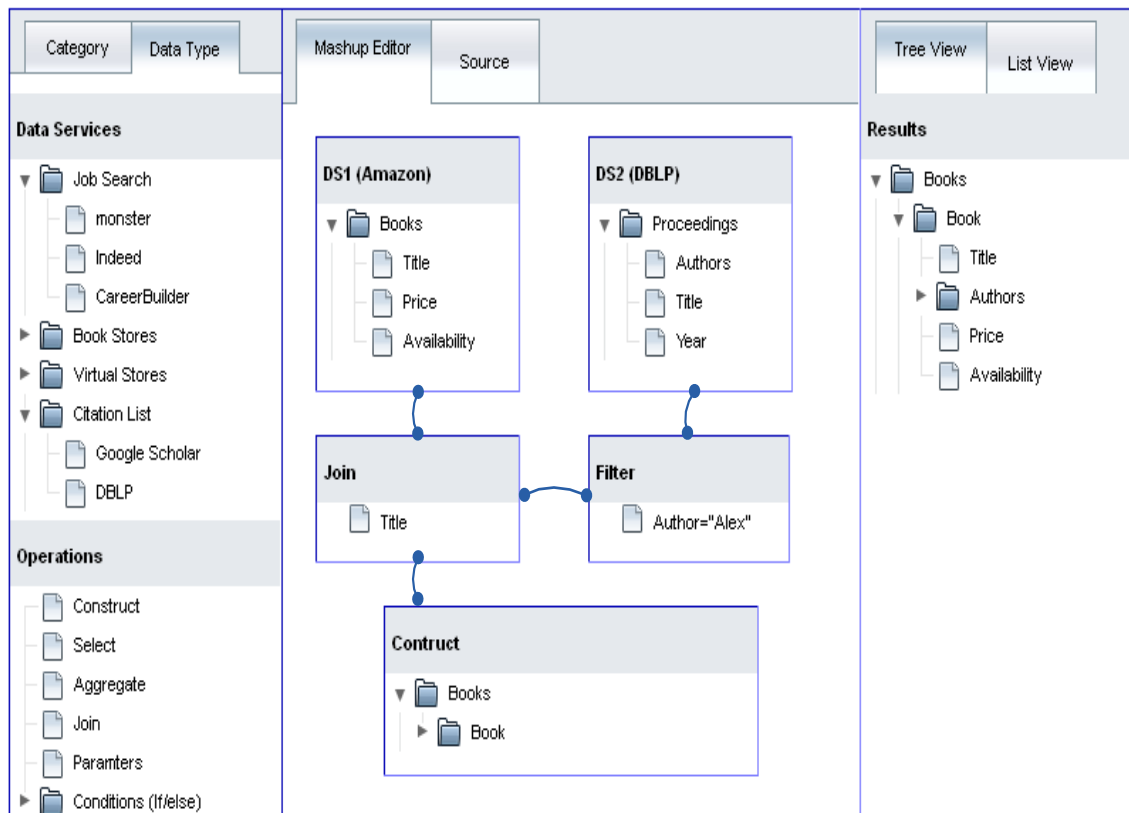


Figure 5.5: DeXIN Mashup Editor

Our tool is similar to MashQL and Deri Pipes, but we focus on the XQuery extension of [3] with additional support of the SPARQL query language. Using our approach, existing data integration support for mashups is further enhanced to formulate a single query containing inside sub-queries of different query languages to deal with heterogeneous data integration.

5.4.1 Architectural Overview

Figure 5.4 gives an architectural layers overview of the mashups generated using the DeXIN mashup tool. Inputs for the mashup can be any data source in XML or RDF data format which provide access to its data by using query interfaces, API's or REST/SOAP protocols. The DeXIN Mashups Editor is a visual interface to generate mashups by using an easy to use graphical interface.

The DeXIN Mashup Editor lies at the presentation layer which is responsible to combine user interface components from several data sources and is also responsible to present the results in the user's desired format. The process layer of a mashup is responsible for the internal logic of a mashup, in case of the DeXIN Mashup tool, the process layer will generate XQuery following the extended XQuery syntax presented in DeXIN. The query generator will automat-

```

for $a in
doc(" http://WISIRISFuzzySearch/License.xml")/agreement,
$b in SPARQLQuery(" SELECT ?Availability ?ExecutionTime
WHERE {
?x <http://www.w3.org/2001/sub#avail> ?Availability .
?x <http://www.w3.org/2001/sub#QoS> ?ExecutionTime "
    }, http://WISIRISFuzzySearch/QoS.rdf)/result
RETURN
  <Result>
    <ServiceTitle>{$a/title}</ServiceTitle>
    <Requirement>{$a/requirement}</Requirement>
    <Availability>{$b/availability}</Availability>
    <ExecutionTime>{$b/ExecutionTime}</ExecutionTime>
  </Result>

```

Figure 5.6: A Sample Query in extended XQuery Format – Generated from the Mashup Visual Interface [3].

ically generate XQuery for DeXIN from the the visual interface of the DeXIN Mashup Editor. Finally, at the data layer of the DeXIN Mashup distributed queries will be executed over heterogeneous data sources and results will be combined and returned to the user. Figure 5.5 shows the interface of DeXIN the Mashup Editor. The main window is divided into three panels, namely data source selection, mashup editor, and query results.

5.4.2 Data Source Registration

The registration of data services at DeXIN mashup tool is not mandatory because DeXIN can interact with any data service at runtime, but providing some metadata about data services by following the registration procedure makes the mashups generation process simpler from user's perspective. Each data service must provide a unique name, should have one of the DeXIN supported data types and querying interface to query the data service. Data providers can provide additional information about schema, user privileges, license and legal issues to facilitate the users to interact with their data sources effectively.

5.4.3 Data Source Selection

All available/registered data services are shown in the left panel of Figure 5.5. Each data service describes its available data source, its functionality and schema (if provided) which help the user to select the most suitable data service. Data services can be arranged in different categories based on meta-data provided while registering a data source. Alternatively, data services can be grouped according to their data format (i.e., XML or RDF) by choosing the "data type" option in the left panel.

5.4.4 Mashup Editor

The central panel in Figure 5.5 is the mashup editor. The user can select any data service from the left panel and can easily drag and drop it into the mashup editor. These data services can be combined via several data operations, which are also selected by drag and drop from the left panel. The mashup is implemented by generating an extended XQuery expression with sub-queries in SPARQL or XQuery following the syntax in [3]. Figure 5.6 shows a sample extended XQuery expression generated by the mashup editor after integrating XML and RDF data sources. This query contains a SPARQL query as a sub-query inside XQuery. The mashup editor has both a design view (by choosing the “mashup editor” option in the central panel) and a command line interface (via the “source” option in the central panel). The design view provides an easy graphical interface while the command line interface is used by an expert user to write queries in the extended XQuery syntax described in [3]. For the creation of the queries from the graphical interface we use a similar approach as described in [52] for XQuery generation and [70] for SPARQL query generation.

5.4.5 Query Results

In the right panel of Figure 5.5, the result of executing the query from the mashup editor is displayed. The data format of the result is always XML. The user can choose between two different views of the XML result: either in tree form (as shown in Figure 5.5) or in table form (by choosing the “list view” option in the right panel).

5.5 Discussion

In this chapter, we proposed the idea of using mashup tools for the dynamic integration of heterogeneous web data sources. Mashups are generated by the combination of the contents, presentation, and/or application functionality of the existing web sources. Several mashups editors provide easy and simple interfaces, which encourage people to build new applications using the massive amount of publicly available contents over the web. Mashups can be considered a great success for the application which require simple aggregation of existing data sources. However, the limitation of the existing tools is that they focus only on simple contents e.g. web feeds. In parallel to the continuous development of the web the amount of (semi)-structured data on the web is also at great increase. The process of building data mashups which require complex data structures integration is an art that is limited to the skilled programmers, because the development of data mashups using existing mashups tools still requires strong programming skills.

In order to better understand the process of the mashups generation, we discussed different mashups types and their architectural layers. We discussed a few of the existing mashups tools, their virtues and compared them to better understand their advantages and limitations.

We proposed a query based aggregation of multiple heterogeneous data sources by combining the powerful querying features of XQuery and SPARQL with an easy interface of a mashup tool for data sources in XML and RDF. We have designed DeXIN as a mashup tool using the three architectural layers of a mashup. We have provided a database oriented mashup tool for

integrating heterogeneous data sources. All the available data sources over the internet are considered as a huge database and each source is considered as a table. Data mashups can generate queries in extended XQuery syntax and can execute to any available data source contributing in the mashup generation. In the end, we showed the implementation of the DeXIN Mashup Editor and explained how the DeXIN framework can be effectively utilized to benefit from the combined features of the powerful query languages with a visual interface of mashups.

Conclusion

In this thesis, we have addressed the problem of data integration by targeting a specific area of distributed heterogeneous web data sources integration. We focused on three specific aspects of this thesis: (i) distributed heterogeneous data integration, (ii) data concern aware querying for data services integration, and (iii) integrating data services using data mashups. A proof of concept is demonstrated with implementation and experimental results.

In this chapter, we shortly summarize the main contributions of this thesis and outline future research goals.

6.1 Summary of the Contributions

Fundamentally, the thesis is based on the three observations which include:

- Web data sources are rapidly growing and there is a strong need for web applications to handle queries over heterogeneous, autonomous, and distributed web data sources.
- The published data over the web is often associated with several data concerns. It must be ensured that data consumers utilize the data in the right way and are bound to the rules and regulations defined by the data and service provider.
- A query based aggregation of multiple heterogeneous data sources should be combined with an easy interface of mashups editors to generate complex data integration applications visually for the novice users.

Based on the above mentioned observations, we designed our research goals which led to the following contributions:

6.1.1 Heterogeneous Web Data Sources Integration

The problem of integrating multiple heterogeneous data sources is not new and many solutions have been proposed over the past (mostly tightly coupled data integration solutions), but the

emergence of web data sources introduces several new challenges for the heterogeneous distributed data sources integration particularly in a single query. To address these challenges, we have presented DeXIN (Distributed extended XQuery for heterogeneous Data INtegration), an extensible framework for distributed query processing over heterogeneous, distributed and autonomous data sources [3]. The idea is to use one data format as a basis (the so-called “aggregation model”) and extend its corresponding query language to execute sub-queries in other query languages. Extensibility is a powerful feature of DeXIN which allows it to select an arbitrary query language and extend its capabilities to execute subqueries in other query languages.

We choose XML as an aggregation model and extended its corresponding query language (XQuery) to execute SPARQL queries as subqueries inside XQuery to retrieve XML and RDF data from a single query. We come up with an extension of XQuery which covers the full SPARQL query language and supports the SPARQL query inside XQuery.

Another important feature of DeXIN is that it avoids the transfer of large amounts of data to a central server for centralized data integration and exonerates the hassle of transforming the entire data sources into one common format to deal with heterogeneity. DeXIN just transforms the query results into a common format (XML in our case) resulting in efficient query processing with low execution time. We carried out several experiments using DBLP and DBPedia as benchmarks, which document the good performance and reduced network traffic achieved using our approach.

6.1.2 DeXIN as a Data Integration Tool

We have implemented DeXIN, as a web based system to integrate data by executing distributed XQuery over heterogeneous data sources [4]. We demonstrate typical use cases of heterogeneous data integration which show that DeXIN is a simple but powerful tool to integrate rapidly changing heterogeneous data sources dynamically. DeXIN can be utilized by many applications where the data sources are unknown at design time, and it eases the integration process from the user’s perspective by not requiring prior knowledge of data sources.

6.1.3 Data Concern Aware Querying for Data Services Integration

We incorporate data concerns associated with data services into query languages for the data integration applications. We design a new concern aware querying system which takes data concerns into account and data concerns are directly integrated into the XQuery language [5].

The proposed querying system

- (i) can take arbitrary data concerns into account,
- (ii) integrates the data concern awareness into the query language, and
- (iii) automatically selects the appropriate data sources depending on current context, user requirements and data concerns.

The problem of integrating data concerns into querying systems is still in its infancy stage and only a handful of published work is available in this area [75, 76]. We laid the foundations

by defining four possible models for querying data which have associated data concerns. Each model can best fit depending on the application requirement, working environment, user context and/or data owner's concerns. We selected one of the models and implemented it using an extension of the XQuery language for data concern aware querying. The concern aware querying system uses meta data information and provides the best solution for integrating data services on the fly while preserving their individual concerns.

6.1.4 Data Services Integration Using Data Mashups

We implemented a mashup editor to dynamically integrate heterogeneous web data sources by using the XQuery extension proposed in DeXIN. Data mashups can generate queries in extended XQuery syntax and can execute any available data source contributing in the mashup generation. We propose a query based aggregation of multiple heterogeneous data sources by combining powerful querying features of XQuery with an easy interface of mashups tools. The novelty of our tool is that it augments the powerful features of database querying to the data mashups tools while providing an easy to use interface of mashups editors to generate complex queries visually for the integration of a multitude of distributed, autonomous, and heterogeneous data sources.

6.2 Future Work

6.2.1 Extend DeXIN to Further Query Languages

An important feature of DeXIN is its flexibility and extensibility. The DeXIN framework allows to choose an arbitrary query language as an aggregation model and extend it to further query languages. So far we have implemented XQuery as an aggregation model and extended it to execute SPARQL query language inside XQuery. A major goal for future work on DeXIN is to extend the data integration to further data formats (in particular, relational data) and further query languages (in particular, SQL).

SPARQL is gaining more attention nowadays because of the popularity and success of the semantic web. Another possibility to extend DeXIN is to use SPARQL as an aggregation model and extend SPARQL to execute further query languages (e.g. XQuery, SQL etc.) inside SPARQL queries.

6.2.2 Optimize the Distributed Query Processing

We are planning to incorporate query optimization techniques (like semi-joins, a standard technique in distributed database systems) into DeXIN for efficient query processing. Currently, SPARQL queries inside XQuery are treated as a set of independent queries which can be executed in parallel. However, the existing version of the DeXIN cannot deal with the dependent dependant or bind joins. Dependant or bind joins are basically nested loop joins where intermediate results from the outer relation are passed to the inner loop to be used as filter, which means that for each value of a variable in outer loop a new subquery is generated for execution at remote site. One of our future agendas is to deal with the dependent or bind joins using the

optimizer described in Chapter 3. In such scenarios the optimizer will first look for all possible values of the variables in outer loop and ground the variables in sub query will all possible values, thus formulating a bundled query to ship at once to remote site.

6.2.3 Bring more Data Concerns into Consideration

Data security, privacy, usage permission, licensing and many other data related concerns need to get more attention while publishing data over the internet using data services. Contrary to enterprise information systems where the domain of the application is within an organization, data services and their integration tools e.g mashups, clouds etc. consider the whole internet as their domain where any resource can be accessed anytime from anywhere. Such technologies provide great facilities for the development of short lived situational applications on demand, but it also raises many new threats of data rights violations, particularly when resources are dynamically discovered and parties involved in the cloud are not known to each other.

So far, we have extended XQuery to deal with some important data concerns (e.g. licensing, data quality, quality of service, etc.), one item on our agenda for future work will be to integrate further data concerns like pricing, data security, auditing model, etc. Our querying system is designed to access data sources via XQuery, another important goal for future work is the integration of our querying system to data sources which expose their data via web services.

6.2.4 Incorporate DeXIN to Other Applications

Cloud computing has recently emerged as a new paradigm for hosting and delivering services over the internet. However, despite the fact that cloud computing offers huge opportunities to the IT industry, the development of cloud computing technology is currently at its infancy, with many issues still to be addressed. We are interested to incorporate data services integration tools into cloud computing architectures particularly for distributed and federated data sources. We are optimistic to merge our existing data services integration research work into cloud architecture for the mutual benefits of both database and web community.

6.2.5 Efficient Resource Allocation based on Data Concerns

One of the key features of cloud computing is the capability of acquiring and releasing resources on-demand. Automated service provisioning is not a new problem. Dynamic resource provisioning for internet applications has been studied extensively in the past. However, it is not obvious how a service provider can achieve the objective to allocate and de-allocate resources from the cloud to satisfy its service level objectives (SLOs), while minimizing its operational cost. An additional actor data provider is involved when it comes about data services clouds, which might have different objectives than the service provider. Furthermore, the consumer wishes to achieve high agility, response time or many other concerns. Other service selection parameters like privacy, licensing, quality of data, quality of service are still not addressed for fully automated service selection. All the resource provisioning decisions must be made online in a cloud environment, we are interested to extend cloud computing architectures to provide better means

for data service integration while dynamically preserving individual concerns of all the actors involved in the cloud.

6.3 Closing Remarks

This work has appeared in the form of conference papers (see Appendix A). In particular, the DeXIN framework was proposed in [3] and its implementation as a tool was demonstrated in [4]. We further extended the concept of taking data concerns into account for data services proposed in [75]. We presented data concern aware querying models and its implementation using XQuery [5, 6]. Finally, a mashup editor was developed and demonstrated in [7] for data services integration using data mashups.

List of Publications

- **DeXIN, an Extensible Framework for Distributed XQuery over Heterogeneous Data Sources**, Muhammad Intizar Ali, Reinhard Pichler, Hong-Linh Truong and Schahram Dustdar, In Proc. ICEIS 2009, volume 24 of Lecture Notes in Business Information Processing, pages 172-183, Springer, 2009.
- **On Using Distributed Extended XQuery for Web Data Sources as Services**, Muhammad Intizar Ali, Reinhard Pichler, Hong-Linh Truong and Schahram Dustdar, In Proc. ICWE 2009, volume 5648 of Lecture Notes in Computer Science, pages 497-500, Springer, 2009.
- **Data Concern Aware Querying for the Integration of Data Services**, Muhammad Intizar Ali, Reinhard Pichler, Hong-Linh Truong and Schahram Dustdar, In Proc. ICEIS 2011, pages 111-119, SciTePress, 2011.
- **On Integrating Data Services Using Data Mashups**, Muhammad Intizar Ali, Reinhard Pichler, Hong-Linh Truong and Schahram Dustdar, In Proc. BNCOD 2011, to appear in volume 7051 of Lecture Notes in Computer Science, Springer, 2011.
- **Incorporating Data Concerns into Query Languages for Data Services**, Muhammad Intizar Ali, Reinhard Pichler, Hong-Linh Truong and Schahram Dustdar, to appear in Lecture Notes in Business Information Processing, Springer, 2011.

Bibliography

- [1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [2] Waseem Akhtar, Jacek Kopecký, Thomas Krennwallner, and Axel Polleres. XSPARQL: Traveling between the XML and RDF Worlds - and Avoiding the XSLT Pilgrimage. In *Proc. ESWC 2008*, volume 5021 of *Lecture Notes in Computer Science*, pages 432–447. Springer, 2008.
- [3] Muhammad Intizar Ali, Reinhard Pichler, Hong Linh Truong, and Schahram Dustdar. DeXIN: An Extensible Framework for Distributed XQuery over Heterogeneous Data Sources. In *Proc. ICEIS 2009*, volume 24 of *Lecture Notes in Business Information Processing*, pages 172–183. Springer, 2009.
- [4] Muhammad Intizar Ali, Reinhard Pichler, Hong Linh Truong, and Schahram Dustdar. On Using Distributed Extended XQuery for Web Data Sources as Services. In *Proc. ICWE 2009*, volume 5648 of *Lecture Notes in Computer Science*, pages 497–500. Springer, 2009.
- [5] Muhammad Intizar Ali, Reinhard Pichler, Hong Linh Truong, and Schahram Dustdar. Data Concern Aware Querying Using Data Services. In *Proc. ICEIS 2011*, pages 111–119. SciTePress, 2011.
- [6] Muhammad Intizar Ali, Reinhard Pichler, Hong Linh Truong, and Schahram Dustdar. Incorporating Data Concerns into Query Languages for Data Services. to appear in *Lecture Notes in Business Information Processing*. Springer, 2011.
- [7] Muhammad Intizar Ali, Reinhard Pichler, Hong Linh Truong, and Schahram Dustdar. On Integrating Data Services Using Data Mashups. In *Proc. BNCOD 2011*, volume 7051 of *Lecture Notes in Computer Science*. Springer, 2011.
- [8] Cristina Aurrecochea, Andrew T. Campbell, and Linda Hauw. A Survey of QoS Architectures. *Multimedia Syst.*, 6(3):138–151, 1998.
- [9] Dave Beckett. RDF/XML Syntax Specification (Revised). W3C recommendation, W3C, February 2004. <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>.

- [10] Dave Beckett and Jeen Broekstra. SPARQL Query Results XML Format. W3C recommendation, W3C, January 2008. <http://www.w3.org/TR/2008/REC-rdf-sparql-XMLres-20080115/>.
- [11] Sonia Bergamaschi, Silvana Castano, and Maurizio Vincini. Semantic Integration of Semistructured and Structured Data Sources. *SIGMOD Record*, 28(1):54–59, 1999.
- [12] Sourav S. Bhowmick, Le Gruenwald, Mizuho Iwaihara, and Somchai Chatvichienchai. PRIVATE-IYE: A Framework for Privacy Preserving Data Integration. In *Proc. ICDE Workshops 2006*. IEEE Computer Society, 2006.
- [13] Nikos Bikakis, Nektarios Gioldasis, Chrisa Tsinaraki, and Stavros Christodoulakis. Querying XML Data with SPARQL. In *Proc. DEXA 2009*, volume 5690 of *Lecture Notes in Computer Science*, pages 372–381. Springer, 2009.
- [14] Sarah Cohen Boulakia, Séverine Lair, Nicolas Stransky, Stéphane Graziani, François Radvanyi, Emmanuel Barillot, and Christine Froidevaux. Selecting Biomedical Data Sources According to User Preferences. In *Proc. ISMB/ECCB (Supplement of Bioinformatics) 2004*, pages 86–93, 2004.
- [15] Tim Bray, François Yergeau, C. M. Sperberg-McQueen, Jean Paoli, and Eve Maler. Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C recommendation, W3C, August 2006. <http://www.w3.org/TR/2006/REC-xml-20060816>.
- [16] Martin Bryan. *SGML - An Authors Guide to the Standard Generalized Markup Language*. Addison-Wesley, 1988.
- [17] Diego Calvanese, Elio Damaggio, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Semantic Data Integration in P2P Systems. In *Proc. DBISP2P 2003*, volume 2944 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2003.
- [18] Don Chamberlin, Jonathan Robie, Daniela Florescu, Scott Boag, Jérôme Siméon, and Mary F. Fernández. XQuery 1.0: An XML Query Language. W3C recommendation, W3C, January 2007. <http://www.w3.org/TR/2007/REC-xquery-20070123/>.
- [19] Donald D. Chamberlin, Denise Draper, Mary F. Fernández, Michael Kay, Jonathan Robie, Michael Rys, Jérôme Siméon, Jim Tivy, and Philip Wadler. *XQuery from the Experts: A Guide to the W3C XML Query Language*. Addison Wesley, 2003.
- [20] Chris Clifton, Murat Kantarcioglu, AnHai Doan, Gunther Schadow, Jaideep Vaidya, Ahmed K. Elmagarmid, and Dan Suciu. Privacy-preserving Data Integration and Sharing. In *Proc. DMKD 2004*, pages 19–26. ACM, 2004.
- [21] Edgar F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, 1970.

- [22] Dan Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3C Recommendation, W3C, September 2007. <http://www.w3.org/TR/2007/REC-grddl-20070911/>.
- [23] Sadie Creese, Paul Hopkins, Siani Pearson, and Yun Shen. Data Protection-Aware Design for Cloud Services. In *Proc. CloudCom 2009*, volume 5931 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 2009.
- [24] Francisco Curbera, Matthew J. Duftler, Rania Khalaf, and Douglas Lovell. Bite: Workflow Composition for the Web. In *Proc. ICSOC 2007*, volume 4749 of *Lecture Notes in Computer Science*, pages 94–106. Springer, 2007.
- [25] Asit Dan, Robert Johnson, and Ali Arsanjani. Information as a Service: Modeling and Realization. In *Proc. SDSOA 2007*. IEEE Computer Society, 2007.
- [26] Christopher J. Date and Hugh Darwen. *A Guide to SQL Standard, 3rd Edition*. Addison-Wesley, 1993.
- [27] Matthias Droop, Markus Flarer, Jinghua Groppe, Sven Groppe, Volker Linnemann, Jakob Pinggera, Florian Santner, Michael Schier, Felix Schöpf, Hannes Staffler, and Stefan Zugal. Embedding XPath Queries into SPARQL Queries. In *Proc. ICEIS 2008*, pages 5–14, 2008.
- [28] Michael Dyck, Jonathan Robie, John Snelson, and Don Chamberlin. XML Path Language (XPath) 2.1. W3C Working Draft, W3C, December 2009. <http://www.w3.org/TR/2009/WD-xpath-21-20091215/>.
- [29] Nicholas L. Farnan, Adam J. Lee, and Ting Yu. Investigating privacy-aware distributed query evaluation. In *Proc. WPES 2010*, pages 43–52. ACM, 2010.
- [30] Mary F. Fernández, Trevor Jim, Kristi Morton, Nicola Onose, and Jérôme Siméon. Highly Distributed XQuery with DXQ. In *Proc. SIGMOD 2007*, pages 1159–1161, 2007.
- [31] James C. French, Allison L. Powell, James P. Callan, Charles L. Viles, Travis Emmitt, Kevin J. Prey, and Yun Mou. Comparing the Performance of Database Selection Algorithms. In *Proc. SIGIR 1999*, pages 238–245. ACM, 1999.
- [32] Fabien Gandon. GRDDL Use Cases: Scenarios of extracting RDF data from XML documents. W3C note, W3C, April 2007. <http://www.w3.org/TR/2007/NOTE-grddl-scenarios-20070406/>.
- [33] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos, and Jennifer Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *J. Intell. Inf. Syst.*, 8(2):117–132, 1997.
- [34] Christian Morbidoni Giovanni Tummarello, Axel Polleres. Who the FOAF Knows Alice ? A Needed Step Toward Semantic Web Pipes. In *Proc. SWAP 2008*, volume 314 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

- [35] Sven Groppe, Jinghua Groppe, Volker Linnemann, Dirk Kukulenz, Nils Hoeller, and Christoph Reinke. Embedding SPARQL into XQuery/XSLT. In *Proc. SAC 2008*, pages 2271–2278, 2008.
- [36] P. Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. A Measurement Study of Napster and Gnutella as Examples of Peer-to-Peer File Sharing Systems. *Computer Communication Review*, 32(1):82, 2002.
- [37] Hakan Hacigümüs, Sharad Mehrotra, and Balakrishna R. Iyer. Providing Database as a Service. In *Proc. ICDE 2002*. IEEE Computer Society, 2002.
- [38] Alon Y. Halevy. Answering Queries Using Views: A Survey. *VLDB J.*, 10(4):270–294, 2001.
- [39] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema Mediation in Peer Data Management Systems. In *Proc. ICDE 2003*, pages 505–516. IEEE Computer Society, 2003.
- [40] Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Data Integration: The Teenage Years. In *Proc. VLDB 2006*, pages 9–16. ACM, 2006.
- [41] Mark Hansen, Stuart E. Madnick, and Michael Siegel. Data Integration using Web Services. In *Proc. DIWeb*, pages 3–16, 2002.
- [42] W. H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., New York, NY, USA, 3rd edition, 2002.
- [43] Mustafa Jarrar and Marios D. Dikaiakos. Querying the Data Web: The MashQL Approach. *IEEE Internet Computing*, 14(3):58–67, 2010.
- [44] Jena. – A Semantic Web Framework for Java, June 2008.
- [45] Jena. – ARQ a query engine for Jena, June 2008.
- [46] Anant Jhingran. Enterprise Information Mashups: Integrating Information, Simply. In *Proc. VLDB 2006*, pages 3–4. ACM, 2006.
- [47] Michael Kay. XSL Transformations (XSLT) Version 2.0. W3C Recommendation, W3C, January 2007. <http://www.w3.org/TR/2007/REC-xslt20-20070123/>.
- [48] Alfred Kobsa. Tailoring Privacy to Users’ Needs. In *Proc. User Modeling 2001*, volume 2109 of *Lecture Notes in Computer Science*, pages 303–313. Springer, 2001.
- [49] Koivunen and Marja-Riitta. W3c semantic web activity. *Semantic Web KickOff in Finland*, 29(1):27–41, 2001.
- [50] Terry A. Landers and Ronni Rosenberg. An Overview of MULTIBASE. In *Proc. DDB 1982*, pages 153–184, 1982.

- [51] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. PODS 2002*, pages 233–246. ACM, 2002.
- [52] James F Brinkley Li Xiang, John H. Gennari. XGI: A Graphical Interface for XQuery Creation. In *Proc. AMIA Symposium 2007*, pages 453–457. American Medical Informatics Association, 2007.
- [53] Yutu Liu, Anne H. Ngu, and Liang Z. Zeng. QoS Computation and Policing in Dynamic Web Service Selection, booktitle = Proc. WWW Alt. 2004, year = 2004, pages = 66–73, publisher = ACM,.
- [54] Giusy Di Lorenzo, Hakim Hacid, Hye-Young Paik, and Boualem Benatallah. Data Integration in Mashups. *SIGMOD Record*, 38(1):59–66, 2009.
- [55] E. Michael Maximilien, Ajith Ranabahu, and Stefan Tai. Swashup: Situational Web Applications Mashups. In *Proc. OOPSLA Companion 2007*, pages 797–798. ACM, 2007.
- [56] E. Michael Maximilien and Munindar P. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 8(5):84–93, 2004.
- [57] Francis McCabe, David Booth, Christopher Ferris, David Orchard, Mike Champion, Eric Newcomer, and Hugo Haas. Web Services Architecture. W3C Note, W3C, February 2004. <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.
- [58] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, W3C, February 2004. <http://www.w3.org/TR/2004/REC-owl-features-20040210/>.
- [59] Frank McSherry. Privacy Integrated Queries: an Extensible Platform for Privacy-preserving Data Analysis. In *Proc. SIGMOD 2009*, pages 19–30. ACM, 2009.
- [60] Wolfgang M. Meier. eXist: Open Source Native XML Database, June 2008.
- [61] Jim Melton. SQL, XQuery, and SPARQL: What’s Wrong With This Picture? . In *Proc. XTech 2006*, 2006.
- [62] Jian Meng and Jinlong Chen. A Mashup Model for Distributed Data Integration. In *Proc. IEEE ICMCG 2009*, pages 168–171, 2009.
- [63] Michaël Mrissa, Salah-Eddine Tbahriti, and Hong-Linh Truong. Privacy Model and Annotation for DaaS. In *Proc. ECOWS 2010*, pages 3–10. IEEE Computer Society, 2010.
- [64] Einar Mykletun and Gene Tsudik. Aggregation Queries in the Database-As-a-Service Model. In *Proc. DBSec 2006*, volume 4127 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 2006.
- [65] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.

- [66] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C recommendation, W3C, January 2008. <http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/>.
- [67] Bastian Quilitz and Ulf Leser. Querying Distributed RDF Data Sources with SPARQL. In *Proc. ESWC 2008*, volume 5021 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 2008.
- [68] Bastian Quilitz and Ulf Leser. Querying Distributed RDF Data Sources with SPARQL. In *Proc. ESWC 2008*, volume 5021 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 2008.
- [69] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, 2003.
- [70] Alistair Russell and Paul R. Smart. NITELIGHT: A Graphical Editor for SPARQL Queries. In *Proc. ISWC (Posters & Demos) 2008*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
- [71] Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Comput. Surv.*, 22:183–236, September 1990.
- [72] David E. Simmen, Mehmet Altinel, Volker Markl, Sriram Padmanabhan, and Ashutosh Singh. Damia: Data Mashups for Intranet Applications. In *Proc. SIGMOD 2008*, pages 1171–1182, 2008.
- [73] Andreas Thor, David Aumueeller, and Erhard Rahm. Data Integration Support for Mashups. In *Interaktion im Web*, 2007.
- [74] Martin Treiber, Hong-Linh Truong, and Schahram Dustdar. SEMF - Service Evolution Management Framework. In *Proc. EUROMICRO 2008*. IEEE Computer Society, 2008.
- [75] Hong Linh Truong and Schahram Dustdar. On analyzing and specifying concerns for data as a service. In *Proc. APSCC 2009*, pages 87–94. IEEE Computer Society, 2009.
- [76] Hong Linh Truong and Schahram Dustdar. On Evaluating and Publishing Data Concerns for Data as a Service. In *Proc. APSCC 2010*, pages 363–370. IEEE Computer Society, 2010.
- [77] Jeffrey D. Ullman. Information Integration Using Logical Views. In *Proc. ICDT 1997*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 1997.
- [78] Andrei Vancea, Michael Grossniklaus, and Moira C. Norrie. Database-Driven Web Mashups. In *Proc. ICWE 2008*, pages 162–174. IEEE Computer Society, 2008.
- [79] M.A. Vouk. Cloud Computing: Issues, Research and Implementations. In *Proc. ITI 2008*, pages 31–40, june 2008.

- [80] Priscilla Walmsley and David C. Fallside. XML Schema Part 0: Primer Second Edition. W3C Recommendation, W3C, October 2004. <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>.
- [81] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, March 1992.
- [82] Stephen J. H. Yang, James S. F. Hsieh, Blue C. W. Lan, and Jen-Yao Chung. Composition and Evaluation of Trustworthy Web Services. *IJWGS*, 2(1):5–24, 2006.
- [83] Naiem Khodabandhloo Yeganeh, Shazia Wasim Sadiq, Ke Deng, and Xiaofang Zhou. Data Quality Aware Queries in Collaborative Information Systems. In *Proc. APWEB/WAIM 2009*, volume 5446 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 2009.
- [84] Nan Zhang and Wei Zhao. Privacy-Preserving Data Mining Systems. *IEEE Computer*, 40(4):52–58, 2007.
- [85] Ying Zhang and Peter A. Boncz. XRPC: Interoperable and Efficient Distributed XQuery. In *Proc. VLDB 2007*, pages 99–110, 2007.
- [86] Gang Zhou, Richard Hull, and Roger King. Generating Data Integration Mediators that Use Materialization. *J. Intell. Inf. Syst.*, 6(2/3):199–221, 1996.
- [87] Patrick Ziegler and Klaus R. Dittrich. Three Decades of Data Integration - All Problems Solved? In *Proc. IFIP Congress Topical Sessions*, pages 3–12, 2004.