A COMPARATIVE SURVEY OF DATASHARING APPROACHES AND THEIR APPLICATIONS IN DISTRIBUTED COMPUTING ENVIRONMENTS

[1]SOFIEN GANNOUNI, [2]HASSAN MATHKOUR, [3]MUTAZ BERAKA
[1]Asstt Prof., Department of Computer Science, King Saud University, Riyadh, KSA
[2]Professor, Department of Computer Science, King Saud University, Riyadh, KSA
[3]M.S. in Computer Science and IT Specialist, Department of Computer Science, King Saud University, Riyadh, KSA
E-mail: gnnosf@ksu.edu.sa,mathkour@ksu.edu.sa, mutaz999@hotmail.com

ABSTRACT

The Internet or distributed computing has evolved from a simple file sharing mechanism to data source sharing and dynamic services. This evolution has made data source sharing an urgent necessity at the present time. Therefore, we should benefit from distributed data sources that are spread across a network and address current data sharing challenges, which include masking the heterogeneity between data sources, and between disparate clients and different communication protocols and formats. In the recent past, efforts have been made by researchers and private companies to propose approaches for accessing remote, heterogeneous and autonomous data sources to share them across a network. Other efforts are looking at these approaches and concepts with the aim of developing applications in both centralized and decentralized environments to provide uniform access to and sharing of data sources. In this paper, we review four data sharing approaches that have been proposed, namely Transaction Processing Monitors, Tuplespace, Resource Description Framework and Data Service Approach. For each approach, we will present its architecture, limitations and problems, as well as applications that have been developed based on its concepts. Moreover, the most important open problems related to data sharing systems are briefly highlighted.

Keywords: *Data Sources, Data Sharing Approaches, Transaction Processing Monitor, Tuplespace, Resource Description Framework, Data Service Approach, Open-Source Applications, Proprietary Applications.*

1. INTRODUCTION

In distributed computing, data is the lifeblood of business enterprises and private users, especially data stored in heterogeneous and autonomous data sources. Additionally, many scientific researches are achieved through the analysis of large amounts of scientific data retrieved from multiple data sources. Sharing existing data sources across network make us take advantage of the enterprise data stored in these sources and open-up the opportunity to integrate data from multiple data sources to gain the holistic understanding about data integration. Therefore, the demand of data sources sharing in distributed computing is more important now than ever before.

By sharing and integrating heterogeneous data sources, the following benefits become gain as follows [1]:

- Eliminates data sources heterogeneity
- Provides valuable resources, which are data sources that available for users
- Accesses and retrieves data from multiple data sources at the lowest cost and short time
- Promotes innovation and potential new data uses
- Leads to new collaborations between data providers and data consumers in distributed computing

The proposed approaches, Transaction Processing Monitor (TPM) [2], Tuplespace [3], Resource Description Frameworks (RDF) [4] and Data Service Approach (DSA) [5], attempt to achieve this type of data sharing in different ways. These approaches differ in the way they deal with the challenges that face users and companies during the development of data sharing systems. However, data sharing approaches realize data locked into heterogeneous data sources and make them available for users to share and exchange in distributed manner. Implementation of these approaches in centralized and decentralized environments produce different applications that attempt to provide a comprehensive solution for data sharing and to address relevant aspects and challenges. These applications can be classified generally into proprietary applications, which require licenses for use, or open-source applications, which are available free for use and allow the use to modify source-code. From an implementation perspective, these data sharing approaches may suffer from some limitations and problems that make development of applications using one approach easier and more useful in certain respects than the other approaches. In addition, other related aspects should be considered when selecting an appropriate approach for developing an application.

So far, the current researches did not mention these approaches in one research. Hence, these researches described approaches of data sharing separately without in-depth review and comprehensive study. In addition, most of these researches ignore review almost applications that have been developed to implement concepts found in these approaches. In this paper, we review and explain in detail approaches of data sharing that have been proposed during the last decade, and related challenges, problems and limitations for each one. In addition, we present applications that have been developed based on these approaches to share data stored in heterogeneous data sources.

The remainder of this paper is structured as follows. Section 1 gives an introduction to the importance of data, share this data and data sharing approaches in distributed computing. Sections 2, 3, 4 and 5 each review one approach to data sharing in detail, including the architecture, some limitations and problems in the approach, and existing applications that have been developed during the last decade. Finally, we present a discussion of these approaches as well as a brief overview of the most common open problems and a conclusion.

2. TRANSACTION PROCESSING MONITORS

A TPM provides an infrastructure for building and administering complex transaction processing systems with a large number of clients and multiple servers[6]. In other words, it considers a standard interface that provides functions to process and execute queries among distributed components. TPM supports mainly services for submitting user queries, routing them through servers for processing, coordinating the two-phase commit when transactions are running over multiple servers and ensuring that each transaction satisfies the Atomicity, Consistency, Isolation and Durability (ACID) properties [2]. These properties guaranty the database's consistency over time and guard against hardware and software errors [6].

As a database middleware, TPM provides a set of tools and an environment to develop and deploy applications that retrieve data from multiple Database Management Systems (DBMSs) distributed over a network. This role discharges the DBMSs from managing data consistency and correctness. However, using the tools provided make integration an easy process because the functionality is directly supported by the TPM [2]. TPM is independent of the persistent layer (databases). It supports flexible and robust business modeling and designs that allow modular, reusable routines and Application Programming Interfaces (APIs) to be added to support other components [7]. Moreover, the flexibility of TPM architecture allows adding and modifying components in a distributed system.

There are three alternative TPM technologies, which are as follows [7]:

- TPM technology is session based
  TPM treats transactions that come from users as messages [7]. The single server provides the services of both database and transaction processing. The session server sends messages to the user to ensure it is still alive until processing and executing the query and sending back the result.

- TPM technology is remote data access
  Remote data access centers allow users to communicate with back-end database servers [7].

- TPM technology is the database approach TPM provides functions to a specific database and its architecture is locked to that database system [7].

Finally, TPM addresses the problems of sharing data from enterprise repositories, providing interfaces and ensuring the ACID properties [2]. In addition, it infers the functions of transaction manager, which are locking, scheduling, logging and recovery, and controls the execution of distributed transactions. Furthermore, TPM may perform load-balancing techniques to enhance performance and throughput [2]. TPM provides a set of administrative tools for administrative functionalities [2].

2.1 Architecture Of TPM

The general architecture of TPM provides an abstract interface that allows developers and programmers to adapt and implement it according to specific application needs. The TPM architecture is shown in Figure 1 [7].

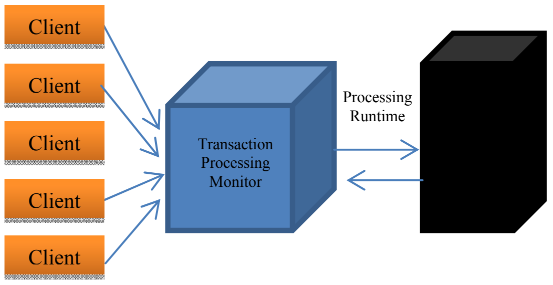

Figure 1. TPM architecture

2.2 Implementation Of TPM

In this paper, we consider only the implementation of TPM that supports sharing and integrating data from multiple data sources. Other implementations are outside the scope of this paper. However, one possible implementation of TPM is as a database middleware system. This system is placed between the application and the database server to provide a standard interface for submitting user queries and applying them on database servers as transactions. In addition, this system may support the integration of collections of data sources distributed over a computer network. We classify the developed applications into proprietary applications and open-source applications.

2.2.1 Proprietary TPM Applications

In this sub-section, we present the proprietary applications, which are commercial applications that implement the TPM approach.

Frameworks/Systems

Active Mediator Object System (AMOSII): itis a DBMS and a distributed mediator system [8]. It provides all database facilities such as a storage manager, transaction manger and AMOSSQL as an object-oriented query language [8]. AMOSII transforms various data models into object-oriented models [8]. In addition, it provides mechanisms for integrating data from data sources. It is a product of research work developed in the EDSLAB at the University of Linköping, Sweden.

Middleware Based On a Code SHipping Architecture (MOCHA): itis a novel database middleware system designed to interconnect data sources distributed over a network [9]. It runs on top of Oracle and Informix and is self-extensible because new application-specific functionalities needed for query processing are deployed to remote sites in an automatic-fashion by the middleware itself [9]. It is a product of research work at the University of Maryland, USA.

Distributed Information Search COmponent (DISCO): itis a system that deals with heterogeneous distributed database systems that allows users to submit their queries [10]. It provides special features to make integration of multiple data sources having the same type easier [10].

Garlic: itis a database middleware system that integrates multiple databases without changing how or where the data is stored [8][11]. It uses a uniform object-oriented data model to represent data from various data sources, and uses an object-oriented SQL as its query language [8]. Garlic is a product of research work at IBM's Almaden Research Center.

Peer Agent System: it is an agent-based transaction in peer-to-peer (P2P) architecture. It provides the ability to exchange data and solves data management heterogeneity problems [12]. Peer agent system is a product of research work at University of Ancona, Italy.

P2P database network (P2PDBN): itimplements the TPM approach in a P2P environment.It is a network of peers without a global transition coordinator, where each peer has their own databases and participates in the network to exchange and share data with other peers [13]. P2PDBN is a product of research work at University of Ottawa, Canada.

Projects/Middleware's

The Stanford-IBM Manager of Multiple Information Sources (TSIMMIS): it provides a set of tools that facilitate the rapid integration of heterogeneous information sources whether the data is structured and unstructured [14]. It allows access to integrated data and ensures this information is consistent [14]. It is a product of research work at Stanford University, USA.

2.2.2 Open-source TPM Applications

In this sub-section, we presentonly one open-source application that implements the TPM approach.

Projects/Middleware's

Concept Relation Assay Value Explorer (CRAVE): it is a database middleware and visualization system that allows users to search, retrieve, and visualize ontologies of phenotypes held in custom database [15].

2.3 Limitations and problems of TPM

The architecture of TPM requires a transaction manager that is responsible for creating and maintaining transaction context, and a resource manager that is responsible for managing the associated database, and for participating in the two-phase commit and recovery protocol [6][16]. Therefore, TPM is a complex system and replacing such a system is not easy so it has a long lifetime. In addition, TPM systems must be able to extend with the changing needs of an enterprise [16].

A limitation of TPM is that the functionality is not well-defined and is based on a specific domain, and it is taught-coupled (system dependent) [2]. Additionally, the implementation cost of TPM technology is not cheap, but is cost-effective, because the result provides significant savings [7]. Regarding the implementation code of TPM, it is usually written in a lower-level language and it is not widely available in visual toolsets [7].

The main problem with TPM as middleware or as a system is satisfying the ACID properties when transactions are running over multiple servers. This requires tracking all transactional operations and databases operated upon. In addition, the transaction context between components should be controlled, the status of transactions monitored and the association between database connections and transactions maintained [16]. Another problem with TPM is performance, especially when serving a large number of users. The load-balancing technique should be implemented to enhance performance and provide fast response times.

Other problems in the TPM technologies mentioned before are described as follows. The session-based technology is not as scalable because when the number of users grows the number of messages increases which effects performance [7]. Remote data access technology also faces the problem of scalability [7]. The database server approach faces the problem of custom implementation of TPM, because TPM provides functions for a specific database.

3. TUPLESPACE

Tuplespace was initially introduced in the Linda parallel programming language [3], which was developedbyDavidGelernterandNicholasCarrieroat YaleUniversity [17].Itprovidesasetofprimitiveoperationsforinserting ,fetchingandretrievingdatafromasharedspace storingusers'data. It allows the data providers to post their data as tuples in the shared space called Tuplespace, and allows data consumers to fetch and retrieve data in these tuples, which matches a certain pattern from that space. Accordingly, Tuplespace is a multiset of tuples, where each tuple is sequence of typed fields, for example, <"Ahmed", "Programmer", "$3000">[3].

Processes running the users' programs communicate through Tuplespace by writing and reading tuples in and out of the shared space [3]. These communications are space decoupling because communication is established in anonymous mode (without knowing the references the other users) [3]. In addition, these communications are time decoupling because two parties of communication should not be available at the same time. Some of the services that Tuplespace provides are [18]:

- Reading a tuple from the shared space with or without removing it. In this case, the calling process is blocked until a matching tuple appears.
- Reading a tuple from the shared space with or without removing it and without blocking. Unlike the previous service, the operation

returns null if no matching tuple exists in the shared space.

- Writing a tuple in Tuplespace.

Tuplespace as a model of computing can be implemented in various programming languages, so it relies on the associated programming languages that developers decide to use. For example, Tuplespace has been implemented in various programming languages such as Java, Ruby, Prolog, the .NET Framework and more.

3.1 Architecture of Tuplespace

The Tuplespace approach can be deployed in a centralized way with one server or in a distributed way with multiple servers. The architecture of centralized Tuplespace is shown in Figure 2[19] and the architecture of distributed Tuplespace is shown in Figure 3. More detail about this classification will be described in the next section.

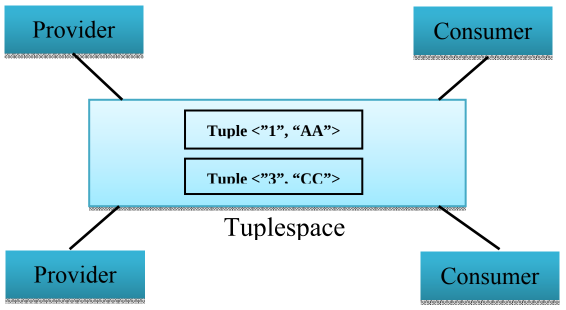

Figure 2. Centralized Tuplespace architecture

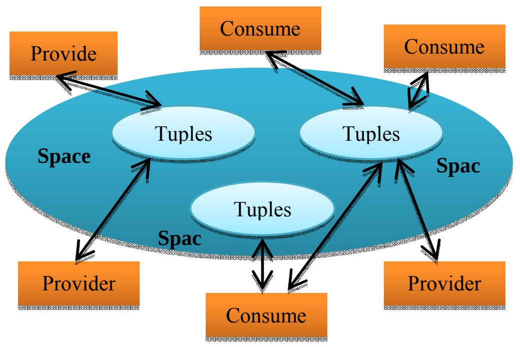

Figure 3. Distributed Tuplespace architecture

3.2 Implementation of Tuplespace

During the last few years, the original model of Tuplespace has been modified and many tuple space applications have been developed [17]. These applications can be classified into two main groups according to the way in which tuples are stored [17]:

- Centralized Tuplespace applications, which store all tuples in a single server.
- Distributed Tuplespace applications, in which tuples in the same space can be stored on different servers and load-balancing techniques can be applied to enhance performance. In other words, the system supports multiple Tuplespaces.

In this paper, we use the above classification of Tuplespace applications and divide them into two groups according to application license: proprietary and open-source applications.

3.2.1 Proprietary Tuplespace Applications

This sub-section summarizes the commercial applications that implement centralized and distributed Tuplespace.

3.2.1.1 Centralized Tuplespace

Projects/Middleware's

Tspaces: A network middleware combines database, Tuplespace, mobile computing and Java technology for the next generation of Tuplespace-based systems[20]. In other words, it combines database functionality and Tuplespace with communications middleware to create platform independent repository that be able to perform new functions and handle new types[20].

3.2.1.2 Distributed Tuplespace

Frameworks/Systems

TIBCO ActiveSpaces: it is a distributed P2P in-memory data grid using virtual shared space. It allows sharing, exchanging and processing of data in real time [21]. This system provides APIs written in Java and C++.

GigaSpaces: it is a platform solution for end-to-end scalability of the application and its data. It also allows cross-language access, which includes the Java, .Net and C++ programming languages [22].

Java PeerSpaces (JPS): it is a system that implements proposed coordination model named PeerSpaces based on the JXTA framework in Java programming language [23]. Peers in a JPS network are connected together, with each peer maintaining local data space, neighbor storage and connection pool [23]. JPS is a product of research work at the University of Bologna, Italy.

Blossom: it is a system that implements distributed Tuplespace in the C++ programming language. Blossom programs do not need a pre-complier to compile them [17] [24]. Blossom is a product of research work in cooperation between Thela Thesis and the Tinbergen Institute.

MTS-Linda: it is an implementation of distributed Tuplespace based on the original Linda model [25][26]. It combines multiple Tuplespaces as first object classes and allows the programmer to maintain them to achieve application specific needs [25][26]. MTS-Linda is a master thesis for Brian Nielsen and Slrensen, Aalborg University, Denmark.

WCL: it is a co-coordination language and runtime systems for geographically distributed agents [25] [27]. It consists of tuple space server, where each server hosts one or more tuple spaces, agents access to tuple spaces through using servers [25] WCL is a product of research work at Cambridge University, UK.

Projects/Middleware's

Comet: it is the communication infrastructure for automated middleware and represents a distributed Tuplespace implementation for Grid like environments [17] [28]. It is research product at Rutgers University, USA.

Tuples On The Air (Tota): it is a middleware for supporting adaptive context-aware applications in dynamic network scenarios [29]. Communication between agents in the Tota network occurs through a distributed Tuplespace [17]. It provides an API that interacts with the middleware easily [29]. Tota is a product of research work at the University of Modena and Reggio Emilia, Italy.

SwarmLinda: it is based on the ant colonies model (swarm intelligence) and implements distributed tuplespace to increase systems scalability [17]. In this model, tuples are considered as food and templates as ants, where an ant tries to find the food [17]. SwarmLinda is a product of research work.

Tupleware: it is a scalable and efficient cluster middleware that implements distributed Tuplespace [25]. It is based on coordination language and incorporates additional techniques to solve scalability and performance problems [25]. Tupleware is a product of research work at the University of Tasmania, Australia.

3.2.2 Open-sourceTuplespace Applications

This sub-section present centralized and distributed Tuplespace open-source applications

3.2.2.1 Centralized Tuplespace

Frameworks/Systems

JavaSpaces: it is a platform for building and developing distributed applications based on the concept of shared Tuplespace (the paradigm of Linda distributed computing) [17] [30].

LighTS: it is a Java implementation of Linda-style Tuplespace [31]. It provides an extensibleframework that makes it easy to introduce extensions tothe Tuplespace, and in general customize the Tuplespace implementation[31].In addition, it provides an adaption layer to allow different Tuplespace implementations to be accessed through it [31].

Projects/Middleware's

Blitz Project: it is an open-source pure Java edition that uses JavaSpaces and JINI thorough the running of essential resources [32].

Rinda: it is a Ruby implementation of the Tuplespace approach (the paradigm of Linda distributed computing) [33].

3.2.2.2 Distributed Tuplespace

Frameworks/Systems

Grinda: it is an implementation of distributed Tuplespaces in structured P2P networks [17]. The server-side is implemented in the Java programming language and the client-side is implemented in the Java and C++ programming languages [17].

Projects/Middleware's

LinuxTuples: it is a C-based open-source Tuplespace server, with a client API written in Python, designed to run on a networked cluster of Linux/Intel boxes [34].

PyLinda: It is an implementation of a distributed tuple space (a paradigm of Linda distributed computing) in the Python programming language

[35]. It supports the implementation of the most widely proposed extensions to the tuple space approach including distributed tuple spaces, garbage collection, sane non-blocking primitives, and bulk tuple operations [35]. However, this system has recently been removed from the PyLinda Web page.

SemiSpace: it is a Java-based open-source interpretation of a distributed tuple space / object space based on JavaSpaces technology [36]. It supports a single space and a cluster of spaces using the Terracotta Integration Module (TIM). The key features offered by SemiSpace are as follows [36]: easy to configure, few dependencies, easy to integrate with Java 2 Enterprise Edition (J2EE) applications, support for generics, ability to distribute space and its content, support multiple tuple spaces and provides a comted-based interface to a SemiSpace Web application.

Linda in a Mobile Environment (LIME): it is a Java-based middleware uses ideas found in the Tuplespace approach and adaptsLinda communication model to provide a coordination layer for designing mobilityapplications[37]. In other words, it is designed to extend the implementation of Tuplespace to support both wired and ad-hoc networks [17][37].

SQLSpaces: it is an implementation of the Tuplespace approach that keeps its API clear and simple [38]. SQLSpaces is intended for relational databases, with the Tuplespace server written in Java and the client API written in the following languages: Java, C#, PHP, Prolog and Ruby [38].

PeerWare: it is a middleware that is designed to support P2P and mobile systems [39]. It adapted the coordination model to allow peers to share data with each other, and allows a set of components to share data and react to any change occurring in these data. It is an open-source middleware.

Fly Object Space: it is a lightweight Object Space can manage information on clusters of computers (multiple tuplespaces) in the form of Objects [40]. This project can use Java, Ruby and Scala via language bindings.

3.3 Limitations and problems of Tuplespace

A limitation of Tuplespace is the representation of information in the form of tuples. Thus, there is no possible distinction between the actual information in tuples and its representation in shared space (there is no standard tuple space interface) [41]. Accordingly, the deadlock problem arise in Tuplespace, which happens when a set of blocked consumers are each holding a tuple and are waiting to acquire a tuple held by another consumer in the set. The implementation of Tuplespace should solve this problem by agreeing on a locking protocol [41].

Tuplespace as a coordination-based model is related to the fixed behavior of the coordination medium. This behavior is set once and for all by the model, and it is not possible to customize it for a specific application [41]. This problem is a clear indication of the control capabilities typically lacking in data-driven coordination models [41].

Another problem with Tuplespace is the amount of memory required for shared Tuplespace. Sharing large amounts of data requires more memory space, which directly affects the performance of the system. This problem in centralized Tuplespace applications leads to a bottleneck [17], whereas in distributed Tuplespace applications affects performance [17]. Therefore, load-balancing techniques should be used to separate the loading, but operation will then be more expensive [17].

A further problem with Tuplespace is the need for expressive matching algorithms to enhance the search for and retrieval of a matching tuple from the space, and to avoid dirty reads and retrieving inconsistent data [3]. A weak matching algorithm will cause scalability problems due to the in() operation (take operation), which blocks if there is no matching tuple [3]. Therefore, a better algorithm means better matching of tuples and a higher level of scalability.

4. RESOURCE DESCRIPTION FRAMEWORK

RDF is an infrastructure that enables and promotes the encoding, exchange and reuse of structured metadata [4]. It is a product of W3C for representing and storing any kind of data as a Web resource on the Web [42]. RDF makes heavily use of XML after adding some constraints on that use that provide unambiguous methods for expressing semantics. In contrast with other approaches, RDF allows human-readable and machine-parseable vocabularies, and is designed to support the reuse of metadata semantics and these vocabularies among different information communities [4].

In RDF, the represented and described Web resources have properties and can be identified by Uniform Resource Identifiers (URIs). These URI references are formed by a URI namespace and a local name [43]. The properties associated with resources are identified by property type, and property types have corresponding values. The values can be atomic (literal) in nature or other Web resources identified by URIs [4].

4.1 RDF Model

The data model of RDF is used to describe resources. This model is shown in Figure 4 [44].

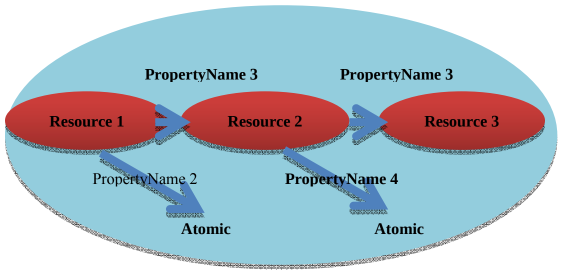

Figure 4: Overview of the RDF model.

RDF provides remote access to unstructured and structured data stored in repositories and has strongly supporting relational databases. It masks the heterogeneity between data sources by exposing them as RDF resources on the Web. This is achieved through two approaches: direct mapping and indirect mapping [45]. The direct mapping approach translates the schema of structured databases such as relational databases to RDF [45], while the indirect approach uses APIs and the application logic provided by Content Management Systems (CMS) as a source of information to be exposed in RDF [45]. In this case, developers of applications are far away from any updates that are performed on the storage layer, because they are using the APIs. From an implementation point of view, there are two approaches to translating relational databases to RDF resources [43]:

1- The static approach, which applies the concept of the extraction, transformation and load (ETL) approach to create the RDF repository from relational databases using mapping rules.
2- The dynamic approach is a query-driven dynamic implementation, which implements mapping dynamically in response to a query.

Finally, RDF as a standard is well suited to representing structured, semi-structured or un-structured data. The representations are suitable for Web semantics and support smart queries about data.

4.2 Implementation of RDF

RDF gains wide acceptance through the many frameworks, systems, APIs and tools that have been developed to create, edit and update RDF models, and to share and query data from those models. These prototypes allow developers to deal with RDF easily and use interfaces provided during development of the application. In this paper, we are concerned only with applications of RDF and no other tools, APIs, and so on. The reader may refer to the W3C RDF homepage to read about these tools. We classify these applications into proprietary applications and open-source applications.

4.2.1 Proprietary RDF Applications

In this sub-section, we present the commercial applications that implement the RDF approach.

Frameworks/Systems

Piazza: it is a peer data management system that allows users to share heterogeneous data in a P2P environment [46][47]. It solves the problems of sharing data in a distributed and scalable way[48]. Piazza is a product of research work in cooperation between University of Washington and University of Pennsylvania

PeerDB: it is a P2P-based system that is based on the BestPeer platform for sharing distributed data stored in relational databases without any schema knowledge [46] [49]. PeerDB is a product of research work at the National University of Singapore and University of Fudan.

GridVine:it is a system that uses P-Grid, a P2P overlay network, to realize semantic overlays [46] [50][51][52]. It addressed issues of semantic interoperability and scalability. GridVine is a product of research in cooperation between Swiss Federal Institute of Technology (EPFL) and Linköping University.

pSearch: it is a system that implements the Distributed Hash Table (DHT) algorithm to solve the problem of data semantic diversity between peers [51]. In pSearch, machines are organized into a semantic overlay that offers an information retrieval service [53]. pSearch is a product of research work at the University of Rochester and HP Laboratories.

GrouPeer: it is a system that focuses on the problem in a random flat unstructured peer-database system [54]. It allows peers in the system to benefit from information or to learn about other peers with similar interests [51]. GrouPeer is a product of research work at the National Technical University of Athens, Greece.

PIER: it is a database-style query engine built on top of DHT, which is intended for querying the Internet [55]. It allows distributed sharing and querying fingerprint information [46]. PIER is a product of research work at the EECS Computer Science Division, UC Berkeley and the International Computer Science Institute

Generic Interoperability Framework: it uses the RDF interface as a generic representation for protocols, languages, data and interfaces. It does not provide creation and manipulation, but provides access to RDF models through a query interface (SQL) [44]. This framework is a product of research work.

Hyperion System: it is a system that is built on top of JXTA to support data sharing for a network of independent Peer Relational Database Management Systems (PDBMSs) [56]. The Hyperion system is a product of research work at University of Toronto, University of Ottawa, University of Edinburgh and University of Trento.

AllegroGraph: it is a system for loading, storing and querying RDF data [57]. It a persistent graph database that supports SPARQL, RDFS++ and Prolog reasoning from several client applications. It is available in a paid edition and a free edition.

Oracle Spatial 11g: it is an RDF management platform based on a graph data model. RDF data (triples) are persisted, indexed and queried, similar to other object-relational data types[57].

4.2.2 Open-source RDF Applications

This sub-section summarizes non-commercial (open-source) applications that implement the RDF approach.

Frameworks/Systems

Bibster: it is an open source P2P-based system built on top of the JXTA framework for exchanging bibliographic data among researchers [58]. It exploits ontologies in order to share these data [58].

Redland: it is a flexible and efficient RDF system that provides object-oriented interfaces for storing and retrieving RDF data [59]. These interfaces are written in C, Perl, Java, Tcl, Python and other programming languages [59]

D2RQ Platform: it an open source platform treating non-RDF databases as virtual RDF graphs. It consists of the D2RQ mapping language, a D2RQ engine and a D2R server [60].

Sesame: it is an open source framework for parsing, storing, interfacing, and querying RDF schema and RDF data [61]. It provides an easy-to-use API for connecting to RDF storage solutions [61].

Jena: it is an open-source Java framework developed by HP for RDF models. It supports statements and resource-centric RDF views [44] and provides an API for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine.

Projects/Middleware's

Edutella: it is an open-source project that is intended for semantic Web design and implements a schema based P2P system [46]. It allows peers to exchange RDF metadata and query RDF repositories across the network [46][62].

4.3 Limitations and problems of RDF

RDF as a data sharing approach requires extensive representation for describing data sources in RDF data and schema models. In addition, reconstructing the RDF graph is not a trivial task. Additionally, RDF is an expensive language because it may impose further semantic conditions in addition to those described in the standard RDF. These conditions are imposed on the meanings of terms in practical vocabularies of RDF.

Another problem with RDF is the implementation of the query language to return results from RDF syntax. The developers decide whether the implementation is statement-centric (asking for matching statements) or node-centric (parsing graph representation of the model relatively to a node) [59]. This choice is critical because there are many things that must be considered during implementation, such as the type of parser that will be used to parse RDF syntax, the matching algorithm, and so on.

Another problem with RDF is the amount of memory required for representing metadata from data sources in the RDF model and parsing the model in terms of a graph that can be queried [59]. Sharing large amounts of data requires more memory space, which directly affects the performance of the system.

There are further problems regarding RDF systems. These systems face two main problems, scalability and semantic interoperability. Some of the systems focus on solving the scalability problem but not semantic interoperability (exchanging and querying semantic data), and others focus instead on the interoperability [46].

5. DATA SERVICE APPROACH

Service-Oriented Architecture (SOA) efforts of any enterprise mainly focus on integrating disparate systems. Behind most of these system silos are heterogeneous data sources. Sharing and exchanging data stored in these sources in a distributed manner is beneficial for all users and companies. DSA embodies SOA principles to expose data stored in heterogeneous data sources. It masks the heterogeneity between different kinds of data sources and makes them available as services by providing a Data Service Layer (DSL). This layer is responsible for accessing structured, semi-structured and un-structured data sources, and exposing them as Web services or as a set of representational state transfer (REST) style Web services. The main advantage of this approach is that it reduces the complexity of developing new applications that integrate data from several data sources [5] [63][64].

DSL is much more than just a data access tool. It provides the missing layer between the persistent layer and SOA. Furthermore, DSL as a single access point to all enterprise data sources exposes heterogeneous data sources as data services. These data services are loosely-coupled, reusable, scalable and define virtual access points to enterprise data sources [5] [63][64]. A DSL should usually provide at least the following services:

- Access to data stored in heterogeneous data sources.
- Exposure of data sources as data services for accessing data. Read operations will be generated for providing the ability to access data. Furthermore, it may provide the ability to manipulate data by providing Create, Read, Update and Delete (CRUD) and customized operations.
- Data services will map data source calls to one back-end data source.

However, to understand the described guidelines well, we describe them as follows [65]:

- Data through services:
  Providing data through services leads to easy access of data stored in heterogeneous data sources, and supports data integration, which becomes very easy.
- Data as a service:
  It facilitates the process of collecting and composing data through service composition or a data integration layer above service abstraction. Also, it provides data operations (for example, CRUD operations) that facilitate software development.

Finally, DSA helps and users benefit from data stored in data repositories by sharing and integrating them in a distributed manner.

5.1 Architecture of Data Service Approach

The general architecture of DSA provides an abstract interface called DSL that allows access to heterogeneous data sources and exposes them as services. These services are available for remote invocation. A general overview of the architecture of DSA is shown in Figure 5 [5].

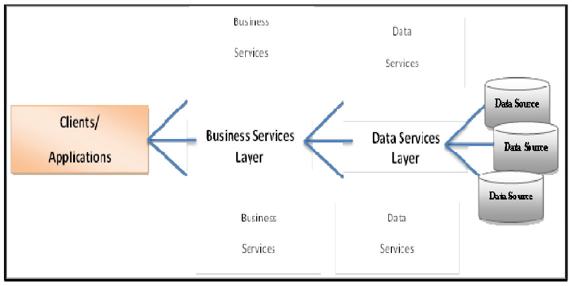

Figure 5: Architecture of DSA

5.2 Implementation of DSA

In our opinion, insufficient effort has been put into the implementation of DSA and have produced only a limited number of applications. In this paper, we classify these applications in proprietary and open-source applications.

5.2.1 Proprietary DSA applications

In this sub-section, we summarizes the commercial applications that implement data service approach.

Frameworks/Systems

ADO.NET Data Services Framework: the main motivation of this framework is to demand separation between the presentation layer and the data layer to build more interactive and responsive Web applications, and for building RESTiful systems [66][67].

BEA AquaLogic Data Services Platform (ALDSP): A platform allows developers to design, develop, deploy and maintain DSL in a SOA world. It models the collection of data sources as a set of data services. These services are available to any consumer that needs them [68]

Projects/Middleware's

OracleAS and DB: this project is intended for the users of the Oracle application server [69]. It may not be fully projected in a P2P environment, but it allows Oracle users to expose databases as Web services and allows them to consume those services.

5.2.2 Open-source DSA applications

This sub-section present the open-source applications that implement data service approach

Frameworks/Systems

WSO2: itis open-source software builton top of WSO2 Carbon, a lightweight high-performance platform for creating data services, and uses Axis2 as the underlying SOAP processing engine [70].

Projects/Middleware's

Axis2 POJO: it is based on creating POJO classes to expose databases as Web services [71]. The process of creating and exposing databases as services is written manually by developers.

5.3 Limitations and problems of DSA

In DSA, data sources are exposed as data services and become available for remote invocation. This service is decoupled from the data source but exposes the functionality of the data source. Therefore, if the underlying structure of the exposed data source changes or the data source is deleted, DSA should provide a mechanism that reacts to this modification and takes appropriate action. Accordingly, DSA should support automation change discovery for any change happening in data sources [5].

Another problem with DSA is supporting a public or private UDDI registry. A public registry requires security mechanisms such as access control, encryption messages, policies, and so on, while with a private registry scalability is the main problem. In addition, a UDDI registry must be available all the time and must maintain a good security level.

A further problem in DSA is the performance issue for both the data access layer and remote invocation. In DSA, the data access layer provides access to and manipulation of data stored in heterogeneous data sources. Therefore, accessing data stored in various data sources should be efficient and flexible because this is the core reason for building data services. As for the issue of remote invocation performance, communication between parties are using SOAP messages and therefore many service invocations means many messages exchanged, so implementing a parallel mechanism has a greater impact on performance. Additionally, the remote invocations may not all be successfully completed because of problems that may occur during execution. Therefore, implementing exception-handling mechanisms is important for catching errors and performing appropriate actions [5][70] [72].

6. DISCUSSION

Based on our review of data sharing approaches, we conclude that TPM and Tuplespace focus on sharing and integrating data stored in specific type of data source, namely databases, while RDF and DSA focus on sharing and integrating data stored in heterogeneous data sources. In addition, RDF and DSA require describing a whole data source as a service or resource, while in TPM there is no need for this as the TPM interface deals with it directly, and in Tuplespace the data provider posts the data as tuples in shared Tuplespace.

Moreover, from our review of applications developed based on these approaches, we conclude that some approaches had received more effort than others. In addition, most of the developed applications that implement concepts found in these approaches are based on P2P architecture and not client server architecture, because P2P architecture provides more advantages over centralized architecture, avoiding both computational performance and information update bottlenecks, and providing robustness, reliability, fault-tolerance and scalability. By focusing on types of data sources, sharing structure data source such as database is acquired most attention than semi-structured and unstructured data sources. This has become clear through two reasons, the first reason is the most of users and companies around the world usually store their data in databases rather than other type of data sources. The second reason is most of applications that have been developed support sharing databases firstly and may support sharing other types of data sources secondly. The main problem facing data sharing applications in these approaches are scalability, interoperability and performance, but the problems are not limited to these.

Finally, TPM as a data sharing approach is not as popular as the other approaches. As for DSA, more research and development efforts are needed during the next decade to develop more applications that implement the concepts found in DSA. Developing a new solution that supports sharing distributed data sources, requires a further study of these approaches carefully and examine their limitations and problems. This depth analysis will help us to decide which approach is suitable for a certain situation to develop an appropriate solution. The selected approach should be capable of integrating data to retrieve data from multiple data sources in a simple way.

7. OPEN RESEARCH ISSUES IN DATASHARING SYSTEMS

Datasharing systems are capable of sharing huge amounts of data among an increasing number of unreliable peers. A survey of these systems shows that the research community is currently investigating several open problems including security issues, search mechanismsexpressiveness, efficiency and robustness.These open problems are common to all datasharing approaches. We provide a brief summary of these important and open research issues. More details are provided in[73].

Research efforts related to search issues aims at increasing the responsiveness of data sharing distributed or P2P systems by exploring new mechanisms allowing users to look up for desired data in an efficient way. The topology of the network adopted by the distributed or P2P system, the strategy of placing data on the peers and the message routing protocol are key factors for the success of the search mechanism. New topologies such Gaussian andEisenstein-Jacobi[74] networks, thathave emerged in the last few years,provide tremendous opportunities for the development of novel and innovative new data placement strategies as well asefficient routing protocols.

Ongoing research is exploring how to make search mechanisms more efficient and robust. Further research is needed to extend search techniques such as key lookup, keyword, ranked keyword, aggregates and SQL to support much larger systems and to incorporate new features with reasonable performance.

The requirements of data sharing systemsin terms of security are organized intothe following areas: availability, authenticity, anonymity, and access control[73].Developing techniques that prevent, detect, manage, and are able to recover from attacks for each of these areas is expected to be a research challenge for some time.

8. CONCLUSION

In this paper, we overview the most known data sharing approaches in detail and review the developed applications that implement these approaches. These applications fail in some aspects and are successful in others. Further research is needed to provide a comprehensive solution for sharing data sources that masks the heterogeneity between user platforms and between data sources.Developing such solutionrequires to extend the comparative study that we present in this paper to define a set of comparison criteria. Such criteria will allow to select the appropriate approach that should be adopted for a target solution. Besides, the expected solution should consider the open research issues highlighted in this paper in order to satisfy the security and search requirements of datasharing systems.

ACKNOWLEDGMENT

This work was supported by the Research Center, King Saud University, Riyadh, Saudi Arabia.

REFRENCES:

[1] V.V. Eynden, L. Corti, M. Woollard, L. Bishop and L. Horton, "MANAGING AND SHARING DATA", UK Data Archive, University of Essex, UK, May 2011.

[2] G. Alonso, "Transaction Processing Monitors (TP-monitors)", Computer Science Department, Swiss Federal Institute of Technology (ETHZ), last visited, May 2011, cited athttp://masteritgov.dia.uniroma3.it/didattica/MW/Middleware-04-TP-Monitors.ppt.

[3] I.K. Herrmann, "Asynchronous Middleware Tuple Spaces", 2011.

[4] M.H. Needleman, "RDF: The Resource Description Framework", Vol. 27, 2001, pp. 58-61.

[5] J. Bloomberg and J. Goodson. "Best Practices for SOA: Building a Data Services Layer", SOA World Magazine, Vol. 8, issue 5, June 2008.

[6] A. Silberschatz , H. F. Korth and S. Sudarshan, "Database System Concepts", sixth edition, McGraw Hill, 2010.

[7] D. Sadoski, D., "Transaction Processing Monitor Technology", last visited, July 2011, cited at: http://www.oernii.sk/sxool/ing_semester1/_kubiki/DP/Podklady/Architektura/Transaction%20Processing%20Monitor%20Technology_files/tpmt_body.htm

[8] J. Gebhardt, "Integration of Heterogeneous Data Sources with Limited Capabilities in the Object-Oriented Mediator Engine AMOSII", Laboratory for Engineering Databases, Link¨oping University, Sweden, 1999.

[9] M. Rodr´ıguez-Martinez and N. Roussopoulos, "MOCHA: A Self-Extensible Database Middleware System for Distributed Data Sources", Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, May 2000.

[10] A. Tomasic, L. Raschid and P. Valduriez, "Scaling Heterogeneous Databases and the Design of Disco", Institut National De Recherche En Informatique Et En Automatique, INRIA, 1995.

[11] M.J. Carey, L.M. Haas, P.M. Schwarz, M. Arya, W.E. Cody, R. Fagin, M. Flickner, A.W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J.H. Williams and E.L. Wimmers, "Towards Heterogeneous Multimedia Information Systems: The Garlic Approach", IBM Almaden Research Center, San Jose, 1995.

[12] L. Penserini, M. Panti and L. Spalazzi, "Agent-Based Transactions into Decentralised P2P", Preliminary Report, Computer Science Institute, University of Ancona, Italy, ACM Publisher, 2002.

[13] M. Masud and I. Kiringa, "Transaction processing in a peer to peer database network", Data & Knowledge Engineering, Vol. 70, 2010, pp. 307-334.

[14] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman and J. Widom, "The TSIMMIS Project: Integration of Heterogeneous Information Sources", Proceedings of IPSJ Conference, Tokyo, Japan, October 1994.

[15] G.V. Gkoutos, E.C.J. Green, S. Greenaway, A. Blake, A. Mallon and J.M. Hancock, "CRAVE: a database, middleware and visualization system for phenotype ontologies". BIOINFORMATICS, Oxford journals, Vol. 21, 2005, pp. 1257–1262.

[16] subbu.org, "Nuts and Bolts of Transaction Processing", last visited, July 2011, cited at: http://www.subbu.org/articles/nuts-and-bolts-of-transaction-processing

[17] S. Capizzi, "A Tuple Space Implementation for Large-Scale Infrastructures", PhD thesis, Department of Computer Science, University of Bologna, Italy, 2008.

[18] F. Fummi, G. PErbellini, R. Pietrangeli and D. Quaglia, "A Middleware-centric Design Flow for Networked Embedded Systems", In Design, Automation & Test in Europe Conference & Exhibition, 2007.

[19] K.A. Hawick, H.A. James, and L.H. Pritchard, “Tuple-space based middleware for distributed computing”, Technical Report DHPC-128, 2002.

[20] T.J. Lehman, S.W. McLaughry, and P. Wycko, "TSpaces: The Next Wave", Proceedings of Hawaii International Conference on System Sciences, 1999.

[21] TibcoActiveSpaces, “TIBCO ActiveSpaces Enterprise Edition”, last visited, July 2011, cited at: http://www.tibco.com/products/soa/in-memory-computing/activespaces-enterprise-edition/default.jsp.

[22] GigaSpaces, “XAP Elastic Application Platform", last visited, July 2011, cited at: http://www.gigaspaces.com/xap/overview

[23] N. Busi, C. Manfredini, A. Montresor and G. Zavattaro, “PeerSpaces: Data-driven Coordination in Peer-to-Peer Networks”, 2003.

[24] R. Goot, “High Performance Linda using a Class Library”, PhD thesis, Erasmus University Rotterdam, 2001.

[25] A.K. Atkinson, “Tupleware: A Distributed Tuple Space for the Development and Execution of Array-based Applications in a Cluster Computing Environment”, PhD. thesis, University of Tasmania, 2009.

[26] B. Nielsen and T. Słrensen, “Distributed Programming with Multiple Tuple Space Linda”, Master Thesis, Aalborg University, 1994.

[27] A. Rowstron, “WCL: A Coordination Language to Geographically Distributed Agents”, World Wide Web Journal, Vol. 1, Issue 3, 1998, pp. 167-179.

[28] Z. Li and M. Parashar. “Comet: A scalable coordination space for decentralized distributed environments”, Proceedings of the 2nd International Workshop on Hot Topics in Peer-to-Peer Systems, 2005, pp. 104 – 112.

[29] M. Mamei, F. Zambonelli and L. Leonardi, “Tuples On The Air: A Middleware for Context-Aware Computing in Dynamic Networks”, 2nd International Workshop on Mobile Computing Middleware at the 23rd International Conference on Distributed Computing Systems (ICDCS), Los Alamitos, CA, USA, IEEE Computer Society, 2003, pp: 342-347.

[30] Q.H. Mamoud, “Getting Started With JavaSpaces Technology: Beyond Conventional Distributed Programming Paradigms”, Oracle technical article, 2005.

[31] G. P. Picco, D. Balzarotti, and P. Costa, "LIGHTS: A lightweight, customizable tuple space supporting contextaware applications", Proceedings of the 20th ACM Symposium on Applied Computing (SAC), Santa Fe, New Mexico, USA, March 2005.

[32] Blitz, “Blitz project”, last visited, July 2011, cited at: http://www.dancres.org/blitz/.

[33] Rinda, “Rinda (Ruby programming language)”, last visited, July 2011, cited at: http://en.wikipedia.org/wiki/Rinda_(Ruby_programming_language).

[34] LinuxTuples, “LinuxTuples”, last visited, July 2011, cited at: http://linuxtuples.sourceforge.net/.

[35] pylinda, “Python”, last visited, July 2011, cited at: http://code.google.com/p/pylinda/

[36] SemiSpace, “SemiSpace”, last visited, July 2011, cited at: http://www.semispace.org/semispace/.

[37] G.P. Picco, A.L. Murphy, and G.-C. Roman, "LIME: Linda meets mobility", Proceedings of the 21st International Conference on Software Engineering (ICSE), May1999. pp. 368–377.

[38] SQLSpaces, “Welcome to SQLSpaces”, last visited, July 2011, last updated, May 2011, cited at: http://sqlspaces.collide.info/.

[39] G. Cugola and G.P. Picco, “PeerWare: Core Middleware Support for Peer-to-Peer and Mobile Systems”, Technical report, Dipartimento di Electronica e Informazione, Politecnico di Milano, 2001.

[40] FlyObjectSpace, “Cloud and Cluster Scaling Software”, last visited, July 2011, cited at: http://www.flyobjectspace.com/

[41] A. Omicini and E. Denti, “From tuple spaces to tuple centres”, Science of Computer Programming, Vol. 41, 2001, pp. 277-294.

[42] N. Alexander and S. Ravada, “RDF Object Type and Reification in the Database”, Proceedings of the 22nd International

Conference on Data Engineering (ICDE'06), IEEE Computer Society, 2006.

[43] S. Zhou, "Exposing Relational Database as RDF", 2nd International Conference on Industrial and Information Systems, 2010, pp. 1-4.

[44] K. Candan, H. Liu and R. Suvarna, "Resource Description Framework: Metadata and Its Applications," ACM SIGKDD Explorations, Vol. 3, 2001, pp. 6-19.

[45] U. Bojārs and J.G. Breslin, "From Online Community Data to RDF", 2009.

[46] V.S. Agneeswaran, "A Survey of Semantic Based Peer-to-Peer Systems", Technical report, LSIR, communicated to International Journal of Computer Science and Software Technolog, 2007.

[47] I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, and P. Mork. "The Piazza Peer Data Management Project", ACM SIGMOD Record, Vol. 32, 2003, pp. 47-52.

[48] A. Roshelova, "A Peer-to-Peer Database Management System", PhD. thesis proposal, University of Trento, 2004.

[49] W. S. Ng, B. C. Ooi, K.-L. Tan and A. Zhou, "Peerdb: A p2p-based system for distributed data sharing", In ICDE, Bangalore, India, 2003, pp. 633–644.

[50] K. Aberer, P. Cudr´e-Mauroux, M. Hauswirth and T.V. Pelt, "GridVine: Building Internet-Scale Semantic Overlay Networks", In Sheila A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, 3th International Semantic Web Conference, Vol. 3298 of Lecture Notes in Computer Science, Springer, Verlag, 2004, pp. 107–121.

[51] V. Kantere, D. Tsoumakos, T. Sellis and N. Roussopoulos, "GrouPeer: dynamic clustering of P2P databases", Technical Report TR-2006-4, National Technical University of Athens, Inf. Syst. J., 2006.

[52] V. Kantere, D. Tsoumakos, T. Sellis and N. Roussopoulos, "GrouPeer: dynamic clustering of P2P databases", Information System, Vol. 34, Issue 1, 2009, pp. 62–86.

[53] C. Tang, Z. Xu and S. Dwarkadas, "Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks", SIGCOMM'03, Karlsruhe, Germany, 2003.

[54] V. Kantere, D. Tsoumakos and T. Sellis, "A framework for semantic grouping in P2P databases", Information Systems, Vol. 33, 2008, pp. 611-636.

[55] R. Huebsch, J.M. Hellerstein, N. Lanham, B.T. Loo, S. Shenker and I. Stoica, "Querying the Internet with PIER", Proceedings of 29th International Conference on Very Large Data Bases, Berlin, Germany, VLDB, September 2003, pp: 321–332.

[56] P. Rodríguez-Gianolli, M. Garzetti, L. Jiang, A. Kementsietsidis, I. Kiringa, M. Masud, R.J. Miller and J. Mylopoulos, "Data Sharing in the Hyperion Peer Database System", Proceedings of the 31st VLDB Conference, Trondheim, Norway, VLDB, 2005, pp. 1291-1294.

[57] W3CSemanticWeb, "SemanticWebTools", last visited, July 2011, cited at: http://www.w3.org/wiki/SemanticWebTools

[58] P. Haase, B. Schnizler, J. Broekstra, M. Ehrig, F.V. Harmelen, M. Menken, P. Mika, M. Plechawski, P. Pyszlak, R. Siebes, S. Staab, and C. Tempich, "Bibster - a semantics-based bibliographic peer-to-peer system", Proceedings of 3rd International Semantic Web Conference (ISWC), Springer, 2004, pp. 122-136.

[59] D. Beckett, "The design and implementation of the Redland RDF application framework", Computer Networks, Vol. 39, 2002, pp. 577-588.

[60] C. Bizer, "The D2RQ Plattform - Treating Non-RDF Databases as Virtual RDF Graphs", last visited, July 2011, last update: November 2010, cited at: http://www4.wiwiss.fu-berlin.de/bizer/d2rq/

[61] Sesame, "home of Sesame", last visited, July 2011, cited at: http://www.openrdf.org/about.jsp

[62] R. Hayek, G. Raschia, P. Valduriez and N. Mouaddib, "Data Sharing in P2P Systems", Handbook of Peer-to-Peer Networking, Springer, 2010, pp. 531-569.

[63] K. Goundar, S. Singh and X.F. Ye, "An Investigation into Concurrency Control Mechanisms in Data Service Layers", 14th

Asia-Pacific Software Engineering Conference, IEEE Computer Society, 2007.

[64] M. Nikoo and Dunstan Thomas Consulting, “The Data Layer – Build or Buy?”, 2003.

[65] J. Bloomberg and R. Schmelzer, “The Data Services Layer: Building a Solid Foundation for SOA”, 2009.

[66] G. Fink, “ADO.NET Data Services Introduction”, last visited, December 2010, cited at: http://blogs.microsoft.co.il/blogs/gilf/archive/2008/08/10/ado-net-data-services-introduction.aspx#comments.

[67] MicrosoftADO.NET, “ADO.NET Data Services Overview”, last visited, December, 2010, cited at: http://msdn.microsoft.com/en-us/library/cc668794(v=VS.90).aspx

[68] BEA Systems, “BEA Aqualogic Data Services Platform™”, Technical Whitepaper, 2006.

[69] OracleDB, “Developing Database Web Services”, last visited, December, 2010, cited at: http://otndnld.oracle.co.jp/document/products/as10g/101300/B25221_03/web.1013/b14434/devdbase.htm.

[70] S. Rubasinghe and A. Anandagoda, “WSO2 Data Services”, 2008.

[71] D. Jayasinghe, “Exposing a Database as a Web Service”, last modified, December, 2009, cited at: http://www.developer.com/db/article.php/3735771/Exposing-a-Database-as-a-Web-Service.htm

[72] OracleODISuite, “Enterprise Data Services in SOA using ODI Suite”, Oracle white paper, 2009, pp. 1-38.

[73] N. Daswani, H.Garcia-Molina and B. Yang, “Open Problems in DataSharing Peer-to-Peer Systems”, Proceedings of the 9th International Conference on Database Theory. Siena, Italy, 2003.

[74] M. Flahive and B. Bose., “The Topology of Gaussian and Eisenstein-Jacobi Interconnection Networks”, IEEE Transactions on Parallel and Distributed Systems, Vol. 21, Issue 8, pp. 1132-1142, August 2010.