TripCom

Triple Space Communication

FP6 – 027324

Deliverable

D3.2

State of the art and Triple Space-specific requirements of semantic query languages

Janne Saarela
Tommi Koivula
Lyndon Nixon
Axel Polleres
David de Francisco

March 23, 2007

Executive Summary

This document presents why querying is needed in the TripCom project. This document then presents general query language features and shows how they are present in existing RDF query languages. The target audience for this document is software engineers who wish to understand technical features of query languages both in general and in the context of the TripCom project.

This document presents how rule languages can be used to add to the expressivity of query language especially in the context of querying RDF data models.

Next, the document presents requirements for a query language for Triple Space, a semantic co-ordination middleware developed in the TripCom project, which originate from three use case domains, namely Semantic Web service communication, Enterprise Application Integration (EAI), and Electronic Patient Summaries in eHealth (EPS).

Finally, this document recommends the use of SPARQL query language with identified extensions to be used as the basis for querying TupleSpaces.

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**	3.2	**Title**	State of the art and Triple Space-specific requirements of semantic query languages
Work Package	**Number**	3	**Title**	Triple Space Interaction

Date of Delivery	**Contractual**	M12	**Actual**	31-Mar-07
Status	version 1.1		final ☐	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Janne Saarela, Tommi Koivula (Profium), Lyndon Nixon (FU Berlin), Axel Polleres (DERI Innsbruck), David de Francisco (TID)			
Resp. Author	Janne Saarela (Profium)		**E-mail**	tripcom@profium.com
	Partner	Profium	**Phone**	+358 (0)9 855 98 000

Abstract (for dissemination)	This document presents semantic query languages, use case driven TripCom specific query requirements and recommendations about what query language to use in TripCom project.
Keywords	TripCom, RDF, SPARQL

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
June 15th, 2006	1	Janne Saarela, Axel Polleres	Initial draft for internal discussions
July 17th, 2006	2	Janne Saarela	Restructuring, fixed typos
September 14th, 2006	3	Janne Saarela	New structure
September 21st, 2006	4	Lyndon Nixon	Added Linda querying chapter
September 26th, 2006	5	Axel Polleres	Refined structure of rules chapter
October 2nd, 2006	6	Janne Saarela	Added assignments for WP partners
October 5th, 2006	7	Janne Saarela, Tommi Koivula	Added data manipulation primitives and elaborated features
December 28th, 2006	8	Janne Saarela, Tommi Koivula	Added feature matrix and sample queries from WP8 scenarios
January 13th, 2007	9	Axel Polleres	Added text to rules and added some requirements
February 4th, 2007	10	Janne Saarela, Tommi Koivula	Completed query language features
February 8th, 2007	11	Janne Saarela	Service description, closure, view and data manipulation API reference
February 9th, 2007	12	Lyndon Nixon	Added scenario sample queries
February 19th, 2007	13	Lyndon Nixon	Added scenario requirements extraction (first version)
February 28th, 2007	14	Janne Saarela	Introduction, Summary and Recommendation
March 1st, 2007	15	Lyndon Nixon	Minor edits, completed candidates for querying section
March 5th, 2007	16	Janne Saarela	Spellcheck before internal QA review
March 15th, 2007	17	Janne Saarela	Incorporated internal QA review comments
March 16th, 2007	18	David de Francisco	Additional internal comments
March 16th, 2007	19	Lyndon Nixon	Pre-final version
March 23rd, 2007	20	Janne Saarela	Final version

Project Consortium Information

Acronym	Partner	Contact
Leopold Franzens University Innsbruck http://www.deri.at	LFUI	Prof. Dr. Dieter Fensel Digital Enterprise Research Institute (DERI) Innsbruck, Austria E-mail: dieter.fensel@deri.org
National University of Ireland, Galway http://www.deri.at	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, Finnland 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	Sara Carro Martinez Telefonica I+D Madrid, España E-mail: tripcom@tid.es

Table of Contents

1 Introduction

This document starts by presenting why querying is needed in TripCom project.

This document then presents general features of semantic query languages. The focus of this document is in query languages that operate on Semantic Web metadata descriptions i.e. the ones based on the Resource Description Framework (RDF) data model. The features and the languages are then presented in a feature matrix which summarizes the current state-of-the-art.

Rules technologies are then presented as they can be used to encode ontologies i.e. formal conceptual models which enable inference processes to derive implicit knowledge from explicit RDF descriptions. In addition, this document presents how rules can be used together with RDF query languages.

The document then presents sample queries which are derived from use cases explored by the TripCom project, namely from Semantic Web Services (SWS), Enterprise Application Integration (EAI) and EPS (Electronic Patient Summaries).

Finally this document presents recommendations as to what RDF query language and rule technology the TripCom project should adopt to satisfy the use case driven requirements.

2 Querying and rules in Linda

Traditional tuplespace computing systems have used the Linda coordination language [24] or extensions thereof to define how clients would interact with the system in a coordinated manner. Hence, it is worthwhile to consider at the beginning how querying has been enabled in classical Linda-based systems and their extensions. Equally, the programmability of the tuplespace has been an issue of relevance to some Linda implementations and it is worthwhile to consider how the provision of programmability has also extended the richness of retrieval operations of such systems. As a result we gain a perspective over the current state of the art of retrieval in Linda-based coordination systems and can assess to what extent this state of the art could be sufficient for the proposed Triple Space.

2.1 Querying: Linda templates

Retrieval is governed in the Linda model by matching tuples against a template, which is defined as a tuple which contains both literals and (possibly typed) *variables*. A match between a template and a tuple occurs when the template and the tuples are of the same length, the field types are the same and the value of constant fields are identical.

Classical Linda performs retrieval by binding the variables in the template to concrete values taken from a single matched tuple found in the space. For example the tuple *("N70241", EUR, 22.14)* - three fields containing a string, a pre-defined type (here, currency codes) and a float - will match the template *("N70241", ?currency, ?amount)* binding the values EUR and 22.14 to the variables *currency* and *amount*, respectively. An alternative, realised in some Linda implementations, is to return a tuple object which matches the provided template rather than perform variable binding.

We can note that this form of retrieval is very simple. In cases which do not need any richer query expressions, it can be a very effective and efficient means to acquire matching tuples. In Linda only the first found matching tuple is returned - hence the entire tuplespace must not be searched. This allows for approaches to retrieval resolution that work on the structure of the tuplespace to find potential matches as effectively as possible.

The cost of matching templates to tuples will depend upon the format of the tuples and the complexity of the data structures they contain. In the case of Triple Space we have a simple tuple format which is based on the RDF data model of (subject, predicate, object) but a more complex data structure (once we begin to include reasoning based on RDF(S) semantics).

In terms of relational querying this corresponds to a SELECT... WHERE... with the variables to be bound given in the SELECT clause and the template contains both the concrete values and variables in the WHERE clause. In classical Linda retrieval, the WHERE clause is restricted to a single statement, and the set of answers are restricted to a single set of variable bindings.

2.2 Querying: Linda extensions

Linda has been extended to include an operation to return *all* matching tuples found in the space ("multiple read"). Here, the set of answers to the retrieval operation is no longer restricted, i.e. all answers are returned as is typical in query languages.

In Jada [7], a *getAny* operation allows a client to provide a set of templates and returns all tuples found in the space that match at least one of those templates. This is a disjunctive query, i.e. the sum of all individual matches. The *getAll* operation allows a client to provide a set of templates and returns all tuples found that match all of those templates. This is a conjunctive query, i.e. the intersection of all individual matches. In terms of the relational query, this removes the restriction on multiple WHERE clauses, where the semantics is either to consider the clauses complementary or disjoint.

For richer querying, we consider the extensions implemented in the IBM TSpaces system [47]. TSpaces extends matching to include consideration of tuple types as well as field types, allows tuple fields to be named so that matching can be performed on a subset of tuple fields and supports expressing ranges of values to find. Furthermore, TSpaces uses the relational database backend to add support for extending matching with AND and OR operators, allowing for more SQL like retrieval operations. In comparison with the Jada operations, TSpaces allows AND and OR to be arbitrarily nested so that complex query trees can be built up. We note however even this further extension of Linda matching is not as expressive as SQL over a relational database, e.g. a JOIN still requires in TSpaces a *scan* over the tuplespace, a *query* operation and then a merge.

2.3 Rules in tuplespaces

More expressive retrieval in a tuplespace may be made possible by the addition of programmability in the space. Clients may be able to insert programmatic components in the space as tuple content which are then resolved within the system and a tuple containing the result data can be obtained. The simplest example of this was the *eval* primitive in the original Linda proposal.

However, it was dependant on the underlying programming language used in the implementation rather than some shared means to express programmability in the space which respects the heterogeneity of both the platforms on which the tuplespace runs and of the clients interacting with that tuplespace.

While programmability is often added to tuplespaces primarily to control agent access, the extension can also be applicable to changing the semantics of the matching operations. For example, MARS [5], among others, supports the specification of "reactions" in the space which are methods of a Java object (MARS uses Java for its programmatic environment) tied to some access event (e.g. a retrieval operation from a specifiable agent with a specifiable template). When a matching access event occurs, the method on the Java object is executed.

Reactions can be considered to be a form of rules specifiable in tuplespace. The expressiveness of the rules depends on the range of functions available in the specification of the reactions. One can either specify that the insertion of certain tuples in a space generates additional tuples which could then be retrieved by the right template, or that retrieval operations of certain types in a space first allow some calculation to

be made before matching takes place. It appears that reactivity is generally applied to single tuples independently of the (non-)existence of other tuples, i.e. applying general methods to fields based on their datatypes such as fixed mathematical operations upon an integer. This suggests that operations over sets of tuples (such as aggregration) may not be possible in Linda (while other coordination languages do support this).

The possible modifications to matching that is made possible by reactivity in a space are not able to fully express a query language such as SQL nor the use of a more expressive language (e.g. Datalog). Fundamentally, reactions are based on a combination of existing operations and a specific template, so that the semantics of the intended operation is tied to the data being queried rather than the query operation. This means the form of data in the space must be knowable in advance for reaction rules to be specifiable, rather than agents being able to rely on the use of a specific query construction to guarantee a particular retrieval semantic.

2.4 Conclusion

This chapter has given a brief overview of retrieval in tuplespace systems. While Linda templates can provide a simple, efficient means to retrieve a tuple from the space, it has very simple retrieval semantics that may not be expressive enough for many scenarios. A number of extensions to Linda can loosen the query expressiveness restrictions, so that e.g. multiple templates may be specified or multiple results returned. However, in comparison with classical query languages, e.g. SQL in the relational database world, Linda does not provide the same level of query expressiveness for retrieval. Programmability is one way to support a number of additional query functions, however in other systems support for other query languages outside of Linda have been added. This has the advantage of allowing clients to use query expressions they may already be programmed for, as well as reusing known standards in the field. For example, a number of tuplespace systems which were extended to support XML data in tuples, e.g. XMLSpaces [44], also supported the use of known XML Query languages to retrieve matching tuples from the space. As a result we recommend a similar approach for Triple Space i.e. tuples are RDF statements which form RDF graphs which can be effectively queried using dedicated RDF query languages. We propose that Triple Space support a dedicated RDF query language for richer retrieval of matching tuples in the space.

3 Semantic query languages

This section presents Semantic Query Language features. The features are enumerated here so that TripCom partners can understand what generic characteristics semantic query languages have. This in turn helps the project formulate the requirements for querying in the context of TripCom.

This section is called 'Semantic query languages' rather than 'query languages' is due to the approach selected by the TripCom project. This approach is based on the use of semantic technologies that enable non-ambiguous descriptions of message structures communicated by different software systems via the Triple Space architecture. In addition to non-ambiguity, semantic technologies such as Resource Description Framework (RDF), RDF Schemas (RDFS) and Web Ontology Language (OWL) enable software systems infer implicit knowledge from explicit knowledge. This inference process adds important capabilities to Triple Space communication by letting computers infer more knowledge from the exchanged messages than the communicating parties have expressed.

3.1 Query language features

This section describes general query language features which can be found in several languages that deal with different data models (e.g. SQL, XQuery, SPARQL).

3.1.1 Data extraction features

Projection

Projection is a generic term for looking for data in the underlying data model. Projection looks for values in a table, in an XML document or in an RDF data model (directed labelled graph) and returns those values typically as variable bindings.

A sample query making use of projection in natural language: **What are the allergies suffered and the vaccination of patient XYZ?**

Constraints on structure

Constraints can be used with projection or other data retrieval primitives. The constraints can take into account parts of the underlying data model and thus limit the results of the query. If a constraint is expressed on the structure, it effectively means that a certain structural part or parts must be present or they must not be present in order to be part of a query result.

There are typically multiple constraints expressed in a query. Multiple constraints must be expressed with clear semantics that require all of the constraints to be satisfied or one of them to be satisfied. This would effectively map to AND or OR semantics in the Boolean algebra. Boolean algebra is often seen as the key ingredient in expressing multiple constraints.

A sample query making use of structural constraints in natural language: **Retrieve the phone number of patient XYZ if (s)he does not have an email address.**

Disjunction/Conjunction

A query may also consider multiple criteria in the structure of the underlying data model. This criteria can be expressed as a disjunction (OR) or as a conjunction (AND).

A sample query making use of disjunction in expressing contraints on structure: **Retrieve the IDs of patients who have Peniciline allergy OR have a tetanus vaccination.**

Query languages/engines which only support conjunctive queries cannot answer disjunctive queries directly. Rather, they can break disjunctive queries down into seperate queries which are answered independently and then the result sets combined.

Set Difference/Negation

A query language may also support set difference or negative queries.

A sample query making use of set difference: **Retrieve the IDs of patients who have Peniciline allergy and do NOT have a tetanus vaccination.**

Please note that these queries are problematic since in an open environment they can only be answered if either negative information is explicit or if the scope of negation is explicit and finite.

3.1.2 Recursion/Reachability

A query language on graphs can support reachability or transitive closure queries. Reachability queries are typical over transitive relations such as asking for descendants: **Give me descendants of patients with inheritable illnesses.**

A *transitive closure* query, similar to negative queries or aggregates, can only be answered if locally complete knowledge is available, e.g. **Give me ALL descendants of patient with ID.**

3.1.3 Ground and inferred data models

A query language may exploit not only an explicitly persisted data model but also a data model which can be inferred from the explicit data model. Sometimes these two data models are called *ground* and *inferred*.

For example, a sample query **Give me descendants of patients with inheritable illnesses** evaluated against a ground data model means the data model holds information about *descendant* relationships as well as about 'inheritable illness' concept. Evaluating the very same query against a query service which supports inferred data models may also determine *descendant* relationships from the ground data model and additional inference rules which might be encoded in ontologies such as RDFS and OWL. This ground and inferred data model capability has no effect on the query language syntax. However, it has an effect on the semantics of the query language expressions.

Constraints on values

If a constraint is expressed on a value, the query language needs to be aware of the type of the value. If the type is not known, typically the only available operator available for value constraints is equality or non-equality. With additional knowledge

about the types of the values, a constraint can use other operators such as *bigger-than*, *smaller-or-equal-than*, *before* or *overlaps*.

A sample query making use of constraints on values in natural language: **What were the drugs taken by patient XYZ during a given period of time?**

Graph construction/closure

A query may not always be a projection whose results are values of a given part of the underlying data model. A query may also create new data in the same data model as the underlying data model. The capability to create query results in the same data model as the underlying data model is known as 'closure'. This feature enables subqueries where a result of one query may be the input for another query.

A sample query making use of graph construction in natural language: **Which patients have had physical contact with during treatment** (This sample assumes the data store on which the query is run does not include such 'physical-contact-with' information explicitly.)

Graph construction can also be seen as a 'view' creation mechanism as it can map underlying data to a different data model at query evaluation time.

There are initiatives such as SPARQL Service Adversitement and Discovery Language (SADDLE)[8] to use graph construction features to describe the properties of query processors. A client could request properties of the query processor before issuing query expression. This helps in determining the capabilities of the query processor and helps to establish a correct query and to expect a correct result type from the processor. For example, a server may support N3 encoding of graphs in the query result. A client can query the server capabilities in a handshake phase and determine the N3 encoding capability before issuing a query request.

Boolean queries

Boolean queries can be used to check for the existence of data in the queried data model. The result is simply true or false depending whether the query criteria are met or not. For example, **Has this medicine been given to treat that allergy?**

Aggregate functions

Query languages can also return aggregate results. Such aggregate results may come from function evaluation which takes as input multiple results (projections or graph construction/closure) and which results in a singular result of a given type.

For example, MIN and MAX are functions which return a single result in e.g. double precision integer value space. Another example would be CONCAT which could result in a string concatenation of multiple strings into a single string.

A sample query making use of an aggregate function in natural language: **How many patients have been diagnosed with some type of cancer?**

3.1.4 Context/Provenance

The evaluation of a query language expression may be distributed. The distribution can be implemented either by a single query service aggregating data from multiple sources or by dispatching the same query to multiple query services. In such an

evaluation environment a query language may provide information about the context or provenance where the query conditions are met.

A sample query such as: **Which TripCom servers hold information about patient XYZ?** requires the query language to return information not only about the data model but about the context or provenance where such information is present.

In the context of the RDF data model, provenance can be addressed via a 4th slot in the traditional 3-tuple/triple data model of RDF. Such quads can encode information about the origin of an individual triple and expose that information for querying.

3.1.5 RDF primitives

Query languages that allow querying data encoded using RDF need to decide whether they support bNodes - resources nodes which are only unique in the context of a surrounding graph - and RDF collections/containers. In addition, RDF has support for higher-level statements i.e. the ability to make statements about statements. In order to do this, RDF defines a process called *reification* which decomposes a 3-tuple/triple to several 3-tuples. Some domains make use of these features in the RDF data model. Running queries in those domains requires support for these primitives.

3.1.6 Data manipulation features

In addition to the data retrieval functions, users typically expect a query language to have some primitives to change the underlying data model. These primitives are enumerated in the following subsections. In TripCom the data manipulation features are to be provided via API calls but they are still presented here for completeness. Query languages are read-only by nature but they may have support for these data manipulation features in their syntax and semantics.

Insert

Adding data to the underlying data model is known as the INSERT operation. Such an operation may succeed or fail depending whether the data meets with validity or integrity constraints associated with the data store system.

Modify

Modifying data in the underlying data model is known as the MODIFY or UPDATE operation. Such an operation may succeed or fail depending whether the data meets with validity or integrity constraints associated with the data store system.

Delete

Deleting data from the underlying data model is known as the DELETE operation. Such an operation may succeed or fail depending whether the data meets with validity or integrity constraints associated with the data store system. In RDF data model context deleting an inferred triple makes no sense. Thus the delete will most probably be restricted to ground triples. This has an implication that a client trying to invoke a delete operation should know from all the triples which ones are ground and which ones are inferred in order to invoke correct delete operations.

3.1.7 Transaction control

A query system may exhibit support for transactions. This effectively means that when multiple parties are operating against a single data store, any modifications for the data need to be controlled so that integrity and validity of the data can be controlled. If a modification is an atomic operation such as adding a single data entry or deleting or single data entry, no transaction control is needed. However, when several changes are executed against a data store, it is desirable to control the changes as an atomic operation instead of individual operations part of which may succeed and part of which may fail.

ACID transactions

Atomicity, Consistency, Isolation and Durability are often associated with transactions. Atomicity means a set of modifications is done completely or none of the individual items of the set is done. Consistency guarantees none of the integrity constraints of the underlying database are broken after the transactions i.e. the database is in a consistent state before and after a transaction. Isolation means the set of modifications will be visible as a whole to other users of the same data store i.e. individual changes cannot be observed by other users. Finally, Durability means the set of modifications is persisted by the data store and they are not lost because of system faults.

Open transactions

Open transactions are often associated with long-running transactions where so-called critical resources cannot be held for a single user at a time. In such environments the different modifications within the transactions are not in isolation but become visible to other users when the transaction proceeds. If the transaction ever needs to be interrupted or roll-backed in transactional terms, the previous modifications need to be cancelled with what is known as 'compensative' operations.

3.2 RDF Query languages

The following sections describe different RDF Query languages. These languages have been selected based on the following criteria: the language should be specified openly in a way that allows its implementation without royalty costs and that the language should have more than 1 implementation. This latter requirement ensures that the language definition has better interoperability if more than 1 party has implemented it from a specification.

3.2.1 SPARQL

SPARQL Query Language for RDF [41] is a technology developed by the Data Access Working Group (DAWG) at the World Wide Web Consortium (W3C) since Spring 2004. The technology is currently in W3C Working Draft state.

3.2.2 RDQL

RDF Data Query Language (RDQL) [1] is a technology developed by the HP Research Labs in Bristol, UK, and first implemented in their Jena toolkit.

3.2.3 RQL

RDF Query Language (RQL) [32] is a technology developed by Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis and Michel Scholl at the Institute of Computer Science in Heraklion, Greece.

3.2.4 SeRQL

SeRQL[2] is a technology developed within the Sesame open source project.

3.2.5 RxPath

RxPath[3] is a technology developed by Adam Souzis and implemented in the Rx4RDF open source project. RxPath is syntactically based on XPath technology but semantically operates on RDF data model via a canonical XML representation of the RDF data model.

3.2.6 Other languages

There are many other RDF query languages. A comprehensive introduction to these other languages is available in a REWERSE project deliverable [23].

3.3 Comparison of RDF query language expressiveness

The feature matrix (Table 3.1) shows the query language features with the query languages.

[1] http://www.w3.org/Submission/RDQL/
[2] http://www.openrdf.org/doc/sesame/users/ch06.html
[3] http://www.liminalzone.org/RxPath

Feature	SPARQL	RDQL	SeRQL	RxPath	FOL
Projection	Yes	Yes	Yes	Yes	Yes/No
Constraints on structure	Yes	Yes	Yes	Yes	Yes
Constraints on values	Yes	Yes	Yes	Yes	No
Graph construction	Yes	No	No	No	n/a
Boolean queries	Yes	Yes	No	Yes	Yes
Aggregate functions	No	No	No	No	No
Data manipulation features	No	No	No	No	No
Transaction control	No	No	No	No	No
Disjunction/UNION	Yes	No	Yes	No	Yes
Set Difference	Yes	No	Yes	No	No
Recursion	No	No	No	No	Yes/No[I]
RDFS support	Yes/No[II]	Yes/No[II]	No	No	Yes[III]
OWL support	Yes/No[IV]	No	No	No	Yes
Context	Yes	No	No	No	Yes/No[V]

[I] Transitivity yes, transitive closure no

[II] Entailment Regimes foresee this in principle, Jena Rules can e.g. emulate large parts of RDFS, but the infinite axiomatic triples forbid full coverage

[III] By encoding ter Horst's or Draltan's version of RDFS rules, infinite axiomatic triples can be coded by a function symbol.

[IV] Entailment Regimes foresee this in principle, some engines such as Pellet partly support SPARQL, OWL Full is quasi impossible

[V] Context can be encoded in FOL, e.g. with an exrtra parameter of predicates, see e.g.[40].

Table 3.1: Features vs. existing query languages

4 Ontology and Rule languages

This chapter introduces various ontology and rule languages, most of which, but not all, are specifically tailored towards RDF.

The investigation of these languages in the context of Triple Space querying is interesting for two main reasons:

- We can not expect all knowledge in the Triple Space to be available as plain RDF graphs. Querying should cover inferences from the facts stored in plain RDF using sets of axioms and rules defined in **ontologies**. To this end, we need to investigate to what extent ontological knowledge can be combined with query languages. The rest of the chapter focuses on aspects of the feasibility of taking particular rules or ontology languages as the basis for entailment regimes in SPARQL.

- Rules languages serve also as query languages. For instance, Datalog is well known to serve well as a powerful query language supporting features such as recursion or negation, not natively available in all standard query languages for RDF.

This chapter is organized as follows. First, a brief review of the two main knowledge representation paradigms for representing ontological knowledge will be given, classical first-order logic and logic programming. There is plenty of introductory literature available on these topics, so focus is on the illustrating main differences and features in the context of RDF/Triple Space querying.

Second, an introduction of several concrete languages is given by example and all their main features as well as their adequacy to represent ontological knowledge, combinability with classical RDF query languages, and usability as a query language itself is presented.

To keep our preliminary research results separate from the state-of-the-art discussion which this deliverable focuses on, this document refers to some recent research results on the relation between RDF query languages outlined in chapter 3.2 and rules languages discussed in this chapter which are provided in a separate Technical Report [39].

4.1 Preliminaries

Before going into details, a short overview about different approaches towards describing ontologies and rules will be given. The most important semantics will be described briefly to clarify the differences. These concepts make it easier to classify and thus understand the expressivity of each of the rule languages.

Combining the different approaches for knowledge representation underlying ontologies outlined in this chapter is an open research topic [11] and thus out of scope of this deliverable which focuses on available techniques and well-investigated semantics of query languages.

4.1.1 Classical (First-order) Logic

Classical first-order Logic (FOL) is not described here. The interested reader if referred to excellent standard introductory texts such as [22] here. FOL serves as a semantic

foundation of OWL [17], KIF [26], Common Logic[18], etc. Relations between classical logic and RDF as well as using classical logic on top of an ontology language will be described.

Writing RDFS in classical logic

In order to extend RDF data by first-order logic formulae, RDF needs to be mapped to first-order logic. There are three possible choices for this.

Properties and classes as predicates A seemingly natural choice is to model all properties except rdf:type as binary predicates and rdf:type, i.e. the "isA" or "classMembership" relation as a separate unary relation for each class.

For example, we will refer to classes and properties from the European Patient Summary (EPS) schema being developed in WP8B and additional FOAF[1] data.

Taken the statements:

- *`http://www.tripcom.org/eps/~retok` is a European patient summary.*
- *The summary `http://www.tripcom.org/eps/~retok` was registered at date 2006-10-30, 18:49:00.*
- *The summary belongs to somebody with foaf:mbox `mailto:reto.krummenacher@deri.org`.*

In RDF using TURTLE [4] notation (which is easier to read than RDF/XML and also close to SPARQL's syntax) we could express this as follows:

```
<http://www.tripcom.org/eps/~retok> a eps:EPS.
<http://www.tripcom.org/eps/~retok> eps:head
            [eps:registrationDate "2006-10-30T18:49:00"].
<http://www.tripcom.org/eps/~retok> eps:head
            [eps#patientContactInformation
               [foaf:mbox <mailto:reto.krummenacher@deri.org>]].
```

Blank nodes, which are heavily used here, resemble incomplete information which can be expressed in first order logic using existential variables.

In total, the triples above would correspond to something like the following FOL formula:

$$\begin{aligned}&\exists x_1, x_2.\\&EPS(retok) \wedge head(retok, x_1) \wedge\\&registrationDate(x_1, "2006-10-30T18:49:00") \wedge\\&patientContactInformation(x_1, x_2) \wedge\\&mbox(x_2, "reto.krummenacher@deri.org")\end{aligned}$$

This notation is perfectly compatible with Description Logics variants such as those corresponding to OWL[17] and Semantic Web Rules proposals such as e.g. SWRL[31], see Section 4.2.3 below.

Note however, that using this notation, even simple rules like the RDFS entailment rules from the RDF semantics document [29] are not expressible in a straight-forward manner.

We know that `foaf:mbox` has `rdfs:domain` the class `rdfs:domain`. In first-order-logic the corresponding triple

[1] see `http://www.foaf-project.org/`

`foaf:mbox rdfs:domain foaf:agent.`

is expressible as follows:

$$\forall x, y.mbox(x, y) \rightarrow agent(x)$$

However, note that in this modelling certain triples referring to RDFS vocabulary are treated differently than others, ie. this is translated as simply:

$$domain(mbox, agent)$$

and indeed this would be entailed in OWL full (but not in OWL DL - which is a subset of OWL fulle, more on this later).

RDFS entailment rules themselves can NOT be expressed in this notation, e.g. rule (rdfs3) from [29, Sections 3.1 and 4.1] says that

> If the RDF graph[2] under consideration contains triple `A rdfs:range B.` and `C A D.` then add `D rdf:type B.`

If we tried to write this down using classes and properties as predicates, we'd end up with higher-order logic:

$$\forall A, B, C, D.range(A, B) \wedge A(C, D) \rightarrow B(D)$$

A formalization of a fragment of RDFS semantics can still be created, see [13].

Triples as predicates In order to avoid this problem, one can alternatively use only one dedicated ternary predicate *triple* to model the relation of all triples in first-order logic. The above sample data would then correspond to the formula:

$$\begin{aligned}\exists x_1, x_2.\quad & triple(retok, a, EPS) \wedge triple(retok, head, x_1) \wedge \\ & triple(x_1, registrationDate, "2006-10-30T18:49:00") \wedge \\ & triple(x_1, patientContactInformation, x_2) \wedge \\ & triple(x_2, mbox, "reto.krummenacher@deri.org")\end{aligned}$$

and also the entailment rule (rdfs3) from above could be modeled staying within FOL:

$$\forall A, B, C, D.triple(A, range, B) \wedge triple(C, A, D) \rightarrow triple(D, a, B)$$

Slotted syntax and F-Logic F-Logic [34, 33] may be seen as a syntactic variant of first-order logic especially tailored to emulate slots in first-order logic and get a higher-order "look-and-feel" in the language. Although, strictly speaking properties in RDF and OWL are not really the same as slots (properties and roles amount to global, binary relations whereas slots are local to an object rather like member variables an a programming language like Java), this notation is often useful in the context of RDF ontology/rule languages such as WRL [2] or WSML [16] allow such slotted notation from F-Logic.

Formally, an F-Logic theory is a set of formulae constructed from so called *molecules*. Let σ denote the universe of possible object identifiers and $\mathcal{V}$ denote a (finite or countably infinite) set of variables.

[2] i.e. set of RDF triples

Definition 1 (Molecule) *A molecule in F-Logic is one of the following statements:*

1. *An* is-a *assertion of the form* $C : D$ *where* $C, D \in \sigma \cup \mathcal{V}$
2. *A* subclass-of *assertion of the form* $C :: D$ *where* $C, D \in \sigma \cup \mathcal{V}$
3. *Data molecules of the form* $C[D \twoheadrightarrow E]$ *where* $C, D, E \in \sigma \cup \mathcal{V}$

An F-Logic molecule is called ground, if it contains no variables.

Here, $C : D$ states that C is a member of class D whereas $C :: D$ states that C is a subclass of D. Data molecules of the form $C[D \twoheadrightarrow E]$ have the meaning that for individual C one value of attribute D is E. Attributes are usually viewed to be set-valued[3]. An F-Logic formula is defined by combining molecules with the usual logic connectives $\wedge, \vee, \leftarrow, \leftrightarrow, \neg$ and quantifiers $\forall, \exists$.

In F-Logic notation, we refer to e.g. [33] for details, the above can be written more concisely as follows:

$$\begin{aligned}\exists x_1, x_2. \quad & retok : EPS[head \twoheadrightarrow x_1] \wedge \\ & x_1[registrationDate \twoheadrightarrow \text{"}2006-10-30T18:49:00\text{"}, \\ & \quad patientContactInformation \twoheadrightarrow x_2] \wedge \\ & x_2[mbox \twoheadrightarrow \text{"}reto.krummenacher@deri.org\text{"}]\end{aligned}$$

F-Logic was proposed as a formalization for RDF/RDFS in [28] especially giving a semantics to reification in RDF by using additional features of F-Logic out of scope here. More recently, the exact relation between F-Logic and ontology languyages such as OWL, RDFS and WSML, was investigated in [14].

Description Logics

Description Logics (DL) may be viewed as another syntactic variant of FOL. They have become important in the context of RDF due to the fact that the DL and Lite variants of the Web Ontology language OWL are based on it. Especially important is the DL $\mathcal{SHOIN}(D)$ underlying OWL DL. Compared to full FOL, most description logics have the property of being decidable, that is there exists a sound and complete algorithm to compute satisfiability of a DL theory in one of these variants [4]. An informal overview in tables 4.1 and 4.2 show correspondence among OWL terminology and their DL and FOL representations.

Classical logic as a query language

Classical logic can be viewed as a query language in the following sense: When RDF data is viewed as a conjunctive formula D (with possibly existential variables arising from blank nodes), then conjunctive queries are translatable into a conjunction of literals Q with possibly free variables as well and finding an answer for a conjunctive query amounts to finding substitutions for the free variables in Q such that

$$D \models Q$$

[3] In the original paper set-valued attributes were denoted by by $\twoheadrightarrow$ instead of $\rightarrow$ but this distinction is not needed here.

[4] refer to the Description Logics Handbook [3] for details

For instance in SPARQL, conjunctive queries correspond to basic graph patterns. A simple SPARQL query

> *What is the telephone number of the primary contact person of citizen ¡CitizenID¿?*

```
PREFIX eps:  <http://www.tripcom.org/eps#>
SELECT  ?phone
WHERE   { ?x a eps:EPS;
            eps:head [eps:patientIdentifier <CitizenID>;
            eps:primaryEmergencyContact ?contact] .
           ?contact eps:phone ?phone .}
```

can be – using the unary/binary predicates FOL notation – viewed as the folllowing formula:

$$\exists x, b_1, x_{contact}. \quad eps(x) \ \wedge head(x, b_1) \wedge \\ patientIdentifier(b_1, \texttt{<CitizenID>}) \wedge \\ primaryEmergencyContact(b_1, x_{contact}) \ \wedge phone(x_{contact}, x_{phone})$$

where x_{phone} is a free variable.

More complex queries

Since ontology languages such as OWL and RDFS can be viewed as FOL theories, automated theorem proving can be used for query answering with respect to D under consideration of an ontology O by proving:

$$D \cup O \models Q$$

First-order theorem provers and DL engines can thus be viewed as query answering engines.

However, neither first-order theorem provers nor description logic provers are tailored for the task of query answering. They are tailored to prove sentences, i.e. formulae without free variables, so, the best such an engine can do is trying out all possible combinations of constants appearing in the underlying RDF data and proving entailment for each of these, using, e.g., Tableaux proofs typical for DL reasoners for each possible such assignment. This fact made common DL reasoners perform badly for query answering until very recently. DL engines tailored towards the task of query answering, such as KAON2[5], appeared only recently.

Secondly, the tasks of answering such a query given an arbitrary FOL theory representing the ontology O is undecidable.

To be able to answer queries in Tripcom on RDF data taking ontological inferences such as taxonomies into account, a subset of classical logic, logic programming, which can be extended with non-monotonic negation, is considered.

[5]http://kaon2.semanticweb.org/

4.1.2 Logic programming

This section discusses different logical languages than classical logic treated in the previous section, namely, languages based in logic programming [37] and deductive databases [1, 45]. Main differences between the logic programing paradigm and classical first-order logic are discussed e.g. in [12, 20], which explain that these two paradigms for knowledge representation do not combine in a straight-forward manner.

Logic programming variants such as Datalog and its extensions are far closer to query languages such as SQL than classical logic.

Horn clauses and Datalog

Logic Programming (LP) is based on the idea of "programming in logic". The formulae allowed for programming are rules similar to classical implications over first-order predicates, where the implication symbol $\leftarrow$ is sometimes written :-[6] In LP rules, all variables are implicitly all quantified, rules are written implications, with variables. The most basic LP rules are definite rules with exactly one head atom and a conjunction of positive atoms in the body. These correspond to so called Horn clauses in classical logic, ie. (universally closed) disjunctions with at most one positive literal.

For instance, the rule

$$\forall A, B, C, D.triple(D, a, B) \leftarrow triple(A, range, B) \wedge triple(C, A, D)$$

from above is logically equivalent to the Horn clause

$$\forall A, B, C, D.triple(D, a, B) \vee \neg triple(A, range, B) \vee \neg triple(C, A, D)$$

In LP notation, this would be written as simply:

$$triple(D, a, B) \text{:-} triple(A, range, B), triple(C, A, D).$$

There are LP dialects which support the slotted syntax of F-Logic mentioned above, such as FLORA-2[7], where:

$$D : B \text{:-} A[range- > B], C[A- > D].$$

In logic programming systems, such rules are evaluated under a minimal model semantics. For Horn programs the canonical minimal model (the so-called Herbrand model) coincides precisely with the set of all entailed ground formulae and can, in absence of function symbols, be computed efficiently. This efficient computability has advantages in terms of query answering, i.e. in the query:

$$D \cup O \models Q$$

if D and O consist of only Horn clauses, then the query answers can be computed efficiently by a forward-chaining algorithm without the need for expensive full first-order proof procedures.

[6] this notation comes from Prolog systems, which usually only support ASCII characters and thus the arrow cannot be written.

[7] http://flora.sourceforge.net/

The ontology language WSML-Core relies on this, being definable in terms of function-free Horn-logic and thus allowing for efficient query answering. Logic programming without function symbols is often called "Datalog". The example in the previous subsection falls in this class. Interestingly, Datalog has especially been successfully applied as a query language. For details how Datalog can be used as a query language, particularly how Datalog can be applied to RDF data and emulate SPARQL please see [39]

Compared with query languages such as SPARQL, Datalog adds recursion and transitive closure not expressible in SPARQL allowing an easy formulation of the descendant query described in English in section 3.1.2.

Assuming the parent relationship is defined for an RDF dataset by triples with the property `parent`. Then one can define a recursive query for all descenants of the patient with ID `pID` simply as follows:

```
descendant(X)  :- parent(pID, X).
descendant(X)  :- parent(Y, X), descendant(Y).
```

Adding negation - Normal logic programs

Negation in logic programming commonly refers to "negation as failure" in which if a logical statement cannot be provide false its "negation" is concluded as being true. A special negation operator `naf`[8] (sometimes written `not`) is used to specify such negated terms in rule bodies.

This is different from classical negation, since it denotes a non-monotonic form of negation. The `naf` operator enables expression of set difference.

The sample query from Section 3.1.1 "Retrieve the IDs of patients who suffer from Penicillin allergy and do NOT have a tetanus vaccination." can be written something like the following in LP dialects with negation as failure:

```
answer(ID) :- hasAllergy(ID,"Penicillin"), naf hasVaccination{ID,"Tetanus").
```

This is different from the logical implication

$$answer(ID) \leftarrow hasAllergy(ID, "Peniciline") \wedge \neg hasVaccination(ID, "Tetanus")$$

in that the latter needs negative information about $hasVaccination$ explicitly derivable, in order to obtain a positive overall answer, whereas naf denotes non-derivability of positive information.

Non-Stratified Logic programs There is a straight-forward extension of minimal Herbrand models for *stratified logic programs*, that is, programs where negation as failure does not appear in a recursive cycle. The word "stratified" comes from the fact that program rules can be ordered into *strata*, i.e. layers which do not recursively depend on each other and negative rule dependencies only go from higher to lower strata.

A problem occurs if negation as failure occurs in combination with non-stratified recursion, as in the following example:

[8] Negation as failure

```
male(ID) :- naf female(ID),patient(ID).
female(ID) :- naf male(ID),patient(ID).
agent(ID) :- female(ID).
agent(ID) :- male(ID).
```

Such undesirable combinations might arise when combining rules with negation from several sources.

Together with the information in RDF data that e.g. `nicola` is a patient, one can neither derive straight-forwardly that `nicola` is male nor that `nicola` is female.

There are two standard approaches to solve this dilemma. Either, to take an agnostic position and use a three-valued interpretation, which says that `female(nicola)` and `male(nicola)` are both `unknown` instead of logically `true` or `false`. This is the position that the *well-founded semantics*[46] takes. The other way to treat such a situation is to accept two possible answers namely either `female(nicola)` is `true` and `male(nicola)` is `false`, or the other way around. This is the approach that the *stable model semantics* [25] takes. Note that, when "asking" the query whether `nicola` is an agent, given the above rule set (describing a kind of ontology), the answer would be `unknown` under the well-founded semantics , and `true` under the stable model semantics.

For discussion about the complexity of evaluating queries over rulesets adhering to these different semantics, please see [10].

As for why the discussed languages/semantics are important in terms of RDF querying, recursive programs become an issue, as soon as rules are allowed in Ontology languages and different ontologies refer to each other. Negation as failure equally becomes an issue, for instance when mapping rules or database views are defined to infer implicit knowledge over RDF data. Please see [40] for details.

4.2 The Semantic Web rules layer

Previous section looked at two primary models for modelling rules on top of ontologies: approaches based on classical (first order) logic and approaches based on logic programming. The definition of a rules layer for the Semantic Web on top of RDF and OWL is split between the choice of one or the other approach [30]. Given the possible need for rules in the Triple Space, either to provide added expressiveness as an extension of the query language or to support richer ontology and concept mediation, we turn to the current rules support in the Semantic Web field.

4.2.1 RDFS

RDF and RDFS are to a large extent mappable to first-order logic. A subset of the RDFS semantics can be defined by rules, by encoding the entailment rules from [29, Sections 3.1 and 4.1] as a set of Horn rules:

```
triple(P,rdf:type,rdf:Property) :- triple(S,P,O).
triple(S,rdf:type,rdfs:Resource) :- triple(S,P,O).
triple(O,rdf:type,rdfs:Resource) :- triple(S,P,O).
triple(S,rdf:type,C) :- triple(S,P,O), triple(P,rdfs:domain,C).
triple(O,rdf:type,C) :- triple(S,P,O), triple(P,rdfs:range,C).
```

OWL property axioms as RDF Triples	DL syntax	FOL short representation
$\langle P$ rdfs:domain $C\rangle$	$\top \sqsubseteq \forall P^-.C$	$\forall x,y : P(x,y) \supset C(x)$
$\langle P$ rdfs:range $C\rangle$	$\top \sqsubseteq \forall P.C$	$\forall x,y : P(x,y) \supset C(y)$
$\langle P$ owl:inverseOf $P_0\rangle$	$P \equiv P_0^-$	$\forall x,y : P(x,y) \equiv P_0(y,x)$
$\langle P$ rdf:type owl:SymmetricProperty $\rangle$	$P \equiv P^-$	$\forall x,y : P(x,y) \equiv P(y,x)$
$\langle P$ rdf:type owl:FunctionalProperty $\rangle$	$\top \sqsubseteq \leqslant 1P$	$\forall x,y_1,y_2 : P(x,y_1) \wedge P(x,y_2) \supset y_1 = y_2$
$\langle P$ rdf:type owl:InverseFunctionalProperty $\rangle$	$\top \sqsubseteq \leqslant 1P^-$	$\forall x_1,x_2,y : P(x_1,y) \wedge P(x_2,y) \supset x_1 = x_2$
$\langle P$ rdf:type owl:TransitiveProperty $\rangle$	$P^+ \sqsubseteq P$	$\forall x,y,z : P(x,y) \wedge P(y,z) \supset P(x,z)$

Table 4.1: Expressing OWL DL Property axioms to DL and FOL

```
triple(C,rdfs:subClassOf,rdfs:Resource) :- triple(C,rdf:type,rdfs:Class).
triple(C1,rdfs:subClassOf,C3) :- triple(C1,rdfs:subClassOf,C2),
                                 triple(C2,rdfs:subClassOf,C3).
triple(S,rdf:type,C2)         :- triple(S,rdf:type,C1),
                                 triple(C1,rdfs:subClassOf,C2).
triple(C,rdf:type,rdfs:Class) :- triple(S,rdf:type,C).
triple(C,rdfs:subClassOf,C)   :- triple(C,rdf:type,rdfs:Class).
triple(P1,rdfs:subPropertyOf,P3) :- triple(P1,rdfs:subPropertyOf,P2),
                                    triple(P2,rdfs:subPropertyOf,P3).
triple(S,P2,O)                   :- triple(S,P1,O),
                                    triple(P1,rdfs:subPropertyOf,P2).
triple(P,rdfs:subPropertyOf,P)   :- triple(P,rdf:type,rdf:Property).
```

However, Ter Horst and Draltan have proven that these rules do not fully cover RDFS semantics[19, 43] and have provided alternative formalizations.

Descriptive Logic Programming (DLP) is defined [27] as an intersection of the Description Logic and Logic Programming subsets of First Order Logics. It is layered onto a subset of RDFS which removes support for the RDFS recursive metamodel (i.e. the unrestricted use of the type relation). A technique called DLP-fusion is introduced to allow for mapping of queries from the DLP fragment of Logic Programming to Description Logics and vice versa.

4.2.2 OWL

As mentioned above, OWL Lite and OWL DL can be viewed a subsets of first order logic as shown in Tables 4.1 and 4.2. However, they are not built on top of RDFS. OWL DLP (see DLP discussion in previous subsection) is the most expressive sub-language of OWL DL that can be efficiently mapped to Datalog. OWL DLP is simpler than OWL Lite and easier to align to the semantics of RDFS. Yet alignment between OWL DLP and RDF(S) can only be achieved through the enforcement of additional modelling constraints and transformations.

OWL Flight [15] is an OWL-based language with its semantics based on Logic Programming rather than Description Logics. It is restricted to the Datalog fragment of FOL allowing query answering to be performed by a Logic Programming implementation.

OWL complex class descriptions*	DL syntax	FOL short representation
owl:Thing	$\top$	$x = x$
owl:Nothing	$\bot$	$\neg x = x$
owl:intersectionOf $(C_1 \ \ldots \ C_n)$	$C_1 \sqcap \ldots \sqcap C_n$	$\bigwedge C_i(x)$
owl:unionOf $(C_1 \ \ldots \ C_n)$	$C_1 \sqcup \ldots \sqcup C_n$	$\bigvee C_i(x)$
owl:complementOf (C)	$\neg C$	$\neg C(x)$
owl:oneOf $(o_1 \ \ldots o_n)$	$\{o_1 \ldots o_n\}$	$\bigvee x = o_i$
owl:restriction $(P$ owl:someValuesFrom $(C))$	$\exists P.C$	$\exists y.P(x,y) \wedge C(y)$
owl:restriction $(P$ owl:allValuesFrom $(C))$	$\forall P.C$	$\forall y.P(x,y) \supset C(y)$
owl:restriction $(P$ owl:value $(o))$	$\exists P.\{o\}$	$P(x,o)$
owl:restriction $(P$ owl:minCardinality $(n))$	$\geqslant nP$	$\exists^n_{i=1} y_i . \bigwedge^n_{j=1} P(x,y_j) \wedge \bigwedge_{i \neq j} y_i \neq y_j$
owl:restriction $(P$ owl:maxCardinality $(n))$	$\leqslant nP$	$\forall^{n+1}_{i=1} y_i . (\bigwedge^n_{j=1} P(x,y_i) \supset \bigvee_{i \neq j} y_i = y_j)$

*For reasons of legibility, we use a variant of the OWL abstract syntax [38] in this table.

Table 4.2: Mapping of OWL DL Complex Class Descriptions to DL and FOL

4.2.3 Semantic Web Rule Languages

This section considers some current rule languages for the Semantic Web [9].

SWRL is a W3C member submission from Ian Horrocks, Peter Patel Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean [10].

SWRL and its FOL extension SWRL-FOL [31] are extensions of OWL by Horn Rules in the direction of extending OWL DL towards full first-order logic, rather than rule languages in the logic programming style. While a subset of SWRL falls inside Horn Logic, a SWRL knowledge base easily goes beyond this fragment, because of the use of classical negation, existentially quantified variables, and disjunction in the head of the rule. A set of Horn Logic formulae can be reduced to standard Logic Programming rules; the Horn Logic formulae and the Logic Programming rules entail exactly the same set of ground formulae. Consequently, SWRL and standard rule languages differ in expressiveness. The advantage of common rule languages which are based on Horn Logic is the efficient reasoning support which has been developed for certain reasoning tasks like query answering.

Still, SWRL syntax has become stablished especially in the OWL community as a common notation to denote conjunctive queries over OWL.

The Web Rule Language (WRL) [2] is a W3C member submission from DERI. WRL layers on a significant part of RDFS. WRL does not allow the use of language constructs in the language itself and does not allow full treatment of blank nodes, because this would require reasoning with existential information, which is typically not possible in the context of standard rule languages. WRL provides a significant extension of RDFS through the possibility of specifying local attributes, range and cardinality constraints for attributes and attribute features such as symmetry, transitivity and reflexivity. Furthermore, WRL provides an expressive rule language which can be used for the manipulation of RDF data.

The W3C is currently working on a Rule Interchange Format (RIF) [11]. While, as the name suggests, the idea is not to specify a rule language for the Semantic Web but rather a normalized means for the exchange of rules in the Semantic Web, the working group will be defining a RIF Core which seeks to establish the common logic subset for Semantic Web rules languages so that a degree of interoperability can be

[9] Some text sourced from `http://www.w3.org/Submission/WRL-related/`

[10] `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`

[11] `http://www.w3.org/2005/rules/wg`

guaranteed across Semantic Web rule systems. This core can be extended in specific directions, e.g. as a Horn Rule Language.

4.2.4 Semantic Web Rule Engines

TRIPLE [42] is a technology developed by Michael Sintek, Stefan Decker, and Andreas Harth. An engine for a rule language supporting Horn rules with slotted syntax, RDF processing, but allowing external calls to OWL reasoners. The system is based on the XSB Prolog system, which supports well-founded negation and thus set difference, but this is to our knowledge not implemented in TRIPLE.

Notation3 (N3) is a technology developed by Tim Berners-Lee and Dan Connolly. It is a native extension of RDF's TURTLE syntax by rules including negation as failure (`n3:notIncludes`(. The semantics of N3 is defined purely by its implementation of the moment (cwm[12]) and not formally defined. It has non-termination problems when confronted with non-stratified negation. Authors' assumption is that cwm works fine for computing perfect models.

FLORA-2 [13] is a rules engine based on a dialect of F-Logic which has been adapted to support F-Logic-based Semantic Web languages such as Web Service Modeling Language (WSML) and OWL. For example, F-OWL [48] is an inference engine for OWL implemented using XSB and Flora-2.

OWL-IM [35] is a Storage and Inference Layer for the Sesame RDF repository that can perform OWL DLP reasoning.

dlvhex [21] is a solver for non-monotonic logic programs which, given the possibility of exchanging knowledge with external sources in a fully declarative paradigm such as answer-set programming (ASP), can be used to deal with external knowledge and reasoners, such as RDF datasets or description-logics knowledge bases. This is achieved through *dl-programs*, which are DL atoms (the knowledge base) plus rules similar to those in logic programming. SPARQL mappings to Datalog with stratified negation as failure allow SPARQL queries (with extensions beyond SPARQL) to be executed on top of dlvhex [39].

4.3 Consequences for TripCom

This chapter has provided an introduction to ontologies and rule languages. In addition, it has provided a description of first-order logic and other logic paradigms such as description logic and logic programming as ways to manage ontological primitives for modelling a data domain.

These technologies are relevant for TripCom because they enable an adaptive approach to managing domain specific data. This means future users of TripCom project results need not develop procedural code to model their data. Instead, TripCom project results can adapt to declarative descriptions of the domain data and make sure the relationships and constraints expressed in those descriptions which are based on ontology languages are respected in query evaluation.

[12] http://www.w3.org/2000/10/swap/doc/cwm.html
[13] `http://flora.sourceforge.net/`

5 Triple Space specific requirements for semantic querying

In the following, we give a "global" requirements list [1] which we should keep in mind concerning Triple Space querying.

1. Query evaluation should extract data sets that are not necessarily complete but **"good enough"**, at minimum correct (according to the semantics). In an open environment such as the Web or the Triple Space, we cannot strive for complete answers (see also [40]). Also, incompleteness of information is inherent with blank nodes in the RDF datamodel. This implies the following sub requirements:

   - Restrict the query language and its semantics in a way which well controls the scope and complexity of queries to remain "manageable".
   - Accept incompleteness of results, as long as they are correct.
   - Negation must be scoped as otherwise results may become unsound, see [40].
   - Aggregates (such as count, sum, avg, etc.) must be scoped otherwise results may become unsound.
   - Scoping of the queries must be possible.
   - A mechanism to name graphs and contexts, see e.g. [6]

2. **Query evaluation time** should be optimized. This includes:

   - The time required for the query plan generation
   - The time of query plan evaluation
   - The time of data transmission

3. **Integration of rule and ontology languages**, especially: WSML and WRL, is needed. Triple Space needs rule languages and ontologies together with at least a minimum of derivation/reasoning within the Triple Space for several reasons:

   - Mapping rules between different vocabularies, discoverable and evaluable within the triple space are needed. Otherwise, each party which poses a query, needs to know the particular RDF vocabulary, or mappings at the time of posing a query. As TripCom intends to allow different and overlappng vocabularies, it should not leave all the burden of mediation to the requester. To push simple mediation rules within the middleware layer needs rules in the middleware layer.
   - Rules and ontologies allow description of intensional knowledge, "views", and allow linking between different sources of knowledge. For instance, the rules in one space might refer to data/queries to be evaluated in another space, instead of duplicating/materializing the data of the referred space. Materialization of views is brittle for reasons of inconsistencies and duplication.

[1] These requirements are partly derived from deliverable S7.3 of the Infomix Project (`http://www.mat.unical.it/infomix`).

The **tradeoff between expressivity and efficiency for evaluation** of rules and queries is an issue for Triple Space.

4. The issue of **data manipulation vs. pure query language** must be addressed: what requirements for the query language come from interaction patterns, and are these sufficiently covered by the Triple Space API?

5. Connection to "pure" web services: As defined in the Annex, the Triple Space is a technology *complementary* to pure web services. Should the Triple Space allow calls to external encapsulated Web services within rule bodies and queries?

6 Sample queries from WP4 and their requirements

In WP4 a Semantic Web services scenario is being realized in which individual and composite WSMO Web services may interact with one another in a coordinated fashion through Triple Space-enabled WSMX platforms. These Web services are described in the Web Service Modeling Language (WSML). To enable interation between service requester and service provider, service descriptions need to be published in a service registry. Service requesters can issue queries on the registry to discover services fullfilling their requirements. In a following step, the service requester can interact with the service.

This chapter lists the requirements for service discovery for the Web services that are described using Web Service Modeling Language (WSML).

6.1 Sample query set

In WSML, the Web service descriptions are composed of three major sections called functional descriptions, non-functional descriptions and behavioral descriptions. The user has to specify its requirements in a WSML Goal in a way similar to WSML Web services descriptions. The discovery process is based on matchmaking of Goal provided by user with the Web service description. Discovery requires different efforts in annotation and description of goals versus services and may deliver results of different accuracy. There are different levels of discovery based on the type of formalism used for matchmaking.

To illustrate this approach consider a few examples from the traveling domain: some client goals and Web service capabilities are modelled which relate to information about flights and train connections. The following informal sample goals are defined. Assuming the requester wants to find adequate Web services, these goals need to be matched with the available Web service descriptions:

- G1: *I want to know about all flights from any place in Ireland to any place in Austria.*
- G2: *I want to know about some flights from any place in Ireland to any place in Ireland which is different from the starting location.*
- G3: *I want to know about all flights from Galway (Ireland) to Dublin (Ireland).*
- G4: *I want to know about some flights from Innsbruck (Austria) to some place in Ireland (the client does not necessarily care which one).*
- G5: *I want to know about some train connections from Galway (Ireland) to Dublin (Ireland).*

For the formalization of a second example, we assume that an appropriate set of ontologies O for geographical data and traveling are in place. A relation `in(x; y)` which states that object `x` is geographically located in object `y`, a relation `flight(f; s; e)` which states that `f` is a flight from location `s` to location `e` and a relation `train(t; s; e)` which states that `t` is a train connection from location `s` to location `e`. Four available Web services W1; : : : ;W4 expose the following informal capabilities:

- W1 offers *information about all flights for any place in Europe to any place in Europe.*
- W2 offers *information about flights from all places in Ireland to all other places in Ireland, but not necessarily information about* all *such flights.*
- W3 offers *information about all flights from all places in Ireland to Innsbruck (Austria).*
- W4 offers *information about all train connections from Galway (Ireland) to Dublin (Ireland).*

Discovery process and its requirements:

Keyword-based discovery is a basic ingredient in a complete discovery process for Semantic Web services. It is matching the non-functional properties specified in Goal and Web services. Performing a simple search at the first step filters a huge number of available services quickly. Keyword-based discovery is considered as a useful technique in the complete Semantic Web service discovery process. Below is a sample Goal and Web service description. The non-functional properties are matched by the discovery engine.

Sample WSML Goal description:

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {_"http://www.gsmo.org/dip/travel/goal#",
dO _"http://www.gsmo.org/dip/travel/domainOntology#",
dc _"http://purl.org/dc/elements/1.1#"}

/*
 * Test Goal
 */
goal _"http://www.gsmo.org/dip/travel/goal.wsml"
nfp
dc#title hasValue "Goal"
dc#contributor hasValue "ABC User"
dc#description hasValue "Express the goal of buying
a plane ticket from Austria to UK"
endnfp
importsOntology _"http://www.gsmo.org/dip/travel/domainOntology.wsml"
capability goalCapability
postcondition
definedBy
?ticket[
dO#from hasValue ?from,
dO#to hasValue ?to,
dO#vehicle hasValue ?vehicle
] memberOf dO#Ticket and
?from memberOf dO#AustrianCity and
```

```
?to memberOf dO#IrishCity and
?vehicle memberOf dO#Airplane.
```

Sample WSML Web service description:

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {_"http://www.gsmo.org/dip/travel/webServiceA#",
  dO _"http://www.gsmo.org/dip/travel/domainOntology#",
dc _"http://purl.org/dc/elements/1.1#"}

/*
 * Web service A
 */
webService _"http://www.gsmo.org/dip/travel/webServiceA.wsml"
nfp
dc#title hasValue "First Web service"
dc#contributor hasValue "XYZ Travel Agency"
dc#description hasValue "Web service for booking plane tickets
  from Austria and Germany to any european city"
endnfp
importsOntology _"http://www.gsmo.org/dip/travel/domainOntology.wsml"
capability webServiceACapability
postcondition
definedBy
?ticket[
dO#from hasValue ?from,
dO#to hasValue ?to,
dO#vehicle hasValue ?vehicle
] memberOf dO#Ticket and
(?from memberOf dO#AustrianCity or ?from memberOf dO#GermanCity) and
?to memberOf dO#EUCity and
?vehicle memberOf dO#Airplane.
```

The next level of discovery requires information in an explicit formalism. One main characteristic of this approach is that the problem domain (or the universe of discourse) is understood as a set of objects that can be grouped together into sets (or classes). Each class captures common (syntactic and semantic) features of their elements. Features can be inherited between classes by defining class hierarchies. This way, a problem domain can be structured as classes of objects and is basically understood as a collection of classes (or sets of things). The description of a set of objects for representing a goal or a Web service can be interpreted in different ways and thus the description by means of a set is not semantically unique. A modeler might want to express that either all of the elements that are contained in the set are requested (in case of a goal description) or can be delivered (in case of a Web service description), or that only some of these elements are requested (or can be delivered). The modeler

has specific domain in mind when specifying such a set of relevant objects for a goal or Web service description and this intention determines whether two descriptions should be considered to match or not. Thus, this domain should be stated explicitly in the descriptions of service requests or service advertisements.

Lightweight discovery, which is the next level in using formalisms in the process of semantic based discovery, uses logical expressions and background knowledge formulated using WSML Flight and WSML Rule. This component can use a Description Logics (DL) reasoner to detect intersection and plug-in matches for a limited set of conjunctive queries. Lightweight discovery uses the ontologies and capability descriptions in WSML descriptions of Goal and Web services and matches them based on Description Logics. A sample (collected from SWS Challenge [3] scenario) Goal and Web service description based on WSML that contains the functional descriptions (in this case, capability description of Web service) is presented below. The functional descriptions of the Web service and Goal are used in this case to find match based on the principles of Description Logic.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {_"http://www.gsmo.org/dip/Goal1b#",
  so _"http://www.gsmo.org/dip/ShipmentOntology#",
dc _"http://purl.org/dc/elements/1.1#"}

/*
 * Goal 3b - test goal for discovery based on destination
 */
goal _"http://www.gsmo.org/dip/Goal3b.wsml"
nfp
dc#title hasValue "Goal 3b"
dc#description hasValue "Goal of shipping a package to New York"
dc#contributor hasValue "TripCom"
endnfp

importsOntology _"http://www.gsmo.org/dip/ShipmentOntology.wsml"

capability goal3bCapability
postcondition
definedBy
?order[so#to hasValue ?to] memberOf so#OrderRequest
and
?to[so#name hasValue "Barney Gumble",
so#address hasValue ?address] memberOf so#ContactInfo
and
?address[so#city hasValue so#NY].
```

Sample WSML Web service description:

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace {_"http://www.gsmo.org/dip/WSRacer#",
  so _"http://www.gsmo.org/dip/ShipmentOntology#",
dc _"http://purl.org/dc/elements/1.1#"}

/*
 * Racer
 */
webService _"http://www.gsmo.org/dip/WSCourier.wsml"
nfp
dc#title hasValue "Courier Web service"
dc#description hasValue "We are shipping packages only to addresses in Asia."
dc#contributor hasValue "TripCom"
endnfp

importsOntology _"http://www.gsmo.org/dip/ShipmentOntology.wsml"

capability racerOrderSystemCapability
postcondition
definedBy
?order[so#to hasValue ?contact] memberOf so#OrderRequest
and
?contact[so#address hasValue ?address] memberOf so#ContactInfo
and
so#addressIsOnContinent(?address, so#Asia).
```

6.2 Requirements extraction

In the discovery process, Triple Space is queried to receive either a single, some or all matches on a simple template as a form of filtering a large set of candidate Semantic Web services. The query returns a set of matches with a ranking of descriptions based on axioms which are expressed e.g. in WSML Flight using F-Logic. This is performed in the WSMX system.

A problem is that the axioms governing input, output, preconditions and effects (IOPE) of Web services or Goals are expressed in WSML using an extended version of the Datalog subset of F-Logic (WSML-Flight), and hence are unavailable to an RDF-based reasoner in Triple Space.

In an RDF-based Triple Space it will be necessarily to separate the Triple Space processing of Web service discovery (based on RDF querying) and the Web service layer processing (with access to WSML querying). It may be that this division proves sufficient: WSML can be serialized as RDF and stored in Triple Space, and full descriptions, encapsulated as Named Graphs, are accessed by the graph read operation of the TS API. They can be processed as WSML graphs, using WSML axioms in extended logical models such as F-Logic, in the Web services layer e.g. in WSMX.

Initial querying is used to extract graphs of Web service descriptions. These query results can be further narrowed by matching on Web service name, imported ontologies,

and non-functional properties (e.g. Dublin Core), hence requirements on a query language are restricted to:

- named graph support (WSMO Web services or Goals accessible as named graphs)
- Literal matching, which can include strings (plus regular expressions), integer and date-time.
- URI equivalence based on normalization of URI strings

WSML is based - in its Flight and Rule sub-languages - on Logic Programming, building on Datalog with LP-fragments of F-Logic, inequality and default negation. In proposing a query language for Triple Space with an extension into the rules space, we could narrow this division and begin to support some WSML-based reasoning in Triple Space querying. However, WSML has been taken in both extended DL (WSML-DL) and LP directions, meeting only at WSML-Full, while taking DLP as the common subset of DL and LP may provide too little. WSML-Flight queries (Datalog extended with inequality and (locally) stratified negation) may allow extended querying of and reasoning with WSML axioms in Triple Space.

Extended queries require these query features:

- disjunction (OR)
- (set) inequality ("different from") / negation ("not" UK City)
- support for RDF containers/collections
- sorting of / limits on result sets (e.g. flights ordered by price / the five cheapest flights)
- grouping on other properties (e.g. average price per airline)
- aggregrate functions (e.g. number of available flights/average price of flights)
- nested queries
- recursive queries

7 Sample queries from WP8A and their requirements

In WP8A an Enterprise Application Integration (EAI) scenario is being realized in which content and Service Providers negotiate the provision of media content to assemble content services which will be offered to final customers. This negotiation process will be performed using a marketplace business model, through the use of auctions to get the best content offer at the moment the content service is being assembled.

7.1 Sample query set

From the WP8A use case perspective, data will be considered in a two level approach. Content services will be stored in the TS and a set of components should be supplied in order to exploit the service:

- The contract between the Service Providers and Content Providers which rule the provision of content.
- The set of users that are subscribed to this service and therefore use it.
- The evaluation of the satisfaction of the users and an value which marks the limit in which negative evaluations from users become disadvantageous for Service Provider interests.

In order to obtain these components, a Service Provider will negotiate with different Content Providers. For negotiation purposes and for a better demonstration of TripCom benefits in this context, information regarding content offers by different Content Providers will be stored in the Triple Space in the context of an active auction.

The EAI use case information can be divided in the following groups of data from a conceptual perspective:

- The *actor's definition* data, where all information about Service Providers, Content Providers and users who interact in this collaborative scenario is stored.
- The *content catalogue* deals with information related to the content offered by each Content Provider
- The *auction register*, which will store the temporal data that must be managed when an auction is created and has to be managed (all active bids) until the temporal deadline specified is reached.
- The *service register*, where all services are stored once an auction process has finished and a service has been assembled. All signed contracts as well as service evaluations should be part of this register.

7.1.1 Activities of the service register

The Digital Asset Management (DAM) use case will need to get information from the service register in order to:

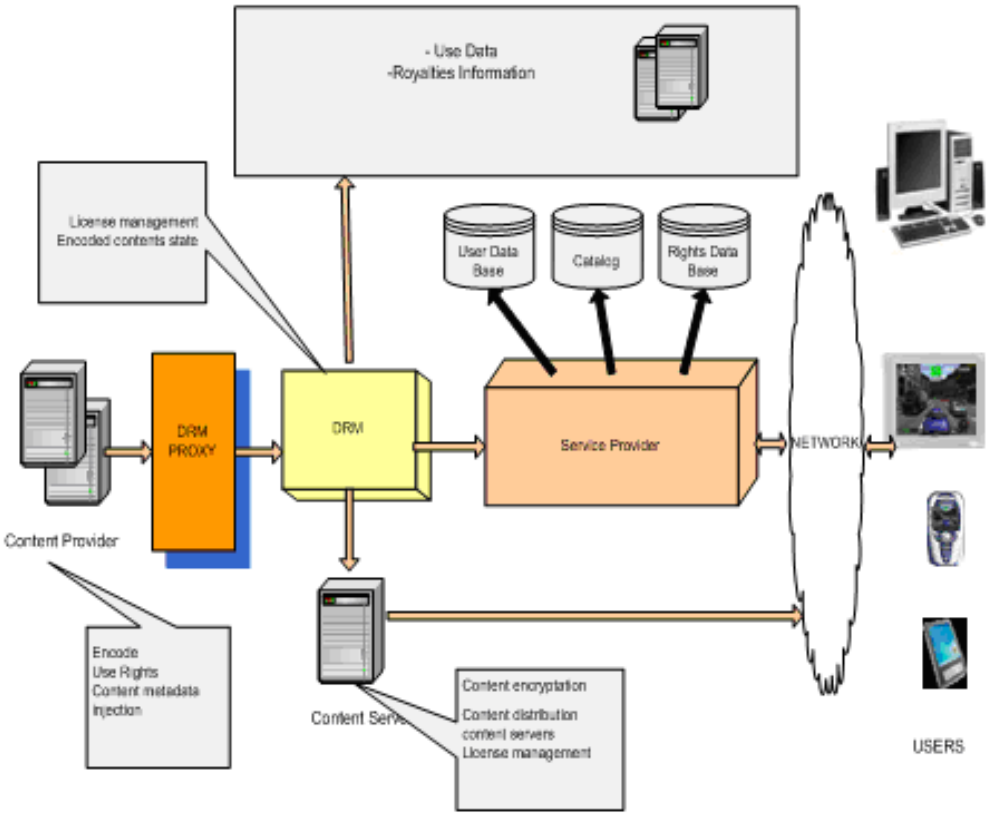

Figure 7.1: The Enterprise Application Integration (EAI) scenario

- Add new content services as they get assembled.
- Consult and modify existing parameters of the services stored in the register. These modifications can come from the Service Provider which holds the content service or from any Content Provider which is involved in the service being part of any contract attached to the service.
- Delete content services that are no longer available.

7.1.2 Information and queries needed by Service Provider

- A Service Provider needs to retrieve descriptions of available services provided by other Service Providers which match some technical details.
- A Service Provider needs to retrieve the status of any content service, even if it not provided by this Service Provider instance. The service being asked for can be explicitly defined (using the service identification or name) or implicitly defined, using the approach defined for previous topic.
- A Service Provider needs to retrieve any information regarding a contract subscribed with a content provider to offer a content service.
- A Service Provider needs to retrieve all bids related to an ended auction it started in order to get the best bid.

- The system needs to publish all relevant information of a designed service on behalf of the Service Provider who is offering this service once the design phase of the content service has finished.
- A Service Provider needs to be able to modify features of any content service it provided for business reasons, e.g. as a result of negative feedback of customers about the whole service or parts of this service.
- A Service Provider needs to be able to halt its provided services in order to stop the availability of those services.

7.1.3 Information and queries needed by Content Providers

- Content Providers for a Service Provider may need to retrieve existing available services provided by Service Providers which match some technical details. This query is for information purposes, so not critical to the system.
- Content Providers of a service need to retrieve any information regarding a contract it has with a service provider.
- A Content Provider needs to publish bids and modify them in the Triple Space in order to hire a service or contents, in the context on an active auction.
- A Content Provider needs to retrieve information from all active bids related to an active auction in which it is participating.
- A Content Provider needs to modify parameters of a contract related to some contents of services or contents hired when the conditions of these change.
- A Content Provider needs to cancel a contract if there is a business reason for doing so, and the terms of the contract permit it, e.g. previous conditions imply the unavailability of some of the contents being hired by a service provider.

7.1.4 Queries made over the content catalogue

The Content Provider will need to get information from the content catalogue in order to consult and modify existing content offers from a service provider which has changed some features of the content offered or has added contents to its existing offer.

7.1.5 Queries regarding content catalogue hierarchy

A Service Provider needs to retrieve existing available contents in order to evaluate which existing content offers fit a content request pattern published by a Service Provider. These queries about available contents can be made using:

- The media type a Service Provider is searching for.
- Area of the content being offered.
- What the content contains.

A Content Provider needs to modify information about contents it offers.

7.2 Requirements extraction

7.2.1 Service Provider query features

Search by content features ("match some technical details about contents offered, access policies, and distribution channels"). Check for existence by content features ("is there content matching my requirements available").

7.2.2 Service Register query features

- Projection: A service provider or a system supervisor needs to know the available content services registered in the server.
- Constraints on the structure: A service provider needs to request existing contents and their features, for example offering F1 drivers' interviews which includes images and audio contents.
- Constraints on values: A service provider needs to know existing contents whose contents offered, for example are no longer than 3 minutes in order to provide a highlights news service.
- Boolean queries: has this service provider X any currently active services?.
- Aggregate functions: Need to know the number of currently active services offering soccer contents from the Spanish League.

7.2.3 Content Catalogue query features

- Projection: The system needs to know about available contents at this moment.
- Constraints on the structure: The system needs to know content offers for soccer summary of past Bundesliga fixtures.
- Constraints on values: The system needs to know content which is suitable for a PDA offer. (resolution no larger than 640x480 and size no larger than 40 Megabytes)
- Boolean queries: Is there any offer of tennis images?.
- Aggregate functions: Need to know the overall length of a content package for a soccer match, with highlights and some interviews with most representative players.

7.2.4 Summary

The WP8a scenario raises these key requirements:

- Datatype support (string, integer, datetype) and comparison operators (equals, lessThan, moreThan, ...)
- Ranking of results (ordering by datatype; ordering by other factors is based on ranking algorithms outside of the querying process)

- Aggregration functions
- Boolean results

It also considers the following features as supportive:

- Filtering (e.g. exclude contents matching a certain graph pattern)
- Negation and disjunction (e.g. give me any ball sport that is Finnish OR is NOT baseball)

8 Sample queries from WP8B and their requirements

WP8B describes the European Patient Summary (EPS) scenario, which is an implementation of an infrastructure for patient summaries at the European level. In this scenario, TripCom is used as a communication infrastructure and as distributed storage capable of managing the most critical health data of each citizen in Europe.

The EPS scenario aims to provide a first step toward a network of complementary health systems. Since the scenario does not force existing systems to be updated or replaced, semantic interoperability techniques will be a key requirement for dealing with heterogeneity of existing eHealth systems and standards. Security policies will be also implemented in order to guarantee that only authorized care givers will have access to the citizens' data.

8.1 Sample query set

This sample query set is produced by WP8B. It is determined to be an expression of the range of sample queries that will be made by the EPS infrastructure on the health data stored in the Triple Space. Many queries have been pseudo-formalized in SPARQL (the RDF schema is still subject to change) with notes regarding missing or questioned expressiveness.

- Is <CitizenID> managed by the EPS infrastructure?

```
PREFIX eps:  <http://www.tripcom.org/eps#>
ASK  { ?x a eps:EPS;
   eps:head [eps:patientIdentifier <CitizenID> ] . }
```

  This is a boolean query that looks into the whole space in order to find data related to such citizen and returns true or false.

- How many persons suffered of heart attack in <LocationID> (with <LocationID> ranging over countries, regions or local administrations) during the years from <StartYear> to <EndYear> inclusive?

```
PREFIX eps:  <http://www.tripcom.org/eps#
PREFIX ts:  <http://www.tripcom.org/ontologies/tsonto#
PREFIX dc:  <http://purl.org/dc/elements/1.1/
COUNT DISTINCT  ?person
WHERE   { ?x a eps:EPS;
   eps:head [eps:patientIdentifier ?person;
   eps:patientContactInformation [eps:country <RegionID> ]];
   eps:body [eps:healthRecord [eps:suffering <HeartAttack> ]];
   ts:hasAccessLogEntry [dc:data ?when] .
FILTER (?when >= <StartYear> AND ?when <= <EndYear>) . }
```

This query requires the possibility to execute aggregate functions on results. SPARQL does not have a support for an aggregate query function such as `count`.

This query requires also to get all the data of a citizen that lives in the region <LocationID>; that is people that lives in all the cities of that location. There will be an ontology that describes the geography of Europe and the query will need to use transitive closure to obtain all the subregions given a region ID.

This requires the expression of a rule (not possible in Description Logic) to bound the set of concepts in the country <LocationID> to the concepts in <LocationID>, i.e. country(PatientID,LocationID) ¡- city(PatientID,CityID),part_of(CityID,LocationID).

This query requires support for datatypes, since date and times are based on xsd:date, xsd:time and xsd:datetime.

This query requires for greaterThan and lessThan and equal operators on the datatypes.

- How many persons suffered before <age> from <SetOfDiseaseIDs> and how many of them died at age between <StartAge> to <EndAge>?

  In order to define the age of the citizens, this query requires the possibility to perform type casting and arithmetic operations with some data types (such as 2007 minus yearOfBirth using the xsd:integer and xd:dateTime datatypes).

  In addition to the previous query, this query requires to express set of reference to instances (SetOfDiseaseIDs). This is semantically the same as a set of disjunctive graph patterns, which is syntactically longer as supporting sets in queries.

- Who are the citizens currently managed by <DoctorID> ?

  This query does not raise any new requirements, though it may depend on how the query shall determine who is "currently" managed by a Doctor. If triples are not deleted, but rather invalidated, a scoped negation is necessary to say e.g. 'citizens managed by <DoctorID> who do NOT have an managedUntilDate property' where the 'managedUntilDate' property acts as a form of temporal invalidation of the managing property between a patient and a doctor.

- Was <CitizenID> cared by <DoctorID> and when?

  This query does not raise any new requirements, if we consider the above approach we query for a relationship of 'cared_by' between a doctor and a citizen and return the begin and end dates of that relationship. If the triple is reified to allow the attachment of this temporal information, the query language must support access to reified statements. Alternatively, quads might be used to attach provenance information in named graphs, but Triple Space specifies tuples as being three fielded, so to the client, it is still necessary to provide access either through reification or a rdfs:seeAlso type propery which points to named graphs.

- Which EPS users accessed the record of <CitizenID> specifying also the datetime of the access, the operation executed and the data accessed.

```
PREFIX eps:  <http://www.tripcom.org/eps#
```

```
PREFIX ts:  <http://www.tripcom.org/ontologies/tsonto#
PREFIX dc:  <http://purl.org/dc/elements/1.1/
SELECT  ?who, ?when ?accessOp ?rdfTree
WHERE   { ?x a eps:EPS;
      eps:head [eps:patientIdentifier <CitizenID> ];
            ts:hasAccessLogEntry [dc:publisher ?who;
                                  ts:accessDate ?when;
                                  ts:accessOperation ?accessOp
                                  ts:accessedData ?rdfTree ].}
```

It should be possible to obtain the result where a single record could contain the following data:

Dr. Smith from Milan e-Health authority - 10th March 2006, 14:34 - Insert - and then the extract of the data inserted in the form of an RDF extract or equivalent:

```
<eps:healthRecord>
      <eps:Medication>
          <eps:dose>
              <eps:Measurement>
                  <eps:value>0.43</eps:value>
                  <eps:unit rdf:resource="&cyc;Grams"/>
              </eps:Measurement>
          </eps:dose>
          <eps:frequency>
              <eps:Measurement>
                  <eps:value>2</eps:value>
                  <eps:unit rdf:resource="&cyc;Monthly"/>
              </eps:Measurement>
          </eps:frequency>
          <eps:product rdf:resource="umls:C0000435"/>
          <eps:publication>2002-05-30T09:30:10.5</eps:publication>
      </eps:Medication>
  </eps:healthRecord>
```

In the TS ontology, AccessLogEntry refers to data where data is a Triple or Graph; the reference to the Graph must be automatically resolved to the Graph itself. In a query, this requires named graph support.

- What are the allergies suffered by <CitizenID> ?

```
PREFIX eps:  <http://www.tripcom.org/eps#
SELECT  ?allergy
WHERE   { ?x a eps:EPS;
       eps:head [eps:patientIdentifier <CitizenID> ];
            eps:body [eps:healthRecord ?r1, ?r2] .
```

```
?r2 a eps:Allergy;
       eps:substance [eps:codeValue ?allergy] .}
```

This query does not raise any new requirements, though a possible direction of the EPS schema could be to model such natural 'sets' of information in RDF containers. Then any query language used to extract this information needs to support RDF collections/containers.

- What are the medication administered to <CitizenID> in the last <Hours> ?

  Here we pass as input the current date-time and, with datatype support, we can get the correct results e.g. by calculating the date-time value a number of hours before the current date-time and applying a filter to results with that date-time range.

- Does <CitizenID> have any contraindications for <DrugID>?

  Here we give a natural language version of the question. The requester's software should convert this into a query for:

```
?X such that
   ?X is a contraindication for <DrugID> AND
   ((?X is a Drug AND
     <CitizenID> takes ?X)
    OR
    (?X is a MedicalCondition AND
     <CitizenID> suffersFrom ?X))
}
```

  This query raises the requirement to support disjunctive (OR) queries.

8.2 Requirements extraction

Requirements identified through the provided sample queries are:

- disjunction
- datatype support (e.g. date-time, integer, boolean) with arithmetic and logical operators (+ - * / < > = != <= >= AND OR NOT)
- aggregation with aggregate functions (count, max, min, average)
- boolean queries
- set of terms or instances (note: syntactic sugar as one can list each term or instance in a disjunctive query)
- reification (note: syntactic sugar as one can access reified statements over the RDF vocabulary)

- transitive closure
- calculation of transitive relationships
- support for RDF containers/collections

9 Candidates for Triple Space querying

The purpose of this document has been geared towards determining candidates for Triple Space querying. Preceding chapters have presented various query language features and RDF query languages and elaborated on the added value of rules technologies in query evaluation.

The following list is a union of all requirements for query languages as identified by the use cases earlier in this document.

- named graph support (WSMO Web services or Goals accessible as named graphs)
- datatype support (string, integer, datetype) and comparison operators (equals, lessThan, moreThan, ...)
- URI equivalence based on normalization of URI strings
- disjunction (OR)
- (set) inequality ("different from") / negation ("not" UK City)
- support for RDF containers/collections
- sorting of / limits on result sets (e.g. flights ordered by price / the five cheapest flights)
- grouping on other properties (e.g. average price per airline)
- aggregrate functions (e.g. number of available flights/average price of flights)
- nested queries
- recursive queries
- reification
- calculation of transitive relationships

Table 3.1 in Section 3.2 presented the features of some of the existing RDF query languages. Comparing that table with the above requirements gives us grounds to recommend SPARQL as a query language to be used in the TripCom project. SPARQL has the best fit to Triple Space querying in terms of feature coverage and life expectancy as it is maturing to an open standard in a W3C working group.

The development of the SPARQL language started with somewhat similar requirements as Triple Space querying. SPARQL early requirements [9] included the following

- conjunction
- variable binding results
- extensible value testing
- subgraph results
- local queries

- optional match
- limited datatype support
- result limits
- streaming results
- disjunction
- WSDL protocol

This list already covers most of the requirements identified by the use cases of the TripCom project. However, the following requirements need to be addressed outside the scope of the original SPARQL requirements.

- aggregation functions (count, min, max, average)
- rule language support
- syntactic support for primitives of RDF data model reification
- support for RDF collections/containers
- calculation of transitive relationships

We also recommend use of CONSTRUCT query type with SPARQL queries in the Triple Space querying context as the query results can then be used for further RDF processing.

We also recommend extending the SPARQL query protocol to support specification of a time parameter specifying how long results should be waited for from a Triple Space before returning them to the query client.

In Work Package 3's next task in the TripCom project we will evaluate the use of SPARQL as a query language in the Triple Space prototype to realise the given scenarios by providing the identified features, with a focus on how restricting the query language in some cases may aid efficiency of querying (given the global aspect of Triple Space) and how extending the query language may be possible to include the additional requirements identified here. Given the involvement of TripCom partners in the W3C RDF Data Access Working Group (DAWG), we expect this further work to be able to provide input to and receive input from the SPARQL standardization process.

10 Summary

This document has presented reasons that querying is needed in the TripCom project. This presentation is followed by a description of general features of semantic query languages.

We have then presented rules technologies to encode ontologies for enabling inference processes to derive implicit knowledge from explicit RDF descriptions. In addition, we have shown how rules can be used together with RDF query languages.

We have then presented sample queries which have been derived from use cases explored by the TripCom project, namely from Semantic Web Service communication, Enterprise Application Integration (EAI), and EPS (Electronic Patient Summary).

Finally we haved presented SPARQL with some identified extensions as a candidate query language for the TripCom project.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Jürgen Angele, Harold Boley, Jos de Bruijn, Dieter Fensel, Pascal Hitzler, Michael Kifer, Reto Krummenacher, Holger Lausen, Axel Polleres, and Rudi Studer. Web rule language (WRL). W3C Member Submission, available from `http://www.w3.org/Submission/WRL/`, June 2005.

[3] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[4] Dave Beckett. Turtle - Terse RDF Triple Language, April 2006. Available at `http://www.dajobe.org/2004/01/turtle/`.

[5] Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. MARS: A programmable coordination architecture for mobile agents. *IEEE Internet Computing*, 4(4):26–35, 2000.

[6] Jeremy Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. Named graphs. *Journal of Web Semantics*, 3(4), 2005.

[7] Paolo Ciancarini and Davide Rossi. Jada - Coordination and Communication for Java Agents. In Jan Vitek and Christian Tschudin, editors, *Mobile Object Systems: Towards the Programmable Internet*, volume 1222, pages 213–228. Springer-Verlag: Heidelberg, Germany, 1997.

[8] Kendall Clark. SPARQL Service Advertisement and Discovery Language (SADDLE). http://www.w3.org/2001/sw/DataAccess/proto-wd/saddle, 2005.

[9] Kendall Grant Clark. RDF Data Access Use Cases and Requirements, March 2005. W3C Working Draft.

[10] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[11] Jos de Bruijn, Thomas Eiter, Axel Polleres, and Hans Tompits. On representational issues about combinations of classical theories with nonmonotonic rules. In *Proceedings of the 1st International Conference on Knowledge Science, Engineering and Management (KSEM'06)*, Lecture Notes in Computer Science, Gullin, China, August 2006. Springer. Invited paper.

[12] Jos de Bruijn, Thomas Eiter, Axel Polleres, and Hans Tompits. On representational issues about combinations of classical theories with nonmonotonic rules. In *Proceedings of the 1st International Conference on Knowledge Science, Engineering and Management (KSEM'06)*, volume 4092 of *Lecture Notes in Computer Science*, pages 1–22, Gullin, China, August 2006. Springer. Invited paper.

[13] Jos de Bruijn, Enrico Franconi, and Sergio Tessaris. Logical reconstruction of normative RDF. In *OWL: Experiences and Directions Workshop (OWLED-2005)*, Galway, Ireland, November 2005.

[14] Jos de Bruijn and Stijn Heymans. Translating ontologies from predicate-based to frame-based languages. In *Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML-2006)*, Athens, Georgia, USA, November 2006. IEEE.

[15] Jos de Bruijn, Rubén Lara, Axel Polleres, and Dieter Fensel. Owl dl vs. owl flight: conceptual modeling and reasoning for the semantic web. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 623–632, New York, NY, USA, 2005. ACM Press.

[16] Jos de Bruijn, Holger Lausen, Axel Polleres, and Dieter Fensel. The web service modeling language: An overview. In *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, volume 4011 of *Lecture Notes in Computer Science*, Budva, Montenegro, June 2006. Springer.

[17] Mike Dean, Guus Schreiber, Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference, February 2004. W3C Recommendation.

[18] Harry Delugach, editor. *ISO Common Logic*. 2006. Available at `http://philebus.tamu.edu/cl/`.

[19] Marin Draltan. A formalization of rdf. Technical Report TR/DCC-2006-8, Universidad de Chile, 2004.

[20] Thomas Eiter, Giovambattista Ianni, Axel Polleres, Roman Schindlauer, and Hans Tompits. Reasoning with rules and ontologies. In *Reasoning Web 2006*, volume 4126 of *Lecture Notes in Computer Science*, pages 93–127. Springer, September 2006.

[21] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. dlvhex: A prover for semantic-web reasoning under the answer-set semantics. *wi*, 0:1073–1074, 2006.

[22] Melvin Fitting. *First Order Logic and Automated Theorem Proving (second edition)*. Springer Verlag, 1996.

[23] Tim Furche, Francois Bry, James Bailey, Sebastian Schaffertand Benedikt Linse, Renzo Orsini, Ian Horrocks, Michael Krauss, and Oliver Bolzer. Survey over existing query and transformation languages, r2.0. Technical Report I4-D9a, REWERSE project, April 2006.

[24] David Gelernter. Generative communication in linda. *ACM Trans. Program. Lang. Syst.*, 7(1):80–112, 1985.

[25] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

[26] M. Genesereth. KIF. knowledge interchange format: Draft proposed American national standard (dpANS). Technical Report NCITS.T2/98-004, ANSI KIF Ad Hoc Group, 1998. Available from `http://logic.stanford.edu/kif/dpans.html`.

[27] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.

[28] Michael Kifer Guizhen Yang. On the semantics of anonymous identity and reification. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE : Confederated International Conferences CoopIS, DOA, and ODBASE 2002. Proceedings*, pages 1047–1066. Springer, 2002.

[29] P. Hayes. RDF semantics. `http://www.w3.org/TR/rdf-mt/`.

[30] Ian Horrocks, Bijan Parsia, Peter Patel-Schneider, and James Hendler. Semantic web architecture: Stack or two towers? In Francois Fages and Sylvain Soliman, editors, *Principles and Practice of Semantic Web Reasoning (PPSWR 2005)*, number 3703 in LNCS, pages 37–41. Springer, 2005.

[31] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an OWL rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM, 2004.

[32] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A declarative query language for RDF. In The 11th Intl. World Wide Web Conference (WWW2002).

[33] M. Kifer. Rules and ontologies in F-Logic. In Norbert Eisinger and Jan Małuszyński, editors, *Reasoning Web: First International Summer School 2005, Msida, Malta*, volume 3564 of *LNCS*, pages 22–34. Springer, Berlin/Heidelberg, 2005.

[34] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[35] Atanas Kiryakov, Damyan Ognyanov, and Dimitar Manov. Owlim - a pragmatic semantic repository for owl. In *WISE Workshops*, pages 182–192, 2005.

[36] J.W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1984.

[37] J.W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1987. second edition.

[38] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax, February 2004. W3C Recommendation.

[39] Axel Polleres. Sparql rules! Technical Report GIA-TR-2006-11-28, Universidad Rey Juan Carlos, November 2006. available at `http://www.polleres.net/publications/GIA-TR-2006-11-28.pdf`.

[40] Axel Polleres, Cristina Feier, and Andreas Harth. Rules with contextually scoped negation. In *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, volume 4011 of *Lecture Notes in Computer Science*, Budva, Montenegro, June 2006. Springer.

[41] Eric Purd'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/, 2006.

[42] M. Sintek and S. Decker. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science*, pages 364–378, 2002.

[43] Herman ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. *Journal of Web Semantics*, 3, 2005.

[44] Robert Tolksdorf and Dirk Glaubitz. Coordinating Web-based Systems with Documents in XMLSpaces. In *Proceedings of the Sixth IFCIS International Conference on Cooperative Information Systems (CoopIS 2001)*, number LNCS 2172, pages 356–370. Springer Verlag, 2001.

[45] J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.

[46] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.

[47] P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T spaces. *IBM Systems Journal*, 37(3):454–474, 1998.

[48] Youyong Zou, Tim Finin, and Harry Chen. *F-OWL: an Inference Engine for the Semantic Web* , volume 3228 of *Lecture Notes in Computer Science*. Springer-verlag, November 2004. (proceedings of the Third International Workshop (FAABS), April 16-18, 2004, Greenbelt, MD, USA).