TripCom

Triple Space Communication

FP6 – 027324

Deliverable

D1.3

High-Performance Storage Implementation

Brahmananda Sapkota
Vassil Momtchev
Omair Shafiq

March 31, 2008

Executive Summary

Deliverable D1.3 - *High-Performance Storage Implementation* describes Triple Space storage as implemented. It presents a high-performance implementation for the main storage functionality, allowing for integration of heterogeneous data sources through various adapters. This document starts with describing a single machine implementation and subsequently the federation of storage over a network. The basic infrastructure based on RDF stores as defined in T1.2 [25] is described first. A partial reference implementation of the API was built using that infrastructure.

An efficient and scalable solution to accessing RDF triples was determined by evaluating the initial implementation. A highly optimized and efficient structure for searching RDF triples is provided as part of an efficient and scalable method of accessing such triples. In order to access, integrate, and distribute data over heterogeneous data sources various adapters are implemented. In addition to the implementation of the adapters, this deliverable also describes the technology used for their implementation as well as an expanded discussion of related work in the areas of each particular adapter.

Document Information

IST Project Number	FP6 – 027324	Acronym	TripCom
Full Title	Triple Space Communication		
Project URL	http://www.tripcom.org/		
Document URL			
EU Project Officer	Werner Janusch		

Deliverable	Number	1.3	Title	High-Performance Storage Implementation
Work Package	Number	1	Title	Storage

Date of Delivery	Contractual	M12	Actual	31-March-07
Status	version x		final ☒	
Nature	prototype ☒ report ☐ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Brahmananda Sapkota (NUIG), Vassil Momtchev (ONTO), Omair Shafiq (LFUI)			
Resp. Author	Brahmananda Sapkota (NUIG)		E-mail	brahmananda.sapkota@deri.org
	Partner	NUIG	Phone	+353 (091) 49-5214

Abstract (for dissemination)	This document describes Triple Space storage. It presents a single machine implementation and potential solutions for distributing the storage over a network. It describes heterogeneous data adapters and the technologies used in implementing them for the purpose of allowing to access, integrate, and federate data over heterogeneous data sources.
Keywords	Triple Space, Semantics, Storage, RDF

Version Log			
Issue Date	Rev No.	Author	Change
27/07/2007	1	B. Sapkota	D1.3 Final version of deliverable structure created
27/07/2007	2	Vassil Momtchev	Virtual super super section outline added
30/11/2007	3	Omair Shafiq	Data Federation chapter updated
21/12/2007	4	Omair Shafiq	More updates in introduction, survey of related work and requirements analysis of Data Federation chapter
10/01/2008	5	B. Sapkota	Restructured the deliverable
13/01/2008	6	B. Sapkota	Rewrote summary and abstract to reflect new deliverable structure
15/01/2008	7	B. Sapkota	Updated introduction to reflect new deliverable structure
08/02/2008	8	Omair Shafiq	Adapted content according to new Table of Contents
09/02/2008	9	Omair Shafiq	Survey of related work for Federation adapter completed
11/02/2008	10	Omair Shafiq	Requirements analysis for Federation adapter finalized
15/02/2008	11	Omair Shafiq	Design of Federation adapter added, and pre-finalized version of chapter 5, federation adapter, is ready
20/02/2008	12	B. Sapkota	Completed Chapter 4
10/03/2008	13	Vassil Momtchev	TRREE and all implementation sections added
10/03/2008	14	O. Shafiq	Addressing Vassil's comments
14/03/2009	15	B. Sapkota	Implemented Vassil's comments, updated executive summary, introduction and conclusions

Project Consortium Information

Acronym	Partner	Contact
Semantic Technology Institute Innsbruck http://www.sti-innsbruck.at	STI	Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria E-mail: dieter.fensel@sti-innsbruck.at
National University of Ireland, Galway http://www.deri.ie	NUIG	Dr. Laurentiu Vasiliu Digital Enterprise Research Institute (DERI) Galway, Ireland Email: laurentiu.vasiliu@deri.org
University of Stuttgart http://www.iaas.uni-stuttgart.de/	USTUTT	Prof.Dr. Frank Leymann Inst. für Architektur von Anwendungssystemen (IAAS) Stuttgart, Germany E-mail: frank.leymann@informatik.uni-stuttgart.de
Vienna university of Technology http://www.complang.tuwien.ac.at/	TUW	Prof.Dr. eva Kühn Institut für Computersprachen Vienna, Austria E-mail: eva@complang.tuwien.ac.at
Free University Berlin http://www.ag-nbi.de/	FUB	Prof. Dr.-Ing. Robert Tolksdorf AG Netzbasierte Informationssysteme Berlin, Germany E-mail : tolk@inf.fu-berlin.de
Ontotext Lab, Sirma Group Corp. http://www.ontotext.com/	ONTO	Atanas Kiryakov, Vassil Momtchev, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: vassil.momtchev@ontotext.com
Profium OY http://www.profium.com/	Profium	Dr. Janne Saarela Profium OY Espoo, Finland E-mail: janne.saarela@profium.com
CEFRIEL SCRL. http://www.cefriel.it/	CEFRIEL	Davide Cerri CEFRIEL SCRL. Milano, Italy E-mail: cerri@cefriel.it
Telefonica I+D http://www.tid.es/	TID	Noelia Pérez Crespo Telefonica I+D Madrid, España E-mail: npc@tid.es

Table of Contents

List of Abbreviations

API	Application Programming Interface
DBMS	Database Management System
D2R	Relational Database to RDF
DAI	Data Access and Integration
DHT	Distributed Hash Table
DL	Description Logic
DO	Domain Ontology
DOMS	Domain Ontology Management System
DTP	Data Tools Platform
EMF	Eclipse Modeling Framework
FRP	Federation Routing Pattern
HTTP	Hyper Text Transfer Protocol
IBM	International Business Machine
JDBC	Java DataBase Connectivity
JMS	Java Message Service
MAAN	Multi Attribute Addressable Network
N3	Notation 3
N3QL	N3 Query Language
OGSA	Open Grid Services Architecture
ORDI	Ontology Representation and Data Integration
OWL	Web Ontology Language
OWLIM	OWL In Memory
P2P	Peer-to-Peer
PAGE	Put And Get Environment
PEPSINT	Peer-to-Peer Semantic Integration
POCS	Predicate Object Context Subject
RDBMS	Relational DBMS
RDF	Resource Description Framework

RDFS	RDF Schema
RDQL	RDF Data Query Language
REST	Representational State Transfer
SAIL	Storage and Interface Layer
SPARQL	SPARQL Protocol And RDF Query Language
SPO	Subject Predicate Object
SPOC	Subject Predicate Object Context
SQL	Structured Query Language
TRREE	Triple Reasoning and Rule Entailment Engine
TripCom	Triple Space Communication
TS API	Triple Space API
WSMX	Web Service Execution Environment
XSLT	Extensible Stylesheet Language Transformations
XML	Extensible Mark-up Language
YARDI	YARS ORDI Adapter
YARS	Yet Another RDF Store

1 Introduction

Triple Space Computing has been envisioned as communication and coordination paradigm for Semantic Web-based applications. It is based on the principle of persistent publishing and reading of data. The need for persistency makes storage one of the important requirements in the realization of Triple Space Computing. There is a need for a RDF data storage facility that will provide a base for the overall Triple Space infrastructure because RDF is the data model used in Triple Space. Several RDF stores exist which can be used for the purpose. This section presents an overall description of the implemented Triple Space storage infrastructure.

Triple Space is designed to be an open, distributed and scalable system. As such, the data may reside in different data stores owned by different organizations and dispersed to different geographical locations and connected over Internet. A standalone storage system may not always suffice to fulfill the requirements of triplespace storage. In addition, a Triple Space environment may include data from different sources using different data models such as RDF and relational data models. Therefore, the storage layer in the Triple Space has to provide an unified view of data abstracting from their models and locations.

In this document, the initial prototype implementation of Triple Space storage is presented and possible solutions for supporting scalable and distributed storage is discussed. This document presents possible approaches for integrating legacy repository systems to Triple Space storage. Integration of various data sources are performed through the use of data adapters. The adapters created for this purpose are the TREEAdapter, YARSAdapter, RDBMSAdapter, and FederationAdapter. The data representation model in Triple Space storage system is tripleset as defined in T1.2 [25]. The queries for retrieving these data are represented in the SPARQL [29] query language.

In addition to the implementation of different adapters, this deliverable presents the technology used for their implementations. In each section describing an adapter, licences of that particular adapter is described.

The features supported by Triple Space storage are: 1) federation of data located on multiple distributed sources (The Virtual Super Store); 2) multiplexing of the information to multiple RDF stores; 3) integration of arbitrary structured data sources and representation via ontology; 4) efficient support of metadata via triplesets - which highly extend the possibilities for optimization of the access to federated and other structured data sources. Such metadata is used to maintain information about the source table and primary key of a relational model to represent the triple.

The implementation of Triple Space storage infrastructure is configurable. Reasoning support as well as performance can be configured and tuned as required through the use of the TREEAdapter. This adapter supports reasoning in both highly contextualized and simple RDF models. It performs forward-chaining entailment on top of RDF extended named graphs and employs the selected reasoning strategy.

The RDBMSAdapter is implemented to allow access to the relational databases. The access to the relational databases used by applications are supported at run time. In addition to providing means for accessing relational databases, the RDBMSAdapter adds semantics to the relational data model based on the ORDI data model. The creation or translation of relational models to ORDI data model for adding semantics is supported offline; the support for such transformation at runtime is not supported

because of the incompatibility in these models. It is deemed infeasible to provide full support of all model operations for this reason and only read operations are supported. The RDBMSAdapter provides a partial support of ORDI data models with read only connection to the tripleset model.

The integration of different data sources can be done with the help of adapters to transform between two data models. The YARSAdapter was implemented to integrate different RDF data models. This adapter is applicable in situations where optimised indexing and query response time is of importance.

To be able to scale indexing structures and query processing, the data has to be distributed across a number of storage nodes. The distribution should not require users to access them each node individually. The distribution of data and data sources has to be invisible from the users, thereby providing a single view of the data. In this respect, Triple Space infrastructure may span multiple organisational boundaries, forming a virtual organisation. In order to support such an abstraction, The FederationAdapter was implemented. It integrates different data sources via data federation providing both enterprise level flexibility and scalability. In addition, this adapter considers some of the desired properties of data federation such as simplicity, ability to handle horizontal growth of data, efficiency, completeness and correctness. Completeness and correctness is supported per RDF virtual data store at the local Triple Space kernel and not at the global scale. Triplespace-wide completeness and correctness support is provided through the Distribution Manager [20]. To keep the distribution mechanism in the Triple Space storage system as simple as possible, the availability of data stores are taken into account while storing data. The data retrieval mechanism is based on simple distribution of queries to all available data stores, execution of queries locally, followed by joining the required results.

The rest of this document is structured as follows. The ORDI implementation built on top of Triple Reasoning and Rule Entailment Engine 3.0 (TRREE) is described in Section 2. The integration of relational databases to the ORDI data model is presented in Section 3. The YARSAdapter, which integrates YARS RDF store to Triple Space storage, is described in Section 4. Section 5 presents federated Triple Space storage. The configuration aspects of Triple Space storage and its evaluation is introduced in Section 6. This document is concluded in Section 7 with some recommendations for future directions.

2 Triple Reasoning and Rule Entailment Engine

This section describes the ORDI implementation built on top of the Triple Reasoning and Rule Entailment Engine 3.0 (TRREE) [1]. TRREE is a high-performance storage inference engine that integrates efficient reasoning and performs forward-chaining of entailment rules on top of RDF extended named graphs. The TRREEAdapter component implements the ORDI specification and manages the differences in functionalities between the specification and the rule engine's low level interfaces. The adapter provides a full implementation of the ORDI model and is used as reference implementation in ORDI framework 0.5.

The TRREEAdapter is recommended to be used in scenarios that requires high-scalability, efficiency and light-weight reasoning.

2.1 Technology

TRREE version 3.0 is complete rewrite of the 2.x engine with major changes in the supported data model, operation semantics, persistent strategy and thus performed optimizations. From this point on TRREE we will refer only version 3.x. It performs forward-chaining of entailment rules on top of RDF extended named graphs and employs a reasoning strategy that can be described as total materialization. The implementation of TRREE relies on a compiling the entailment rules into chunks of Java code and merging the resultant code to generate the main entry point for inference. Reasoning and query evaluation are performed in-memory. The full content of the repository is loaded into and maintained in a proprietary representation format in a computer's main memory, which makes possible very efficient retrieval and query answering.

TRREE can be configured with a set of entailment rules that determine the supported semantics. Each rule has a set of premises that conjunctively define its body. The premises are, in their turn, RDF statements, which may contain variables in any of their positions (subject, predicate, or object). The head of the rule comprises one or more consequences, each of which is also an RDF statement. The consequences may contain free variables, i.e. such that are not used within the body of the rule. In the later case the variables are bound to new (unique) anonymous nodes. TRREE supports a rule language that is more expressive than the one used for the definition of the RDFS semantics. This language is almost identical with the R-Entailment defined by Horst; the major difference is that at present TRREE provides no support of the R-Entailment's axiomatic triples and inconsistency rules.

Deliverable D2.3 [24] provides an extensive overview about the different approaches to reasoning in triplespace.

2.2 Implementation

The ORDI data model provides high-level statement repository-like interfaces. The underlying TRREE is optimized for efficiency, thus the internal interfaces provide only very low level functionality. The TRREEAdapter provides full support of the ORDI data model specification and extends the underlying engine in the following directions:

[1] http://www.ontotext.com/trree/

- Support of RDF types: The data model of the TRREE engine only uses integer values. This ensures very good performance in the evaluation of internal operations, but requires additional transformations when the data is represented in the ORDI data model. Custom implementation of a disk-based hash table to map the RDF types to integers is included. The adapter implements an efficient loading mechanism, which minimize the transformation overhead.

- Statement lazy-loading: ORDI operations are designed to provide maximum usability at reasonable efficiency. The resolving of the RDF types and retrieval of the triplesets is done only after a client request.

- Isolation of the client interaction using sessions: There is no support of connection in TRREE, which makes difficult the memory management for the clients of the ORDI data model. The adapter implements closable connections, which deal with all associate memory resources and faciliate the resource allocation.

- Possibility to monitor the newly asserted implicit statements: By default the TRREE engine works in black-box mode, because the increased performance measures. However, in specific scenarios like the Model-View-Controller architecture pattern, this is not feasible since the controller has to observe the changes in the model enforced by the reasoner. The TRREEAdapter wraps the original model and provides sophisticated capabilities to register listeners to receive notifications from the engine.

3 Relational Database Management System

This section discusses the integration of relational databases to the ORDI data model. Still, there are large sets of data which remains locked into systems that lack semantics. Often data migration has not been feasible because of the high integration costs for a complete system redesign. Thus, a new flexible way is required to add semantics to such existing data sets.

The RDBMSAdapter is recommended for configurations that require integration of legacy data sets whose migration to semantic storage systems is not feasible.

3.1 Technology

The RDBMSAdapter realises a practical and efficient way to interact with information stored in relational databases using ORDI data model operations. It does not provide a full implementation of all model operations; only read operations are supported. Bidirectional support (allowing write operations to the database) will not be supported, because the principle differences between the two technologies makes such mapping unfeasible. This makes it difficult to implement relational database consistency constraints in a general knowledge representation language. Motik suggests in [26] an extension of DL semantics to control the degree of incompleteness in ontology based on the database integrity constrains. However, such an approach requires further research and may not be directly applicable to databases that rely on heavy denormalisation. The scope of the RDBMSAdapter is limited two general scenarios:

- Provide online access for a semantic aware application to relational models used by legacy applications. It is difficult to migrate the whole IT infrastructure especially if a complex transactional model is required by the business process. In these scenarios the constant database replication to a semantic form is not feasible mainly because of synchronisations delays and heavy system overhead to read the complete data from the database. The RDBMSAdapter virtualises the ORDI model capable of executing SPARQL queries on top of the data.

- Offline translation of a relational model to ontologies and adding semantics. Still, the main part of knowledge is locked in databases, which lack semantics. The RDBMSAdapter implements a simple algorithm to allow the migration of the data to ontologies.

The transformation process is described with the help of an ontology serialized in N3 format. Figure 3.1 describes the main ontology classes. The descriptor may be further extended to provide transformations for other structured formats like XML. One descriptor may be used to access multiple data sources. Here is a more detailed description of the most important classes in the ontology:

- DataSource - is a concept to specify connection to data source, which is able to process queries specified in a PredicatePattern.

- PredicatePattern - specifies the generation of quad pattern like (?s, ?p, ?o, ?ng). The predicate (?p) and graph name (?ng) variables are bound using static values for the pattern. The subject and object values are bound to the result set

returned from the DataSource. For a SQLPredicatePattern, an SQL query is required to return two columns interpreted as subject and object.

- DataHandler - is java code which can transform subject or object values to the original data source value.

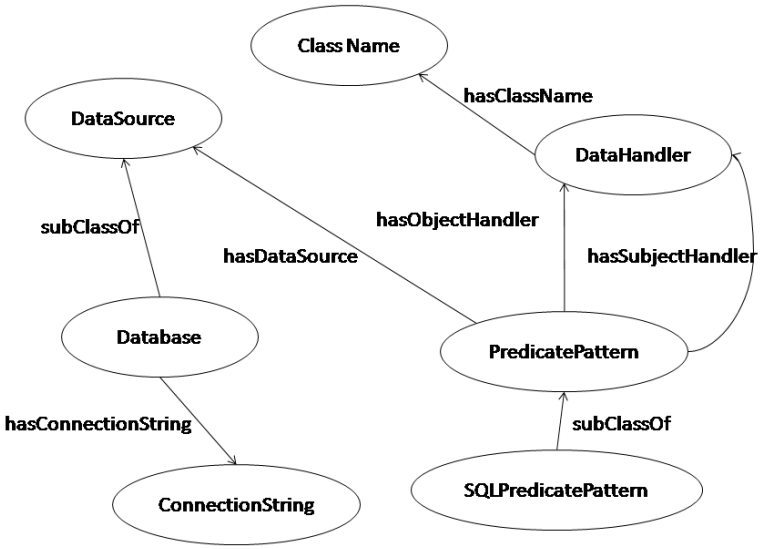

Figure 3.1: Ontology mapper descriptor

3.2 Implementation

The RDBMSAdapter provides partial support of the ORDI data model. All connections to the tripleset model are in read-only mode and write operations generate ORDIReadOnlyExceptions by default. The behaviour may be altered in specific scenarios using the FederationAdapter, where no exception is generated. The adapter integrates an SQL parser distributed with the Eclipse Data Tools Platform (DTP) Project [1] internally to validate the syntactic and semantic (whether the query is a select statement that returns two columns) correctness of every SQL statement associated with an SQLPredicatePattern. There are several performance optimizations to maximize the query evaluation performance:

- The size of the predicate pattern result is maximally cut into the database. The SQL parser generates variations of the WHERE clause depending on the bound variables.
- The three variation of SQL query WHERE clause for a given predicate pattern (subject is bound; object is bound; subject and object are bound) are generated and prepared at initialization time.

[1] http://www.eclipse.org/datatools/

- If multiple predicate patterns have to be executed all the queries are run in parallel.

SPARQL query execution is made using the default query engine of the ORDI framework. All SPARQL queries are decomposed to a list of search operations of the type (?s, ?p, ?o, ?ng). After the successful execution, the query engine combines the result of the search operations. Further optimization of the RDBMSAdapter could be the implementation of a special SPARQL engine customized for relational database operation, which minimizes the volumes of data transfered through the JDBC interfaces. This can be realized by pushing all conjuction operation to more complex SQL queries where possible using the Eclipse EMF SQL Parser.

For security reasons it is strongly encouraged to not store database passwords in the descriptor files, but to specify them in the command line.

4 Yet Another RDF Store

This section describes integration of the YARS data model [1], i.e. N-Triples with context, and the ORDI data model. YARS is an optimised quad store that supports inserts, deletes and keyword query operations. To support these operations, it provides a N3QL entry point over HTTP. The indexing structure in YARS is implemented over BerkeleyDB [2] for quad indexing and the Apache Lucene [3] is used for keyword search functionality. The YARSAdapter component implements and manages the differences in the functionalities of the YARS and ORDI interfaces with respect to insert, delete, and query functionalities. The YARSAdapter is the recommended implementation to be used in scenarios which require quad storage, deletion, and keyword search.

4.1 Technology

YARS (Yet Another RDF Store) is a persistent RDF storage and retrieval system which can be embedded into an application and adapted for special application needs. It is a lightweight open-source Java implementation and provides an efficient RDF indexing structure [16]. It defines and realizes a complete index structure including full-text indices for RDF triples with context in order to support fast storage and retrieval of large amount of RDF data (in the order of billions of triples). Persistency is supported through the use of B+ trees. Internally, each RDF triple is extended with *context* (i.e., the source of the RDF triple) and stored as quadruples (*Subject, Predicate, Object, Context*) encoded in N3 [5] format. Information is indexed for each combination of (*S, P, O, C*) thus enabling fast retrieval of quadruples, given any combination of subject (*S*), predicate (*P*), object (*O*) and context (*C*). Triple pattern and N3QL queries are supported in YARS. The triple pattern queries are defined as ((*S, P, O*),*C*) where every parameter is either a constant or a wildcard. The N3QL is extensible, i.e., the query itself is a graph that adheres to the RDF data model but is extended with variables and grouping of graphs.

The current version of YARS supports tree-shaped Datalog queries with one shared variable. It provides support for import and export of RDF triples to N3 format. It has HTTP access interface built upon REST (Representational State Transfer) [14] and provides a higher abstraction layer than those provided by other RDF stores such as Jena or Redland. HTTP PUT, HTTP GET and HTTP DELETE are the operations currently supported in YARS [1]. These operations can be used for adding data, issuing queries, and deleting data from YARS.

The features supporting the integration of YARS with the Triple Space storage infrastructure are: support for optimized indexing structures for retrieval of RDF statements including context (via quadruples) while minimizing the need for joins, n-gram full text indexing for efficient keyword searches with wildcards, and simple HTTP-based access interface.

As the data model chosen for Triple Space storage is the *tripleset* i.e. the data encoded in the form $< S, P, O, C >$ T_1,T_2,...,T_n, where S, P, O, C and T_i represent the subject, predicate, object, context and tripleset respectively. However unlike in YARS,

[1] http://sw.deri.org/2004/06/yars/
[2] http://www.oracle.com/database/berkeley-db/je/index.html
[3] http://lucene.apache.org/java/docs/

the context in tripleset does not imply the source of the triples. The source of a triple is represented by T_i in tripleset. Therefore by altering and adjusting those parameters (e.g., $< S, P, O, T_1\$\#\$C >$, $< S, P, O, T_2\$\#\$C >$, ..., $< S, P, O, T_n\$\#\$C >$), YARS can be integrated with TripCom storage infrastructure. The character sequence \$#\$C can be used in order to reformulate the queries when needed. The operation-wise mappings between Triple Space storage and YARS is described in Section 4.2.

4.2 Implementation

The Triple Space storage infrastructure is built on the ORDI framework. Through the Triple Storage Adapter (c.f. Section 4.1), ORDI allows integration of various repositories in order to support storage of data represented in different schemata (e.g., RDF, Flat-File, and Relational Schema). As a persistent semantic storage backend, YARS could be integrated with ORDI because of various desirable features such as persistency, efficient indexing, fast storage and data retrieval, and scalability. However, the internal data model used in YARS and Triple Space storage are not fully compatible. In order to adapt the data format from one system to another, a layer needs to be placed between the YARS and ORDI systems [17]. We call this layer an YARDI adapter and provides the following functionality.

- Allow ORDI to use a particular instance of YARS as triplestore;
- Convert the ORDI primitive operations to those of YARS;
- Accept *TripleSet* from ORDI and convert them to *quadruples* that YARS accepts and vice versa;
- Propagate any YARS error messages to ORDI;

These functionalities can be achieved by exploiting the ORDI and YARS models with respect to storage and querying. In either case, the basic adaptation required is internal data model adaptation. This can be performed as specified in Section 4.1. The following table shows how ORDI primitive operations can be mapped to those of YARS.

Primitive Operations	
ORDI	**YARS**
addStatement(s, p, o, c)	createStatement(c), executeInsert(s, p, o)
removeStatement(s, p, o, c)	executeDelete(s, p, o, c)
associateTripleset(s, p, o, s, o)	executeInsert (s, p, o, s$#$o)
deassociateTripleset(s, p, o, s, o)	executeDelete (s, p, o, s$#$o)

Table 4.1: Mapping Between ORDI and YARS Primitives

The table above shows a simple mechanism for mapping between ORDI and YARS primitive operations. However, at implementation time special attention should be taken. When associating triplesets, for example, it is necessary to determine whether the triple that the triplesets should be associated with is already stored in YARS or not. Similar attention needs to be taken care while dissociating the triplesets from a triple. In dissociation, the triple should exist but the associated triplesets should be removed i.e., it involves *executeQuery*, *executeDelete* and *executeInsert* operations over a YARS statement.

5 ORDI Federation

The FederationAdapter is an implementation of the ORDI data model that allows data storage and access in multiple RDF data repositories local to each Triple Space kernel. This chapter provides an introduction to federation mechanisms for data integration of multiple RDF repositories and a detailed description of FederationAdapter architecture.

This implementation must be applied in scenarios that require simultaneous access to multiple autonomous ORDI data models connected to a high-speed local network.

5.1 Data Integration through federation

Data federation is the combining of data from various data sources into one single virtual data source or data service. The data can then be accessed, managed and viewed as if it were part of a single system. Federation techniques are used to meet the demands of Triple Space infrastructure to operate flexibly and efficiently while accessing the data from different RDF stores over the Web. It will help the storage layer in providing the upper layers with standardized access to integrated views of data and will hide the complexity of the underlying data sources.

Data federation is useful especially when the environment is dynamic (e.g., data storage nodes may go online and offline), flexible (different users based on different applications may use Triple Space for communication) and scalable (the number of users and data may increase significantly), and an efficient way is required to retrieve the data.

Several benefits have been recognized by different commercial data integration and federation solutions, i.e.

- **Enterprise level flexibility** Data federation techniques can incorporate data from any source over the Web. It further helps in providing the updated data at the real-time.
- **Enterprise level scalability** enables increased capacity easily and incrementally, scaling from a single project to an enterprise data layer that supports many applications and data sources.

5.2 Technology

Data federation has to be supported so that information from multiple data stores can be accessed in an integrated fashion, while maintaining the abstraction that the data resides in a single virtual store. Triple Space data integration has to be considered at the level of Enterprise Information Integration, where different data sources require different implementations. For instance using the FederationAdapter, specific named graphs may be routed to different implementation of the ORDI data model. Our emphasis is not only on correctness and tractability but also on speed and simplicity.

- **Simplicity** The integration technique has to be simple and easily understandable for the developers as well as the users (i.e. upper layers to use Triple Space storage infrastructure).

- **Able to handle horizontal growth of data** It must be ready to handle higher orders of information especially on a horizontal scale (i.e. an increasingly larger number of machines) and be generic enough for usage by any application. Considering the web-based scalability and openness of Triple Space, a huge amount of data is expected to be stored and retrieved from million of users over the internet.

- **Efficiency** Data storage retrieval from the distributed RDF store should efficient enough in terms of time taken and resource consumption. Time taken for storage and retrieval of data is very curtail for communication using Triple Space over the Web. A time and resource consuming mechanism is required to determine in which repositories data has been stored to match a query.

- **Completeness** Federation of distributed repositories may have the information distributed in chunks across different repositories. It fosters the need for complete retrieval of data from all the repositories.

- **Correctness** Different repositories are supposed to be accessed by multiple users where the information can be changed, updated or deleted without any notification to other data repositories or the data integration component of Triple Space kernel. Therefore, it is important that all the data collection queries should ensure fresh retrieval of data from the repositories, in order for the query result to be valid or correct and not out-dated.

The federation adapter is supposed to ensure completeness and correctness up to only one specific RDF virtual data store at the local Triple Space kernel and not to the global scale, which is to be done by Distribution Manager. Thus, the federated data models have to be connected to high-speed local area network.

5.2.1 Triple Space Virtual Data Super-store

While taking into consideration the requirements analysis performed above, this section presents architecture and functionality of Triple Space virtual data super-store. As name presents, it is supposed to be virtually single but actually will be based on several independent RDF data stores connected to the Triple Space FederationAdapter.

The Triple Space data layer is supposed to provide access to data and does not itself provides any reasoning support. The reasoning support in Triple Space is provided by a dedicated Query Processor. Data layer is only concerned with providing a single view based access and storage of data. The distribution mechanism in the Triple Space data store has been planned to be as simple as possible. The data is stored based on availability of data stores. The data retrieval mechanism is based on simple distribution of queries to selectively available data stores, execution of queries locally, followed by joining the returned results.

5.3 Implementation

The Triple Space virtual super-datastore is managed by a FederationAdapter that keeps the individual data stores bound together and hides the distribution complexity for the ORDI data model. The other components of Triple Space kernel accessing

ORDI data model primitive operations	**FederationAdapter operation implementation**
AddStatement(<s, p, o, ng>)	foreach model do if FRP return true then AddStatement(<s, p, o, ng>)
RemoveStatement(<s, p, o, ng>)	foreach model do RemoveStatement(<s, p, o, ng>)
AssociateTripleset(<s, p, o, ng>, $\{ts_1,\ldots,ts_n\}$)	foreach model do AssignToTripleSet(<s, p, o, ng>, $\{ts_1,\ldots,ts_n\}$)
DeassociateTripleset(<s, p, o, ng>, $\{ts_1,\ldots,ts_n\}$)	foreach model do DeassociateTripleset(<s, p, o, ng>, $\{ts_1,\ldots,ts_n\}$)
Search(<s, p, o, ng, ts>) : Statement[ ]	foreach model do Seach(<s, p, o, ng, ts>) merge with PreviousStatement[ ]

Table 5.1: ORDI primitive operations and FederationAdapter operations

the data layer do not have to take distribution into account. The FederationAdapter provides virtually a single view of data which may be stored at different data stores and distributed globally.

The FederationAdapter is initialised with a list of ORDI data models associated with a federation routing pattern (FRP). The FRPs are used to decide at runtime whether a specific statement must be routed to a specific model. The precise interface specification is available on the ORDI project website [1]. The implemented FederationAdapter operations are shown in Table 5.3 where ng represents a named graph and ts_i represents a tripleset.

Where required, data replication (or mirroring of data) is also supported to solve problems of unavailability. If a data store is likely to be unavailable shortly, or remains unavailable from time t_0, or is required to be taken offline permanently, the data can be replicated from one data store to any other available data store.

The design of the FederationAdapter is carried out to keep the federation mechanism as simple as possible.The Triple Space data layer aims to provide single virtual global view of data that is actually stored in different individual data stores. The process of data distribution is also simple, as the data is distributed across any possible data store. The data retrieval mechanism is simple as it is based on the simple union of all the results obtained. The solution is able to handle growth of incoming data by increasing underlying data stores. The fewest possible interactions of the FederationAdapter and the data stores are planned to enable the solution to scale as the amount of data and number of data stores increase. Efficiency is ensured by multitasking in the monitoring of data stores, distribution of queries, and simultaneous local execution on appropriate data stores, and storing the data on fastest accessible data store. The proactive mechanism of a Data Store Router ensures completeness and correctness in the data retrieved from a single data stores by keeping up-to-date information about data store availability.

[1] http://www.ontotext.com/ordi/

6 Configuration

This section gives a more practical overview of the ORDI framework and what are the minimal configuration sets to be applied. The full software documentation is available on the ORDI website [1].

The ORDI framework supports default runtime parameters in the **ordi.properties** file. The file is optional and must be resolvable by the current context classloader under a resource name **ordi.properties**. Here is a reference example for its content:

```
#Default parameters for ORDI implementation
explicit-collection=(100000,10000,10000,10000)
map-collection=(1000000,100000,100000)
inferred-collection=(100000,10000,10000)
# Uncomment the line below to not load the previously saved data after restart
# The parameter presence is disputable and may result lost of information!
#load-persisted-data=false

########################################
# Default implementation for interfaces
########################################

#Default ORDI tripleset model implementation
com.ontotext.ordi.tripleset.TSource=com.ontotext.ordi.trree.TRREEAdapter
```

The file uses a typical Java property format [2], where each line corresponds to single key and assigned value. All lines starting with # or ! are comment lines. To change the default implementation of ORDI data model the user has to alter the key value of **com.ontotext.ordi.tripleset.TSource** to a class name that implements the new interface. The first three parameters **explicit-collection**, **map-collection** and **inferred-collection** are used to tune the performance of the underlying TRREE engine and define the expected size of the collections to hold the statements. For additional information please refer TRREE documentation [3].

The TRREEAdapter requires at least 256MB. To instantiate the ORDI model using the default configured implementation use:

```
import com.ontotext.ordi.Factory;
import com.ontotext.ordi.tripleset.TSource;

...

TSource source = Factory.createDefaultTSource();
```

The ORDI data model has six basic data manipulation operations. Extensive tests are available to test the operation compatibility of a specific implementation with the specification. Each version of the ORDI data model specification brings a list of operations and strict recommendation for their implementation. To verify the implementation conformation with the formal specifications a suite of tests is distributed. The tests cover the basic ORDI model operations and verify the correct implementation of the java interfaces.

Further information and the latest software documentation are available at the ORDI framework SourceForge website [4].

[1] http://ordi.sourceforge.net
[2] http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html
[3] http://www.ontotext.com/trree/index.html
[4] http://ordi.sourceforge.net

7 Conclusions

A high-performance storage system is a key prerequisite to achieve the goals of Triple Space. This document presents the initial prototype of high performance Triple Space storage, which allows integration of heterogenous data sources, such as OWLIM, YARS and RDBMS, through various adapters. Then a single view-based access and storage for data is made possible through the FederatioAdapter. This storage is implemented to support data integration for heterogenous data sources as well as to allow federation of multiple data sources across network.

The integration of heterogeneous data is achieved based on different adapters such as TRREEAdapter, YARSAdapter, RDBMSAdapter. These adapters are based on the ORDI data model and enable integration of different data models with persistency and reasoning support. Only read operations are supported for integrating relation data model, as the support for other operations were either undesired or infeasible.

The federation of multiple data sources across network is achieved through FederationAdapter based on the ORDI data model. This adapter integrates multiple data sources that are local to a Triple Space kernel, while keeping the autonomy of individual data sources. These multiple independent data sources or storage systems are federated into a single virtual data store. Through this integration and federation strategies for data integration, Triple Space storage not only supports a scalable, efficient and distributed storage, but also provides an approach to integrate external legacy data repositories with the Triple Space storage infrastructure.

The table below summarises the four different adapters built for accessing Triple Space data store.

Table 7.1: Triple Space Adapters

TRREE Adapter	TRREE stands for Triple Reasoning And Rule Entailment Engine. The TRREE Adapter provides efficient reasoning and forward-chaining of entailment rules on top of RDF extended named graphs.
YARS Adapter	The YARS Adapter allows persistent storage and retrieval of RDF triples in YARS. The YARSAdapter implements the ORDI data model. It manages mismatches between the YARS and ORDI interfaces with respect to insert, delete and query functionalities.
RDBMSAdapter	The RDBMSAdapter bridges the ORDI data model and relational databases. It brings a flexible way to add semantics to existing data sets and enables read-only access to information stored in relational databases using the ORDI data model.
FederationAdapter	The FederationAdapter is an implementation of the ORDI data model. This adapter federates data into multiple data stores which are local to each Triple Space kernels. It keeps the individual data stores bound together and hides the distribution complexity for the ORDI data model.

References

[1] YARS: Yet Another RDF Store. URL: http://sw.deri.org/2004/06/yars/.

[2] Stefan Decker Andreas Harth. Optimized Index Structures for Querying RDF from the Web. In *In 3rd Latin American Web Congress*, Buenos Aires, Argentina, November 2005.

[3] Stefan Decker Andreas Harth, Jrgen Umbrich. MultiCrawler: A Pipelined Architecture for Crawling and Indexing Semantic Web Data. In *5th International Semantic Web Conference (ISWC 2006)*, Athens, GA, USA, November 2006.

[4] Sesame Open Source RDF Schema based Repository and Querying facility.

[5] Tim Berners-Lee. An RDF language for the Semantic Web. W3C Recommendation, 1998. Available at: http://www.w3.org/DesignIssues/Notation3.html.

[6] Tim Berners-Lee. Relational Databases on the Semantic Web, September 1998.

[7] Min Cai and Martin Frank. RDFPeers: A Scalable Distributed RDF Repository based on A Structured PeertoPeer Network. In *In Proceedings of the 13th International World Wide Web Conference (WWW 2004)*, New York, NY, USA, May 2004. ACM Press.

[8] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. *Journal of Grid Computing*, 2(1), March 2004.

[9] Isabel F. Cruz, Huiyong Xiao, and Feihong Hsu. Peer-to-Peer Semantic Integration of XML and RDF Data Sources. In *in proceedings of workshop on Agents and Peer-to-Peer Computing (AP2PC), held at 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2004)*, New York, NY, USA, July 2004. ACM Press.

[10] J. Grant D. Beckett. Semantic Web Scalability and Storage: Mapping Semantic Web Data with RDBMSes, SWAD-Europe deliverable, W3C (2003), September 2003.

[11] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, Baltimore, Maryland, USA, June 2005.

[12] A. Ghioni E. Della Valle, A. Turati. PAGE: A Distributed Infrastructure for Fostering RDF-Based Interoperability. In *in proceedings of Distributed Applications and Interoperable Systems (DAIS 2006)*, Bologna, Italy, June 2006.

[13] Lee Feigenbaum, Sean Martin, Matthew N. Roy, Benjamin Szekely, and Wing C. Yung. Boca: an open-source RDF store for building Semantic Web applications. *Journal of Briefings in Bioinformatics*, 2(2), 2005.

[14] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures. Ph.D. Thesis, University of California, Irvine, 2000. Available at: http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm.

[15] H. Stuckenschmidt G. Adamku. Implementation and evaluation of a distributed RDF storage and retrieval system. In *in The 2005 IEEE/WIC/ACM International Conference Web Intelligence*, September 2005.

[16] Andreas Harth and Stefan Decker. Optimized Index Structures for Querying RDF from the Web. In G. Goos, J. Hartmanis, , and J. van Leeuwen, editors, *Proc. of the 3rd Latin American Web Congress.* IEEE Press, November 2005.

[17] Andreas Harth, Gabor Nagypal, and Damyan Ognyanoff. DIP D2.5: Ontology Repository. DIP WP 2 Deliverable, June 2006. Available at: http://sw.deri.org/2005/03/diprdf/wp2.5/.

[18] Kien A. Hua and Chiang Lee. Handling Data Skew in Multiprocessor Database Computers using Partition Tuning. In *Proc. of 17th VLDB Conference*, 1991.

[19] Maksym Korotkiy and Jan L. Top. From relational data to RDFS models. In *in the proceedings of International Conference on Web Engineering (ICWE 2004)*, July 2004.

[20] Reto Krummenacher, Elena Simperl, doug foxvog, Vassil Momtchev, Dario Cerizza, Davide Cerri, Brahmananda Sapkota, Kia Teymourian, Daniel Martin, Hans Moritsch, Omair Shafiq, and David de Francisco. D6.5 Towards a Scalable Triple Spac. TripCom Deliverable, March 2008.

[21] M. Lenzerini. Data Integration: A Theoretical Perspective. In *In Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2002)*, Madison, Wisconsin, USA, June 2002.

[22] et. al. M. Antonioletti, N. Chue Hong. OGSA-DAI 3.0 - The Whats and the Whys. In *Proceedings of the UK e-Science All Hands Meeting 2007*, September 2007.

[23] Holger Märtens. A Classification of Skew Effects in Parallel Database Systems. In *Proc. of Euro-Par 2001 Conference*. Springer Verlag, August 2001.

[24] Vassil Momtchev, Stijn Heymans, Jos de Bruijn, and Axel Pollers. D2.3 Triple Space Knowledge Representation. TripCom Deliverable, March 2008. Available at: http://tripcom.org/docs/del/Tripcom-D23.pdf.

[25] Vassil Momtchev and Atanas Kiryakov. D1.2 Specification of the Store Architecture and Interfaces. TripCom Deliverable, October 2006. Available at: http://tripcom.org/docs/del/Tripcom-D12.pdf.

[26] Boris Motik, Ian Horrocks, and Ulrike Sattler. Bridging the Gap Between OWL and Relational Databases. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proc. of the 16th International World Wide Web Conference (WWW 2007)*, pages 807–816, Banff, AB, Canada, May 8–12 2007. ACM Press.

[27] B. Omelayenko. RDFT: A Mapping Meta-Ontology for Business Integration. In *Proceedings of theWorkshop on Knowledge Transformation for the Semantic Web at the 15th European Conference on Artificial Intelligence (KTSW2002)*, 2002.

[28] E. Oren, R. Delbru, S. Gerke, A. Haller, and S. Decker. ActiveRDF: Object-oriented semantic web programming. In *In Proceedings of the International World-Wide Web Conference (WWW 2007)*, Banff, Canada, May 2007.

[29] Eric Prud'hommeaux and Andy Seaborne, editors. *SPARQL Query Language for RDF*. W3C Candidate Recommendation, apr 2006.

[30] Brahmananda Sapkota, Sven Groppe, Vassil Momtchev, and Janne Saarela. D1.1 State-of-the-Arts and Requirements Analysis. TripCom Deliverable, April 2006. Available at: http://tripcom.org/docs/del/Tripcom-D11.pdf.

[31] Heiner Stuckenschmidt, Richard Vdovjak, Jeen Broekstra, and Geert-Jan Houben. Towards distributed processing of RDF path queries. *International Journal of Web Engineering and Technology*, 2(2), 2005.

[32] Heiner Stuckenschmidt, Richard Vdovjak, Geert-Jan Houben, and Jeen Broekstra. Index structures and algorithms for querying distributed RDF repositories. In *in Proceedings of the 13th international conference on World Wide Web (WWW 2004)*, New York, NY, USA, July 2004.