Mapping Relational Databases to Semantic Web using Domain-Specific Knowledge

Wondimagegn Yalew Mallede

A thesis submitted in partial fulfilment of the requirements of the London Metropolitan University for the degree of Doctor of Philosophy

2014

BRITISH LIBRARY

IMAGING SERVICES NORTH

Boston Spa, Wetherby
West Yorkshire, LS23 7BQ
www.bl.uk

BEST COPY AVAILABLE.

SOME FAINT TEXT

Acknowledgements

First and foremost, I would like to thank my supervisors- Professor Farhi Marir (Director of Study), Dr Vassil Vassilev (Lead Supervisor) and Dr Yanguo Jing. Professor Farhi was very encouraging and supportive from the beginning and made the very crucial decision of introducing Dr Vassil to the team. I would like to sincerely thank Dr Vassil for the professional, honest, and dedicated effort he has made in shaping my ever wondering thoughts and ideas.

I would like to also pass my appreciation to my previous adviser from University of Westminster- Dr Epaminondas Kapetanios. Dr Epaminondas strengthened and encouraged my research ideas during my stay at University of Westminster. The company I'm working for also gave me the platform to exercise my research ideas as well as allowing me to work from university fortnightly so I could easily consult my supervisors face to face. Thank you to all the managers I discussed the matter with at Sapienza consulting Ltd. A special thank you to Tori Dewhirst for her encouragement during some of the difficult times, for proof reading some of my "creative" English grammar and endure through all the endless "lectures" about my research.

I dedicate this research to my family who always encouraged me to pursue my education. My father Mr Yalew Mallede who is a great believer in education and has done his utmost for his children's future in education. I couldn't have achieved what I have achieved so far without him. My mother Mrs Kassanesh Abeje who if ever I have demonstrated persistence, kindness or hope, these are but a small reflection of the depth, beauty and strength of her character. I would like to also pass my thanks to my brothers, sisters, cousins, friends etc. for their encouragement in their own way. Last but not least I would like to thank my beautiful wife Zelalem Tadesse Benti for being there throughout the time we have been together. For your generous love and respect and above all for our lovely smiley princess baby girl- Lucy Betsinat Mallede. Waiting for your arrival gave me the energy to finish up my thesis and here I'm almost done to play and smile with you all the time ☺.

Table of Content

Contents

List of Algorithms

List of Figures

List of Tables

Abbreviations

API	Application Program Interface
DDL	Data Definition Language
DML	Data Manipulation Language
HTML	Hypertext Markup Language
JDBC	Java Database Connectivity
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SPARQL	SPARQL Protocol and RDF Query Language
SPI	Service Provider Interface
SQL	Structured Query Language
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
XML	eXtensible Markup Language

Chapter 1: Introduction

The current web experience gives us fairly abundant data. Using a few keywords and common search engines, it usually doesn't fail to return search results. With all its openness, the web gives anyone a chance to contribute ideas to be shared by the whole world about any topic. However, the web often feels like it is a mile wide, but only an inch deep. How can we build a more integrated, consistent, deep web experience? (Allemang & Hendler, 2008).

Semantic Web framework provides a platform and architecture to share and exchange data across different applications and platforms. Data sharing and standardisation saves a lot of time and resource in the current fast changing information age. Semantic Web framework is the answer to most of the questions with regard to making the web more universal, interoperable and machine processable. For this ultimate goal, meta-data plays a fundamental role as a source of additional information (Nilsson, et al., 2002). As the technology grows the metadata will be more dispersed and losses its uniformity. Languages like Resource Description Framework (RDF) are used to represent metadata information. Resource Description Framework is an XML-based language for describing resources (Daconta, et al., 2003). If HTML and the Web make all the online documents look like one huge book, RDF schema and inference languages will make all the data in the world look like one huge database (Berners-Lee, 1999).

Mapping existing relational models to ontological repositories will facilitate the process of upgrading legacy systems to Semantic Web. Taking advantage of the existing data without the need to reintroduce the data in semantic Resource Description Framework (RDF) will save us a great deal of time and resource. But this process also requires an efficient model to map not only the data but also the inter-relationship, pattern, validation, encryption, security and relational model attributes so the end semantic data could be processed with all the elements related to the raw data.

Semantic Web is a framework which allows information to be represented not only syntactically using suitable structural definitions ("data schema"), specific instances of them ("data") and their use ("access rights"), but also semantically using a

logical model which allows formal interpretation and sound logical inferencing about the information ("knowledge"). Relational models are limited to the interpretation of purely relational data and do not provide sufficiently rich means for interpreting the knowledge associated with the use of the data. Relational languages lack the expressiveness to represent *imprecise, uncertain, partially true* and *approximate* knowledge (Sheth, Ramakrishnan, & Thomas, Semantics for the Semantic Web: The Implicit, the Formal and the Powerful , 2005). This lack of capability to represent and process knowledge while it is still in a relational model is the trigger for this research project's initiative. This research project focuses on the mapping of the relational data stored in a database to an ontological repository as a step towards representing and processing the knowledge on the Web.

Once the data is extracted, transformed, loaded, managed and stored into the different data marts, it will be used as a source for the mapping. Resource Description Format (RDF) is "lingua franca" of the Semantic Web as HTML is the language of the classical Web. RDF/XML is a particular serialised format of RDF which can be used in conjunction with RDF Schema (RDFS) and Web Ontology Language (OWL) to represent semantically rich information in the repositories of Semantic Web.

The semantic meaning of the information represented in relational data can be derived from the properties of the relations and the associated operations in the domain of use. This leads to the introduction of a consensually shared view of concepts called Ontologies. An Ontology is a formal, explicit specification of a shared conceptualisation (Studer, Benjamins, & Fensel, 1998). The specifications use relations, functions, constraints, axioms etc. to conceptualise the abstract model. 'Formal' in the ontology definition refers to the fact that the expressions must be machine readable; hence natural language is excluded (Staab & Studer, 2009). RDF was designed for situations where Web data needs to be processed and exchanged by applications, rather than being displayed for people. The ability to exchange data between different applications means that the data may be made available to applications other than those for which they were originally intended (Marx, Salas, Breitman, & Viterbo, 2012).

Semantic data models and frameworks help to interpret the knowledge about specific domains and share this knowledge between systems. The models help to

see the 'semantic' from different angles and refine the meaning by working on the ambiguity while reusing its commonality. When two (or more!) viewpoints come together in a web of knowledge, there will typically be *overlap, disagreement,* and *confusion* before *synergy, cooperation* and *collaboration* (Allemang & Hendler, Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL, 2008). By transforming the relational data currently in a relational database into an ontological repository, the data will be available directly to Semantic Web applications. The different Semantic Web languages will also allow the restructure and enrichment of information so that it is better suited to its semantic use. Reasoning logic and rules are also accounted for in the mapping process.

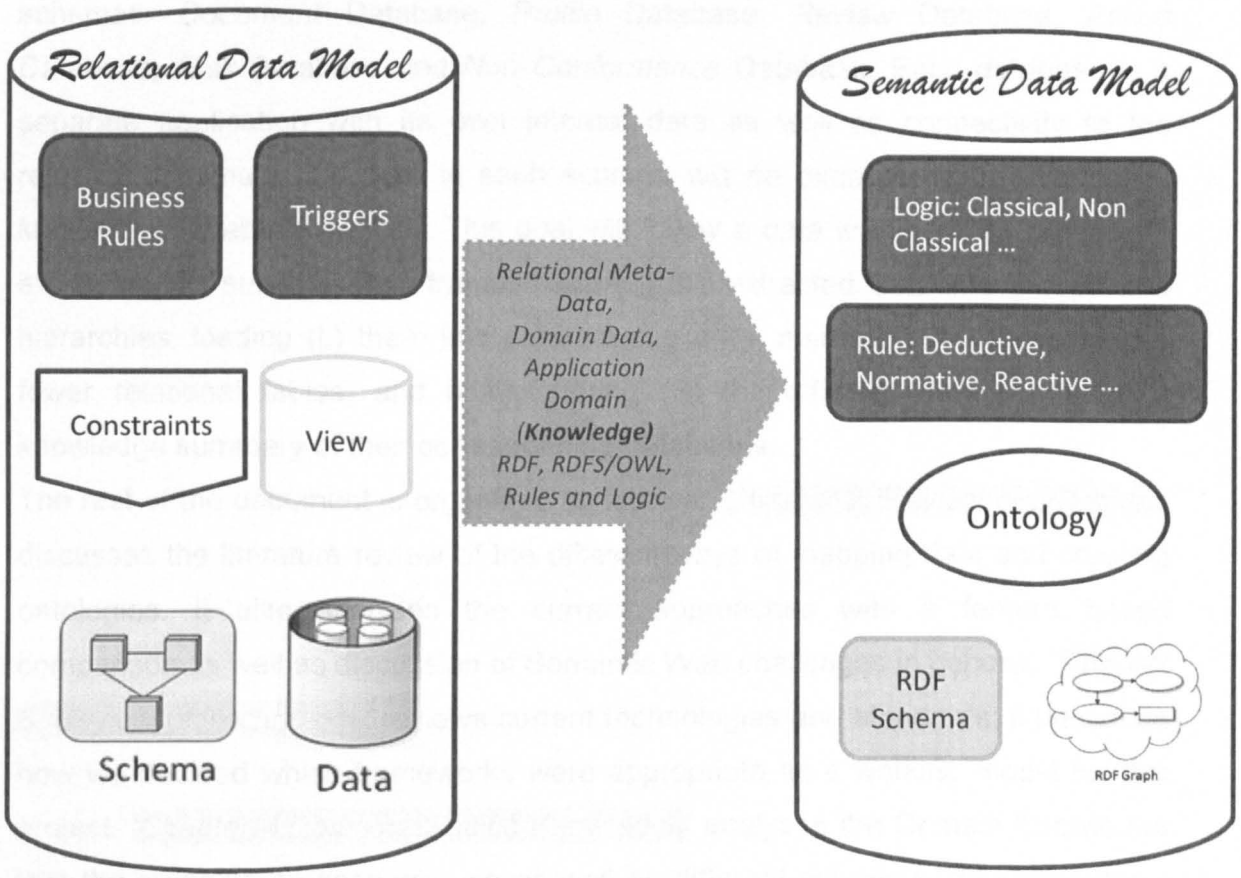


Figure 1- Relational to Semantic Mapping Model

Sematic Web will lead us to a more organised way of processing information with its architectural framework and semantic web languages at different levels of data representation (RDF), data schema (RDFS), ontology language (OWL), rules, logic etc. The future of the web in modelling unorganised information, representing

ontological relations and expressing different concepts will take advantage of the standardised Semantic Web components that will also be part of this research.

The research will use 'Space Project Management' as a case study to formulate relational to semantic mapping heuristics. The relational models that the mapping is based on are databases used by space missions and projects to support activities like earth observation; human space exploration; space launch and navigation; operational maintenance etc. The range of terminologies, standards, unit measurements, definitions etc. refer to the space domain ontology. The mapping process follows this space domain ontology as a 'dictionary' to represent, process, and return knowledge. The research project will focus on six major schemas- *Document* Database, *Profile* Database, *Review* Database, *Action* Database, *Risk* Database and *Non Conformance* Database. Each database is a separate application with its own internal data as well as connectivity to the relevant schemas. The data in each schema will be extracted with an ultimate knowledge database in mind. This goal will follow a data warehousing process in extracting (E) suitable data, transforming (T) the extracted data into groups and hierarchies, loading (L) them into data staging area, managing (M) the data into fewer relational tables, and finally storing (S) the different data marts as a knowledge summary of their corresponding databases.

The rest of the document is organised as follows: '*Chapter 2: Review of Literature*' discusses the literature review of the different ways of mapping data and creating ontologies. It also analyses the current approaches with a feature based comparison as well as discussion of Semantic Web challenges in general. '*Chapter 3: Review of Technology'* reviews current technologies and standards; it describes how we decided which frameworks were appropriate as a working model for this project. *'Chapter 4:Domain Specific Knowledge'* analyses the Domain Knowledge that the mapping process will use as well as different categories of knowledge - *Relational Database Area Knowledge, Domain User Knowledge, Domain Data Knowledge,* and *Application Domain Knowledge.* 'Chapter 5: Relational to Ontological Mapping Algorithm' outlines the different algorithms used to map relational databases to RDF repositories. 'Chapter 6:Relational and Ontological Model' sketches the Relational models and corresponding business knowledge flowcharts. '6.2 Ontological Repository Model' presents the ontological models on

the other side of the mapping paradigm and model view algorithms. 'Chapter 7: Relational to Ontological Mapping Implementation' describes the implementation of the algorithms presented in 'Chapter 5:Relational to Ontological Mapping Algorithm'. It uses different built-in, third-party and custom reasoners based on various ontological languages. 'Chapter 8: Evaluation' uses both statistical and feature based analysis to evaluate the algorithm implementation. 'Chapter 9:Conclusion and Future Work' concludes the thesis with a summary and plans for future work.

In this research project the following software specifications are used as part of the relational database and semantic web frameworks.

Table 1- Software Framework Specification

Framework	***Corporation/ Organisation***	***Version***
Relational Database	Oracle® Database	Oracle Database 11g Release 11.2.0.2.0 - 64bit Production
RDF Repository	Oracle® Spatial	Oracle Spatial 10g Enterprise Edition Release 10.2.0.1.0
Jena	Apache	Version 2.6.4
RDFS/OWL Languages	World Wide Web Consortium (W3C®)	RDFS 1/OWL 2
Application Server	Oracle® Application Server	Oracle10G (10.1.3) Containers for J2EE

Chapter 2: Review of Literature

Currently Semantic Web framework has different data representation, ontology language, semantic rule, query language and related standards to be considered as a way of representing complex models like relational model. World Wide Web Consortium's (W3C) effort to standardise and organise some of the already existing languages contributed to the current stage of being an important platform for Semantic Web's credibility. The journey from Web 1.0 through Web 2.0 to Web 3.0 has amassed a lot of experience and participation from different expertise for a well-reviewed Semantic Web framework.

The approach of mapping relational databases to ontological repositories is broadly divided by whether the mapped RDF repository is based on an already existing ontology or a new one that is initially produced as part of the mapping.

2.1 New Ontology

In this approach the ontology is produced as part of the mapping. The new ontology is either an automatically generated schema ontology or a preliminary domain-specific ontology that could be further refined by an expert user.

Automatically generated schema ontologies rely solely on the relational model. The richness of an automatically generated ontology depends on how exhaustively and efficiently the meta-data of the relational database are used. This ranges from a simple relation/ column to class/property up to complex relational patterns. The automatic nature of the new ontology has a disadvantage of not describing more than what is represented in the source relational database. An obvious advantage is the fast automatic ontology output. It's semantically easier to derive ontologies from Deep Web data sources, especially well-structured relational back-end databases, than from unconstrained natural-language text documents (Geller, Chun, & An, 2008).

Domain-specific ontologies use the domain as source of reasoning for the ontology. Usually a third party is involved in the ontology construction in one form or another. An expert in the domain can be involved in fine tuning the ontology. The involvement of a human significantly affects the automation speed. It also limits the domain knowledge to the expert's interpretation and expressiveness.

There is also an overhead on familiarising the expert with the ontology model. Another third-party option is the use of configuration files. The domain user configures what and how the mapping should happen and this opens an option to fine tune the configuration through a re-use of vocabularies to characterise their meaning in the ontology. A good combination of expert domain user and fine-tuned configuration file opens the option to share the configuration to other similar ontologies. Sharing and comparing configuration files helps to establish a refined configuration and as a result a mature ontology.

Instead of using a domain expert and/-or a configuration file, there is also an option to use the relational database as a source of ontology. By reverse engineering the logical model to the conceptual model, an Entity Relationship (ER) model can be achieved. Although reverse engineering a legacy relational database model that has been through a number of re-designs and normalisations is easier said than done, theoretically ER conceptual models represent domain ontology. The expected discrepancies when reverse engineering requires a domain expert to validate the ontology.

2.2 Existing Ontology

Mapping based on an existing ontology requires matching the defined terms of the ontology to appear in the mapped repository. Since the likelihood of the same party that designed the ontology also being responsible for the mapping is unlikely, there is a higher chance of the need for user intervention. Defining the ontology beforehand allows the possibility of replacing less used vocabularies than if a strict ontology definition is followed. Similarly, if there is a vocabulary noticeably appearing continuously, the option of introducing it to the ontology is also challenging.

One of the main advantages of having an existing ontology during mapping is the possibility of representing a visual graphical interface to facilitate ontology matching. Mapping using an existing ontology primarily uses lexical matching to determine the destination of the source relational object. The proximity between the source relational database and the ontology elements determines the mapping success. Extending lexical matching methodology to "synonym matching"

increases the richness of the mapped data. In addition to lexical and synonym matching a well-designed "pattern matching" cab even help to discover new data.

2.3 *Current Approaches*

The World Wide Web Consortium (W3C) RDB2RDF Working Group (WG) has conducted a survey of current approaches for mapping of relational databases to RDF (Sahoo, et al., 2009). The report summarises the different mapping implementations, query implementations, application domains, and approaches to mapping creation, mapping representation and accessibility.

The mapping implementation is using either a static Extract Transform Load (ETL) or a dynamic on demand query-driven implementation. The static data warehousing approach has its own drawback in accounting the current data. The queries are run at periodic intervals without compromising the current performance using mapping rules. It also gives an opportunity to analyse the data with respect to validation rules. The dynamic approach on the other hand costs a lot of performance time even though the outcome is a current reflection of what is in the relational database.

The query implementation follows two paths. The SPARQL-> RDF or the SPARQL-> SQL-> RDB path. SPARQL treats each RDF graph as a RDB table with three columns, *?subject, ?predicate* and *?object*. Each row corresponds to one tuple and the query result of SPARQL constitutes a table of RDF nodes (Cyganiak, 2005).

Automatic mapping of RDB table to RDF class node and RDB column to RDF predicate doesn't satisfy the goal of mapping the semantics of the data. Tools like Viruoso RDF View (Blakeley, 2007) expanded the above notion to map RDB unique identifier (primary key) to RDF subject and column values as RDF object. Even though these automatic mapping tools could be used as a starting point, there is still a lot to do to analyse, refine and process the data.

The survey also suggested the use of pre-existing public ontology resources such as the National Centre for Biomedical Ontologies at http://bioportal.bioontology.org/ or an automatic domain-specific mapping tool such as D2RQ (Bizer & Cyganiak, 2007) that also allows custom user mapping rules. This approach also helps to reduce the amount of redundant knowledge. In one of

the projects (Byrne, 2008) based on the Royal Commission on the Ancient and Historical Monuments of Scotland (RCAHMS), 1.5 million entities of the database are converted into 21 million RDF triples. Using the domain semantics-driven generation the size of the RDF dataset is reduced by 2.8 million.

Domain-specific ontology is a reasoning tool to standardise and link terminologies between databases and semantic web. Rather than using a manual approach, automatic generation of ontologies from Web sources would be preferable, but this is well-known to be a difficult task (Wu, Doan, Yu, & Meng, 2005). The mapping methodology should follow an algorithm which is open to modification and refinement. There should be a mechanism to solve semantic ambiguity between domains and elements within a domain.

Existing ontology languages like RDF and OWL have their own restrictions in representing relational data in semantic web. RDF Schema is a vocabulary description language for describing properties and classes of RDF resources (Harmelen, 2008). This expressive language is further developed based on Description Logic that provides an algebra for constructing complex classes out of simpler ones (Boley, Kifer, Patranjan, & Polleres, 2007). OWL is a richer vocabulary description language for describing properties and classes, such as relations between classes (e.g., disjointness), cardinality (e.g.- "exactly one"), equality, richer typing of properties, characteristics of properties (e.g.- symmetry), and enumerated classes (Harmelen, 2008).

Different deductive, normative and reactive rules will allow us to better map the business rules, establish privacy policies (security), control access levels and implement other in-depth negotiations between Semantic Web entities. The {event, condition, action} tuple case of reactive rule interchange is a typical example that should be mapped and applied in Semantic Web framework as would have been the case in a RDB trigger implementation. The survey by the W3C RDB2RDF incubator group also recommended the use of the W3C Working Group Rule Interchange Format (RIF) to represent RDB to RDF mappings (Sahoo, et al., 2009).

A further "feature-based comparison" between the major mapping languages (Direct Mapping, eD2R, R_2O, Relational.OWL, Virtuoso, D2RQ, Triplify, R2RML, R3M) based on RDB2RDF WG report (Sahoo, et al., 2009) also shows the

different features of existing mapping languages (Hert, Gerald, & Gall, 2011). The paper compares the mapping languages using four categories: *direct mapping, read-only general-purpose mapping, read-write general-purpose mapping, and special-purpose.*

2.3.1. Direct Mapping [DM, SquirrelRDF, SQL2SW]

Direct mapping bases its implementation on finding a correlation between relational databases and ontologies. The formulation of relational databases using *First Order Logic* (FOL) (Rybiński, 1987) helps to describe a database structure and properties of the database constructs. On the other hand the use of *Description Logic* (DL) (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2010), which is a subset of FOL, by semantic ontology languages like OWL makes the mapping more feasible.

The different relations, relationships and attributes in the relational schema are used to choose the corresponding classes and properties. The *pre-processing* phase identifies the strong and weak relations to be mapped into ontology classes. The relationships are also classified as those that can be mapped as a class by its own and as an object property of an existing class. The attributes are mapped based on their participation in a foreign key relationship. Those that are used as a foreign key in a relational relationship are mapped as object properties while non-foreign key attributes are mapped as data type properties.

Mapping languages like SquirrelRDF allow a SPARQL query access to relational databases via JDBC. SquirrelRDF exposes the mapped store in a rather "raw" form. It makes no attempt, for example, to reveal implicit relations between objects (suggested by foreign keys), or normalise denormalised data. This simplifies Squirrel's task, focusing it on mapping to RDF and ignoring the complex task of transforming between vocabularies or ontologies, which are better left to pure RDF tools (SquirrelRDF, 2013). SQL2SW mapping language goes a bit further to use OWL as an ontology repository language. SQL2SW relies on the availability of the most accurate relational model in SQL DDL form. It also assumes the normalisation of the relational schema at least up to third normal form. Starting from the relational model which is typically represented in SQL DDL form already incurs a loss of semantics from the preceding Entity Relationship (ER) conceptual

model. Since a relational schema does not support all constructs of a conceptual schema, some of the semantics captured in the conceptual schema – e.g. inheritance – will necessarily be lost when translating the schema from conceptual to relational (Benslimane, Benslimane, & Malki, 2006) (Alalwan, Zedan, & Siewe, 2009). Nevertheless, an automatic transformation of SQL DDL statements to OWL DL ontologies goes further than earlier efforts in that the entire system is expressed in first-order logic (Tirmizi, Sequeda, & Miranker, 2008).

2.3.2. Read Only General Purpose Mapping [Virtuoso, D2R MAP, D2RQ]

The simplest approach for a read only general purpose mapping language is a single table with the three Subject, Predicate, Object columns and an additional Graph column that forms a Quadruple. This approach offers both SPARQL and SQL support. SPARQL queries are typically translated into SQL during query parsing. With an assumption that all the data is in one big table, the translation takes a one to one approach with `OPTIONAL` becoming `LEFT OUTER JOIN` and SPARQL `UNION` becoming SQL `UNION`. There is also a support for basic SQL aggregate functions and relational data manipulation (insert, delete, update) using SPARQL /Update extension- SPARUL (Seaborne, et al., 2008). Virtuoso mapping language (Virtuoso, 2013) uses the above approach to map relational data to RDF ontologies.

Other approaches are also being tried by mapping languages like D2RQ (D2RQ, 2013) to treat non-RDF relational databases as virtual RDF graphs. D2RQ provides SPARQL query endpoints over the database as well as Apache Jena API to access information. The two important terms in D2RQ mapping language are ClassMap and PropertyBridge. ClassMap represents a group of similar classes of the ontology. It specifies how URIs (or blank nodes) are generated for the instances of the class. It also has a set of property bridges, which specify how the properties of an instance are created. Property values are created either directly from the database or using patterns and translation tables. D2RQ also supports conditional mapping to distinguish between accessible and non-accessible (confidential or out-dated) information. For performance optimisation it also has hint properties to be added to property bridges to speed up queries. Providing hints like `d2rq:valueMaxLength`, `d2rq:valueRegex` and

`d2rq:valueContains` for property bridges which are using `d2rq:column` on large tables and on columns that are not indexed by the database optimises the performance of queries.

2.3.3. Read Write General Purpose [R3M]

Read-Write approach has a goal in mind to take advantage of both relational databases for their scalability and response time as well as semantic web's universal, exchangeable and explicit representation that offers semantic web reasoning support. The approach sees the pros and cons of mapping relational data as a one-time conversion to RDF repositories versus keeping the relational database and providing on-demand RDF output.

Current RDF triple stores are not as scalable as the mature relational database management systems in handling the ever increasing data. The existence of enterprise applications that are designed with relational databases in mind won't help in the decision to migrate everything to RDF repositories as well. This leaves us with using both relational databases and RDF repositories for different and equally important purposes. Then the critical question is how to keep the consistency between the two endpoints. The answer is to upgrade the current read-only access to the mapped relational data to read-write that allows changes to RDF data and propagates back to the source relational database. The most important advantage in this read-write approach is the possibility of persisting the current memory based reasoning that up to now can't exceed the memory size. The implementation of this read-write approach employs SPARQL and SPARQL/ Update extension (SPARUL) (Seaborne, et al., 2008). R3M is the only read-write mapping language currently implementing this approach.

2.3.4. Special Purpose Mapping [eD2R, R_2O, Relational.OWL, Triplify]

eD2R: one of the Special Purpose Mapping languages is eD2R (extended D2R) which is an extension of D2R MAP mapping language. eD2R is a declarative XML-based RDB to RDF mapping language based on RDFS. On top of D2R's ClassMap and PropertyBridge attributes, eD2R extends the language by adding operations and condition items. This allows the definition of complex and conditional transformations on field values based on techniques such as keyword

search, regular expression matching, and natural language processing (Barrasa, Corcho, & G´omez-P´erez, Fund Finder: A Case Study of Database-to-Ontology Mapping, 2003). The main goal of extending D2R (eD2R) is to cover mapping situations involving databases that are lightly structured or not in first normal form (Hert, Gerald, & Gall, 2011). The declarative nature of eD2R is manifested by an independent mapping document that outlines the source relational entities, relationships and attributes and the corresponding classes, object and data properties. The mapping document and the relational database are both inputs to the mapping language to produce an ontological repository. Mapping documents can also be reusable to other relational databases.

R_2O: while mapping RDB to RDF is a very important step in managing knowledge and making the data more universal and interoperable, it is also apparent to keep "upgrading" an existing RDF repository from different RDB sources. R_2O has been designed with this Special Purpose Mapping capability. Unlike D2R MAP and eD2R, R_2O is fully declarative with a better expressiveness for writing complex mapping transformations (Barrasa, Corcho, & G´omez-P´erez, R2O, an Extensible and Semantically based Database-to-Ontology Mapping Language, 2004). The complexity of R_2O answers some of the issues arising when the similarity between the RDB and the Ontology model is so far apart that it's difficult to draw a simple correlation. R_2O's RDBMS independent high level expressiveness and its compatibility with any SQL standard implementation is summarised with the following features (Barrasa, Corcho, & G´omez-P´erez, R2O, an Extensible and Semantically based Database-to-Ontology Mapping Language, 2004).

1. R_2O can be used to suit the current RDB data by setting a mapping definition which can be executed as a batch or on-demand process via a query translation.
2. R_2O can be used to express mappings generated by another mapping discovery tool.
3. The full declarative nature of R_2O can also be used to automatically detect inconsistencies and ambiguities within its own implementation.
4. R_2O's mapping definitions can be used to verify the integrity of the RDB according to an Ontology.

5. R_2O's mapping definition can also be used to automatically characterise data sources to allow dynamic query distribution in intelligent information integration approaches.

Relational.OWL: Relational.OWL is another Special Purpose Mapping language that specialises in Peer-to-Peer database management where peers have the autonomy to decide whether to join or to leave an information sharing environment at any time (Perez de Laborda & Conrad, 2005). The mapping language addresses issues with Volatile Peers, Data Distribution, Relational data, and Data Evolution. Peers in the P2P network have the right to join or leave an information sharing environment anonymously. This wide possibility on the life time of peers makes the negotiation of an exchange protocol very challenging. The distribution of data across different data sources also creates multiple data formats that cause unmanageable data and schema representation. Unlike classical P2P file sharing networks where the file contains all the necessary information to understand the content, relational databases contain not only data but also other schema and related meta-data information. The exchange of data has to be managed by separating the data from the meta-data for a correct interpretation for other peers. Relational.OWL also looks into the ever evolving relational data that might not be necessarily a challenge for a typical P2P file exchange format like binary music or video file.

Triplify: previous Special Purpose Mapping languages (eD2R, R2O, and Relational.OWL) aim at providing ontological representation of the database while Triplify focuses on providing a query access methodology for users. Triplify is based on mapping HTTP-URI requests onto relational database queries. (Auer, Dietzold, Lehmann, Hellmann, & Aumueller, 2009). Unlike other mapping languages that work independently to generate ontological repositories, Triplify provides a lightweight software component that can be easily integrated and installed inside web applications. This provides an easier approach to encourage web application developers to participate in facilitating RDF support without the need to learn a new mapping language.

Triplify is based on the definition of relational database views for a specific Web application (or more generally for a specific relational database schema) in order to retrieve valuable information and to convert the results of these queries into RDF, JSON and Linked Data (Auer, Dietzold, Lehmann, Hellmann, & Aumueller, 2009). By avoiding the complexity of mapping languages and only relying on a few important SQL statements, the amount of RDF triples and Linked Data can be significantly increased. SQL has many features like aggregation and grouping functions as well as table joining that are not readily available in many mapping languages. Using SQL also improves the scalability as the major SQL operation can be pushed down to the database management system. The conversion of SQL query results into RDF model follows three simple conventions. Firstly, projection of the SQL query result must contain an identifier like a primary key to generate instance URIs. Secondly, column names of the SQL projection will be used to generate property URIs. Lastly, individual column values are converted into triple objects. These three steps are followed by annotations to determine the output types. Mapping to existing vocabularies by giving the same SQL column alias results in merging to a common namespace definition. Symbols like `'->'` will generate URIs, `'^^'` hints Triplify to generate RDF literals with matching XSD data type while `'@'` is used as language tag.

Triplify provides both on-demand using a URL which is part of the Triplify namespace or in advance using SQL statements. On-demand database conversion searches for a matching URL pattern for the requested URL by replacing potential placeholders in the associated SQL query (Auer, Dietzold, Lehmann, Hellmann, & Aumueller, 2009).

The legend and table below further summarises the above nine mapping languages based on the supported features (F1 – F15).

Table 2- RDB-to-RDF Mapping Languages Comparison Legend (Hert, Gerald, & Gall, 2011)

Legend	Description
F1	Logical Table to Class
F2	M:N Relationships
F3	Project Attributes
F4	Select Conditions
F5	User-defined Instance URIs

F6	Literal to URI
F7	Vocabulary Reuse
F8	Transformation Functions
F9	Datatypes
F10	Named Graphs
F11	Blank Nodes
F12	Integrity Constraints
F13	Static Metadata
F14	One Table to n Classes
F15	Write Support

Table 3- Summary table of RDB-to-RDF mapping language comparison (Hert, Gerald, & Gall, 2011)

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
Direct Mapping	(✓)	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
eD2R	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	(✓)	✗	✓	✗
R_2O	✓	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	(✓)	✗	(✓)	✗
Relational.OWL	(✓)	✗	✓	✗	✗	✗	✗	✗	✓	✗	✓	(✓)	✗	✗	✓
Virtuoso	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	(✓)	✗	✓	✗
D2RQ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	(✓)	✓	✓	✗
Triplify	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	(✓)	✗	✓	✗
R2RML	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	(✓)	✓	✓	✗
R3M	(✓)	✓	✓	(✓)	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓

What is being consistently missing from the existing mapping approaches is the lack of accounting for implicit knowledge which can be derived from the explicitly represented relational model. It can also be checked using "application domain predicates" and/or executed using "application domain procedures/functions". The deficiencies of the existing mapping languages can be categorised into the following major levels of "knowledge".

1. Absence of means to account for all the available relational database area objects and their variant meta-data combinations.

2. Absence of means to account for domain data knowledge like data-patterns (disjointness, symmetry, transitive chain, etc.)

3. Absence of means to account for application domain knowledge in the form of predicate, function and procedure.

2.4 Challenges for the Semantic Web

Out of the many challenges that Semantic Web faces at its current embryonic stage, availability of content and ontology, scalability, multilinguality, intuitive visualisation and standardisation are some of the general challenges for the semantic web evolution (Benjamins, Contreras, Corcho, & Gómez-Pérez, 2002).

2.4.1. Content Availability

Static HTML Content (Surface Web):

The internet consists of large amounts of static HTML data which is not suitable for semantic web processing. The contents of those pages are only readable to humans and are not understandable by computers. It is extremely difficult to annotate those HTML pages with ontological information. This makes the challenge towards semantic web more difficult and risks the exclusion of existing static page information from semantic web.

There are a few approaches to add semantic to existing static HTML pages. A number of web sites nowadays give an option of tagging data. This approach is especially useful for multimedia web sites where implementing collaborative intelligence algorithms is more challenging in identifying the content. To

overcome the challenge of content availability, there are a few tools that provide a facility for the end user to tag the information. This process is called Folksonomy. A folksonomy is a collaboratively generated free-form categorisation system that allows internet users to categorise online content using tags (Henson, 2007). It's a less costly approach to facilitate the process of tagging than making the content developer fill all the attributes of the information. This process is becoming part of the path to creating a semantic web (Ronkowitz, 2012).

The main contributors of tags in folksonomy are not expert users. This makes the process more user-oriented with the help of facilities like synonyms and thesaurus. Some web sites like del.icio.us (http://www.delicious.com/) list popular tags and tag bundles to organise tags, for example "cats", "dogs" and "fish" tags could be grouped into a bundle called "pets". Some web sites also allow combination of several tags. This will help to achieve a more precise search result.

Lack of precision is one of the main problems of folksonomy. People argue that tagging can only hurt precision even though it is widely assumed that it will improve recall (Jörgensen, 2007). On the other hand folksonomies are discovery systems, without the powerful search capacity of a hierarchical taxonomy; they are going to characteristically have low precision rates (Kroski, 2005).

Phonetic tag representation, misspelling and lack of synonym control are another set of problems that make information representation and retrieval more challenging. The fact that end users make their tag choices makes the tagging process more subjective and prone to semantic ambiguity. Conversely, systems employing free-form tagging that are encouraging users to organise information in their own ways are supremely responsive to user needs and vocabularies, and involve the users of information actively in the organisational system (Mathes, 2004). One of the problems with collaborative tagging is the issue of 'sloppy tags'. These tags are hindrance to classification and searching of tags. Some experts suggest "tidying up" approach and its underlying assumptions, highlighting issues surrounding removal of low-quality, redundant or nonsense metadata, which then lead to the potential risks of tidying too neatly and thereby losing the very openness that has made folksonomies so popular (Guy & Tonkin, 2006). Accepting the context as a tag when the amount of information

to be retrieved is greater than the limited information to be provided by the user is another alternative way of refining 'sloppy tags' (Kim & Kwon, 2008).
The user has a freedom to choose or invent a tag that may not belong to any structure or hierarchy. This feature makes tagging more popular and "democratic" with a consequence of less precise search results and difficult synonym management. For taggers, it's not about the right or the wrong way to categorise something and it's not about accuracy or authority, it's about remembering (Kroski, 2005).
Lack of precision, lack of synonym control and lack of hierarchy of contents are the main challenges in enriching content in semantic web.

Dynamic Database Content (Deep Web):

The other information source on the web is the dynamically generated HTML pages from databases (Deep Web). One of the main challenges in Semantic Web is the already existing vast relational data in relational databases. Relational databases have their own way of categorizing and storing data that is not understandable in the Semantic Web framework. There is more information in the deep web databases than in static web pages and this information is only accessible through web services and application interfaces. Beyond those trillion pages lies an even vaster web of hidden data: financial information, shopping catalogues, flight schedules, medical research and all kinds of other material stored in databases that remain largely invisible to search engines (Wright, 2009). Deep Web databases are typically under-represented in search engines due to the technical challenges of locating, accessing, and indexing the databases (Casanova, Barbosa, Breitman, & Furtado, 2012).
The mapping process makes use of ontology systems rather than tags which are driven by decentralised communities from the bottom up, sometimes called Web 2.0 or social software (Shadbolt, Berners-Lee, & Hall, 2006). This research project will focus on tackling this issue by going further than just annotating the content. This includes the mapping of data as well as the representing of knowledge surrounding the data from relational databases to ontological repositories.

2.4.2. Visualization

The weaknesses of tagging static HTML pages (folksonomy), as we have mentioned them (lack of precision, lack of synonym control and lack of hierarchy of contents), led many of the promoters of such classification to explore new interfaces to help the user among the mass of tags. Some interfaces are basic and just offer the possibility to type tags or search for them but many are more original in the way they propose to interact with information related to content (Glassey, 2007). Tag clouds are one of the most popular tools used to help categorise tags using font sizes and colours.

Tag clouds are visualisation features that help formalise tagging as a taxonomy process. The list of words and their font size indicate the number of posts and popularity of the tags while the display order is usually alphabetical. Websites like Buzznet (http://*www.buzznet.com/*) and Flickr (http://www.flickr.com/) provide a list of popular photo image tags while del.icio.us (http://www.delicious.com/) and Furl (http://www.diigo.com/) give a list of bookmarks under each tag. Technorati (http://technorati.com/) even provide search on blogs. The list of tag cloud keywords educates the user and makes the websites more semantic which helps users to search and analyse the information when needed. The problem with these tags is the lack of defined taxonomic hierarchy or structure and it only helps the user to structure the web sites from his/her point of view.

2.4.3. Lack of Standards

The other big challenge in the Semantic Web is lack of standards. The World Wide Web Consortium (W3C) is playing major role in setting most of the existing standards but there are still parts of the overall standards which are work in progress with a challenge of not going in the same direction. The current Semantic Web standards (XML, RDF, RDFS, OWL, SPARQL etc.) are well founded standards to base on but there are still challenges on different "Mapping", "Encryption", "Trust", "Security" and other related areas that are not yet fully researched.

2.4.4. Ontology Availability

Ontology is the dictionary of Semantic Web that gives a meaning to the data. The availability of ontology is one of the challenges in semantic web. There are

domain specific ontologies that are used in specific fields like human sciences and cultural heritage like National Centre for Biomedical Ontologies (NCBO, 2012) and Royal Commission on the Ancient and Historical Monuments of Scotland (RCAHMS, 2012).

Even though the availability is increasing on many more subject fields, the refinement and usability is still a challenge for the semantic web.

2.4.5. Scalability

There is a challenge on how to manage and process semantic data according to a particular knowledge domain. As most of the data and logic standards in semantic web are text based with XML type formats, the method of storing, organizing, indexing, querying, and maintaining is something that needs careful consideration. A study on RDF data management techniques proposes six key factors to describe the characteristics of an RDF data D to evaluate and measure the space efficiency of RDF data storage and indexing technologies (Jalali, Zhou, & Wu, 2011).

1. |D|: the total number of triples in D;
2. |*pred*(D) |: the number of unique properties in D;
3. |*val*(D) |: the number of unique values in D;
4. |*cls*(D)|: the number of classes in D;
5. *avgPred*(D): the average number of properties belonging to the same class in D; and
6. |*lft(D)*|: *the number of triples whose subjects do not belong to any class* or whose predicates are multi-value predicates.

Using Barton (Abadi D. J., Marcus, Madden, & Hollenbach, 2007), Lehigh University Benchmark (LUMB) (LUMB, 2013), Yago (Suchanek, Kasneci, & Weikum, 2007), and LibraryThing (LibraryThing, 2013) datasets as RDF benchmark, the table below summarises their data characteristics. |*cls*(D)|, *avgPred*(D), and |*lft(D)*| schema related data characteristics are not included due to the absence of schema information in the datasets.

Table 4- Summary of Data Characteristics for RDF Benchmarks (Jalali, Zhou, & Wu, 2011)

Benchmark	|pred(D)|	|val(D)|	|D|
Barton	285	19M	50M
LUBM	18	1M	7M

Yago	93	34M	40M
LibraryThing	338824	9M	36M

Commercial databases like Oracle are incorporating semantic technology that would enable semantic data to share the performance and scalability of an existing technology. Oracle Database 11g Enterprise Edition delivers advanced semantic data management capability not found in any other commercial or open source triple store. The semantic features are part of the Oracle Spatial 11g option (Oracle, 2009). With native support for RDF/RDFS/OWL/SKOS standards, this semantic data store enables application developers to benefit from an open, scalable, secure, integrated, efficient platform for RDF and OWL-based applications. These semantic database features enable storing, loading, and DML access to RDF/OWL data and ontologies, inference using RDFS, OWL and SKOS semantics and user-defined rules, querying of RDF/OWL data and ontologies using SPARQL-like graph patterns embedded in SQL, and ontology assisted querying of enterprise (relational) data.

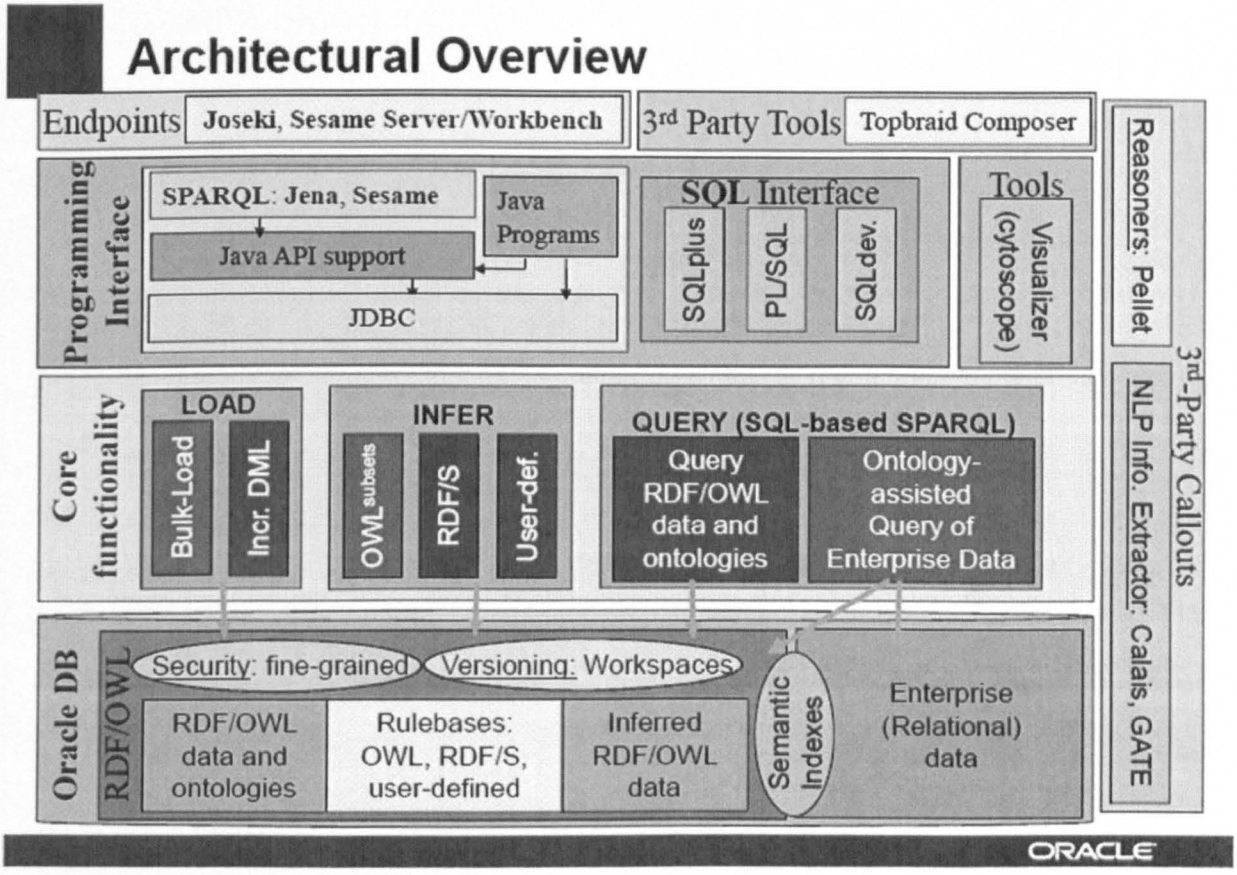


Figure 2- Oracle Semantic Architectural Overview (Lopez, 2009)

2.4.6. Multilinguality

The usage of different languages on the internet complicates the way we represent semantic data. The semantic search should also address the different lexical, semantical and structural issues in Ontology Development (Chau & Yeh, 2005). The ever increasing non-English language usage in the World Wide Web makes the challenge even greater.

Chapter 3: Review of Technology

Review of Technology chapter explains the semantic web architecture and the technology to be used for mapping relational databases to Semantic Repositories. It starts from the least expressive part of the architecture to the most expressive modelling languages. Way before the use of computer languages to represent entities, the study of Semiotics or Semiology dates back centuries that uses signs, symbols and their use of interpretation (Chandler, 2007).

From the early seventies the use of relational model to manage databases paved the way for its current popularity. Entity relationship (ER) is used as a conceptual specification for the data model. Although the relational model still works well in contexts relying on standard databases, it imposes certain restrictions, not inherent in ER specifications, which make it less suitable in Web environments (Casanova, Barbosa, Breitman, & Furtado, 2012). Semantic Web "Linked Data" approach uses triplification to represent entities as "subject, predicate, object". The focus of this research is not only to map the relational schema and data into RDFS repositories but also to represent the implicit knowledge into the ontology.

The current semantic web architecture layer consists of IRI/URI, XML, RDF, RDFS, Ontology, Rules, Query, Logic and other top layers. The sections below elaborate those layers in accordance with their use as standard utilities for the modelling, mapping, implementation and management of the research project.

The entities that are represented in Semantic Web are commonly referred as *resources*, hence the name as part of the basic semantic framework- *Resource Description Framework* (RDF). A resource is represented by the very bottom component of the Semantic Web Architecture layer (Figure 3) - URI/IRI. Universal/Internationalised Resource Identifiers (URI/IRI) are resource locators in the World Wide Web that constitute a network location (URL) to a resource and an optional fragment identifier preceded by a hash (#) symbol. The use of URI helps to identify the resource and present it by adding discriminators.

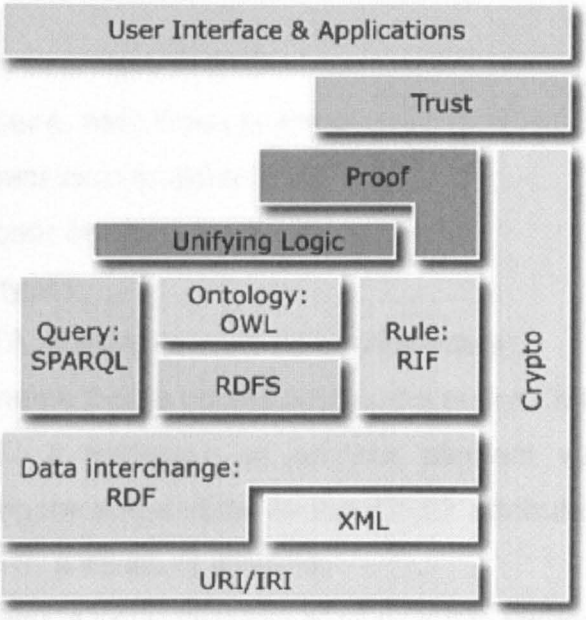


Figure 3- Semantic Web Architecture Layer

3.1 XML

XML is a metalanguage for markup: it does not have a fixed set of tags but allows users to define tags of their own (Harmelen, 2008). XML unlike RDF is an ordered node structure. This rigid structure makes it difficult to represent the whole "relational information" in Semantic Web. XML's tree structure is also another hindrance unlike RDF's "graph" structure which is more flexible to represent different axioms and relations.

- the structure of an XML document must be defined by writing a DTD, the older approach, or an XML schema, the modern approach that will gradually replace DTDs
- the first character of a tag must be a letter, an underscore, or a colon; and that no name may begin with the string "xml" in any combination of cases (such as "Xml" and "xML")
- an attribute is a name-value pair inside the opening tag of an element

```
<!ELEMENT family (person*)>
<!ATTLIST...
                name        CDDATA     #IMPLIED
                address             CDDATA     #REQUIRED
                encoding    (mime|binhex) "mime"
                file        CDDATA
<!ELEMENT text (#PCDATA)>
```

- cardinality operators in DTD
  - ?: appears zero times or once
  - *: appear zero or more times
  - +: appear one or more times
- Attribute types
  - CDATA, a string (sequence of characters),
  - ID, a name that is unique across the entire XML document,
  - IDREF, a reference to another element with an ID attribute carrying the same value as the IDREF attribute
  - IDREFS, a series of IDREFs,
  - (v1|...|vn), an enumeration of all possible values
- value types
  - #REQUIRED
  - #IMPLIED
  - #FIXED (a value given in an XML document is overridden)
  - "value"

XML Schema allows one to define new types by extending or restricting already existing ones. In combination with an XML-based syntax, this feature allows one to build schemas from other schemas.
XML schema provides a sophisticated set of data types that can be used in XML documents (Harmelen, 2008)
(DTDs were limited to strings only)
<xsd:schema
xmlns:xsd="https://www.w3.org/2000/10/XMLSchema"
version="1.0">
<element name="..."/>; optional attributes like type, minOccurs, maxOccurs
<attribute name="..."/>; optional attributes like type, use (optional, required, prohibited), default
unlike DTDs, XML Schema provides powerful capabilities for defining built-in and user-defined (simple or complex) data types

XML namespaces are used to uniquely identify named elements and attributes in an XML document

Example:

xmlns:prefix="location"

```
<?xml version="1.0" encoding="UTF-16"?>
<vu:instructors
        xmlns:vu="http://www.vu.com/empDTD"
        xmlns:gu="http://www.gu.com/empDTD"
        xmlns:uky="http://www.uky.edu/empDTD"
        <uky:faculty
         uky:title="assistant professor"
         uky:name="John Smith"
         uky:department="Computer Science"/>
        <gu:academicStaff
         gu:title="lecturer"
         gu:name="Mate Jones"
         gu:school="Information Technology"/>
</vu:instructors>
```

3.2 *Resource Description Framework and Schema (RDF/RDFS)*

RDF is a data-modelling language specified in an object-attribute-value triple, called a *statement* (all about resources, properties and statements). RDF is a data model for objects and relations between them that can be represented in XML syntax. In RDF, each data entity has a Unique Resource Identifier (URI) and each relationship between two data entities is described via a triple within which the items take the roles of subject(S), predicate (or property)(P) and object(O) (Jalali, Zhou, & Wu, 2011). RDF graph is a representation of a set of RDF triples that constitutes a subject node, predicate, and object node. RDF Graph has nodes and labelled arcs to link two nodes. Each node is RDF URI, RDF Literal, or blank nodes.

RDF Schema is a vocabulary description language for describing properties and classes of RDF resources. XML Schema constructs the *structure* of XML documents, whereas RDF Schema defines the *vocabulary* used in RDF data models. This makes RDF a richer representation tool which is based on a set of vocabularies unlike XML which is based on structural hierarchy. In RDFS we

can define the vocabulary, specify which properties apply to which kinds of objects and what values they can take, and describe the relationships between objects. The Classes and Properties in RDF description vocabulary language are similar to any object-oriented programming language like Java. The only difference is that instead of defining the properties of a class through its instances, the RDF vocabulary description language allows to describe the properties of the classes in terms of the resources using the "domain" and "range" mechanism. Data sets included in another data sets can be defined as a rdfs:subClassOf in the RDFS , whereas a taxonomic structured code set such as the animal kingdom, vegetable kingdom, and products classification code can be defined as a rdfs:subPropertyOf in the RDFS (Choi, Moon, Baik, Wie, & Park, 2010).

1. Classes:

Different resources could be divided into classes. Each member of a class is called *instances* of the class. Classes are resources by themselves and are described using RDF properties. "rdf:type" is used to state that a resource is an instance of a class.

1.1 rdfs:Resource

Everything described in RDFS are called resources. This is an instance of rdfs:Class. All other classes are subclasses of this class.

1.2 rdfs:Class

This is the class of resources that are RDF classes. Rdfs:Class is an instance of rdfs:Class.

1.3 rdfs:Literal

The class rdfs:Literal is the class of literal values such as strings and integers. Property values such as textual strings are examples of RDF literals. Literals may be plain or typed. A typed literal is an instance of a data type class. This specification does not define the class of plain literals.

rdfs:Literal is an instance of rdfs:Class. rdfs:Literal is a subclass of rdfs:Resource.

1.4 rdfs:Datatype

rdfs:Datatype is both an instance of and a subclass of rdfs:Class. Each instance of rdfs:Datatype is a subclass of rdfs:Literal.

1.5 rdf:XMLLiteral

The class rdf:XMLLiteral is the class of XML literal values. rdf:XMLLiteral is an instance of rdfs:Datatype and a subclass of rdfs:Literal.

1.6 rdf:Property

rdf:Property is the class of RDF properties. rdf:Property is an instance of rdfs:Class.

2. Properties

2.1 Rdfs:range

rdfs:range is an instance of rdf:Property that is used to state that the values of a property are instances of one or more classes.

The triple

P rdfs:range C

states that P is an instance of the class rdf:Property, that C is an instance of the class rdfs:Class and that the resources denoted by the objects of triples whose predicate is P are instances of the class C.

Where P has more than one rdfs:range property, then the resources denoted by the objects of triples with predicate P are instances of all the classes stated by the rdfs:range properties.

The rdfs:range property can be applied to itself. The rdfs:range of rdfs:range is the class rdfs:Class. This states that any resource that is the value of an rdfs:range property is an instance of rdfs:Class.

The rdfs:range property is applied to properties. This can be represented in RDF using the rdfs:domain property. The rdfs:domain of rdfs:range is the class rdf:Property. This states that any resource with an rdfs:range property is an instance of rdf:Property.

2.2 rdfs:domain

rdfs:domain is an instance of rdf:Property that is used to state that any resource that has a given property is an instance of one or more classes.

A triple of the form:

P rdfs:domain C

states that P is an instance of the class rdf:Property, that C is an instance of the class rdfs:Class and that the resources denoted by the subjects of triples whose predicate is P are instances of the class C.

Where a property P has more than one rdfs:domain property, then the resources denoted by subjects of triples with predicate P are instances of all the classes stated by the rdfs:domain properties.

The rdfs:domain property may be applied to itself. The rdfs:domain of rdfs:domain is the class rdf:Property. This states that any resource with an rdfs:domain property is an instance of rdf:Property.

The rdfs:range of rdfs:domain is the class rdfs:Class. This states that any resource that is the value of an rdfs:domain property is an instance of rdfs:Class.

2.3 rdf:type

rdf:type is an instance of rdf:Property that is used to state that a resource is an instance of a class.

A triple of the form:

R rdf:type C

states that C is an instance of rdfs:Class and R is an instance of C.

The rdfs:domain of rdf:type is rdfs:Resource. The rdfs:range of rdf:type is rdfs:Class.

2.4 rdfs:subClassOf

The property rdfs:subClassOf is an instance of rdf:Property that is used to state that all the instances of one class are instances of another.

A triple of the form:

C1 rdfs:subClassOf C2

states that C1 is an instance of rdfs:Class, C2 is an instance of rdfs:Class and C1 is a subclass of C2. The rdfs:subClassOf property is transitive.

The rdfs:domain of rdfs:subClassOf is rdfs:Class. The rdfs:range of rdfs:subClassOf is also rdfs:Class.

2.5 rdfs:subPropertyOf

The property rdfs:subPropertyOf is an instance of rdf:Property that is used to state that all resources related by one property are also related by another.

A triple of the form:

P1 rdfs:subPropertyOf P2

states that P1 is an instance of rdf:Property, P2 is an instance of rdf:Property and P1 is a subproperty of P2. The rdfs:subPropertyOf property is transitive.

The rdfs:domain of rdfs:subPropertyOf is rdf:Property. The rdfs:range of rdfs:subPropertyOf is rdf:Property.

2.6 rdfs:label

rdfs:label is an instance of rdf:Property that may be used to provide a human-readable version of a resource's name.

A triple of the form:

R rdfs:label L

states that L is a human readable label for R.

The rdfs:domain of rdfs:label is rdfs:Resource. The rdfs:range of rdfs:label is rdfs:Literal.

Multilingual labels are supported using the language tagging facility of RDF literals.

2.7 rdfs:comment

rdfs:comment is an instance of rdf:Property that may be used to provide a human-readable description of a resource.

A triple of the form:

R rdfs:comment L

states that L is a human readable description of R.

The rdfs:domain of rdfs:comment is rdfs:Resource. The rdfs:range of rdfs:comment is rdfs:Literal.

A textual comment helps clarify the meaning of RDF classes and properties. Such in-line documentation complements the use of both formal techniques (Ontology and rule languages) and informal (prose documentation, examples, test cases). A variety of documentation forms can be combined to indicate the intended meaning of the classes and properties described in an RDF vocabulary. Since RDF vocabularies are expressed as RDF graphs, vocabularies defined in other namespaces may be used to provide richer documentation.

Example

<academicStaffMember>Grigoris Antoniou</ academicStaffMember>

< lecturer>Michael Maher</ lecturer>

<course name="Discrete Mathematics">

<isTaughtBy>David Billington</isTaughtBy>

</course>

Question: What is the list of academic staff members?

Xpath will only return Grigoris Antoniou (//academicStaffMember)

human readers would have also included Michael Maher and David Billington because all lecturers are academic staff members (that is, *lecturer* is a sub-class of academicStaffMember. Courses are only taught by academic staff members.

RDF Schema Example

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns=http://www.w3.org/2000/01/rdf-schema#>
<rdfs:Class rdf:ID="lecturer">
    <rdfs:subClassOf rdf:resource=" academicStaffMember"/>
</rdfs:Class>
<rdfs:Class rdf:ID=" academicStaffMember">
    <rdfs:subClassOf rdf:resource=" staffMember"/>
</rdfs:Class>
<rdfs:Class rdf:ID="course">
    <rdfs:comment>The class of course</rdfs:comment>
</rdfs:Class>
```

RDF Schema Example

```
<rdfs:Property rdf:ID="involves">
    <rdfs:comment>
        It relates only courses to lecturers
    </rdfs:comment>

    <rdfs:domain rdf:resource="#course"/>
    <rdfs:domain rdf:resource="#lecturer"/>
</rdfs:Property>
<rdfs:Property rdf:ID="isTaughtBy">
    <rdfs:comment>
        Inherits its domain ("course") and range ("lecturer") from its
superproperty "involves"
    </rdfs:comment>
    <rdfs:subPropertyOf rdf:resource="#involves"/>
</rdfs:Property>
.....
</rdf:RDF>
```

3.3 Web Ontology Language -OWL 2

OWL takes the basic fact-stating ability of RDF and the class and property structuring capabilities of RDF Schema and extends them in important ways (Horrocks, Patel-Schneider, & Harmelen, 2003). OWL is a richer vocabulary description language for describing properties and classes, such as

- relations between classes (e.g., disjointness),
- cardinality (e.g., "exactly one"),
- equality,
- richer typing of properties,
- characteristics of properties (e.g., symmetry), and
- enumerated classes.

OWL 2 is an extension of OWL 1 to make the web more accessible to facilitate sharing and enrich ontology. RDF/XML syntax is the main and compatible exchange syntax in Semantic Web. There are also other serialization syntaxes such as Turtle (Berners-Lee & Beckett, Turtle - Terse RDF Triple Language - W3C Team Submission, 2008), XML serialization (Motik, Parsia, & Patel-Schneider, OWL 2 Web Ontology Language: XML Serialization, 2009) and Manchester Syntax (Horridge & Patel-Schneider, 2009).

There are two ways of assigning semantics to OWL 2, Direct Semantics (Motik, Patel-Schneider, & Grau, OWL 2 Web Ontology Language: Direct Semantics, 2009) and the RDF-Based Semantics (Schneider, OWL 2 Web Ontology Language: RDF-Based Semantics, 2009). The Direct Semantics approach uses part of the first order logic by assigning semantics directly to ontology structures. However, conditions like transitive properties cannot be expressed in the ontology structure. Ontologies that use Direct Semantics and restricted to those conditions are called OWL 2 DL (Description Logic) ontologies.

Unlike Direct Semantics approach, RDF-Based Semantics uses RDF graphs to interpret the semantics. The meaning reaches the ontology structure indirectly through the mapping between the ontology structure and RDF graph as shown in Figure 4 (W3C-OWLGroup, 2010). RDF-Based Semantics extends the RDF conditions and is fully compatible with RDF Semantics. Since any OWL 2 Ontology can be mapped to RDF and RDF-Based semantics is fully compatible with RDF, RDF-Based semantics can be applied to any OWL 2 Ontology. RDF

graphs considered as OWL 2 ontologies can be interpreted using RDF-Based Semantics and they are informally referred as "OWL 2 Full".

There is a close relationship between Direct Semantics and RDF-Based Semantics. Direct Semantics with OWL 2 DL ontology will still be interpreted in RDF-Based Semantics if the OWL 2 Ontology is mapped to an RDF graph as stated in Section 7.2 of the RDF-Based Semantics Document (Schneider, OWL 2 Web Ontology Language: RDF-Based Semantics, 2009).

There are three OWL 2 Profiles that are applicable in specific scenarios. They are sub-languages (syntactic subsets) of OWL 2 and more restrictive than OWL DL. They are named OWL 2 EL, OWL 2 QL, OWL 2 RL.

OWL 2 EL – used for large but relatively simple ontologies with good time performance rather than expression power. It uses polynomial time reasoning complexity algorithms.

OWL 2 QL – used to directly access data using relational query (e.g. SQL) stored in databases.

OWL 2 RL – used to interoperate with rules engines and rule extended DBMSs on data in the form of RDF triples. It uses polynomial time reasoning complexity algorithms.

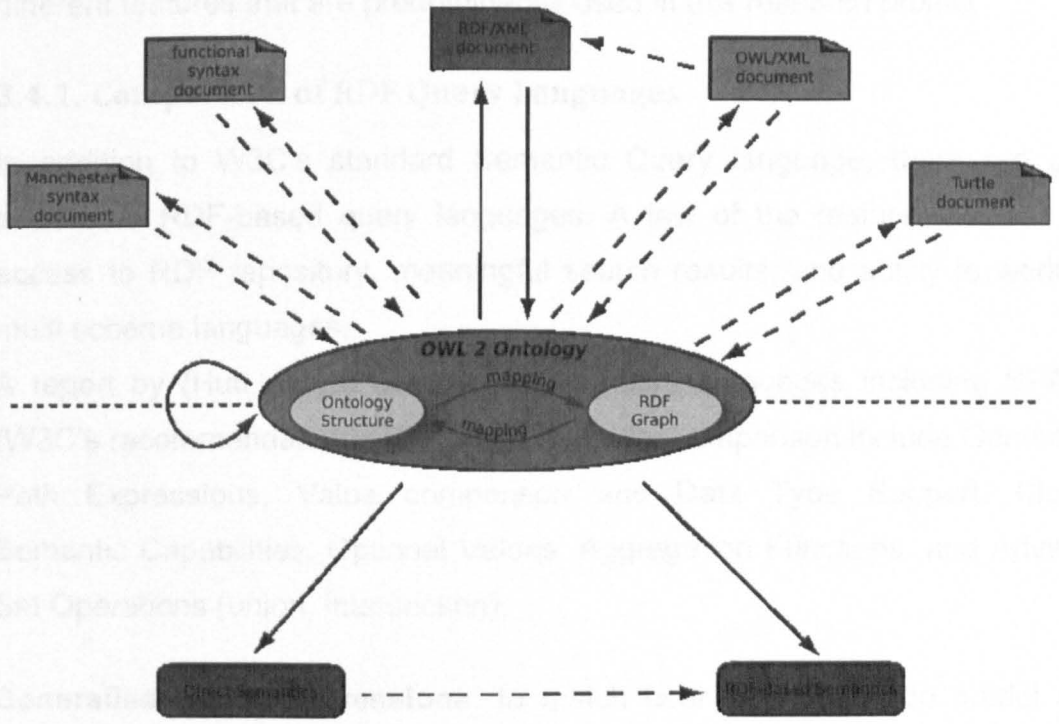


Figure 4- Structure of OWL 2 (W3C-OWLGroup, 2010)

OWL 2 is an extension of OWL 1 and is backward compatible with OWL 1. OWL 2 introduced new functionalities like syntactic sugar (e.g. disjoint union of classes) and new expressions like keys; property chains; richer datatypes, data ranges; qualified cardinality restrictions; asymmetric, reflexive, and disjoint properties; and enhanced annotation capabilities. The above three profiles (OWL EL, OWL QL, OWL RL) and OWL 2 Manchester Syntax are also new additions in OWL 2. All the new OWL 2 features are documented in OWL 2 New Features and Rationale document (Golbreich & Wallace, 2009).

3.4 Query Languages

The Semantic Web query language draws its analogy from is Structured Query Language (SQL). In SQL, a database is a closed world; the FROM clause identifies the tables in the database; the WHERE clause identifies constraints and can be extended with AND; by analogy, *the web is the database* and the FROM clause identifies the RDF models (Miller, Seaborne, & Reggiori, 2002).

The first sub-section compares the different available RDF query languages. It compares the different query language features based on the semantic web standards to choose a suitable query language.

The second sub-section discusses the chosen query language and explains the different features that are predominantly used in this research project.

3.4.1. Comparison of RDF Query Languages

In addition to W3C's standard Semantic Query language, there are a few alternative RDF-based query languages. A few of the features include easy access to RDF repository, meaningful search results, and ability to work with multi schema languages.

A report by (Hutt, 2005) compares four query languages including SPARQL (W3C's recommendation). The criteria used for comparison include Generalised Path Expressions, Value comparison and Data Type Support, Closure, Semantic Capabilities, Optional Values, Aggregation Functions, and Advanced Set Operations (union, intersection).

Generalised Path Expressions: to match user input to graph model using subject-property-object pattern, path expression syntax is required. These

include searching, substituting properties with variables, and constraining values using Boolean expressions.

Value comparison and Data Type support: to compare the values of properties, the data types in RDF model shall be recognised by the query languages.

Closure: the query run against RDF graph shall result the same RDF graph data pattern which in turn can also be used as an input to another query.

Semantic Capabilities: the query language shall provide support for built-in RDF meaningful search results, and ability to work with multi schema languages.

Optional Values: in addition to exact graph pattern matching, query languages shall provide optional (partial matching) of the graph pattern. This is equivalent to "outer join" in SQL.

Aggregate Functions: query languages shall provide support to aggregate functions like COUNT, MAX, AVG as in the case for SQL.

Advanced Set Operations: in addition to the three basic operations (selection, projection, Cartesian product), intersection and union shall be supported as part of Advanced Set Operations.

Table 5- Summary of Data Characteristics for RDF Benchmarks

	Data Type Sup	Path Exp	Closure	Seman Funct.	Opt Values	Agg Funct	Adv Set Ops
SeRQL	Yes	Yes	Yes	Yes	Yes	No	No
SPARQL	Yes	Yes	Yes	No	Yes	No	No
RDQL	Yes	Yes	No	No	No	No	No
XsRQL	Yes	Yes	No	No	No	Yes	No

SeRQL with strong support from the open source community and SPARQL with strong support from the W3C are leading the charge towards standardization (Hutt, 2005). Since then the introduction of features like "filter" makes SPARQL stronger than ever and it is currently W3C's standard Semantic Web Query Language.

3.4.2. SPARQL

SPARQL is W3C's standard Semantic Web Query Language that makes use of IRIs (a subset of RDF URIs without spaces), literals, language tags (@), blank node, etc. SPARQL has the same triple pattern as RDF except that each of the subject, predicate, and object may be a variable.

A simple SPARQL query consists of a SELECT and WHERE clause. The SELECT clause specify the projection that includes the number and order of the output results and the WHERE clause provides the basic graph pattern to match against the data graph. There is an optional FROM clause that you can specify the source and if not specified it assumes the whole knowledge base.

Building RDF Graphs

RDF uses a graph data model, where different entities are vertexes in the graph and relationships between them are represented as edges (Huang, Abadi, & Ren, 2011). RDF repositories represent RDF graphs using triples as "subject, predicate, object". When relational database tables are used as RDF repositories, a table of three columns are used to store each triple value. RDF graph representation uses an edge to connect the "subject" with the "object". Hence each edge represents a triple and as a result the number of RDF triples in a dataset is equal to the number of edges in a RDF graph (Huang, Abadi, & Ren, 2011).

The "CONSTRUCT" query form returns an RDF graph based on a template for RDF triples by matching the graph pattern of the query.

Data:

```
@prefix org:    <http://example.com/ns#> .

_:a  org:employeeName   "Alice" .
_:a  org:employeeId     12345 .

_:b  org:employeeName   "Bob" .
_:b  org:employeeId     67890 .
```

Query:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
PREFIX org:    <http://example.com/ns#>

CONSTRUCT { ?x foaf:name ?name }
WHERE  { ?x org:employeeName ?name }
```

Results:

```
@prefix org: <http://example.com/ns#> .

_:x foaf:name "Alice" .
_:y foaf:name "Bob" .
```

which can be serialised in RDF/XML as:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  >
      <rdf:Description>
            <foaf:name>Alice</foaf:name>
      </rdf:Description>
      <rdf:Description>
            <foaf:name>Bob</foaf:name>
      </rdf:Description>
</rdf:RDF>
```

Restricting values

"FILTER" in SPARQL is equivalent to "WHERE" clause in SQL. SPARQL uses functions like "regex" to match only plain literals with no language tag. "Regex" could also be used together with "str" function to match the lexical forms of other literals.

Query:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
        FILTER regex(?title, "^SPARQL") }
```

Query Result:

title
"SPARQL Tutorial"

Regular expression matches may be made case-insensitive with the "i" flag.

Query:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
        FILTER regex(?title, "web", "i" ) }
```

Query Result:

Title
"The Semantic Web"

JOIN Examples:

SPARQL uses "." operator to support SQL's "Join" and "OPTIONAL" keyword to support SQL's "Left Outer Join".

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?cWHERE{ ?c rdf:type rdfs:Class}

- **Implicit Join:**

  PREFIX uni: <http://www.mydomain,org/uni-ns#>

  SELECT ?x ?y WHERE{ ?x rdf:type uni:Lecturer ; Uni:phone ?y .}

  SELECT ?x ?y WHERE{ ?x rdf:type uni:Lecturer ?x Uni:phone ?y . }

- **Explicit Join:**

  Name of all course taught by the lecturer with ID 123

  SELECT ?n

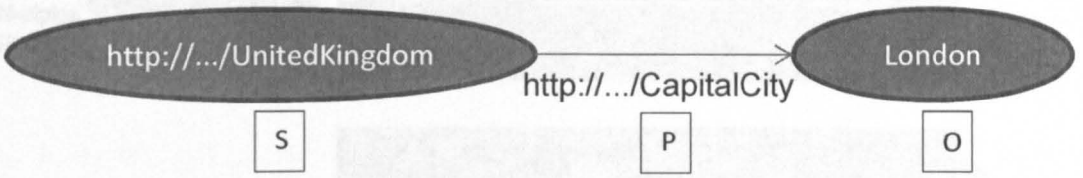
  WHERE{ ?x rdf:type uni:Course ; Uni:isTaughtBy :123 .

  ?c Uni:name ?n .

  FILTER (?c = ?x) . }

3.5 *Jena Framework*

Jena is a Semantic Web framework with features to interface RDF repositories and manage ontologies. After choosing a modelling language for the instances of data (RDF), ontology language for the repository (RDFS/OWL) and the query language (SPARQL) the next stage is choosing an appropriate ontology management framework. Jena is a java based open source ontology management utility considered as a tool to manage the semantic repository in this research project.



Data in Jena framework is structured in sets of RDF triples called *RDF Graph*. An RDF graph is simply a set of triples (S, P, O), where P names a binary predicate over (S, O) (Carroll, Dickinson, Dollin, Reynolds, Seaborne, & Wilkinson, 2004).

Jena supports several serialization languages like RDF/XML, N3, N-triple and turtle. It also has an option for a memory, file-based or database RDF persistence. Jena architecture provides different persistent, inference, RDF, Ontology, Query and related API's that could be invoked using Java programming language. Jena also provides HTTP interface to RDF data.

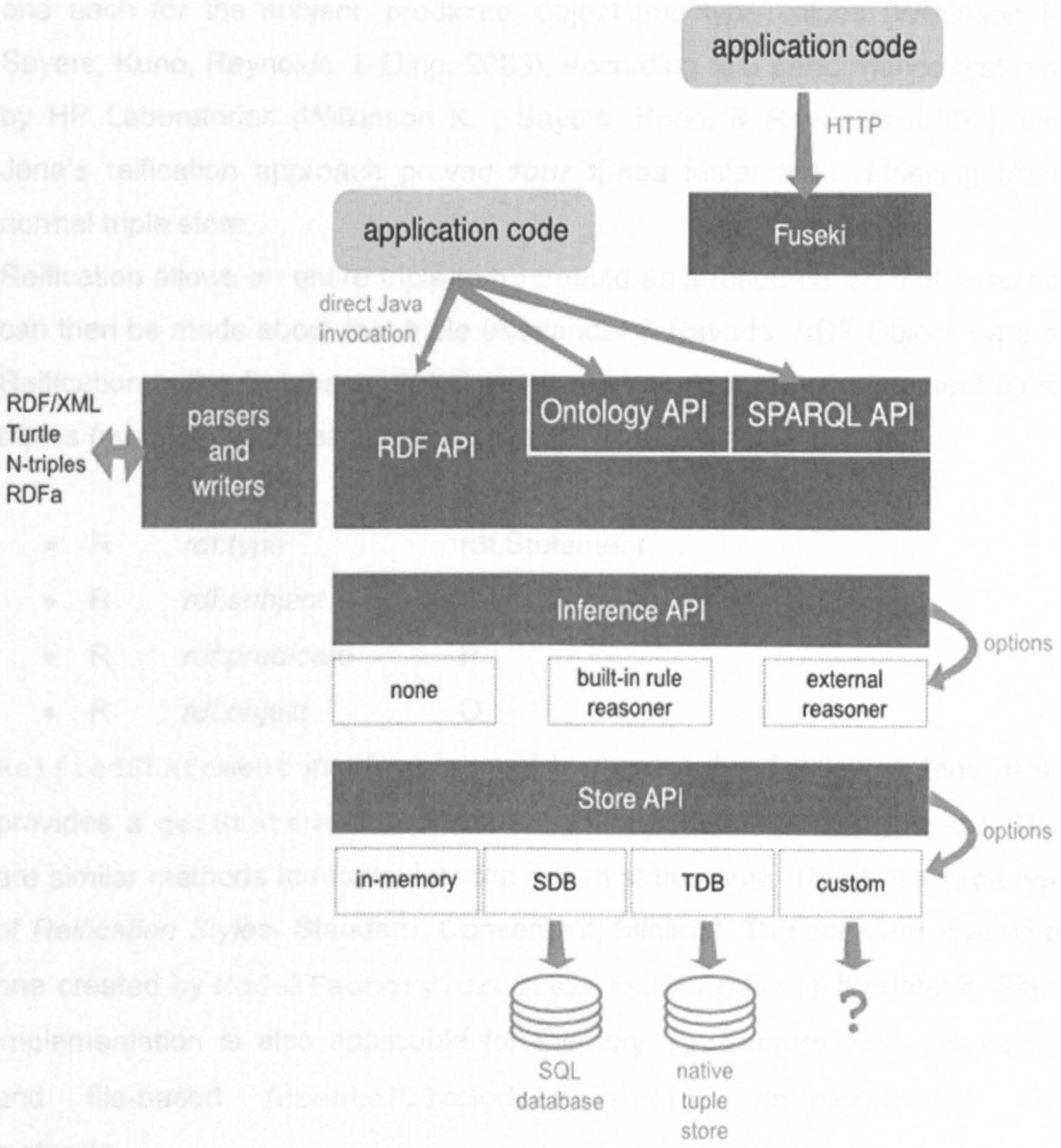


Figure 5- Jena2 Architecture Overview (Jena A. , 2012)

3.5.1. Reification

Data in Semantic Web is represented using RDF Graphs as (S, P, O) triples. In RDF, a reified triple is a description of a triple-token using other RDF triples (Hayes, 2004). An RDF Graph is considered a *statement* when a triple has an additional "*RDF:type*" property of value "*RDF:Statement*". RDF reification was intended as a mechanism for making provenance statements and other statements about RDF triples with a special vocabulary which includes rdf:Statement (Watkins & Nicole, 2006).

Reification in Jena is the representation of the original triple and the additional "statement type" property as a reification quad (four triples). It would be inefficient to store each reified statement as four separate triples. For reification, Jena uses a property table of five columns, one for the statement identifier and

one each for the subject, predicate, object and type values (Wilkinson K. , Sayers, Kuno, Reynolds, & Ding, 2003). According to a performance test report by HP Laboratories (Wilkinson K. , Sayers, Kuno, & Reynolds, 2003), using Jena's reification approach proved ***four times*** faster than retrieving from a normal triple store.

Reification allows an entire triple to be treated as a resource, so that assertions can then be made about that triple (Alexander & Ravada, RDF Object Type and Reification in the Database, 2006). Jena represents a reified statement as four triples (quadlet) to represent a resource R.

- R *rdf:type* rdf:Statement
- R *rdf:subject* S
- R *rdf:predicate* P
- R *rdf:object* O

`ReifiedStatement` interface is used to represent reification in Jena. It also provides a `getStatement()` method to find statements being reified. There are similar methods to manipulate the reified statements. There are three types of *Reification Styles*- Standard, Convenient, Minimal. The standard style is the one created by `ModelFactory.createDefaultModel()` by default. Similar implementation is also applicable for memory `(createMemModelMaker())` and file-based `(createFileModelMaker())` `ModelFactory` class methods.

- Convenient- reification quadlets are not visible in the `listStatements()` but statements that are added to the model contribute to the `ReifiedStatement` construction.
- Minimal – reification quadlets play no role at all in the construction of `ReifiedStatements`, which can then be created only by methods discussed earlier

3.5.2. Jena RDF API

Jena provides a java API that contains interfaces and classes to create and manipulate RDF repositories. The API provides different transactional, query, operational and navigation interface classes to work with the RDF models.

Each (S, P, O) triple set in RDF model that conforms to the reification definition in the previous section is called a *resource*. The subject (S) and object (O)

nodes are linked by a predicate (P). Predicate (P) linking the two nodes can only be a named resource (has URI). The subject (S) can be a named or unnamed resource, while the object (O) can be a named, unnamed, or a literal.

A Set of statements or triples represent an *RDF Model*. A Model in Jena is created using "ModelFactory" as:

```
Model model = ModelFactory.createDefaultModel();
```

A model can be read from a URL or a file. A model output could also be written to a file with an optional language and base URI using one of the following serialization languages.

- RDF/XML
- RDF/XML-ABBREV
- N3
- N-TRIPLE
- TURTLE

```
// create a model
Model model = ModelFactory.createDefaultModel();
model.read( "file:/test.rdf" );
model.write( System.out );
```

Different operations can be performed on RDF models using Jena API. Models can be merged and their output could be explicitly specified using one of the available serialization languages.

```
// read the RDF/XML files
model1.read(new InputStreamReader(file1), "");
model2.read(new InputStreamReader(file2), "");

// merge the Models
Model model = model1.union(model2);

// print the Model as N3
model.write(system.out, "N3");
```

Intersection and difference operations can also be performed with a different output serialization.

```
// intersect the Models
Model modelInt = model1.intersection (model2);

// print the Model as RDF/XML
modelInt.write(system.out, "RDF/XML ");

// difference the Models
Model modelDiff = model1.difference (model2);

// print the Model as N-triple
modelDiff.write(system.out, "N-TRIPLE");
```

Resources in RDF graphs with equivalent URI's are considered the same and they can also be created within a program and added to models.

```
// create the resource
Resource resource = model.createResource();

// add the property
r.addProperty(RDFS.label,
model.createLiteral("London", "en"));

RDFList list=model.createList();
list.add(resource);
```

Storing RDF (Store API) is based on a simple RDF Graph data structure. RDF persistency can be done in memory, using files or databases. Many relational database systems are compatible to RDF persistence data structure (MySQL, PostgreSQL, Oracle, Microsoft SQL Server, HSQLDB, Derby)

Table 6- Summary of Jena RDF API Sample Java methods

Type	Method	Description
Transactional	createResource	Creates a resource for them model

	read	Reads RDF statement from RDF/XML or other specified language source
	write	Writes a serialised representation of the model in a specified language
Navigation	getResource	Returns a resource instance
	getProperty	Returns a property instance
Querying	listStatements	List all statements
	listSubjects	List all resources specified as subjects of statements
	listObjects	List all objects in a model
Operations	union	Creates a new model containing all the statements of the two models
	intersection	Creates a new model containing all the statements which are common in both models
	difference	Creates a new model containing statements that are in this model and not in the other
Containers	BAG	Unordered collection of literals or resources
	ALT	Unordered collection of literals or resources intended to represent alternatives
	SEQ	Ordered collection of literals or resources

3.5.3. Jena Ontology API

Jena provides a java API that contains interfaces and classes to manage ontology repositories. The API provides different ontology languages, reasoners, inference, and complex expressions to work with the ontology models.

The simplest ontology language compatible to RDF is *RDF Schema* (RDFS). Jena is also compatible to the three different levels of OWL ontology language- OWL Lite, OWL DL, OWL Full. Jena Ontology API provides an interface which is language neutral that can use a *profile* to set specific java classes and properties. For instance, the URI for "ObjectProperty" in DAML (DARPA Agent Markup Language) profile is `daml:ObjectProperty` while in OWL it is `owl:ObjectProperty`. The same URI in RDFS is `null` as there is no "ObjectProperty" implementation in RDFS profile.

Jena supports basic characteristic of polymorphism at the RDF level by considering that the Java abstraction (OntClass, Restriction, DatatypeProperty, etc.) is just a view or *facet* of the resource (Carroll, Dickinson, Dollin, Reynolds, Seaborne, & Wilkinson, 2004). For example, if we declare a resource *#DigitalCamera* as an ontology class that could be represented by a java instance of *OntClass*.

```
<owl:Class rdf:ID="DigitalCamera">
</owl:Class>
```

This same resource can be an OWL *Restriction* which proves that there is no unique mapping between RDF resources and Java abstraction.

```
<owl:Class rdf:ID="DigitalCamera">
     <rdf:type owl: Restriction/>
</owl:Class>
```

Jena provides *as()* method to create a new facet on run-time depending on the resource property. The following example creates a resource (res) and instantiates two facets of the same resource that shows the flexibility of Jena in managing Ontology.

```
    Resource res = myModel.getResource( myNS +
"DigitalCamera" );
    OntClass cls = res.as(OntClass.class);
    Restriction rest = cls.as(Restriction.class);
```

An ontology model is an extension of the Jena RDF model, providing extra capabilities for handling ontologies. Ontology models are created through the Jena ModelFactory (Jena O. , 2012).

```
// create ontology model
Model model = ModelFactory.createOntologyModel();
```

This creates an ontology with the following default settings:

- OWL-Full language
- In-memory storage
- RDF inference (i.e. entailments from sub-class and sub-property)

If the ontology model is for a simple model display, then inferencing is unnecessary and a model should be created with no reasoner (OWL_MEM).

```
OntModel model = ModelFactory.createOntologyModel(
OntModelSpec.OWL_MEM );
```

To create an ontology model with a built in or custom specification ModelFactory should be invoked as follows.

```
OntModel m = ModelFactory.createOntologyModel( <model spec>
);
```

In OWL a meta-data about the ontology can be set in the ontology using owl:Ontology.

```
<owl:Ontology rdf:about="SpaceKnowledgeManagement">
    <rdfs:comment>Space                    Management
Ontology</rdfs:comment>
    <rdfs:label>Mapping                    Relational
Databases</rdfs:label>
    <owl:versionInfo>1.0</owl:versionInfo>
</owl:Ontology>
```

The resources represented in the Ontology create a taxonomic hierarchy. The relationship between the different classes, properties, relations, restrictions, axioms etc. create a direct (*asserted*) and indirect (*inferred*) link amongst the RDF graph.

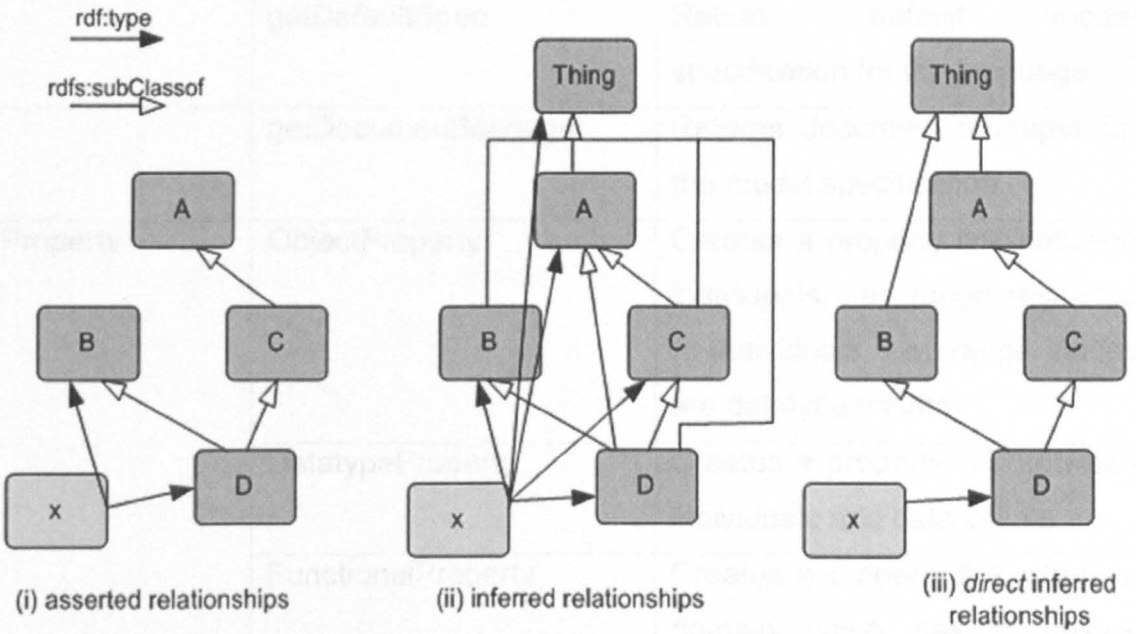


Figure 6- Asserted and Inferred Relationships (Jena O. , 2012)

The distinction between the asserted and inferred relationships helps to organise the Ontology into "facts" and "deductions". Jena's listRDFTypes() is one of the methods to list different types of resources in the Ontology.

```
// Shows direct relationships only if direct=true,
// else shows indirect relationships
listRDFTypes( boolean direct )
```

Table 7- Summary of Jena Ontology API Sample Java methods

Type	Method	Description
Model	createOntologyModel	Creates a new ontology model based on ontology specification, existing model, and/or specified language
	createInfModel	Builds an inferred model using an inferred graph, RDF model, and/or a specified reasoner
	createRDFSModel	Returns both instance data, and RDFS assertions
Specification	OntModelSpec	An encapsulation of ontology model, reasoner and language specification
	getProfile	Returns ontology language profile

	getDefaultSpec	Returns default model specification for the language
	getDocumentManager	Returns document manager for the model specification
Property	ObjectProperty	Creates a property link between individuals, i.e., range restricted to individuals, i.e., range values are datatype values
	DatatypeProperty	Creates a property link between individuals and data values
	FunctionalProperty	Creates a property link where a domain value has a unique value in the range
	InverseFunctionalProperty	Creates a property link where a range value has a unique value in the domain
	TransitiveProperty	A property that denotes transitivity, i.e., if `x p y` holds, and `y p x` holds, then `x p z` must also hold
	SymmetricProperty	A property that denotes symmetricity, i.e., if `x p y` holds, and `y p x` must also hold
Restriction	hasValue	Restricts the property to the specified values
	minCardinality	Restricts the property to at least **n** values
	maxCardinality	Restricts the property to at most **n** values
Expression	intersectionOf	Selects individual resources that belong to both classes
	unionOf	Selects individual resources that belong to any of the classes

	complementOf	Selects individual resources from the domain that don't belong to the specified class

3.5.4. Reasoners and Rule Engines

Jena provides an open platform to use both built-in and third party inference engines. Based on RDFS and OWL ontology languages, Inference API provides "*reasoners*" that could be registered to the Model and produce additional resources on top of the asserted statements.

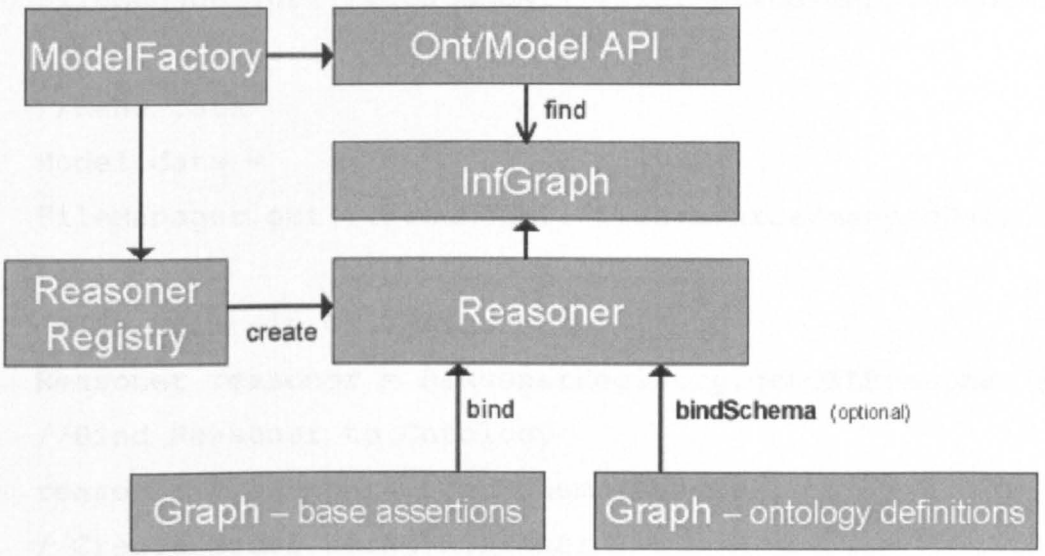


Figure 7- Jena2 Inference Overview (Jena I. , 2012)

`ModelFactory` is used to associate reasoners with a `Model`. Inference is implemented at Graph SPI level so the different model interfaces could share the result. Ontology API provides `OntModelS` to link reasoners to models. Jena also provides `InfModel`, which is an extension of `Model` that provides additional control over the underlying `Graph`.

Methods like `createRDFSModel` provide built-in RDFS inference rules with basic implementation. For different built-in and generic reasoning systems, `Reasoner`s are required. `ReasonerRegistry` static class is used to register reasoners dynamically ranging from built-in *transitive, RDFS, OWL* to *generic user-defined* rule reasoners.

- getOWLReasoner(): prebuilt standard OWL inference reasoner
- getRDFSReasoner(): prebuilt standard RDFS inference reasoner

- getTransitiveReasoner(): prebuilt subclass and subproperty transitive closure reasoner
- Generic User-defined: different forward/backward chaining and hybrid executions

The example below shows an excerpt of a Jena inference implementation using *OWL Ontology Schema, RDF Data* and a built in *OWL*.

```
//Read Ontology
Model schema =
FileManager.get().loadModel("file:source/mappedSchema.ow
l");
//Read Data
Model data =
FileManager.get().loadModel("file:source/mappedData.rdf"
);
//Get built-in OWL Reasoner
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
//Bind Reasoner to Ontology
reasoner = reasoner.bindSchema(schema);
//Create Model using Reasoner
InfModel infmodel =
ModelFactory.createInfModel(reasoner, data);
```

A similar Jena implementation below shows an inference program excerpt using *user-defined* rule instead of a built in reasoner.

```
//Read Ontology
Model schema =
FileManager.get().loadModel("file:source/mappedSchema.ow
l");
//Read Data
Model data =
FileManager.get().loadModel("file:source/mappedData.rdf"
);
/* Set User-defined rule*/
```

```
String ruleString = [ transiveChainSubClassOf: (?x
rdfs:subClassOf ?y),(?y rdfs:subClassOf ?z) -> (?x
rdfs:subClassOf ?z) ];
//Parse Rule
List rules = Rule.parseRules(ruleString);
//Create User-defined Reasoner
Reasoner reasoner = new GenericRuleReasoner(rules);
//Bind Reasoner to Ontology
reasoner = reasoner.bindSchema(schema);
//Create Model using Reasoner
InfModel inf = ModelFactory.createInfModel(reasoner,
data);
```

3.6 *Rule Interchange Format (RIF) as an ontology of Rules*

Rules are used as a way of representing complex knowledge in semantic web. Rule Interchange Format (RIF) is used as part of the "Rule Layer" in the Semantic Web Architecture to represent more knowledge on top of the raw data to share and interchange rules.

RIF is used to share and exchange rules by embedding RIF *meta-information* in an RDF Graph to make use of the rules as an intuitive form of knowledge representation. RIF is compatible with Web Ontology Languages- RDFS/OWL and also has a framework to accommodate dialects of many rule languages.

RIF represent different kinds of rules (deductive, normative, and reactive rules) and can process complex reasoning that are difficult for Web Ontology languages such as

- "default reasoning (default logic)": E.g. Risk severity for mission "X" is 10% unless stated otherwise
- "uncertainty reasoning": E.g. what shall be the outcome if the conclusions from the exploration data are uncertain
- "paraconsistent reasoning": E.g. How shall the system deal with the inconsistency when merging the corresponding reference numbers between project "X" and "Y"

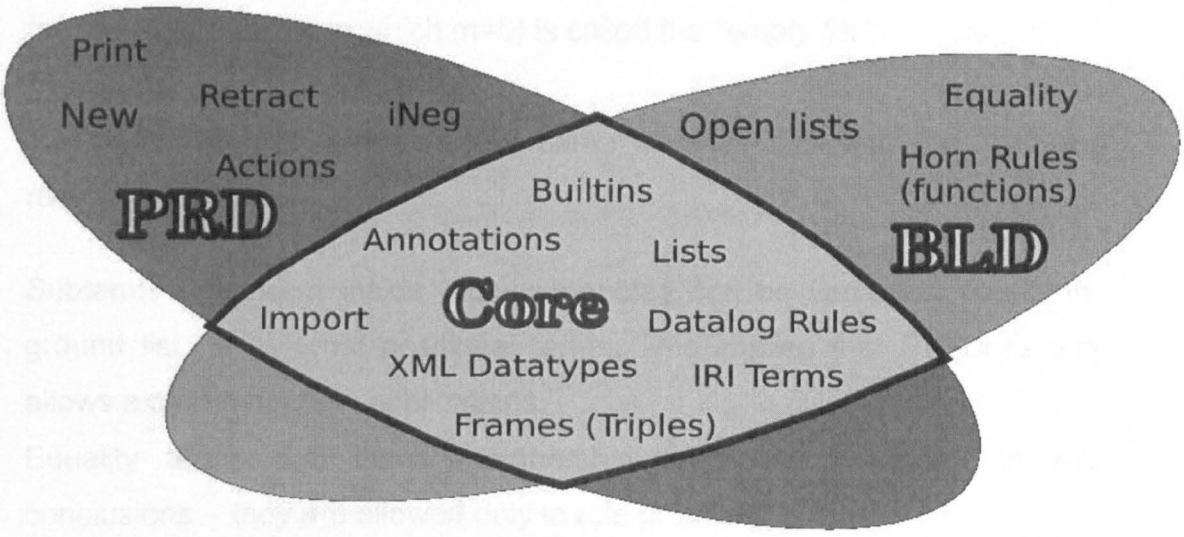


Figure 8- RIF Framework

3.6.1. RIF – Core

RIF-Core implements Horn logic without function symbols i.e. DataLog

if p(...) and q(...) then r(...)

This dialect consists of set of rules implemented by all the dialects. It uses IRI's to name predicate values (same as RDF). It supports XML datatypes: xsd number, strings, date, etc.

RIF-Core is a syntactic subset of both RIF-BLD and RIF-PRD. Its XML-syntax is also a subset of the XML Syntax of RIF-BLD except "Subclass", "sub", and "super" XML tags.

RIF-Core Presentation Syntax

1. **Alphabet**

   Alphabet of RIF- Core is the same as RIF-BLD without the "subclass" symbol(##) and the set of "ArgNames" (used for named-argument uniterms)

2. **Terms**

   The "Terms" of RIF-Core are also the same as of RIF-BLD with the exclusion of "subclass" and "named arguments" terms. In RIF-Core there are only "closed ground" lists.

A "closed ground list" has the form List(t1 ... m), where m>=0 and t1,..., tm are ground terms (no tail and no variables are allowed). A closed list of the form List() (i.e., a list in which m=0) is called the *"empty list"*.

3. **Formulas**

RIF-Core formulas are also the same as RIF-BLD with the following restrictions.

- Subterms that occur inside atomic formulas can be variables, constants, ground list, or external positional terms. This implies that RIF-Core only allows external function applications.
- Equality terms and class membership terms *cannot* occur in rule conclusions -- they are allowed only in rule premises.
- Terms with named arguments and subclass terms are excluded from RIF-Core.

4. **Annotations**

Annotation in RIF-Core is optional and is done in the same way as in RIF-BLD. The only restriction on RIF-Core is on functions. The frame formulas allowed as part of the annotation shouldn't have function symbols.

5. EBNF Grammar (**Extended Backus–Naur Form**)

The entire EBNF is divided into three parts as derived from EBNK for RIF-BLD with restrictions.

- Rule language
- Condition language
- Annotations

The whole EBNF syntax could be read at online as Rule Interchange on the Web (Boley, Kifer, Patranjan, & Polleres, 2007).

3.6.2. RIF-Core as a Specialization of RIF-PRD

1. **Alphabet**

Alphabet of RIF- Core is the same as RIF-PRD presentation language (Conditions, Actions, and Rules) without the symbols ##, *such that*, *Not*, *INeg, Do, Assert, Retract, Modify, Execute*, and *New*

2. **Terms**

The "Terms" of RIF-Core are also the same as of RIF-PRD with the exclusion of "subclass". In RIF-Core there are only "closed ground" lists.

3. **Formulas**

   RIF-Core formulas are also the same as RIF-PRD with the exclusion of *negation* formulas.

4. **Annotations**

   Every term and formula is optionally annotated in RIF-Core and is done in the same way as in RIF-PRD. The frame formulas must be syntactically correct for RIF-Core.

5. **Rules and Groups**

   A RIF-Core rule is a well-formed RIF-PRD rule with no nested forall, no binding pattern, and where the action block is a single atom, a single frame, or a conjunction of atoms and/or frames. A RIF-Core group is a RIF-PRD group without *strategy* and without *priority*.

3.6.3. RIF – Basic Logic Dialect (BLD)

The Basic Logic Dialect of Rule Interchange Format is a specialization of the *RIF Framework for Logic Dialects (*RIF-FLD). It uses Horn rules without negation or action.

if p(...,f(?x, 3), ..) ... Then z(..., g(?y), ...)

It supports objects and frames like F-Logic ontology language. It uses Internationalised Resource Identifiers (IRI) and XML schema data-types. It is also web aware language that can import RDF and OWL (Bruijn, 2010).

RIF-BLD doesn't support negation while "Condition" and "Conclusion" are monotonic. You cannot change the value of any predicate, all statements are either true or false (as in OWL or RDF) unlike RIF- Production Dialect which we will see in the next section.

RIF-BLD Presentation Syntax

XML is the only concrete syntax for RIF-BLD as both mathematical English and Extended Backus-Naur Form (EBNF) don't fully support some of the symbols. Mathematical English doesn't representation delimiters, escape symbols, precedence of operators, parenthesizing, and the like. EBNF also lacks the representation of context-sensitive syntactic constraints.

1. **Alphabet of RIF-BLD**

The Alphabet of RIF-BLD presentation language consists of:

- constant symbols (Const). They are written as "literal"^^symspace, where literal is a sequence of Unicode characters and symspace is an identifier for a symbol space.
- variable symbols (Var) which are disjoint from Const and are written as Unicode strings preceded with the symbol "?"
- argument names (ArgNames) which are disjoint from both Const and Var. They are written in Unicode strings that must not start with a question mark, "?"
- connective symbols (And, Or, :-)
- quantifiers (Exists, Forall) and the following symbols.
- the symbols =, #, ## (for formulas that define equality, class membership, and subclass relationships), -> (for terms that have named arguments and in frame formulas), External (indicates that an atomic formula or a function term is defined externally), Import (import directive), Prefix, and Base (enable compact representations of IRIs)
- the symbols Group (to organise RIF-BLD formulas into collections) and Document (to specify RIF-BLD documents)
- the symbols for representing lists: List and OpenList.
- the auxiliary symbols (,), [,], <, >, and ^^

The above alphabets are used to construct RIF-BLD language as a set of formulas according to the rules given below.

2. Terms

The terms in RIF-BLD include: constants and variables, positional terms, terms with named arguments, plus equality, membership, subclass, frame, and external terms.

Definition (Term).

- Constants and variables. If t ε Const or t ε Var then t is a simple term
- Positional terms. If t ε Const and t_1, ..., t_n, n≥0, are base terms then t(t_1 ... t_n) is a positional term.
- Terms with named arguments. A term with named arguments is of the form t(s_1->v_1 ... s_n->v_n), where n≥0, t ∈ Const and v_1, ..., v_n are base terms and s_1, ..., s_n are pairwise distinct symbols from the set ArgNames.
- List terms. There are two kinds of list terms: open and closed.

- A closed list has the form List(t_1 ... t_m), where m≥0 and t_1, ..., t_m are terms.
- An open list (or a list with a tail) has the form OpenList(t_1 ... t_m t), where m>0 and t_1, ..., t_m, t are terms. Open lists are usually written using the following: List(t_1 ... t_m | t).

A closed list of the form List() (i.e., a list in which m=0, corresponding to Lisp's nil) is called the empty list.

- Equality terms. t = s is an equality term, if t and s are base terms.
- Class membership terms (or just membership terms). t#s is a membership term if t and s are base terms.
- Subclass terms. t##s is a subclass term if t and s are base terms.
- Frame terms. t[p_1->v_1 ... p_n->v_n] is a frame term (or simply a frame) if t, p_1, ..., p_n, v_1, ..., v_n, n ≥ 0, are base terms.

  Membership, subclass, and frame terms are used to describe objects and class hierarchies.
- Externally defined terms. If t is a positional or a named-argument term then External(t) is an externally defined term.
- External terms are used for representing built-in functions and predicates as well as "procedurally attached" terms or predicates, which might exist in various rule-based systems, but are not specified by RIF.

3. Formulas

- **Atomic Formula**: any term of the form p(...), where p is a predicate symbol.
  - Terms equality, membership, subclass, and frame.
  - External defined term of the form External (φ), where φ is an atomic formula, is also an atomic formula.

  Simple terms like constants and variables are not formulas. With the help of logical connectives more general formulas can be constructed from atomic formulas.
- **Condition Formula**: is an atomic formula or one of the following forms
  - Conjunction : a formula that is always true, tautology

    If φ1, ..., φn, n ≥ 0, are condition formulas then so is And(φ1 ... φn)
  - Disjunction: a formula that is always false

    If φ1, ..., φn, n ≥ 0, are condition formulas then so is Or(φ1 ... φn)

- Existentials: If φ is a condition formula and ?V1, ..., ?Vn, n>0, are distinct variables then Exists ?V1 ... ?Vn(φ) is an *existential* formula.

Condition formulas are intended to be used inside the premises of rules as seen in the next bullet points.

- **Rule implication**: φ :- ψ is a formula, called rule implication, if:
  - φ is an atomic formula or a conjunction of atomic formulas,
  - ψ is a condition formula, and
  - none of the atomic formulas in φ is an externally defined term (i.e., a term of the form External(...)). Note: external terms can occur in the arguments of atomic formulas in the rule conclusion. For instance, p(func:numeric-add(?X,"2"^^xs:integer)) :- q(?X).
- **Universal rule**: If φ is a rule implication and ?V1, ..., ?Vn, n>0, are distinct variables then Forall ?V1 ... ?Vn(φ) is a formula, called a universal rule.
  - It is required that all the free variables in φ occur among the variables ?V1 ... ?Vn in the quantification part.
  - An occurrence of a variable ?v is free in φ if it is not inside a subformula of φ of the form Exists ?v (ψ) and ψ is a formula.
  - Universal rules will also be referred to as RIF-BLD rules.
- **Universal fact**: If φ is an atomic formula and ?V1, ..., ?Vn, n>0, are distinct variables then Forall ?V1 ... ?Vn(φ) is a formula, called a universal fact, provided that all the free variables in φ occur among the variables ?V1 ... ?Vn.
  - Universal facts are often considered to be rules without premises.
- **Group**: If φ1, ..., φn are RIF-BLD rules, universal facts, variable-free rule implications, variable-free atomic formulas, or group formulas then Group(φ1 ... φn) is a group formula. As a special case, the empty group formula, Group(), is allowed and is treated as a tautology, i.e., a formula that is always true.

  Non-empty group formulas are used to represent sets of rules and facts. Note that some of the φi's can be group formulas themselves, which means that groups can be nested.
- **Document**: An expression of the form Document(directive1 ... directiven Γ) is a RIF-BLD document formula (or simply a document formula), if

- Γ is an optional group formula; it is called the group formula associated with the document.
- $directive_1$, ..., $directive_n$ is an optional sequence of directives. A directive can be a base directive, a prefix directive or an import directive.

A document formula can contain at most one Base directive. The Base directive, if present, must be first, followed by any number of Prefix directives, followed by any number of Import directives.

In the definition of a formula, the component formulas φ, φi, ψi, and Γ are said to be subformulas of the respective formulas (condition, rule, group, etc.) that are built using these components.

3.6.4. RIF – Production Dialect (PRD)

RIF-PRD has actions instead of logical statements in the "then" part of the conditional statement.

ECA rules: ON Event IF Condition DO Action

Condition and conclusion are non-monotonic

IF (user.level = "admin") THEN (user.access = 100%)

This will change the value of user.access if it was 20% for example, before the rule (as in a programming language)

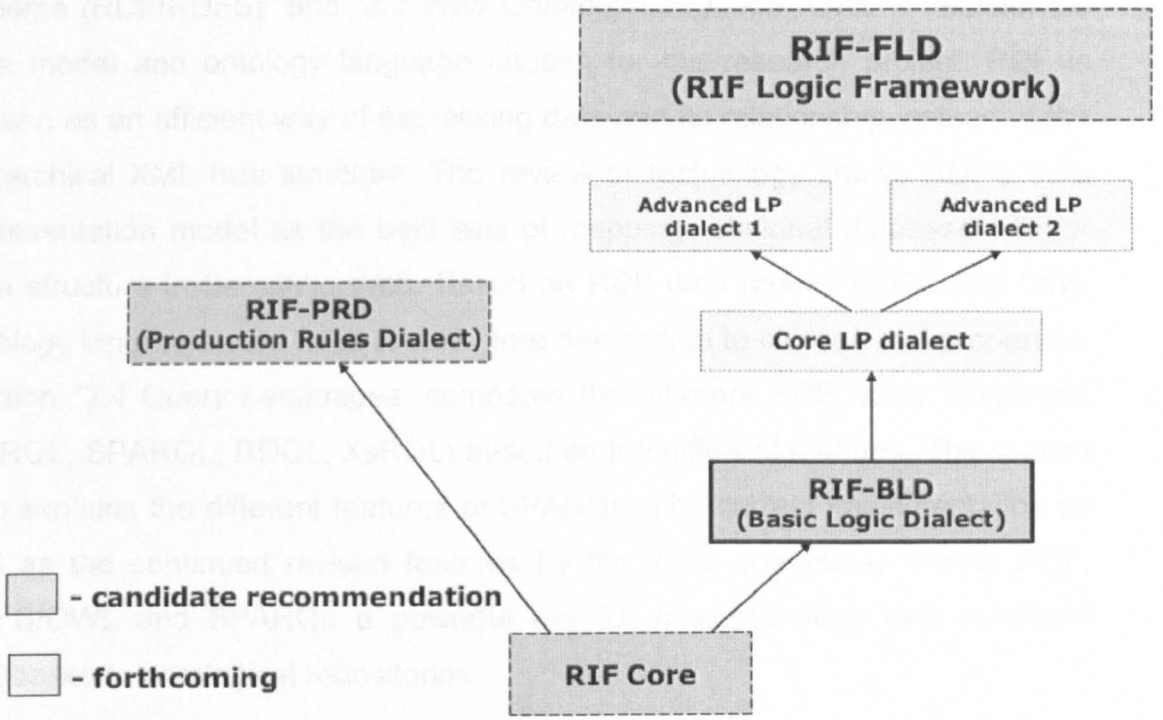


Figure 9- Current RIF Architecture

RIF Example

- RDF triples of the form s p o
  - ex:john ex:brotherOf ex:jack .
  - ex:jack ex:parentOf ex:mary .
- RIF frame formulas of the form s'[p' -> o']

```
Forall ?x ?y ?z (?x[ex:uncleOf -> ?z] :- And(?x[ex:brotherOf -> ?y]
                ?y[ex:parentOf -> ?z]))
```

3.7 Summary (Review of Technology)

Mapping relational database to ontological repositories involves most of the utilities analysed in sections 3.1 to 3.6. The utilities can be categorised into three groups based on their use in this research project.

a) W3C standard and used: *URI/IRI, XML, RDF, RDFS, OWL, SPARQL*

b) Non-W3C standard and used: *Jena*

c) W3C standard but not used: *RIF*

Section *"3.1* XML*"* analyses the basic framework of the Semantic Web world and its hierarchical data structure. RDF, RDFS and OWL base their implementation on XML. Sections *"3.2* Resource Description Framework and Schema (RDF/RDFS)*"* and *"3.3 Web Ontology Language -OWL 2"* explain the data model and ontology language chosen for the research project. RDF is chosen as an efficient way of expressing data and its relationship instead of the hierarchical XML tree structure. The review of technology shows RDF's data representation model as the best way of mapping relational database's linear data structure in Semantic Web. Based on RDF data model, RDFS and OWL ontology languages are used to add more description to classes and properties. Section *"3.4 Query Languages"* compares the different RDF query languages (SeRQL, SPARQL, RDQL, XsRQL) based on the different features. The section also explains the different features of SPARQL. The current implementation as well as the continued revised features by the W3C community makes RDF, RDFS/OWL and SPARQL a powerful tool for mapping deep web relational databases to ontological repositories.

Section *"3.5 Jena Framework"* discusses one of the major open source ontology management frameworks that will be used to manage the mapped ontology. Even though the framework is not yet indorsed by W3C community, it fills the gap for a lack of ontology management tool in Semantic Web. Section *"3.6 Rule Interchange Format (RIF) as an ontology of Rules"* is the last section in this chapter that looks into the "inter-ontology" or "inter-rule/logic" exchange to make the ontology open to different rules and logic. RIF is adopted as W3C standard and by the time of writing this thesis it is in its draft phase. The discussion about RIF and its dialects is included in this chapter as a future implementation and promising standard for a more complete relational to ontological mapping implementation.

Based on the review of literature and technology, suitable Semantic Data Structure (RDF), Ontology Languages (RDFS/OWL), Query Language (SPARQL), and reasoning framework (Jena) are selected as a mapping tool in this research project. The ontology is going to be based on space project management domain that would define high-level terms and concepts that are used in the space project management. Currently there is no generally accepted space project management ontology and in the process of this research sample space-domain ontology will be outlined to test the mapping.

Chapter 4: Domain Specific Knowledge

The mapping algorithms for converting relational databases into ontological repositories account several different types of knowledge: related to the relational model itself (*relational model knowledge*), related to the data stored in the database (*domain data knowledge*), related to the use of data (*domain users knowledge*) and knowledge about the database application (application knowledge). The conversion of the relational database is performed in three subsequent stages: *pre-processing*, during which the keys, constraints, relationships, and patters are identified, *in-processing*, which incrementally maps the relational schema and data and *post-processing*, which enriches the generated ontological repository to do semantic inferencing and account additional application domain knowledge. After mapping the relational database to semantic RDF repository, different semantic rules are also applied to analyse the domain for further understanding and discovery of knowledge.

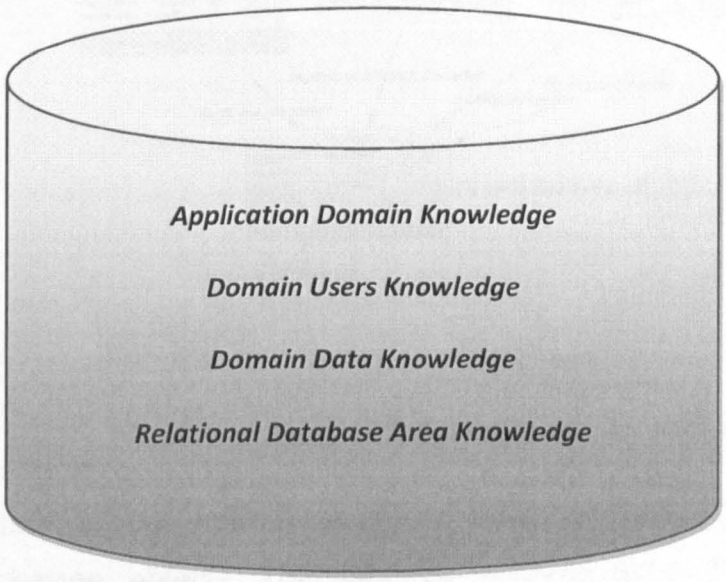


Figure 10- Mapping Knowledge Layer

The use of the existing meta-data and different levels of knowledge as listed above in Figure 10- Mapping Knowledge Layer facilitates the efficient mapping representation of relational databases in Semantic Web.

4.1. Relational Database area Knowledge

The relational database area knowledge is used to identify the tables and columns to be considered in the mapping as well as the database constraints and data type restrictions on table columns. The use of data dictionary, which is

specific to a particular relational database to be mapped, allows us to program the mapping as an iterative procedure that accounts different relational objects in the relational schema (relations, attributes, keys, constraints etc.).

The relational database consists of different relational objects that are grouped into relational schemas. The tables and columns contain the data that is going to be mapped. In relational database constraints are used to keep the integrity of the data. The constraints need to be mapped together with the data to maintain the integrity after mapping.

The database uses data type restrictions to guarantee data consistency during storing, retrieving and processing operations. The standard SQL data types are considered during the *pre-processing* phase where all the existing data types are identified. These SQL data types are also mapped using an equivalent semantic RDF data type during the *in-processing* phase where the mapping transformation is executed in an incremental iterative procedure.

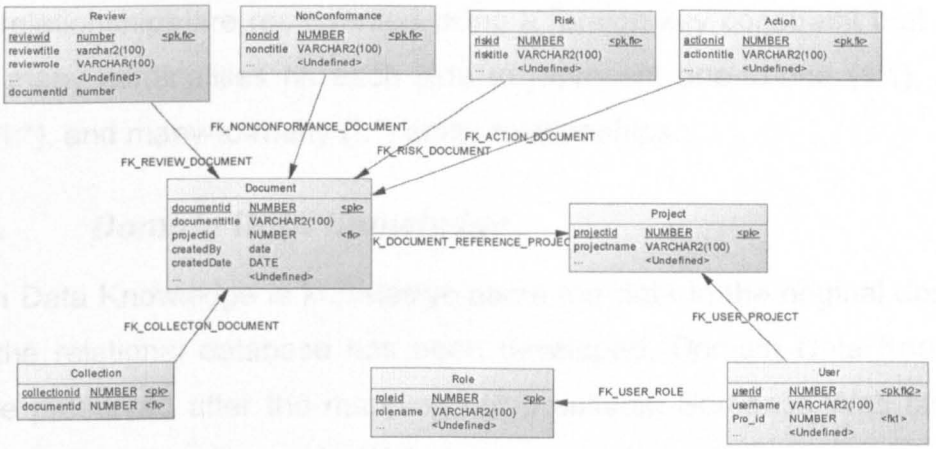


Figure 11- Relational Database Schema Overview

The relational database consists of database schemas (S) as a way of grouping relational objects. The database schema S(T1, T2, ..., Tn) where n is the number of relational tables T is referred as the 'owner' of all the database objects under the schema. The relational data is stored in tables T(A1, A2, ..., Am) where A is the column attribute and m is the number of column attributes in the table. Each table consists of different column attributes A1, A2, ..., Am where each column attribute has its own domain dom(A) and range ran(A).

The database constraints considered during mapping are primary key 'pk', foreign key 'fk', UNIQUE 'unq', NOT NULL 'nn', and CHECK 'ck' which are represented as pk(T), fk(T), unq(A), nn(A) and ck(A) respectively.

Each table T is a set of tuples t1, t2, ...,tn, where n is the number of tuples in a table. Each tuple t is a set of values <v1, v2, ..., vn>, where vi is the value corresponding to column attribute Ai for current tuple 1<=i<=n. Individual attribute values in a tuple are represented using the attribute and value pair as t(Ai, vi).

A relationship in relational databases is a situation that exists between two relational tables indicated by a foreign key constraint. The relationship between two tables is commonly referred as binary relationship. A group of binary relations may form a pattern that involve three (ternary), four (quaternary) or more tables that are commonly referred as N-ary relationships.

The tables involved in a relationship are classified as "strong" or "weak" tables depending on where the foreign key is placed. A "strong" table is indicated by a primary key (pk) database constraint using one or more column attributes while a "weak" table uses a foreign key to refer to the "strong" table.

Binary relationships are represented using a foreign key constraint that involve one or many cardinalities on each side to represent one-to-one (1:1), one-to-many (1:*), and many-to-many (*:*) entity relationships.

4.2. Domain Data Knowledge

Domain Data Knowledge is knowledge about the data in the original domain in which the relational database has been developed. Domain Data Knowledge must be preserved after the mapping using suitable Semantic Web language axioms. In addition, Semantic Web language axioms help represent implicit knowledge, which is not represented in the original relational database, explicitly in the RDF repository after the mapping.

Domain Data Knowledge accounts the classification of the data in the domain (data types, their classification and constraints) and the regularities in the structure of the relations (the data patterns). Domain Data Knowledge can be formulated as a specification of additional dependency between the relations as well as analysis of the data types and constraints. Data patterns that are based on specific configuration of the relationships and their combination with specific constraints as defined in the data dictionary can be represented as domain knowledge using semantic web knowledge representation languages like OWL-Web Ontology Language.

Setting specific relationship patterns to look for enables us to build a data pattern that can be analysed structurally as well as statistically (i.e., percentage of occurrence). The domain-specific knowledge is used to 'prune' the semantic data at different stages of the *pre-processing* phase. The domain-specific knowledge is also a source for pattern discovery and interpretation at different stages of the *in-processing* and *post-processing* phase.

N-ary relationships that involve more than two tables create number of different types of patterns. These patterns are discovered on the base of the analysis of the keys in the relation- "*relational database area knowledge*". One of the criteria of identifying patterns is the structural grouping- i.e., "chain", "star", "triangle", "bridge", "square" etc.

For instance, if there is a pattern where a database column is used both as a *primary key* 'pk' and a *foreign key* 'fk', then the referencing table is a "*subClass*" of the referenced table. This "*predicate*" valuation uses primary key 'pk' and foreign key 'fk' "*relational database knowledge*" to create a "subClass" relationship using a *symmetric*-type pattern on a column that is being used both as a primary key 'pk' and foreign key 'fk' as illustrated in *Table 8- Domain Data Knowledge "predicate"*.

Table 8- Domain Data Knowledge "predicate"

```
Referencing Table T, Referenced Table T',
Column attribute A, primary key pk(T), foreign
key fk(T)
Begin
If( (A in (fk(T))) AND (A in (pk(T')))) then .

<owl:Class rdf:ID="T" >
<rdfs:subClassOf rdf:resource="#T'"/>
</owl:Class>

End if .
End .
```

Data Patterns

1) <u>*Transitive chain of Relations*</u>

Transitive chain of relations is the relationship between set of tables that create a one direction chain like relationship linked using foreign keys. The transitive chain indicates a "transitive relationship knowledge" which might be recursive as the chain grows.

For instance for tables (T_1, T_2, T_3), if there is a relationship between T_1 and T_2, and another relationship between T_2 and T_3, then this chain of tables indicates a transitive chain of relations. If there exists another relationship between T_3 and T_4, then another chain of relation is formed between T_2 and T_4.

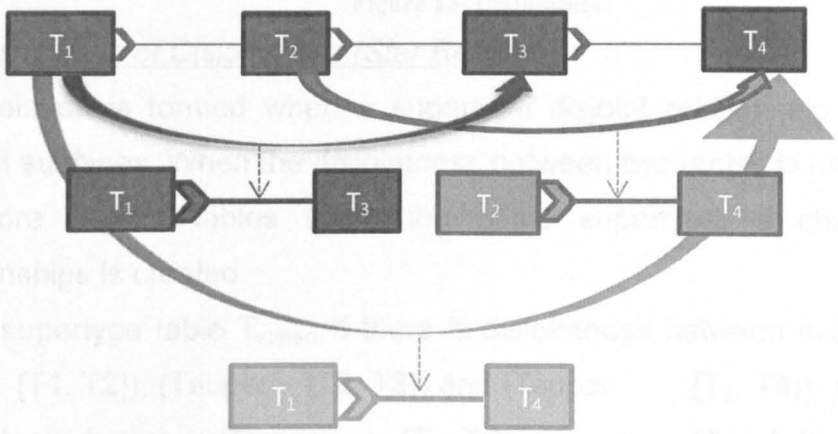


Figure 12- Transitive Chain of Relations

2) Disjointness

The other type of N-ary relationship is disjointness. Disjointness is a special kind of ternary relationship where the two subtypes are disjoint, i.e. their binary relationships with the supertype is non-overlapping.

If a supertype table T_x has a relationship between two subtypes (T_y,T_z), disjointness is defined when there is no relationship between T_y and T_z.

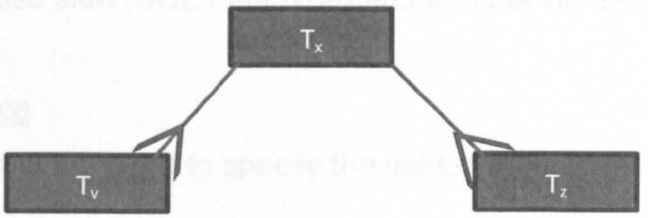


Figure 13- Disjointness

3) Association of Disjointness (Star Relation)

Star relation is formed when a supertype disjoint relationship involves more disjoint subtypes. When the disjointness between two tables is further expanded by more disjoint tables using the same supertype, a chain of disjoint relationships is created.

For a supertype table T_{super}, if there is disjointness between subtypes (Tsuper {T1, T2}), (Tsuper {T2, T3}) and (Tsuper {T_3, T4}), then there is a disjointness between (Tsuper {T_1, T_3}), (Tsuper {T1, T4}), and (Tsuper {T2, T4}).

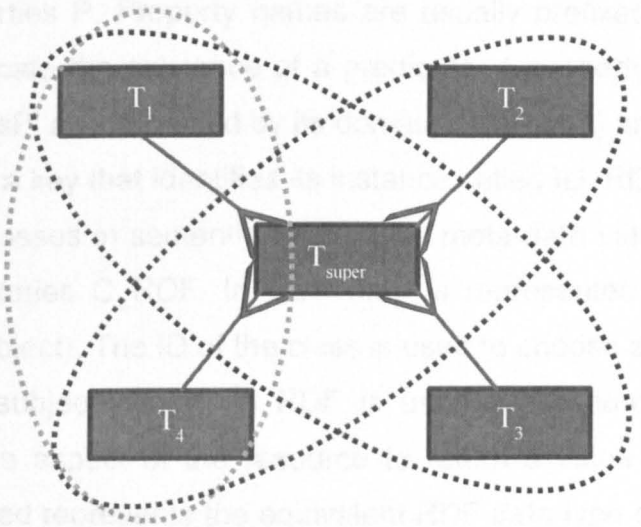


Figure 14- Star Relation

Data Types

The data types used in the mapping process are paired with one on the source relational database side (SQL data type) and another on the semantic web side (RDF).

1) SQL Data type

SQL data types are used to specify the values of the column attributes in the table rows.

i) *String-* an alphanumeric data type with an optional size limit

ii) *Character-* a single letter, digit or special character

iii) *Date-* a data type composed of YEAR, MONTH, DATE, HOUR, MINUTE and SECOND. The omission of time sets the date to a "00:00:00" date.

iv) *Numeric-* an integer or floating point data type where a comma separates

the precision from the scale in the latter case (NUMERIC(i,j), where i is the precision and j is the scale)

v) *Boolean-* a data type with a TRUE or FALSE value

2) RDF Data type

RDF data types are used to specify the values of the tuples in RDF repository. Data in RDF repository is represented using classes. Each class C has one or more properties P. Property names are usually prefixed by verbs like 'is' and 'has' to indicate the existence of a predicate. A property P is represented in a class as 'hasP' accompanied by its domain dom(hasP) and range ran(hasP).

A class has a key that identifies its instance called ID. RDF repository is used to represent classes in semantic web. Class meta-data values are used to create RDF repositories C_RDF. In RDF data is represented using triples (subject, predicate, object). The ID of the class is used to choose a subject sub(M) for the tuple. The subject sub(M) in RDF is used as resource while the predicate pre(M) is the aspect of the resource to return a value as object obj(M). The value returned represents the equivalent RDF data type of the source SQL data type as illustrated in *Table 9- SQL and RDF Data Types* comparison table.

RDF data types in semantic web are classified into Uniform Resource Identifiers (URIs), blank nodes and literals. The following namespace definitions are used in the examples below:

('rdf', 'http://www.w3.org/1999/02/22-rdf-syntax-ns#')

('rdfs', 'http://www.w3.org/2000/01/rdf-schema#')

('xsd', 'http://www.w3.org/2001/XMLSchema#')

i. *Subject (sub)*: a URI or blank node.
   - Example: http://www.example.org/relational/database

ii. *Predicate (pre)*: a URI.
   - Example, http://www.example.org/relational/vendor

iii. *Object (obj)*: a URI, blank node or literal (plain or typed).
   - Example, oracle or "oracle"^^xsd:string

Table 9- SQL and RDF Data Types

Data Type	SQL	RDF/XML
String	varchar, char, long	xsd:string
Date	date, timestamp	xsd:dateTime
	time	xsd:time
Numeric	integer	xsd:int
	decimal	xsd:decimal
	float	xsd:float
	real, number	xsd:double
Boolean	boolean	xsd:boolean

4.3. Domain Users Knowledge

Domain Users Knowledge accounts for the user privileges and user roles in the relational database management system. A configurable data dictionary as illustrated by

Table 10- User Privilege Data Dictionary Configuration is used to read the meta-data and map them to ontological repository to preserve database user access security.

Table 10- User Privilege Data Dictionary Configuration

```
- <Oracle_Database>
  - <table>
      <name>all_tab_privs</name>
      <table_schema>table_schema</table_schema>
      <table_name>table_name</table_name>
      <privilege>privilege</privilege>
    </table>
  - <table>
      <name>dba_role_privs</name>
      <grantee>grantee</grantee>
      <granted_role>granted_role</granted_role>
      <default_role>default_role</default_role>
    </table>
  </Oracle_Database>
```

The users are grouped into different levels of roles that range from visitor (read only) to admin (read/write/delete, etc.). During the *in-processing* phase, user rights and privileges are associated with the corresponding RDF class. The user rights and privileges also build the user profile.

4.4. *Application Domain Knowledge*

Application Domain Knowledge is used by the application which is based on a relational database. The 'Space Project Management' (SPP) application domain knowledge is used for evaluating the mapping algorithm. The applications used in SPP have different concepts and terminologies in the application domain. The concepts that are used in the application domain knowledge are summarised as *Documents*, *Risks*, *Non Conformances*, *Reviews*, *Actions*, *Profiles* and *Projects*. These concepts have unique as well as common definitions amongst applications. The unique knowledge is used to identify the concept while the common ones are used to associate and correlate the concepts. The different concepts and sub-concepts establish application domain concept hierarchy.

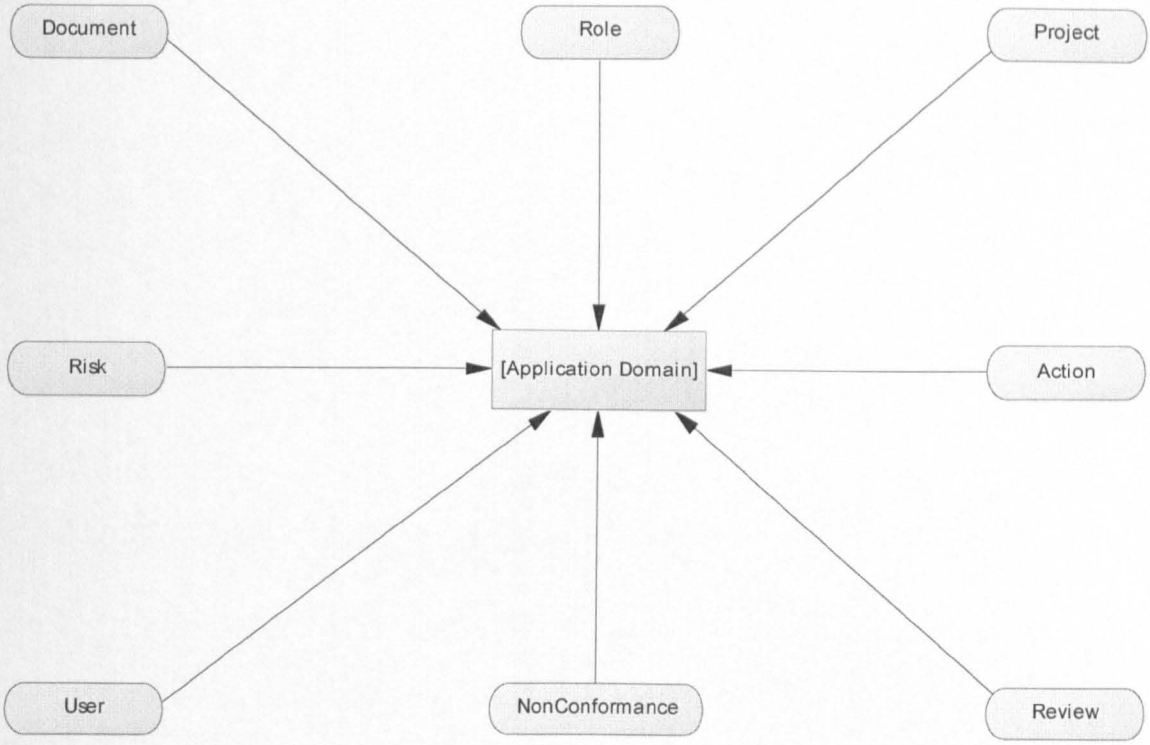


Figure 15- SPP-Application Domain Star Diagram

Each "concept" has attributes that define and explain the concept. There are also sub-concepts that are related to the concept through relationships. Some attributes like "status type" are used to identify an application "status" within SPP application domain.

Both "concepts" and "Relationships" may have further sub-concepts and sub-relationships represented by a nested square bracket ([]).

- Attributes- [URI, Definition, Title, {Status}]

- Status Types (sample)- [Register, Acknowledge, Assign Controller, Reduce, Accept, Resolve, Close]

- Sub-Concepts (sample)- [Domain, Scenario, Criticality *[Likelihood, Severity],* Rank, Trail]

- Relationship Types (sample)- [hasDomain, ofScenario, hasCriticality *[hasLikelihood, hasSeverity]*, hasRank, hasTrail]

The concept URI attribute is a unique representation of the particular concept in SPP application domain. The "definition" attribute has the formal detailed

definition list as shown in *Figure 16- Concept Attribute Star Diagram*. It also has the status and relationship type sub-lists.

The application domain knowledge relies on attributes like "status" to determine the semantic status of the concept in SPP application domain. The status attribute "values" are defined per concept that may have common values with other concepts. These "values" are used to draw different "transitive", "disjoint", "star" and other data patterns that help identify application domain knowledge.

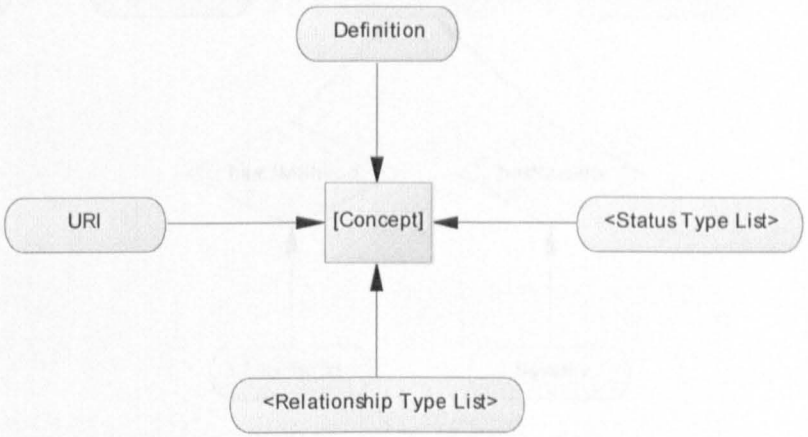


Figure 16- Concept Attribute Star Diagram

The concepts may have sub-concepts to further explain the relationship. Each sub-concept is related to the parent concept using a "Relationship Type" as illustrated by *Figure 17- Sub-Concept Relationship Type*. If the "sub-concept has a further relationship to a "sub-sub-concept", that would create a chain of concept relations that forms a graph like *Figure 18- Risk Concept Relationship Graph*.

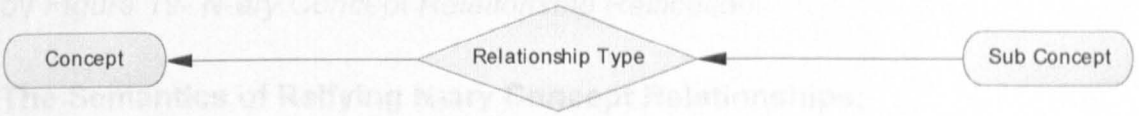


Figure 17- Sub-Concept Relationship Type

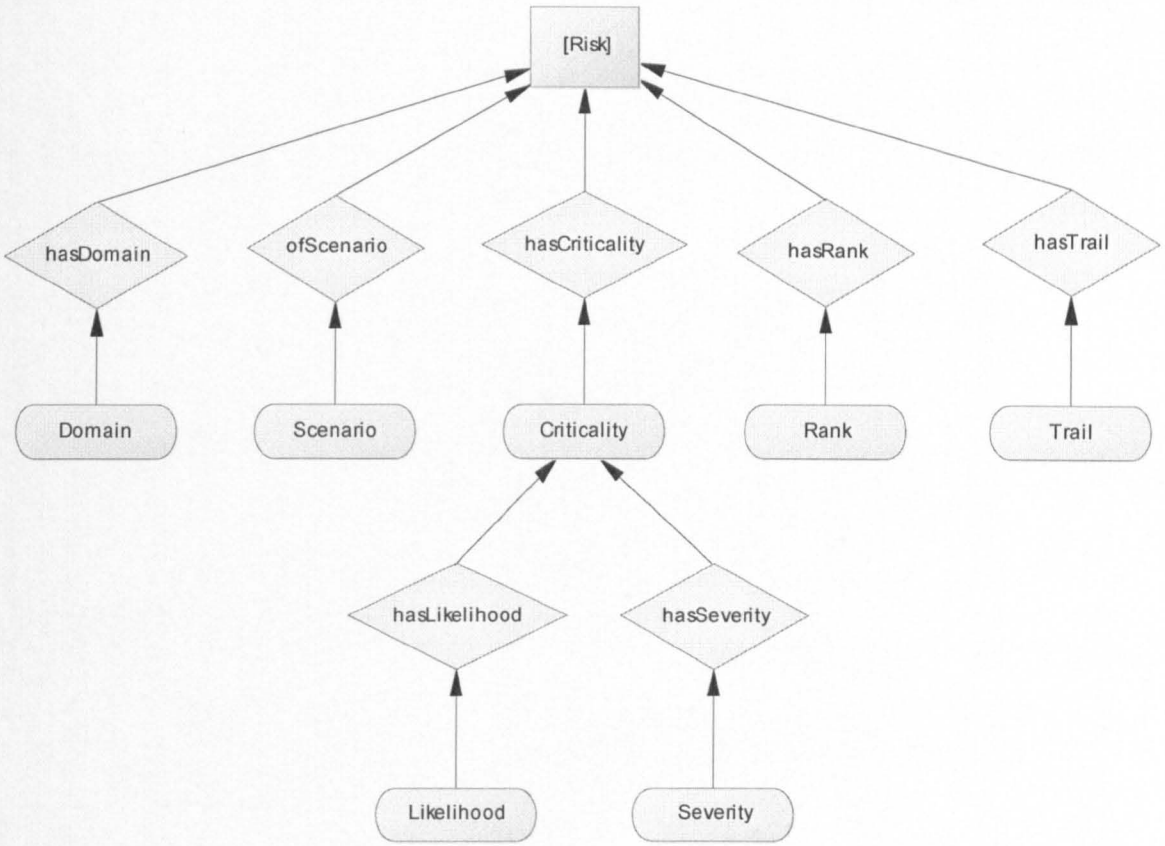


Figure 18- Risk Concept Relationship Graph

Similar approaches that have been used in "domain data knowledge" are also used in "application domain knowledge" identification. The "transitive", "disjoint" and "star" patterns are also applied on the relational "raw data".

The concepts create their own pattern that is interpreted as "application domain knowledge". The tables that use these common concepts create N-ary relationship amongst the tables involved. The N-ary relationship is reified into N binary relationships using the "concept" and the associated table as illustrated by *Figure 19- N-ary Concept Relationship Reification*.

The Semantics of Reifying N-ary Concept Relationships:

- N-ary relationships are reified into n binary relationships
- For each N-ary relationship between Tables T_1, T_2, …, T_n, n number of binary relationship are created
- Each binary relationship is represented by (1:1) cardinality on the relationship class side while maintaining the same cardinality on the relation class side.
- Each binary relationship is a weak relation type that depends on the existence of the referenced relation.

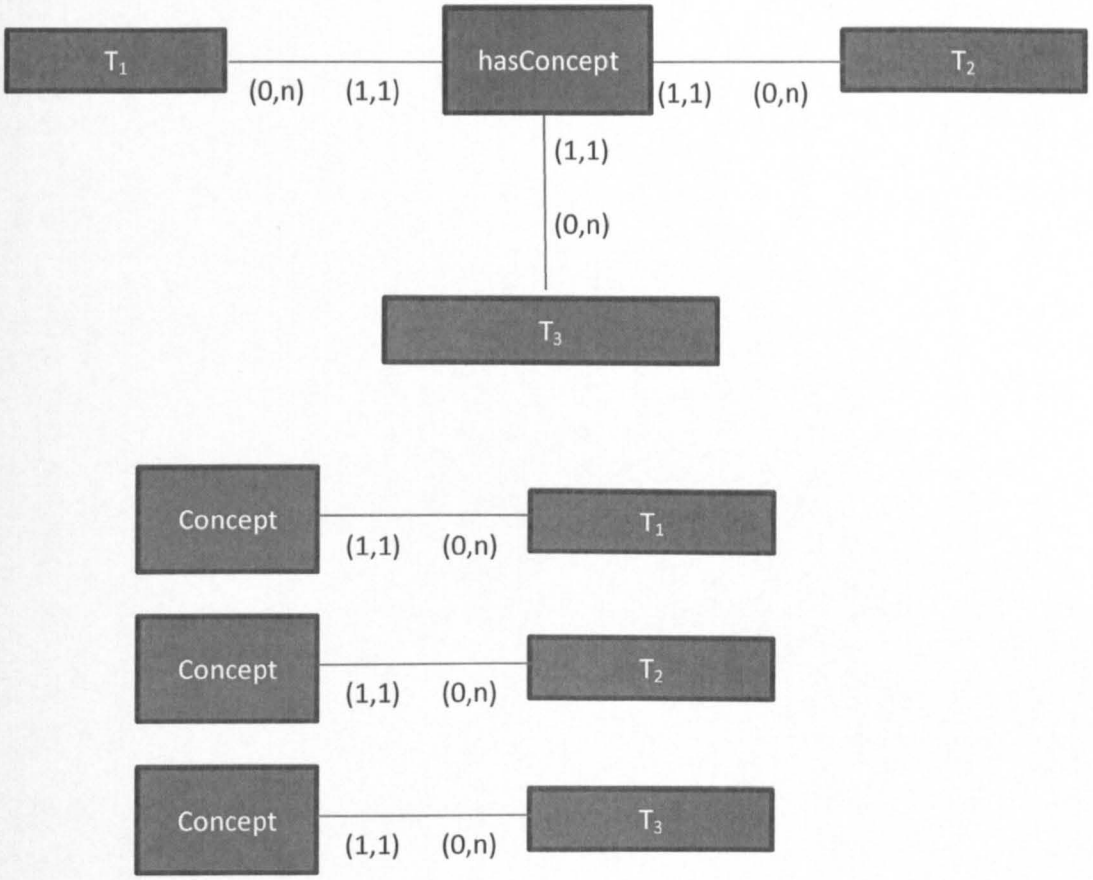


Figure 19- N-ary Concept Relationship Reification

Chapter 5: Relational to Ontological Mapping Algorithms

The mapping algorithms mirror the three *Pre*, *In*, and *Post processing* stages. Each stage has one or more algorithms to cover the different *Relational Database*, *Domain Data*, *Domain User*, and *Application Domain* areas. As part of the *pre-processing* stage *mapTable()*, *mapColumn()*, and *mapConstraint()* algorithms are used to identify the tables, columns, keys, constraints, relationships, and related *Relational Database Area Knowledge* components.

The database mapping procedure constitutes the above three algorithms to form *mapDatabase()* that loops through all tables T and their column attributes A within the schema S. The database mapping starts by mapping the tables- *mapTable()*. The tables are mapped into classes C. Corresponding to each class an RDF Repository C_RDF object is also created. The structure of the RDF repository is based on a TRIPLE format (subject, predicate, object).

After mapping the tables, *mapColumn()* algorithm maps the columns in each table into property column "hasP" of the existing class table C. *mapColumn()* procedure uses *mapDatatype()* procedure to return RDF equivalent data types for each relational attribute.

The different relational database constraints are also mapped using *mapConstraint()* procedure. The procedure maps constraints like primary key 'pk', foreign key 'fk', UNIQUE 'unq', NOT NULL 'nn', and CHECK 'ck'.

```
Procedure mapDatabase (S)
Input: Schema S
Begin

mapTable(S) .
mapColumn(S) .
mapConstraint(S) .

End .
```

Algorithm 1- mapDatabase

The mapping serialization is accompanied by an iterative algorithm that covers all the processes between entities and relationships. The relational tables,

columns, keys, and constraints are used to draw data patters using domain specific heuristics incrementally. This heuristics approach calls sub-programs (*mapRelationship()*, *mapDatatype()*, *mapApplicationKnowledge()*) to apply sets of heuristic rules iteratively. The rules are triggered by different *Domain Data*, *Domain Users*, and *Application Domain Knowledge* level scenarios. Although the mapping algorithm is iterative on top level it also embeds sub-algorithms *(checkRelationship()*, *checkTransitiveChain()*, *checkDisjointness()*) to apply sets of heuristic rules during the serialization which are used recursively. Candidates for suitable heuristics could be rules based on the data patterns and regularities.

During the *Post-Processing* stage the domain and application knowledge is used as an additional source of knowledge to the ontological repository. Since the data is in serialised form, an additional iterative analysis can be applied to further infer more statements about the ontological repository. Candidates for suitable heuristics are also used as reification rules based on ontological discovery. Built-in and custom inference reasoning are also applied for a richer semantic interpretation.

5.1 *mapTable()*

The Semantic Web equivalent of the relational algebra relations is a Class. During mapping the same name for the table T is used for the mapped class C. The class name is used to map subsequent relational columns into semantic class properties.

The classes in the Semantic Web repository can be considered repositories of data to hold the relational data after the mapping. Each Class table C represents the relational table in semantic web. To represent the class data in RDF triples (subject, predicate, object), a separate RDF repository C_RDF is created. In C_RDF the class ID (i.e. primary key equivalent of the source table) is used as a 'subject' while the rest of the properties are used as 'predicates'. Each property *"value"* corresponding to the class ID is the 'object' of the RDF triple.

In addition to the class table C and RDF repository C_RDF, an OWL class is created using the class table C as an ID.

```
Procedure mapTable (S)
Input: Schema S
Output: Class C, RDF Repository C_RDF, OWL
Class
Begin
For each table Ti in S loop .

        Create Class Table Ci .
        Create RDF Repository Ci_RDF
        using Class Table Ci and  a TRIPLE type
        attribute .

        <owl:Class rdf:ID="Ci" />

End loop .
End .
```

Algorithm 2- mapTable

5.2 *mapColumn()*

The column attributes in the relational database are mapped as *properties* in semantic classes. A property in a class can describe an entity class or a relationship class. Each property has a set of allowable domain values that could be shared with one or more properties.

The columns are broadly divided as "key columns" that are used to identify an occurrence of a relation and "simple columns" that only describe a relation. During mapping columns into properties, the following column types are used as criteria to choose the appropriate representation in the semantic web.

Column types

- Candidate Key (CK): minimal set of attributes that uniquely identifies each occurrence of an entity type.
- Primary Key (PK): candidate key selected to uniquely identify each occurrence of an entity type.
- Foreign Key (FK): referencing a primary key (PK) in another relation
- Simple Column (SC): a non-candidate key that describes an entity type.

The column attributes of a table is denoted as $T(A_1, A_2, \ldots, A_n)$

$T(A_1, A_2, \ldots, A_n) = \{PK, \{CK_1, CK_2, \ldots, CK_x\}, \{FK_1, FK_2, \ldots, FK_y\}, \{SC_1, SC_2, \ldots, SC_z\}\}$

Where x, y, z is a whole number.

When the cardinality of x is greater than 1, candidate keys (CK) are treated as *composite* keys.

The column attributes in the relational database are mapped to corresponding class properties. The constraint type associated with the attribute determines the cardinality. Each column attribute A is mapped to property P prefixed by the word "has" as *"hasP"*.

```
Procedure mapColumn (S)
    Input: Schema S, Table T, Column attribute A

Output: Property P, OWL:DatatypeProperty
Begin
  For each table Ti in S loop .

For each Column Aj in Ti loop .

                get mapped Class Table Ci of Ti.

                        set Aj as Property Column hasAj .
                        get_&xsd;type_equivalent (Aj) .

                <owl:DatatypeProperty rdf:ID="hasAj">
                <rdfs:domain rdf:resource="#Ci" />
                <rdfs:range rdf:resource
                 ="&xsd;type_equivalent" />
                </owl:DatatypeProperty>
        End loop .
   End loop .
End .
```

Algorithm 3- mapColumn

5.3 *mapConstraint()*

mapConstraint() algorithm maps relational database constraints into their equivalent semantic web representation. The algorithm reads both table level as well as column level constraints. For primary key pk(T) constraints, it creates "InverseFunctionalProperty" and "maxCardinality" OWL properties. For foreign key fk(T) constraints, it creates "ObjectProperty" OWL property. If a foreign key column is also part of the primary key pk(T) constraints, then the referencing table (T) is set as a "subClass" of the referenced table (T').

For UNIQUE 'unq(A)', NOT NULL 'nn(A)', and CHECK 'ck(A)' database constraints, the algorithm creates equivalent "InverseFunctionalProperty", "minCardinality", and "hasValue" OWL property restrictions respectively.

If the column attribute is a primary key pk(T) of the table, the maximum cardinality of the property car(P) is set to one. If the column attribute has a unique constraint unq(A), the maximum cardinality of the property car(P) is set to one. On the other hand if the property has a NOT NULL constraint nn(A), the minimum cardinality of the property car(hasP) is set to one.

mapConstraint() procedure maps column attribute(s) A using the table T and its constraints (primary key, foreign key, UNIQUE, NOT NULL, CHECK) to a semantic property P and OWL properties.

```
Procedure mapConstraint (S)
    Input: Schema S, Table T, Referenced Table T', Column attribute A,
    primary key pk(T), foreign key fk(T), UNIQUE unq(A), NOT NULL
    nn(A), and CHECK ck(A)
Output: RDFS subClassOf, Property P, OWL cardinality
properties
Begin
For each table Ti in S loop .
 For Column Aj in Ti loop .

 get mapped Class Table Ci of Ti.

 If (Aj in (pk(T'i))) then .

 <owl:InverseFunctionalProperty rdf:resource="# hasAj "/>
```

```
/* set maximum car(hasAj) to 1 . */
<rdfs:subClassOf>
    <owl:Restriction>
    <owl:maxCardinality
     rdf:datatype="&xsd:nonNegativeInteger">1
      </owl:maxCardinality>
    </owl:Restriction>
</rdfs:subClassOf>

 Else
 if (Aj in (fk(Ti))) then .
   If (Aj in (pk(T'i))) then .
    <rdfs:subClassOf rdf:resource="#C'"/>
   End if .
       <owl: ObjectProperty rdf:ID="hasA">
       <rdfs:domain rdf:resource="#C" />
       <rdfs:range rdf:resource="#C'" />
       </owl: ObjectProperty >
 Else
 if (unq(Aj)) then .

 <owl:InverseFunctionalProperty rdf:resource="# hasAj "/>

 Else
 if (nn(Aj) and (!pk(Aj)) then .
       /*set minimum car(hasAj) to 1 .*/
    <owl:Restriction>
          <owl:minCardinality
          rdf:datatype="&xsd:nonNegativeInteger">1
          </owl:minCardinality>
       </owl:Restriction>

 Else
 if (ck(Aj)) then .
<rdfs:subClassOf>
<owl:Restriction>
       <owl:onProperty rdf:resource="#hasAj" />
```

```
        <owl:hasValue rdf:datatype="&xsd;string" > v(Aj)
        </owl:hasValue>
</owl:Restriction>
</rdfs:subClassOf>
 End if .
End loop .
End loop .
End .
```

Algorithm 4- mapConstraint

5.4 mapRelationship()

A relationship between tables is identified by a *foreign key*. mapRelationship() calls a series of sub-procedures to identify the type of relationship between the two tables and discover any patterns with the rest of the tables in the schema.

```
Procedure mapRelationship (S)
Input: Table T, Column attribute A, foreign key fk(T)
Output: Class C, Property P
Begin
For each table Ti in S loop .

        For each column Aj in Ti loop .
                If (Aj = fk(Ti)) then
                        CheckRelationship(Ti, T'i)
                        CheckTransitiveChain(Ti, T'i) .
                        CheckDisjointness(Ti, T'i)

                where T'i is referenced table by Ti
                End if .
        End loop .
   End loop .
End .
```

Algorithm 5- mapRelationship

Check Relationship()

CheckRelationship algorithm uses the referencing table T and the referenced table T' to determine whether to create a separate class to represent the relationship or only add a property column to the existing class.

If the foreign key fk(T) is referring to a primary key pk(T') and the foreign key has a NOT NULL constraint, then a new relationship class is created using the two tables (T, T'). An additional "subClass" parameter is also passed to create a "subClass" axiom between the "Class" equivalents of the tables T and T'.

If the above criterion is not satisfied, the foreign key fk(T) would have been added as a property to the referring table equivalent class C using the mapColumn() procedure above.

```
Procedure CheckRelationship(T, T')
Input: Table T, primary key pk(T), foreign key
fk(T),NOT NULL nn(T)

Begin
If (fk(T) = pk(T') and (fk(T) = nn(fk(T))) then .

        CreateRelationship(T,T', subClass) .

End if .
End .
```

Algorithm 6- CheckRelationship

Check Transitive Chain

Transitive Chain of relations is tested using the foreign key/primary key column attributes between three relational tables.

For any relational tables T1, T2, T3 in Schema S, if there is a foreign key relationship between T1 and T2 and if there is also a foreign key relationship between T2 and T3, then there is a transitive chain between T1 and T3.

The algorithm uses the referenced table's (T') columns to find further foreign key relationship to the rest of the tables. If another relationship other than the one between T and T' is found, a new relationship class is created using the two tables (T, Ti). An additional "Transitive" parameter is also used to create a

"transitive" axiom between the "Class" equivalents of the starting table T and the new third table Ti.

```
Procedure CheckTransitiveChain(T, T')
Input: Table T, Column attribute A, primary key pk(T),
foreign key fk(T)
Begin

For each column Ai in T' loop .
        If (Ai in fk(T')) then .
        For each table Ti in S loop .
                If ((Ai in pk(Ti)) and (Ti != T)) then .

                createRelationship(T, Ti, Transitive) .

                End if .
        End loop .
        End if .
End loop .
End .
```

Algorithm 7- CheckTransitiveChain

Check Disjointness()

Disjointness is a relationship between two "SubType" tables that share a common "SuperType" but has no relationship between each other.

The foreign key/primary key column attributes between the tables is used to determine the disjunction between tables.

For any relational tables T_1, T_2, T_3 in Schema S, if there is a foreign key relationship between T_1 and T_2 and there is also a foreign key relationship between T_2 and T_3 but there is no relationship between T_1 and T_3, then there is disjointness between T_1 and T_3.

The algorithm uses the referenced table's (T') columns to find further foreign key relationship to other tables. If another relationship other than the one between T and T' is found and there is no foreign key relationship between the new table T_i and the starting table T, then a "*disjointWith*" axiom is created between the starting table T and the new table T_i.

N.B. A group of disjoint relationships create a "Star" relation with the "SuperClass" in the middle and the disjoint classes as a branch.

```
Procedure CheckDisjointness(T, T')
Input: Table T, Column attribute A, primary key pk(T),
foreign key fk(T)
Output: RDFS subClassOf, OWL Class, disjointWith
Begin

For each column Ai in T' loop .
        If (Ai in fk(T')) then .
                For each table Ti in S loop .
                If ((Ai in pk(Ti)) and (Ti != T)) then .
                        if ((ALL) fk(Ti) NOT in
                        (ALL) pk (T)) then .

                <owl:Class rdf:ID="Ti">
                        <rdfs:subClassOf rdf:resource="#T"/>
                        <owl:disjointWith rdf:resource=" #T "/>
                </owl:Class>
                End if .
                End loop .
        End if .
End loop .

End .
```

Algorithm 8- CheckDisjointness

Create Relationship()

CreateRelationship algorithm is used to create a new relationship class table to represent the relationship between the referencing table T and the referenced table T'. If there is a foreign key in table T that references to a primary key in table T', a new class table is created using a class symbol C and the name of the referencing and referenced tables respectively separated by an underscore as "*C_T_T'* ".

The primary keys of both tables are added as property columns to the new class by adding "*has*" as a prefix as "*hasP*" and "*hasP' "*.

In addition to the semantic class, a repository is also created to represent the class table data in RDF triples (subject, predicate, object). The RDF repository reads the class table data and presents it in RDF triples. It is named using the class table names suffixed by "*RDF*" as "*C_T_T'_RDF*".

```
Procedure CreateRelationship(T, T', TYPE)
Output: RDFS subClassOf, OWL Class, ObjectProperty
Begin
If (fk(T) = pk(T')) then .

        create Class C _T_T'.
        set pk(T) as Property hasP of Class C_T_T' .
        set pk(T') as Property hasP' of Class C_T_T' .

        create RDF Repository C_T_T'_RDF
                using Class Table C_T_T' and a TRIPLE
                type attribute .

                get mapped Class Table C of T .
                get mapped Class Table C' of T'.

        If (TYPE = 'subClass') then
                <owl:Class rdf:ID="C">
                        <rdfs:subClassOf
                         rdf:resource="#C'" />
                </owl:Class>

        Else
        if (TYPE = 'Transitive') then

                <owl:ObjectProperty rdf:ID="pk(T)">
                <rdf:type rdf:resource
                        ="owl;TransitiveProperty"/>
                <rdfs:domain rdf:resource="#C" />
                <rdfs:range rdf:resource="#C'" />
```

```
                </owl:ObjectProperty>

        End if .

End .
```

Algorithm 9- CreateRelationship

5.5 *mapDatatype()*

```
Procedure get_&xsd;type_equivalent (A)
Input: Column attribute A

Output: &xsd;type
#########################################################
Begin
        if ("datatype" in ("varchar", "char", "long")) then .
                return "xsd:string"
        else
        if ("datatype" = "integer") then .
                return "xsd:int"
        else
        if ("datatype" = "decimal") then .
                return "xsd:decimal"
        else
        if ("datatype" = "float") then .
                return "xsd:float"
        else
        if ("datatype" in ("double", "number")) then .
                return "xsd:double"
        else
        if ("datatype" in("date", "timestamp") then .
                return "xsd:dateTime"
        else
        if ("datatype" = "time") then .
                return "xsd:time"
        else
        if ("datatype" = "boolean") then .
                return "xsd:Boolean"
        End if .
End .
```

Algorithm 10- mapDatatype

5.6 *mapApplicationKnowledge()*

After mapping the relational data to ontological repositories, the algorithm uses application domain knowledge to represent application level knowledge. The

application domain knowledge uses "Space Project Management" SPP-Ontology. The "concepts" in SPP-Ontology are explored for similar relationship patterns as "relational database knowledge" using common application level concept attributes. In Semantic web, not only can we represent data but also knowledge as part of the semantic data.

The following transactional fields are used as a way of representing a common "application knowledge" that spans in more than one application knowledge domain.

mapStatusKnowledge()

A "*status*" is one of the common attribute columns used to represent a particular phase in SPP-Ontology. Depending on the concept type, *"status"* column values may start from "*Draft"/"Create*" and end with "*Approved"/"Rejected*". These statuses represent different "application knowledge" at each stage of the transaction process.

To illustrate the algorithm, here is an example of a status "*accept*" in SPP-Ontology involving concepts *"Review", "Document",* and *"Project".* In SPP-Ontology, accepting a "Review" process ties a "Document" that details the matter to be reviewed and the "Project" where the document belongs.

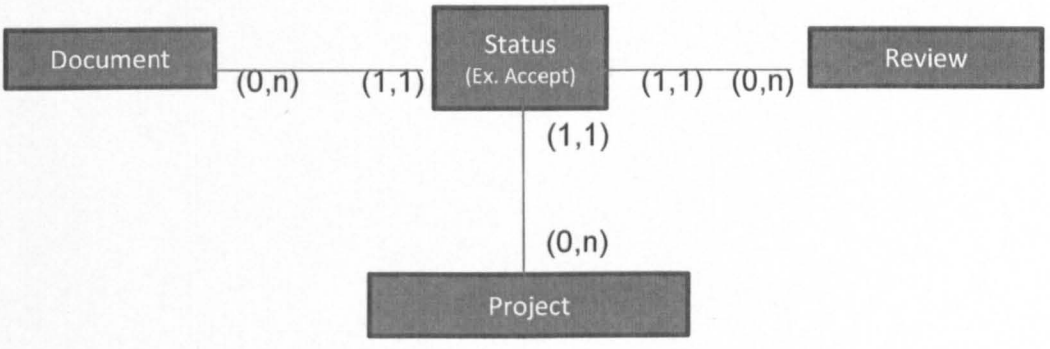


Figure 20- Ternary "Status" relationship- status(document, review, project)

This ternary "Status" relationship-"*status(document, review, project)*", shall be reified into multiple binary relationships before being considered as a source for the mapping.

As a result of the ternary "Status" relationship-"*status(document, review, project)*", this ternary relationship is reified into three binary relationships

- Document_Status(document, status)
- Review_Status(review, status)
- Project_Status(project, status)

```
Procedure mapStatusKnowledge(T, T', A)
Input: Table T, Transaction_Attribute_Column A,
##############################################################
Begin

        create Class C_T_T'.
                set pk(T) as Property hasP of Class C_T_T' .
                set A as Property hasA of Class C_T_T' .

                create RDF Repository C_T_T'_RDF
                        using Class C_T_T' and  TRIPLE type attribute .

                <owl:Class rdf:ID="C_T_T'" />
                get_&xsd;type_equivalent (A) .

                <owl:DatatypeProperty rdf:ID="hasP">
                        <rdfs:domain rdf:resource="#C_T_T'" />
                        <rdfs:range rdf:resource="&xsd;type_equivalent" />
                </owl:DatatypeProperty>

                <owl:DatatypeProperty rdf:ID="hasA">
                        <rdfs:domain rdf:resource="#C_T_T'" />
                        <rdfs:range rdf:resource="&xsd;type_equivalent" />
                </owl:DatatypeProperty>

End .
```

Algorithm 11- mapStatusKnowledge

Chapter 6: Relational and Ontological Model

6.1 Relational Model and Application Domain Workflow

For the practical implementation of the mapping algorithms, a relational model is designed. A relational database implementation of the relational model is used to validate the algorithms. The relational database is also populated with a sample Space Project Management application data which uses the database for monitoring and decision making.

The relational database objects represent the different relational model entities and relationships. The tables in the relational database represent the relations in the relational model. The relational database consist of various groups of tables that help manage "space projects" in a specific application domain. Information about the entities summarised in *Figure 21- Space Project Management High Level Relational Model* are included in the application database. One of the application domain knowledge used in the application domain is the "status" of entities. "Status" application data is used to validate *mapStatusKnowledge()* algorithm in the previous chapter. Most of the relational models have accompanying application "status" workflow. The different relational models of the relational database are summarised as follows.

Document: data about the document that consist of zero or more attachments. It also includes data about the creator and workflow status.

Attachment: data about the attachment file, size, type, location, mime Type etc.

Collection: Is a tree structure to group documents into folders and sub-folders.

Review: this module consist of corrections and dispositions to evaluate a design against the original specification by different level of users such as initiator, contractor, panel coordinator, project manager, board etc.

Risk: data about the risks raised in a project. Depending on the severity and likelihood of the risk, different verifications and explanations are stored in this module

Non Conformance: data in this module consists of non-conformances that need to be traced depending on the "action" and/or "waiver" according to its impact.

Action: actions are usually decisions that involve different participants that will end in a conclusion. It consists of data that surround the discussion that may involve zero or many documents.

Incident: incidents are instances of actions on missions, researches, explorations etc. that need to be investigated, fixed, looked after etc.

Resolution: resolutions are documented solutions to known **instances**.

Profile: this module consists of all registered users with their different privileges, access level and personal details

Role: this module consists of the different roles available and their grouping with respect to hierarchy and privilege.

Project: this module consists of the different projects that make part of the "Profile" together with the "role" and "user" modules.

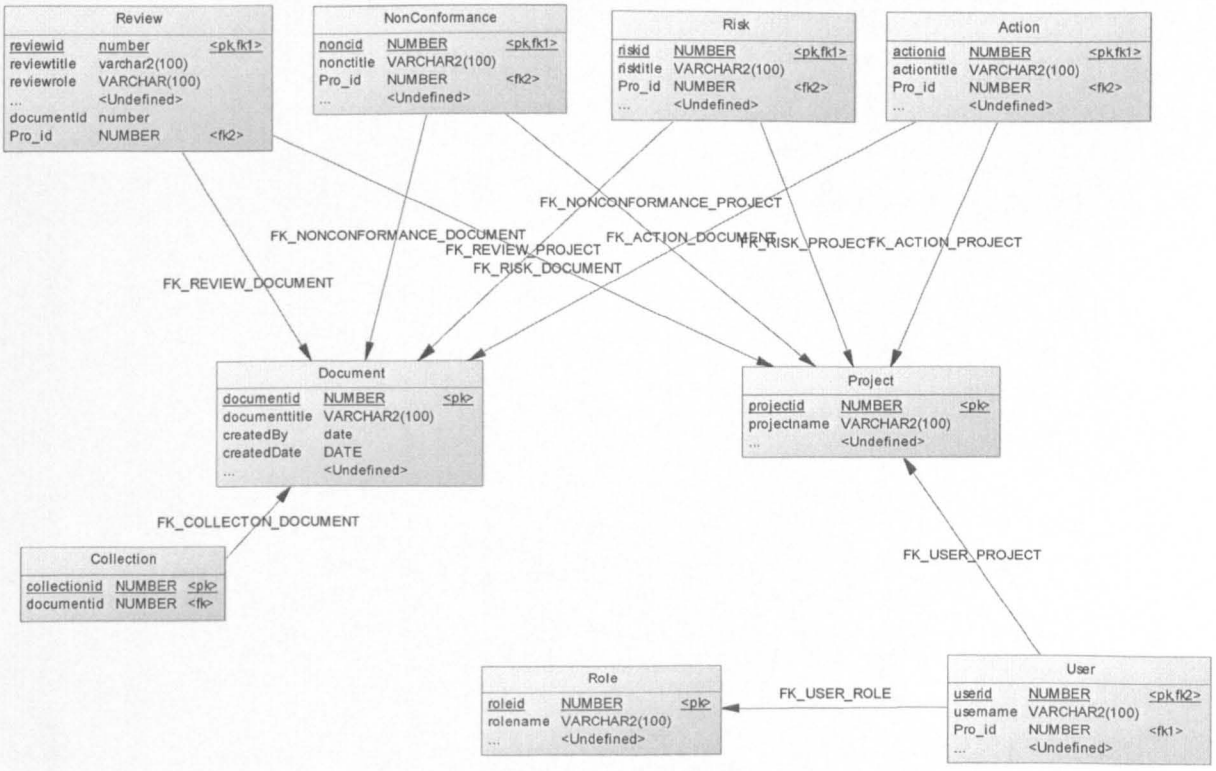


Figure 21- Space Project Management High Level Relational Model

6.1.1 Document

Document table contain metadata information describing documents. The metadata information like title, description, created date, and created by describe the document. Each document can have one or more attachment files. Related documents can also be grouped into "Collections".

Documents are objects that are often exchanged and shared between different relational models. The representation of the document metadata in semantic web data model makes the document more universal and interoperable.

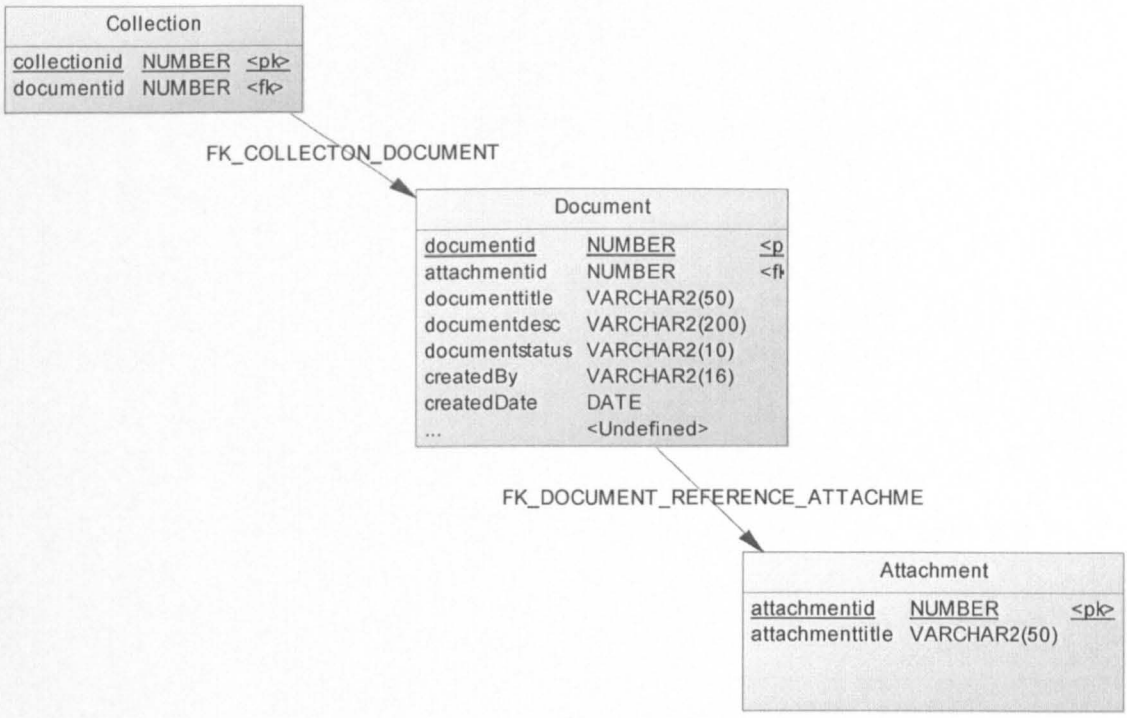


Figure 22- Document Relational Model

Document Relational Model also has a workflow that primarily shows the "status" of the document. This application domain knowledge needs to be mapped together with the workflow process to represent the "knowledge" on the other side of the mapping. The document goes through a review cycle that goes through "create", "Under Revision", "Review", and "Approve" statuses. An approved document is ready to be used as a base in different stages of the project cycle.

6.1.2 Profile

A project is setup to achieve a specific aim. Every project has list of *users* with a specific *role* in the project. Users can belong to one project at a point in time with a specific role.

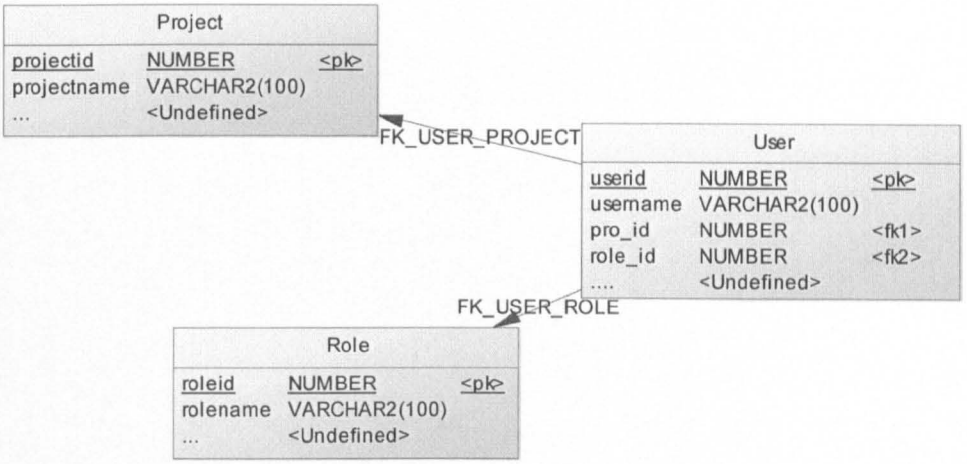


Figure 23- Project Relational Model

6.1.3 Review Discrepancies

Reviews are processes to evaluate a design against the original specification. The review process is conducted at different levels of the project life cycle. It starts from the feasibility study stage and continoues through the design, implementation, testing as well as routine maintenance stages.

The current relational data representation of the different reviews is only accessible through application interfaces and the entity-relationship is unique in its own way which is not something that you can share, merge or exchange with other reviews. The representation of reviews in semantic web data model is crucial step to fulfil all the deficits of the current data representation with respect to universality, sharing and interoperability.

The "review" knowledge at feasibility study stage of a project cycle is crucial for the future implementation. The "Reviews" in a semantic web data model at this stage will help as a base for future feasibility studies and are easier to find and process according to the current specification.

The design stage also raises a lot of "reviews". This stage will involve actual representation of the feasibility study outcome. The different classification and analysis of past "reviews" help produce a better deign if the past "reviews" are represented in an interoperable universal Semantic Web data representation.

Similar processes are also involved after the design stage during implementation. During implementation the pen and paper only process is replaced by an actual craft of building/configuring/assembling. This process naturally raises even more "reviews" and might also involve going back to the drawing board. These all processes are best executed if there is a semantic

web based knowledge database to facilitate the representation, search, and exchange of data

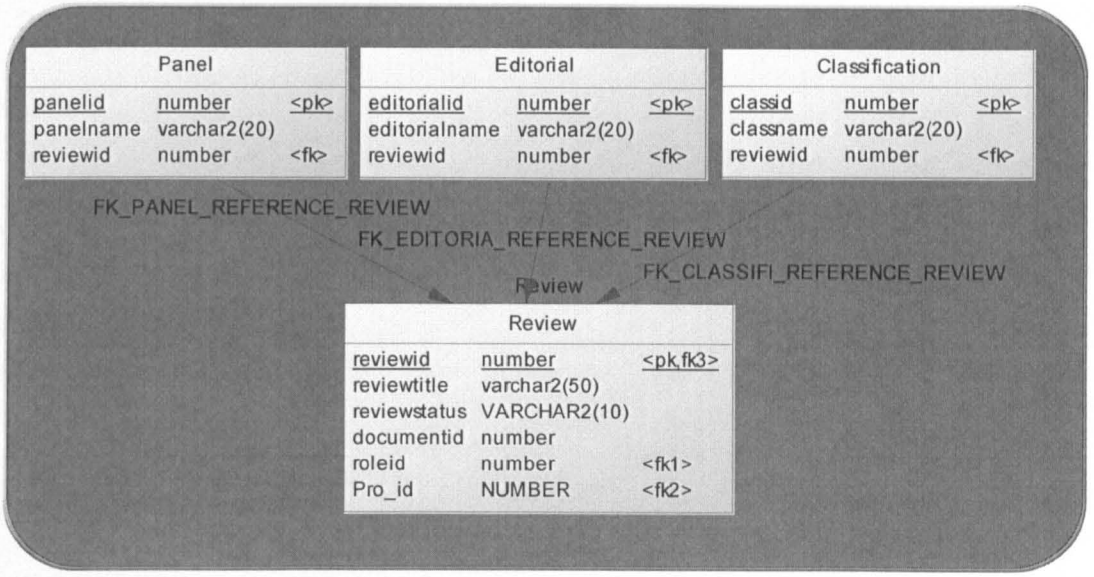


Figure 24-Review Relational Model

The Review workflow involves different roles between the discrepancies being "detected" to the final "closing" of review. After the discrepancy is detected, the reviewer initiates a review for "analysis" with a minor/ major classification. If the result of the analysis (withdraw, accept, reject) is "Accept", then a "User" with a "Chairperson" role assigns a "Contractor" to come up with a "Correction/ Prevention". If the solution is approved by the "Board", then a "user" with a "Coordinator" role raises an action and come up with final "Disposition".

The review workflow gives us the current "status" of the review and the different roles involved in closing the review. The mapping process involves not only the review relational model but also application domain knowledge as represented by the workflow. A review in semantic web data model is more descriptive if it shows not only the review metadata but also the current status of the review, the roles involved from initiating to closing the review as well as the resulted "disposition" to rectify the discrepancy.

6.1.4 Action

Actions are used to follow-up activities. Actions metadata include title, description, audit trail and roles like owner, viewer, contributor, actionee etc. Actions can also include attachments that could be related to Documents. Actions go through a sequence of workflow statuses. A closed action can be

referred as an **"Incident"** log for future reference or as a **"Resolution"** action to deal with similar instances.

The representation of the "action" data in semantic web data model is a useful tool to refer as a knowledge base if similar "actions" are being dealt at the moment. "Actions" like documents are also integral part of the whole process and their representation in an interoperable model is a cornerstone to strengthen not only the main processes but also the different "actions" involved within and across projects.

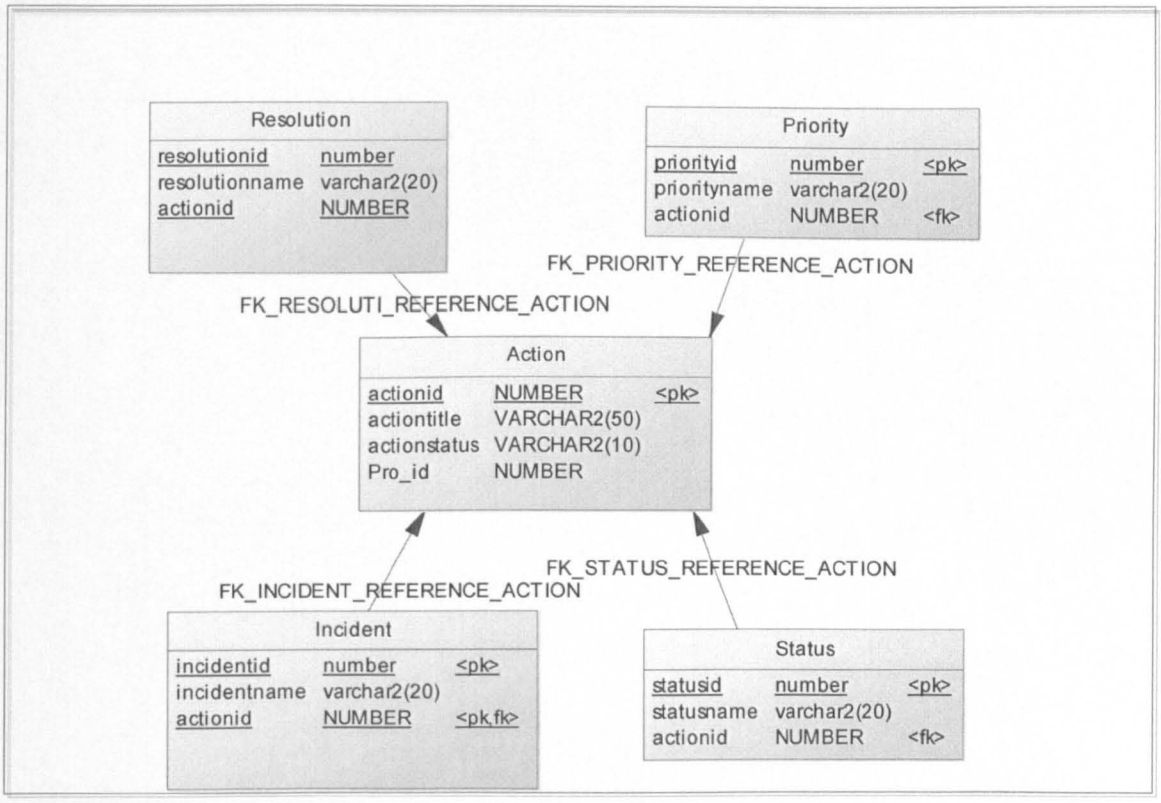


Figure 25- Action Relational Model

The action workflow shows the stages of an action from being created to being closed. After an action is created the participants of the action are selected/ assigned. The workflow goes through a continuous "respond" status until the creator is satisfied and sets the action to a "complete" status. Mapping the relational model and the corresponding "status" shows the current stage of the action that helps to determine if the action is an in an on-going or concluded stage. The application domain knowledge represented by the "status" is used as a source of information that decides the weight of the solution that could be provided by the related "resolution" or "Incident" log.

6.1.5 Risk

Risks are used to **acknowledge**, **rank**, **measure** and **follow up** issues. Risks are measured by their **severity** (Negligible, Significant, Major, Critical, and Catastrophic) and **likelihood** (Minimum, Low, Medium, High, and Maximum). The Risk also includes "**cause**", "**consequence**", "**domain**" (e.g. affecting cost, schedule etc.) and other additional metadata.

The severity of a risk against its likelihood determines the progress of the project or mission to the next step. This knowledge of risks can be used to determine similar risks in the future and to analyse past risks for a better understanding of future prototypes/designs/configurations.

The representation of the risks in the relational model is project specific and difficult to cross-reference across the different projects and missions. The representation of the knowledge base in a semantic interoperable model makes the process of understanding and analysing risks easier and exchangeable. The different "acknowledged" risks and their "resolution" or "reduction" methodologies is a base knowledge to come up with a relatively less risky process/design/methodology for similar subsequent implementations. The knowledge base also helps us to identify "acceptable" risks that might not need further action for the continuity of the process.

All the knowledge bases that are currently in a redundant decentralised relational data model needs to be represented in a semantic web data model to take advantage of the vast data already stored in the relational databases. The representation of this data in semantic web data model makes it more visible and sharable across similar projects.

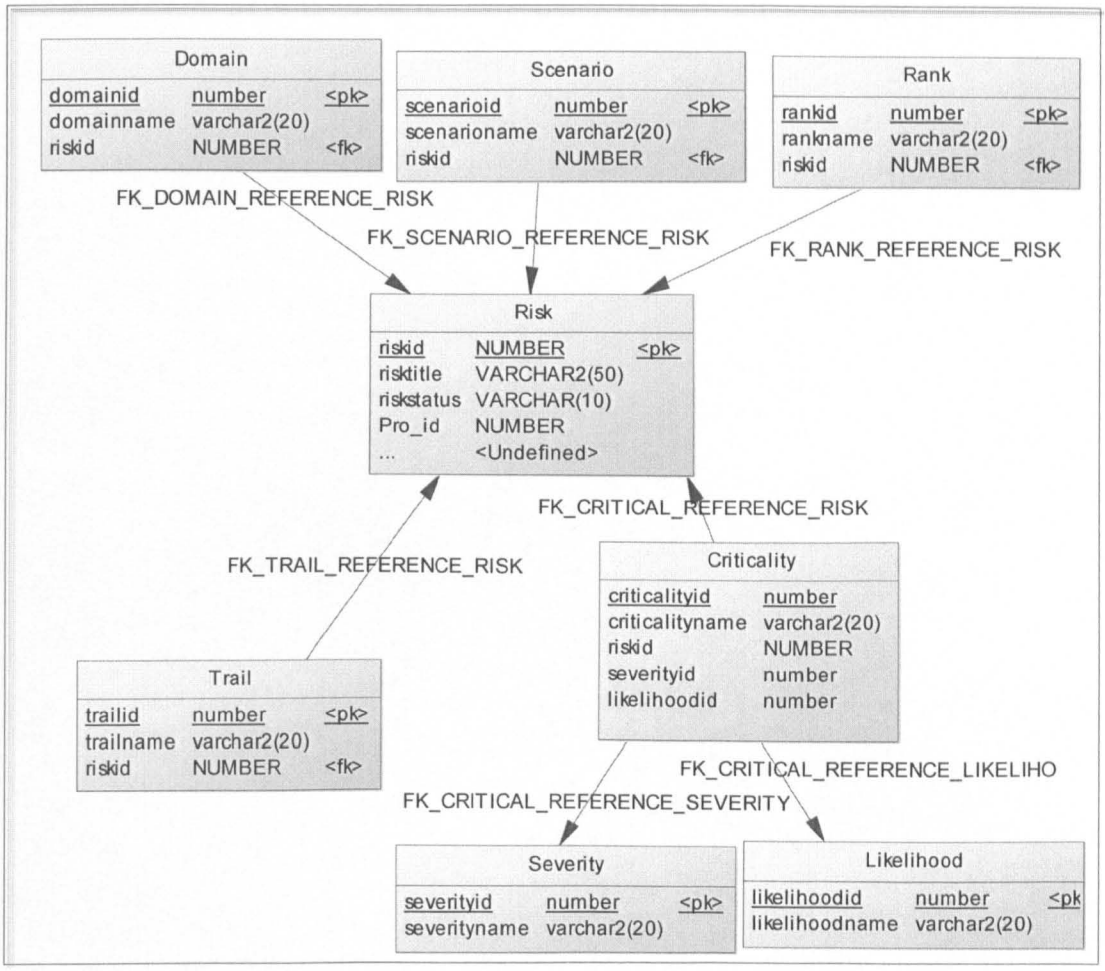


Figure 26- Risk Relational Model

Risk workflow starts with the Risk being "registered". A reviewer decides whether the risk is worth acknowledging or can be rejected. If the risk is accepted, a chairperson assigns a controller to "access" the risk and assign a "reduce", "accept", or "resolve" verdict that results in the risk being closed. Risk status workflow shows the stages to determine the significance of the associated risk. Application domain knowledge like knowing whether the risk is in a "reduce", "accept", or "resolve" state helps to further understand the risk relational model. Mapping not only the relational model but also the application domain status knowledge makes the mapped model a richer and interoperable representation compared to the source relational model.

6.1.6 Non Conformance

Non Conformances are items that do not conform to the specification of the design process. Non Conformances involve component anomalies, operational problems, item defects etc.

The process below depicts the process of tracking non-conformances using the different identification, categorisation, tracking, waiver, resolution and other stages.

The database of previous non-conformances is used as a knowledge base to resolve similar non-conformances in the future. The different categories of non-conformances and their impact is a knowledge that needs to be represented in an interoperable data model to represent, compare and analyse against other future projects.

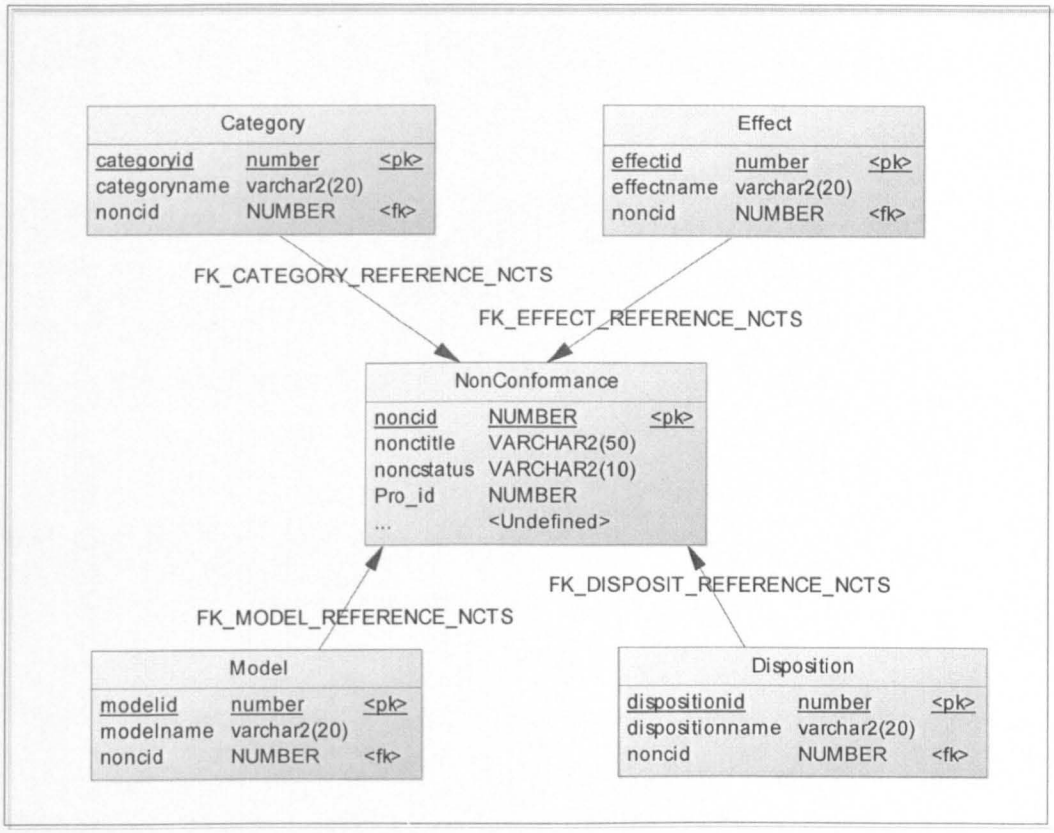


Figure 27- Non Conformance Relational Model

Non-conformances status workflow start with a non-conformance being raised, categorised and analysed by the reviewer. The analysis covers the cause of the non-conformance and explains the consequences. If the "impact" of the non-conformance is major, it is referred to a review board that would deliberate and suggest an "action". If the "impact" is minor, then it goes straight to "action". Based on the type of non-conformance, a prevention or correction is proposed for "verification". Non-conformance relational model refers to action relational model that in turn refers to project and document relational models. Mapping application domain status knowledge together with the relational model helps to explain the non-conformance as well as map the corresponding application-domain knowledge of the "action".

6.2 Ontological Repository Model

According to researches, the approach to store ontologies can be summarised into the following three possibilities (Gali, Chen, Claypool, & Uceda-Sosa, 2004).

1. The first one is to build a special purpose database system that will be used as RDF repository.
2. The second one is to use object oriented database management system (OODBMS).
3. The third one is to use a relational database management system (RDBMS).

In this research relational database systems are used to store ontologies. This makes the ontologies take advantage of the scalable relational database management system (RDBMS) and also allows the integration of both relational and ontological data in a single database system. RDF modelling language is used to represent the ontological model of the original database relational data after the mapping, RDF is used. The tables and views in the relational database are used as a starting point for generating RDF datasets. Each Triple (Subject, predicate, object/object-attribute-value) map represents a logical table. RDF Triples generated from the logical table contain subjects that correspond to a row in a table. Each subject contains one or more (predicate, object) pairs to generate RDF triples for a logical table row. The major advantage of using RDF model in contrast to relational model is its capability to merge data-sets. Unlike relational databases that is record oriented and prone to repeating "subjects" (add row) to add more information about the subject matter, RDF stores follow a graph pattern to rather point to the new data rather that repeat the rest of the data about the subject. There is also a similar mathematical representation difference between relational and RDF query languages that use "relational algebra" and "predicate calculus" respectively to represent axioms, relations, entities etc. RDF entailment is relatively simple, mostly consisting of taxonomic inference using rdfs:subClassOf and rdfs:type, and similar inference for rdfs:subPropertyOf (Pan & Heflin, DLDB: Extending Relational Databases to Support Semantic Web Queries, 2003).

The Ontology model follows the relational model to fit RDF model requirements with respect to the "triple" data model. The possible data values in RDF model are blank nodes, Uniform Resource Locators (URIs) or literals. The literals can

de 'plain literals' which are strings with an optional language tag as part of their value or 'typed literals' which are strings combined with data-type URI. The following three principles are followed in the Ontology Model representation.

- A subject can only be a URI or a blank node
- A predicate (or property) can only be a URI
- An object can be a URI, a blank node, or a literal.

There is a choice between storing "resources URIs" and "literals" in a separate lookup table or together with the statement. In the case of the first Semantic Web application framework –Jena1, the normalised approach of storing resource URIs and literals in a separate table from the main statement table lead to "expensive" table joins (Wilkinson K. , Sayers, Kuno, & Reynolds, 2003). Jena2 adopts different approach, which trades space for time. It uses denormalised schemes with resource URIs and literals stored in the same table. The fundamental principle behind the ontological model from relational model is the use of "class" to represent "table" and "property" to represent "relationship". "Relationship" to "property" mapping follows the ER (Entity Relationship) to Relational database design methodology (Ramakrishnan & Gehrke, 2000). For every relational table, there is a corresponding ontological "class" table commonly described as "property-class table" " (Abadi D. J., Marcus, Madden, & Hollenbach, 2009) but without the so called "left-over triples. Oracle also adopts a property table-like data structure (they call it a "subject-property matrix") to speed up queries over RDF triples. Their utilization of property tables is slightly different from Jena2 in that they are not used as a primary storage structure, but rather as an auxiliary data structure – a materialised view – that can be used to speed up specific types of queries (Abadi D. J., Marcus, Madden, & Hollenbach, 2009). Additional new ontological "property" table is created for every relational "relationship" between two relational tables.

6.2.1 Document Ontology Schema

The Ontology Schema for document's *subject* consists of 'Attachment' and 'Collection' *objects* connected by 'hasAttachment' and 'belongToCollection' *predicate* tables. The data in all the schema column values are represented by URI that constitute a URL for the online address and protocol, ontology model name ('document', 'attachment', 'collection'), id and title.

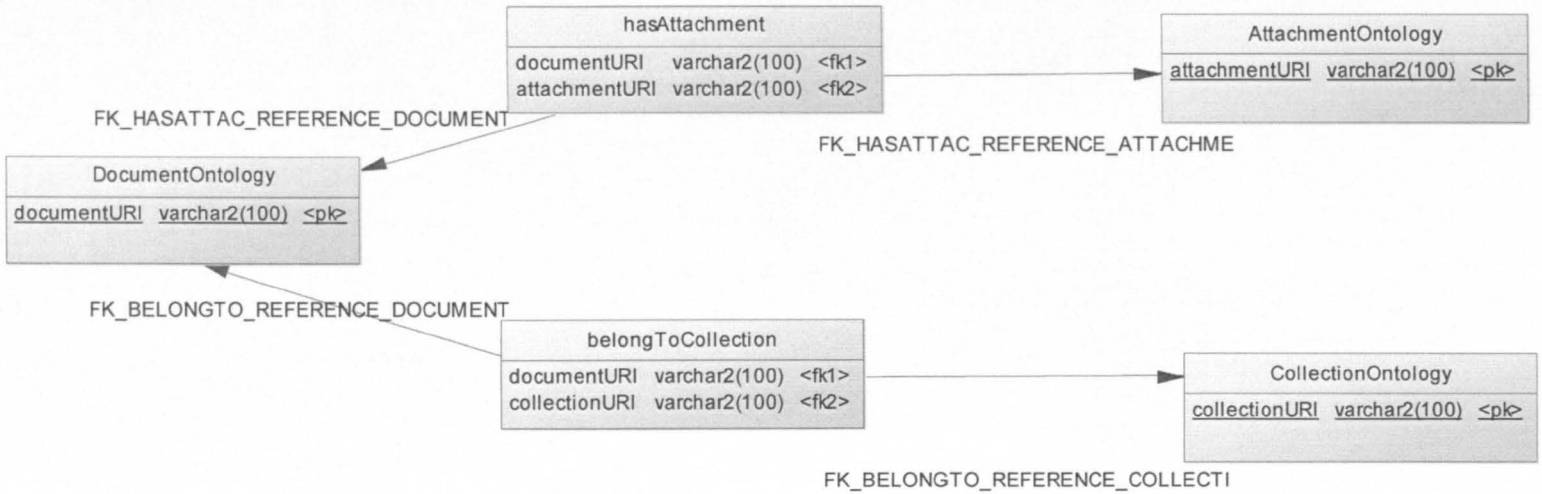


Figure 28- Document Ontology Schema Model

6.2.2 Profile Ontology Schema

The Ontology Schema for user's *subject* consists of 'Project' and 'Role' *objects* connected by 'belongToProject' and 'hasRole' *predicate* tables. The data in all the schema column values are represented by URI that constitute a URL for the online resource address, ontology model name ('user', 'project', 'role'), id and title.

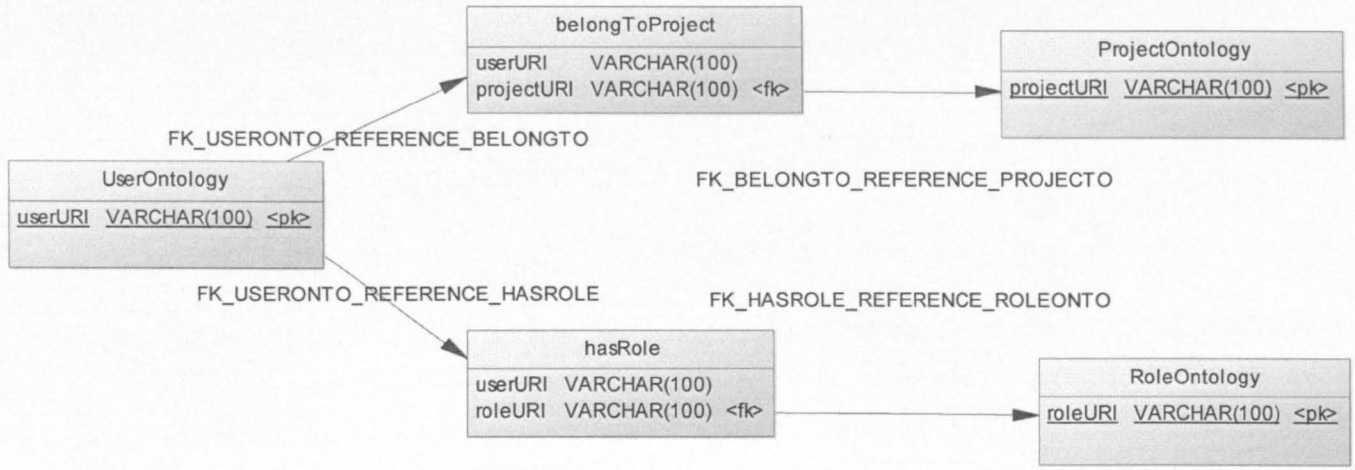


Figure 29- Profile Ontology Schema Model

6.2.3 Review Ontology Schema

The Ontology Schema for review's *subject* consists of 'Panel', 'Editorial', 'Role' and 'Classification' *objects* connected by 'hasPanel', 'hasEditorial', 'hasRole' and 'hasClassification' *predicate* tables. The data in all the schema column values are represented by URI that constitute a URL for the online address and protocol, ontology model name ('review', 'panel', 'editorial', 'role', 'classification'), id, title and/or name.

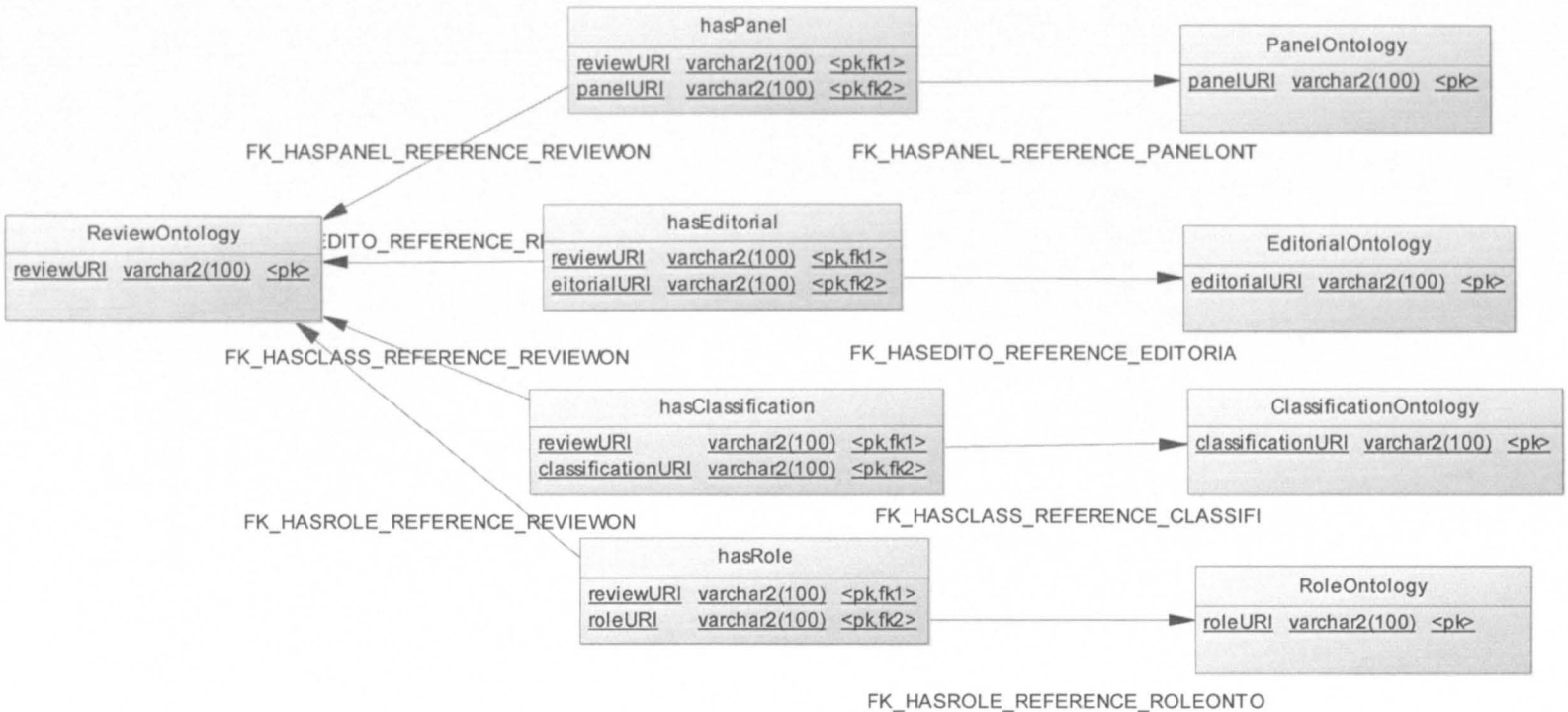


Figure 30- Review Ontology Schema Model

6.2.4 Action Ontology Schema

The Ontology Schema for action's *subject* consists of 'Resolution', 'Priority', 'Incident' and 'Status' *objects* connected by 'hasResolution', 'hasPriority' 'hasIncident', and 'hasStatus' *predicate* tables. The data in all the schema column values are represented by URI that constitute a URL for the online address and protocol, ontology model name ('action', 'resolution', 'priority', 'incident', 'status'), id, title and/or name.

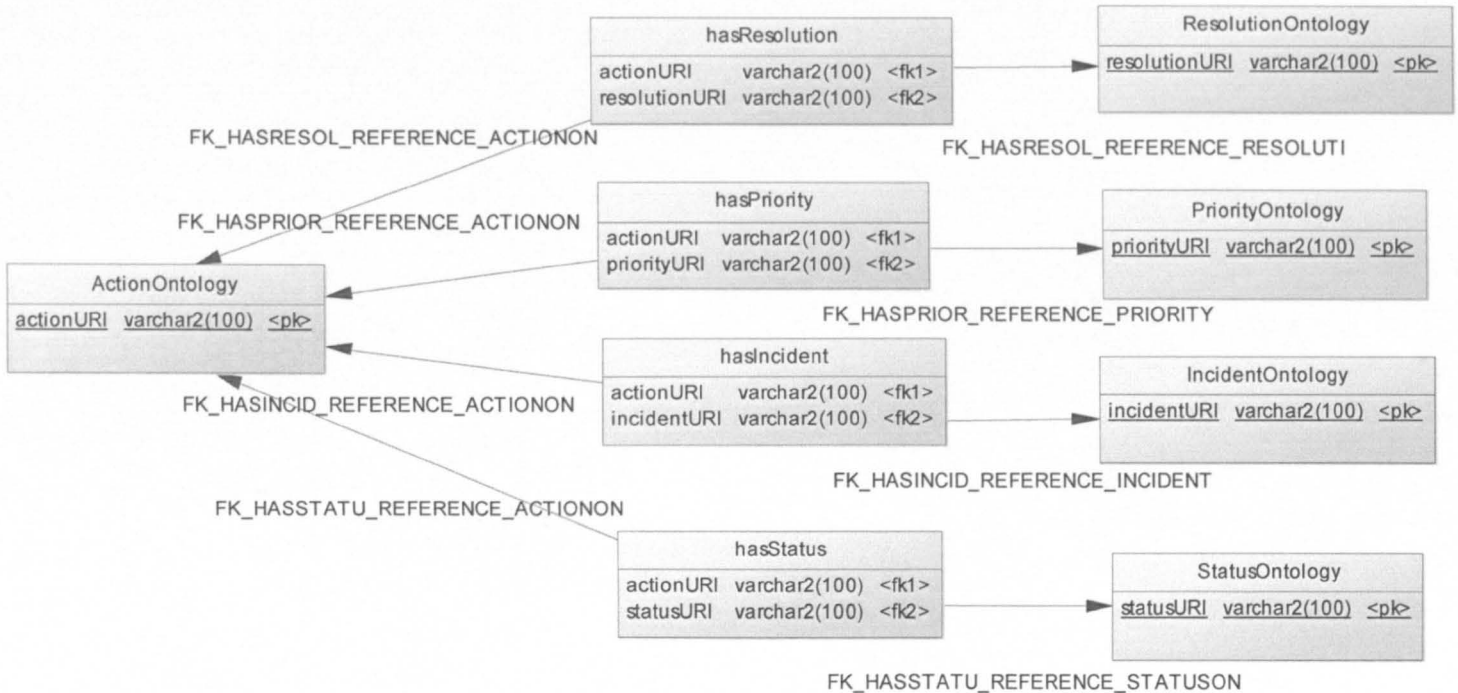


Figure 31- Action Ontology Schema Model

6.2.5 Risk Ontology Schema

The Ontology Schema for risk's *subject* consists of 'Domain', 'Scenario', 'Criticality', 'Rank' and 'Trail' *objects* connected by 'hasDomain', 'hasScenario' 'hasCriticality', 'hasRank' and 'hasTrail' *predicate* tables. The data in all the schema column values are represented by URI that constitute a URL for the online address and protocol, ontology model name ('risk', 'domain', 'scenario', 'criticality', 'rank', 'trail'), id, title and/or name.

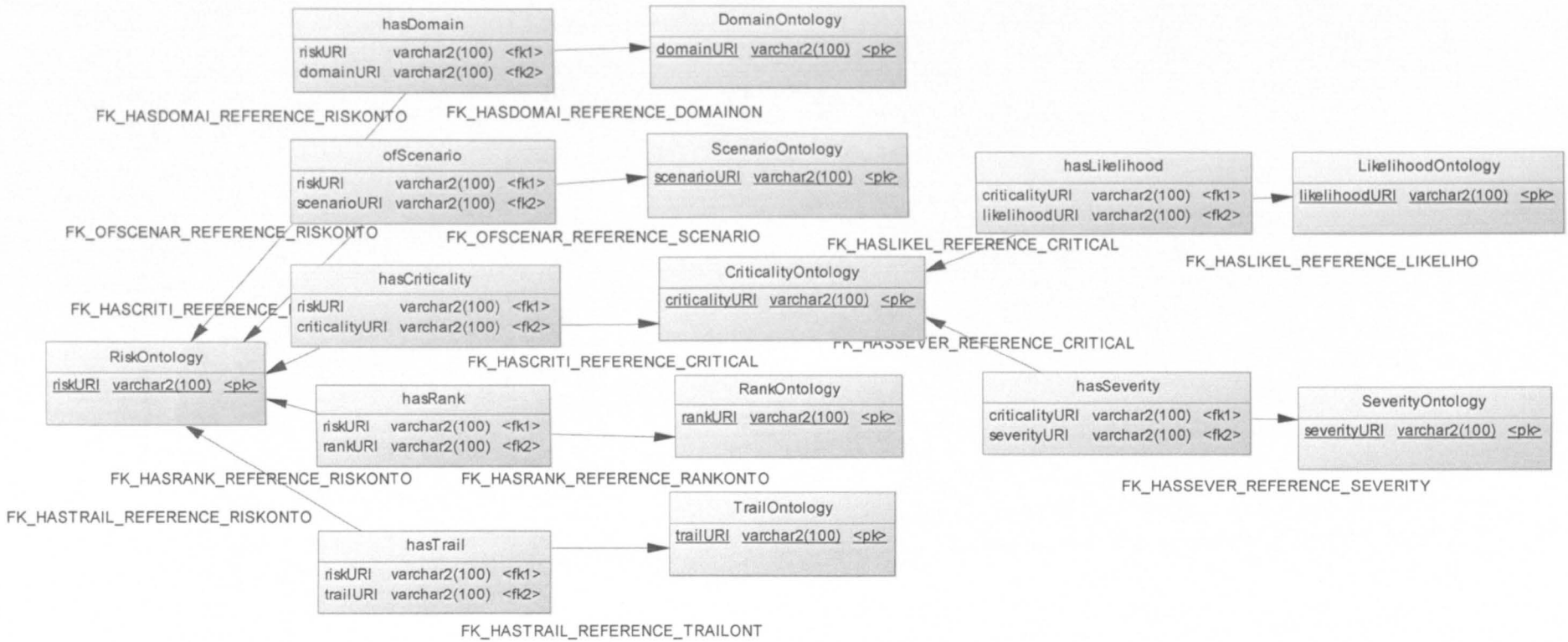


Figure 32- Risk Ontology Schema Model

6.2.6 Non Conformance Ontology Schema

The Ontology Schema for non-conformance's *subject* consists of 'Category', 'Effect', 'Model' and 'Disposition' *objects* connected by 'hasCategory', 'hasEffect', 'hasModel' and 'hasDisposition' *predicate* tables. The data in all the schema column values are represented by URI that constitute a URL for the online address and protocol, ontology model name ('nonConformance', 'category', 'effect', 'model', 'disposition'), id, title and/or name.

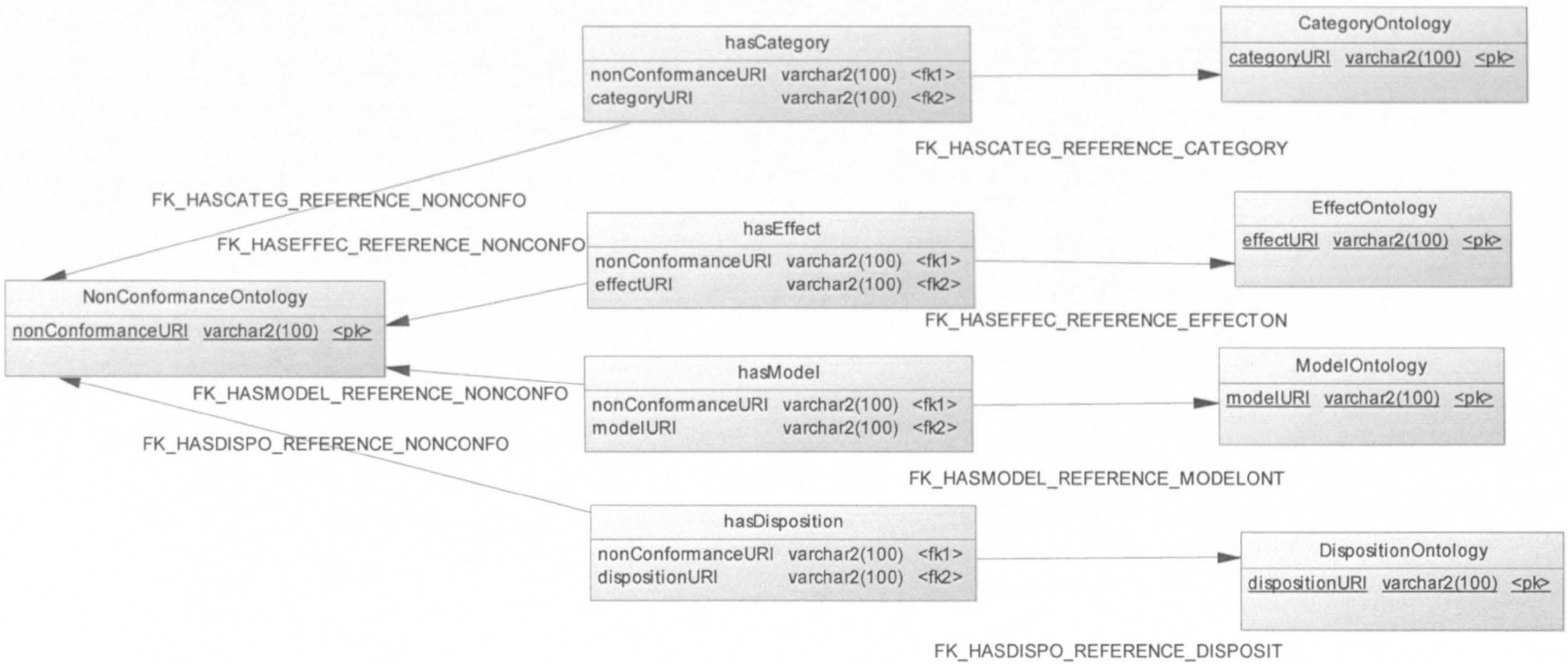


Figure 33- Non Conformance Ontology Schema Model

6.2.7 Summary (Ontology Schema)

The Ontology Schema models outlined in the sections above are used to implement RDF repository using relational database management systems. The relational tables and their relationships are represented by a "class" and "property" table respectively. Since relationships are turned into "property" tables, the number of ontological schema objects is significantly higher than the relational schema objects.

The methodology of using relational databases as both relational and ontological schema relies on the database management system's efficiency and scalability to manage the schemas. The database system's (Oracle in the case of this research project) capability to entertain Semantic Web as well as the availability of compatible ontology management frameworks (Jena in the case of this research project), greatly helps the mapping implementation.

Jena2 provides the capability to store RDF statements using JDBC connectivity. This approach makes relational databases not only being used as relational data store but also as "triple store" for RDF triple statements. Jena2' architecture as discussed in 3.5 Jena Framework covers different storage, inference and ontology layers. It also has transaction management and bulk load support. The upgrade of Jena1 to Jena2 introduced OWL and reasoning support that makes it the most compatible framework for the current Semantic Web Architecture.

The combination of using relational databases as RDF graph store and the use of Jena2 for OWL and logical reasoning makes the mapping of relational model to semantic model more practical and feasible. The following chapters present the actual implementation and evaluation based on the technologies Chapter 3: Review of Technology, Chapter 4: Domain Specific Knowledge, Chapter 5: Relational to Ontological Mapping Algorithms, and Chapter 6: Relational and Ontological Model.

Chapter 7: Relational to Ontological Mapping Implementation

The mapping of relational data to RDF starts by setting up the relational database system. In general, the database system consists of the different tablespace allocations, privileges, a system administrator and different application schemas with their corresponding objects. Using a system administrator account, all the schema owners are created and the necessary system privileges are granted. The schema owner names are used as application ontology concept names during the mapping. Each schema owner has its own set of relational objects like tables and referential integrity constraints.

The implementation is executed using four phases.

1. *Preliminary*:
   a) establish database connection and read configuration and property file.
   b) read the data dictionary and build working tables.
2. *Pre-Processing:*
   a) identify keys, constraints and relationships and create RDFS/OWL equivalent.
   b) look for domain data patterns within relations, data types, constraints etc.
3. *In-Processing*:
   a) apply the mapping algorithms in an incremental iterative transformation.
   b) map the relations and relationships into RDF model
   c) map SQL data type based data to RDF data type based URI
   d) apply different simple and complex data pattern mapping algorithms
4. *Post-Processing:*
   a) analyse the RDF for consolidation, unification, reification etc. using built in rules and logic.
   b) apply custom user-defined rules to reflect application domain predicates/functions/procedures and analyse RDF output.

This chapter presents the mapping implementation for preparing and mapping the relational database into a Semantic Web repository using *relational database area knowledge, domain data knowledge, domain users knowledge* and *application*

domain knowledge in the form of heuristic rules, logical reasoning and reification. The mapping software architecture has a Data Access Object (DAO) package to establish connection to the relational database and read keys, constraints, and relationships to identify what needs to be mapped during the *pre-processing* phase. In the *in-processing* phase the mapping algorithms are applied to the relational schema to produce RDFS/OWL schema using Service and Model components of the software architecture. The relational data and privilege are also mapped as RDF repository using RDFS/OWL schema as descriptors. *In-processing* phase also include different domain-specific data pattern mapping algorithms as well as application domain functions and predicates. Once the RDFS/OWL schema and RDF repository are mapped into a "Triple Store", reasoner component of the architecture is used to apply further reification as part of the *post-processing* phase. The presentation tier uses JSPs to display a tabular representation of the mapping. The application also uses different serialisation formats like RDF/XML and N-triples to display the mapping output.

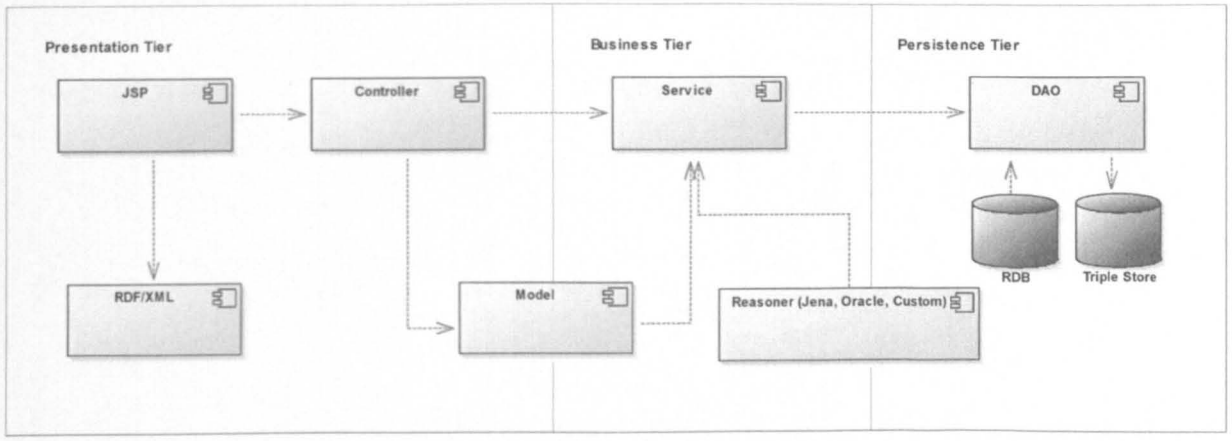


Figure 34- RDB to Ontological Repository Mapping Software Architecture

An interactive java application is developed for the mapping implementation. The application is divided into four broad menus.

1. **Relational**: menu that contains sub-menus to list tables, columns, constraints, relationships, privileges as well as sub-menus to map relational schema, data, and privilege.
2. **Ontological**: menu that contains sub-menus to display mapped ontology in different formats, grouping as well as the mapped tuple data.

3. **Reification**: contains sub-menus to display different Jena built-in, third party, custom as well as Oracle proprietary inference outputs.
4. **Configuration**: contains sub-menu to setup different relational database's connectivity and meta-data values.

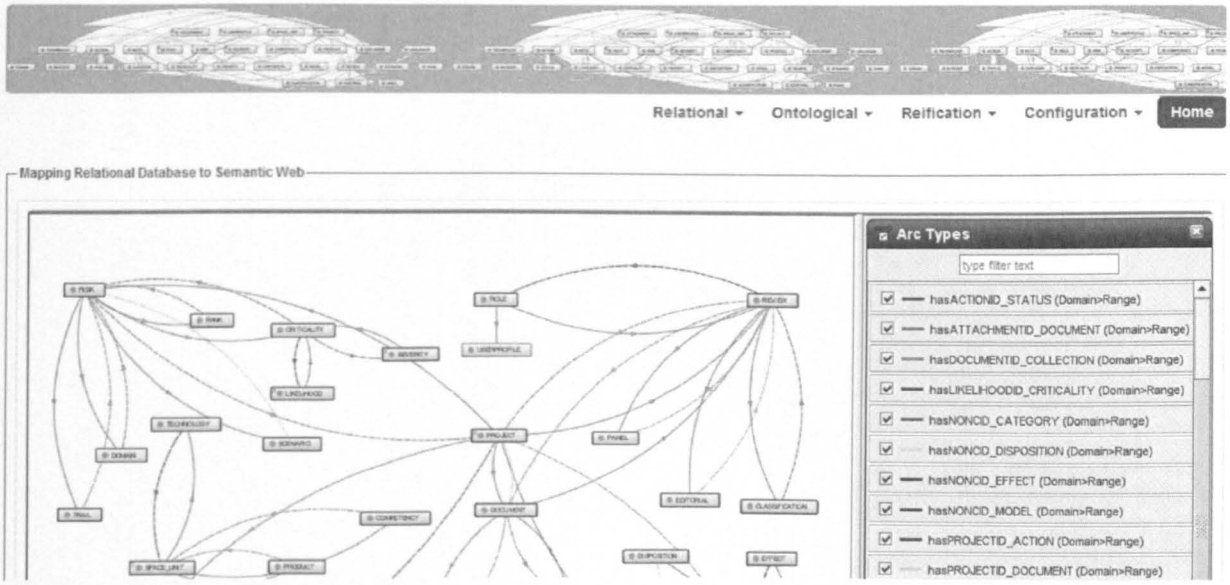


Figure 35- Mapping Java Application Home Page

7.1 Relational to Ontological Schema Mapping Implementation

The relational schema mapping implementation relies on the relational database area knowledge, i.e., relational database meta-data. The meta-data is stored in data dictionary tables that contain various information about each relational database object. Data dictionary is specific for each database system. To make the implementation independent on a specific database it uses a configuration file in which all DBMS specific features are listed. This file is read using DOM XML parser during the execution of the preliminary phase of the implementation and used throughout the interactive session. The Configuration of Oracle DBMS using XML is shown in *Table 11- Oracle Database Meta-data Configuration File* below.

- *name* : name of the relational table that contains the meta-data
- *owner*: relational schema owner of the table/ or database name
- *table_name*: name of the relational table that contains the relational data
- *column_name*: name of the column in the relational table
- *data_type*: SQL or non-SQL data type equivalent data type

- *data_length*: data length in bytes
- *data_precision*: data precision for NUMBER and FLOAT data types
- *data_scale*: number of digits after the decimal point
- *nullable*: specifies if the column allows NULL or not
- *constraint_name*: name of constraint
- *constraint_type*: type of constraint
  - List of Oracle constraint types
    - P: primary key
    - R: referential integrity
    - U: unique
    - C: check constraint
- *r_owner*: owner of table referred by a referential constraint
- *r_constraint_name*: referenced table unique constraint name
- *search_condition*: check constraint search condition text

Table 11- Oracle Database Meta-data Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<Oracle_Database>
  <table>
    <name>all_tab_columns</name>
    <owner>owner</owner>
    <table_name>table_name</table_name>
    <column_name>column_name</column_name>
    <data_type>data_type</data_type>
    <data_length>data_length</data_length>
    <data_precision>data_precision</data_precision>
    <data_scale>data_scale</data_scale>
    <nullable>nullable</nullable>
  </table>
  <table>
    <name>all_cons_columns</name>
    <owner>owner</owner>
    <table_name>table_name</table_name>
    <column_name>column_name</column_name>
    <constraint_name>constraint_name</constraint_name>
  </table>
  <table>
    <name>all_constraints</name>
    <owner>owner</owner>
    <table_name>table_name</table_name>
    <constraint_name>constraint_name</constraint_name>
    <constraint_type>constraint_type</constraint_type>
    <r_owner>r_owner</r_owner>
    <r_constraint_name>r_constraint_name</r_constraint_name>
    <search_condition>search_condition</search_condition>
  </table>
```

```
</Oracle_Database>
```

Ontology Classes

RDFS/OWL classes in the ontology are the equivalent of tables in the Relational Model. The database configuration file contains information about the standard names used by the particular DBMS. In this mapping process *mapTable()* algorithm is used to implement the procedure that maps the tables to "*classes*". The implementation uses *"constraint_type"* as a criteria to identify the primary key (PK) of the table. Here is the sample list of relational tables that *mapTable()* procedure loops through.

Table 12- Relational Tables and Primary key(s) Sample List

Relational Tables

29 items found, displaying 1 to 20.

[First/Prev] 1, 2 [Next/Last]

Table Name	Primary Key	Constraint Name
ACTION	ACTIONID	PK_ACTION
ATTACHMENT	ATTACHMENTID	PK_ATTACHMENT
CATEGORY	CATEGORYID	PK_CATEGORY
CLASSIFICATION	CLASSID	PK_CLASSIFICATION
COLLECTION	COLLECTIONID	PK_COLLECTION
CRITICALITY	CRITICALITYID	PK_CRITICALITY
DISPOSITION	DISPOSITIONID	PK_DISPOSITION
DOCUMENT	DOCUMENTID	PK_DOCUMENT
DOMAIN	DOMAINID	PK_DOMAIN
EDITORIAL	EDITORIALID	PK_EDITORIAL
EFFECT	EFFECTID	PK_EFFECT
INCIDENT	ACTIONID	PK_INCIDENT
	INCIDENTID	PK_INCIDENT
LIKELIHOOD	LIKELIHOODID	PK_LIKELIHOOD
NONCONF	NONCID	PK_NONCONF
PANEL	PANELID	PK_PANEL
PRIORITY	PRIORITYID	PK_PRIORITY
PROJECT	PROJECTID	PK_PROJECT
RANK	RANKID	PK_RANK

Using *mapDatabase()* as the main algorithm, *mapTable(), mapColumn(), mapConstraint(),* and *mapRelationship()* algorithms are executed incrementally. The result of *mapTable()* algorithm implementation in *Table 13- OWL Classes List* shows a sample of the mapped "classes" together with the functional properties as in Table 14- Sample OWL Functional Properties.

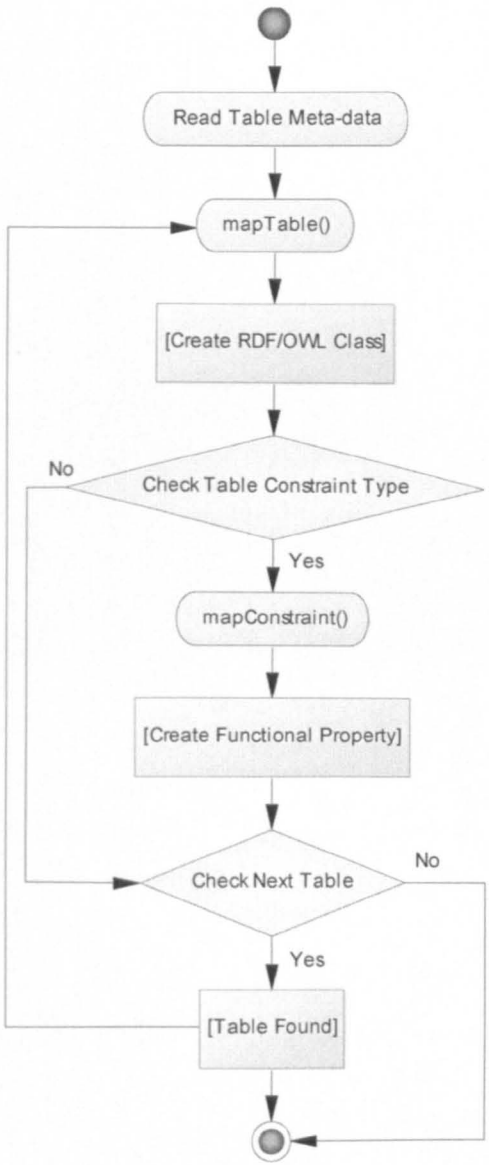


Figure 36- Mapping Tables and Constraints Activity Diagram

Table 13- OWL Classes List

```
<!--

///////////////////////////////////////////////////////////////////////////////
//////////////
    //
    // Classes
    //

///////////////////////////////////////////////////////////////////////////////
//////////////
    -->
<owl:Class rdf:about="urn:spaceProj-Mgmt/ACTION"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/ATTACHMENT"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/CATEGORY"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/CLASSIFICATION"></owl:Class>
```

```
<owl:Class rdf:about="urn:spaceProj-Mgmt/COLLECTION"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/CRITICALITY"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/DISPOSITION"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/DOCUMENT"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/DOMAIN"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/EDITORIAL"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/EFFECT"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/INCIDENT"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/LIKELIHOOD"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/MODEL"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/NONCONF"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/PANEL"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/PRIORITY"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/PROJECT"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/RANK"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/RESOLUTION"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/REVIEW"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/RISK"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/ROLE"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/SCENARIO"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/SEVERITY"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/STATUS"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/TRAIL"></owl:Class>
<owl:Class rdf:about="urn:spaceProj-Mgmt/USERPROFILE"></owl:Class>
```

Table 14- Sample OWL Functional Properties

```
<!--

///////////////////////////////////////////////////////////////////////
//////////////
    //
    // Functional properties
    //

///////////////////////////////////////////////////////////////////////
//////////////
     -->

    <!-- urn:spaceProj-Mgmt/hasACTIONID -->

    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="&xsd;double"/>
    </owl:DatatypeProperty>

    <!-- urn:spaceProj-Mgmt/hasATTACHMENTID -->

    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasATTACHMENTID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="&xsd;double"/>
    </owl:DatatypeProperty>


    <!-- urn:spaceProj-Mgmt/hasCATEGORYID -->

    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCATEGORYID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="&xsd;double"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CATEGORY"/>
```

```
    </owl:DatatypeProperty>

    <!-- urn:spaceProj-Mgmt/hasCLASSID -->

    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCLASSID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="&xsd;double"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CLASSIFICATION"/>
    </owl:DatatypeProperty>


    <!-- urn:spaceProj-Mgmt/hasCOLLECTIONID -->

    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCOLLECTIONID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="&xsd;double"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/COLLECTION"/>
    </owl:DatatypeProperty>



    <!-- urn:spaceProj-Mgmt/hasCOMPETENCYID -->

    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCOMPETENCYID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="&xsd;double"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/COMPETENCY"/>
    </owl:DatatypeProperty>


    <!-- urn:spaceProj-Mgmt/hasCRITICALITYID -->

    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasCRITICALITYID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="&xsd;double"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CRITICALITY"/>
    </owl:DatatypeProperty>

    <!-- urn:spaceProj-Mgmt/hasDISPOSITIONID -->

    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasDISPOSITIONID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="&xsd;double"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DISPOSITION"/>
    </owl:DatatypeProperty>

    <!-- urn:spaceProj-Mgmt/hasDOCUMENTID -->

    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="&xsd;double"/>
    </owl:DatatypeProperty>
```

Ontology Data Properties

After the mapping of relational tables to OWL classes, the next stage of the mapping implementation is the mapping of relational columns into OWL data properties. The implementation procedure uses *mapColumn()* and *mapDatabase()* algorithms. The procedure also uses *mapDatatype()* procedure to map relational SQL data types to RDF data types.

The implementation relies on the configuration file to identify the table name, column name, data type (including data type, data precision, scale, nullablity) etc.

Table 15- Relational Columns Sample List

Relational Columns

91 items found, displaying 1 to 20.

[First/Prev] 1, 2, 3, 4, 5 [Next/Last]

Table Name	Column Name	Data Type	Data Length	Data Precision	Data Scale	Nullable
ACTION	ACTIONID	NUMBER	22	0	0	False
	DOCUMENTID	NUMBER	22	0	0	True
	ACTIONDATE	DATE	7	0	0	True
	ACTIONSTATUS	VARCHAR2	50	0	0	True
	ACTIONTITLE	VARCHAR2	50	0	0	True
ATTACHMENT	ATTACHMENTID	NUMBER	22	0	0	False
CATEGORY	CATEGORYID	NUMBER	22	0	0	False
	NONCID	NUMBER	22	0	0	True
	CATEGORYNAME	VARCHAR2	20	0	0	True
CLASSIFICATION	CLASSID	NUMBER	22	0	0	False
	REVIEWID	NUMBER	22	0	0	True
	CLASSNAME	VARCHAR2	20	0	0	True
COLLECTION	COLLECTIONID	NUMBER	22	0	0	False
	DOCUMENTID	NUMBER	22	0	0	True
CRITICALITY	CRITICALITYID	NUMBER	22	0	0	False
	LIKELIHOODID	NUMBER	22	0	0	True
	RISKID	NUMBER	22	0	0	True
	SEVERITYID	NUMBER	22	0	0	True
	CRITICALITYNAME	VARCHAR2	20	0	0	True
DISPOSITION	DISPOSITIONID	NUMBER	22	0	0	False

The relational columns and their attributes are mapped into the OWL classes as data properties using the equivalent RDF data types.

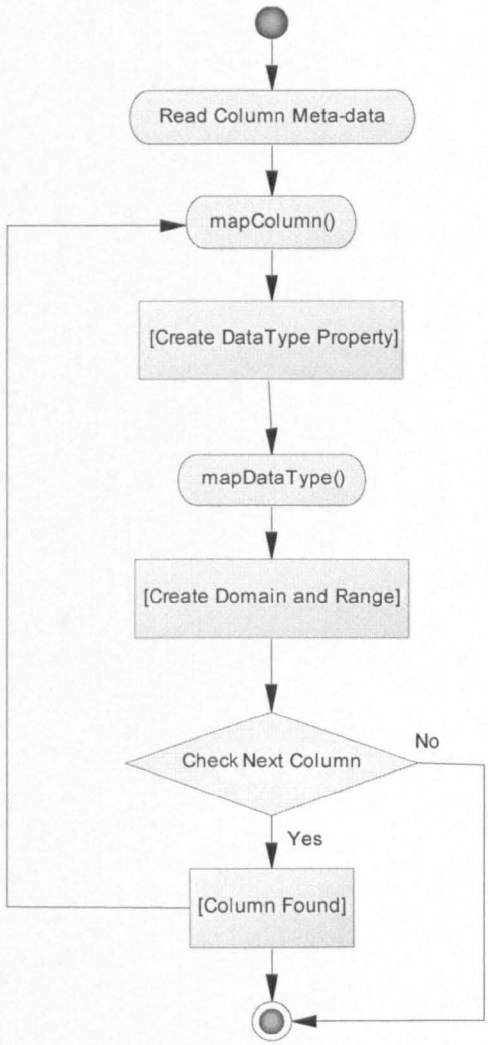


Figure 37- Mapping Columns and Data Types Activity Diagram

Table 16- Sample OWL Datatype Properties shows the output of the mapColumn() and mapDatatype() algorithms using OWL data type property and RDFS domain and range axioms.

Table 16- Sample OWL Datatype Properties

```
<!--
    ///////////////////////////////////////////////////////////////////////
    //
    // Data properties
    //
    ///////////////////////////////////////////////////////////////////////
     -->


    <!-- urn:spaceProj-Mgmt/hasACTIONDATE -->

    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONDATE">
        <rdfs:range rdf:resource="&xsd;dateTime"/>
```

```
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION"/>
</owl:DatatypeProperty>



<!-- urn:spaceProj-Mgmt/hasACTIONID -->

<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:range rdf:resource="&xsd;double"/>
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION"/>
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/INCIDENT"/>
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PRIORITY"/>
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RESOLUTION"/>
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/STATUS"/>
    <rdfs:range rdf:resource="urn:spaceProj-Mgmt/&xsd;double"/>
</owl:DatatypeProperty>



<!-- urn:spaceProj-Mgmt/hasACTIONSTATUS -->

<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONSTATUS">
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION"/>
</owl:DatatypeProperty>



<!-- urn:spaceProj-Mgmt/hasACTIONTITLE -->

<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONTITLE">
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION"/>
</owl:DatatypeProperty>
```

Ontology Constraints

The mapping of relational constraints uses the "constraint_name" and "constraint_type" meta-data of the configuration XML file to determine the corresponding property in OWL. The constraint types considered are the main SQL constraints like primary key 'pk', foreign key 'fk', UNIQUE 'unq', NOT NULL 'nn', and CHECK 'ck'. *mapTable()* procedure already used primary key constraint to determine a functional property to the mapped OWL classes.

The implementation procedure uses *mapConstraint()* and *mapDatabase()* algorithms to identify the relational constraints.

Table 17- Relational Constraints Sample List

Relational Constraints

56 items found, displaying 1 to 20.

[First/Prev] 1, 2, 3 [Next/Last]

Table Name	Column Name	Constraint Name	Constraint Type
ACTION	ACTIONID	PK_ACTION	P
	DOCUMENTID	FK_ACTION_DOCUMENT	R
	ACTIONSTATUS	CK_ACTION_STATUS	C
ATTACHMENT	ATTACHMENTID	PK_ATTACHMENT	P
CATEGORY	CATEGORYID	PK_CATEGORY	P
	NONCID	FK_CATEGORY_REFERENCE_NONCONF	R
CLASSIFICATION	CLASSID	PK_CLASSIFICATION	P
COLLECTION	COLLECTIONID	PK_COLLECTION	P
	DOCUMENTID	FK_COLLECTON_DOCUMENT	R
CRITICALITY	CRITICALITYID	PK_CRITICALITY	P
	LIKELIHOODID	FK_CRITICAL_REFERENCE_LIKELIHO	R
	RISKID	FK_CRITICAL_REFERENCE_RISK	R
	SEVERITYID	FK_CRITICAL_REFERENCE_SEVERITY	R
DISPOSITION	DISPOSITIONID	PK_DISPOSITION	P
	NONCID	FK_DISPOSIT_REFERENCE_NONCONF	R
DOCUMENT	DOCUMENTID	PK_DOCUMENT	P
	ATTACHMENTID	FK_DOCUMENT_REFERENCE_ATTACHME	R
	PROJECTID	FK_DOCUMENT_REFERENCE_PROJECT	R
DOMAIN	DOMAINID	PK_DOMAIN	P
	RISKID	FK_DOMAIN_REFERENCE_RISK	R

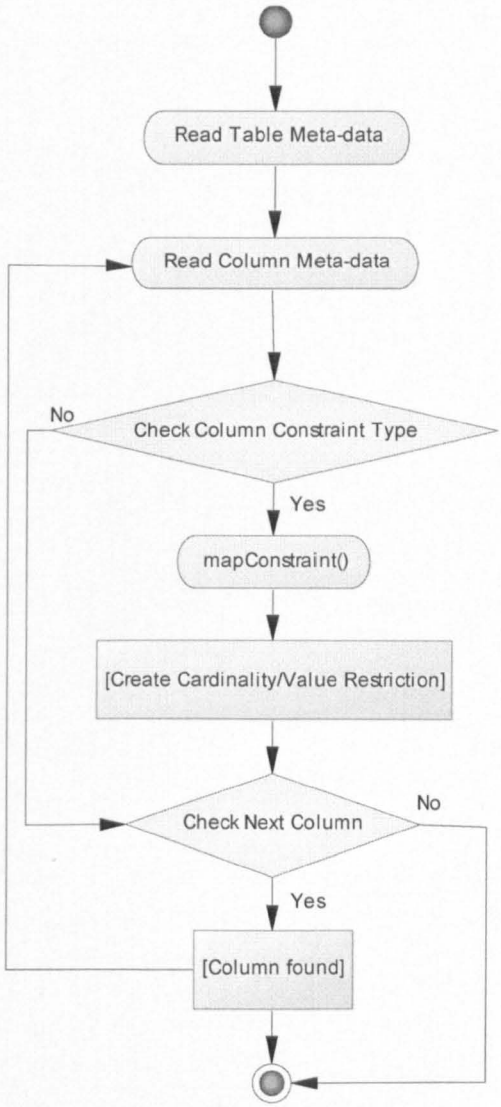


Figure 38- Mapping Column Constraints Activity Diagram

The constraint types are used to keep the entity, referential and data integrity. The entity and referential integrity mapping are demonstrated in OWL Classes and OWL Object Properties. *Table 18- OWL Value and Cardinality Constraints Property Restriction* shows a sample OWL "value constraint" on "hasACTIONSTATUS" property and OWL "cardinality constraint" on "hasACTIONID" property of ACTION OWL class.

Table 18- OWL Value and Cardinality Constraints Property Restriction

```
<!--
    ///////////////////////////////////////////////////////////////////////
    //
    // OWL Restriction and Cardinality
    //
```

```
    ////////////////////////////////////////////////////////////////////////
     -->

<owl:Class rdf:about="urn:spaceProj-Mgmt/ACTION" >
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasACTIONID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
    <rdfs:subClassOf>
        <owl:Restriction>
         <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasACTIONSTATUS" />
            <owl:hasValue rdf:datatype="&xsd;string">OPEN
        </owl:hasValue>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
         <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasACTIONSTATUS" />
            <owl:hasValue rdf:datatype="&xsd;string">CLOSED
        </owl:hasValue>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
         <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasACTIONSTATUS" />
            <owl:hasValue rdf:datatype="&xsd;string">ARCHIEVED
        </owl:hasValue>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Ontology Object Properties

The relational relationships mapping implementation relies on the "foreign key" constraint to identify the relationship amongst relational tables. The configuration XML file's *"constraint_name"* attribute is used in conjunction with the *"r_constraint_name"* attribute to map the relationship between the referencing and the referenced tables.

Table 19- Relational Relationships Sample List

Relational Relationships

29 items found, displaying 1 to 20.

[First/Prev] 1, 2 [Next/Last]

Table Name	Column Name	Constraint Name	Constraint Type	Referenced Constraint Name	Referenced Table Name
ACTION	DOCUMENTID	FK_ACTION_DOCUMENT	R	PK_DOCUMENT	DOCUMENT
CATEGORY	NONCID	FK_CATEGORY_REFERENCE_NONCONF	R	PK_NONCONF	NONCONF
CLASSIFICATION	REVIEWID	FK_CLASSIFI_REFERENCE_REVIEW	R	PK_REVIEW	REVIEW
COLLECTION	DOCUMENTID	FK_COLLECTON_DOCUMENT	R	PK_DOCUMENT	DOCUMENT
CRITICALITY	LIKELIHOODID	FK_CRITICAL_REFERENCE_LIKELIHO	R	PK_LIKELIHOOD	LIKELIHOOD
CRITICALITY	RISKID	FK_CRITICAL_REFERENCE_RISK	R	PK_RISK	RISK
CRITICALITY	SEVERITYID	FK_CRITICAL_REFERENCE_SEVERITY	R	PK_SEVERITY	SEVERITY
DISPOSITION	NONCID	FK_DISPOSIT_REFERENCE_NONCONF	R	PK_NONCONF	NONCONF
DOCUMENT	ATTACHMENTID	FK_DOCUMENT_REFERENCE_ATTACHME	R	PK_ATTACHMENT	ATTACHMENT
DOCUMENT	PROJECTID	FK_DOCUMENT_REFERENCE_PROJECT	R	PK_PROJECT	PROJECT
DOMAIN	RISKID	FK_DOMAIN_REFERENCE_RISK	R	PK_RISK	RISK
EDITORIAL	REVIEWID	FK_EDITORIA_REFERENCE_REVIEW	R	PK_REVIEW	REVIEW
EFFECT	NONCID	FK_EFFECT_REFERENCE_NONCONF	R	PK_NONCONF	NONCONF
INCIDENT	ACTIONID	FK_INCIDENT_REFERENCE_ACTION	R	PK_ACTION	ACTION
NONCONF	DOCUMENTID	FK_NONCONF_DOCUMENT	R	PK_DOCUMENT	DOCUMENT
NONCONF	PROJECTID	FK_NONCONF_PROJECT	R	PK_PROJECT	PROJECT
PANEL	REVIEWID	FK_PANEL_REFERENCE_REVIEW	R	PK_REVIEW	REVIEW
PRIORITY	ACTIONID	FK_PRIORITY_REFERENCE_ACTION	R	PK_ACTION	ACTION
RANK	RISKID	FK_RANK_REFERENCE_RISK	R	PK_RISK	RISK

The relational relationships identified are mapped to their corresponding "Object Properties" using the *mapRelationship()* and *mapDatabase()* procedures.

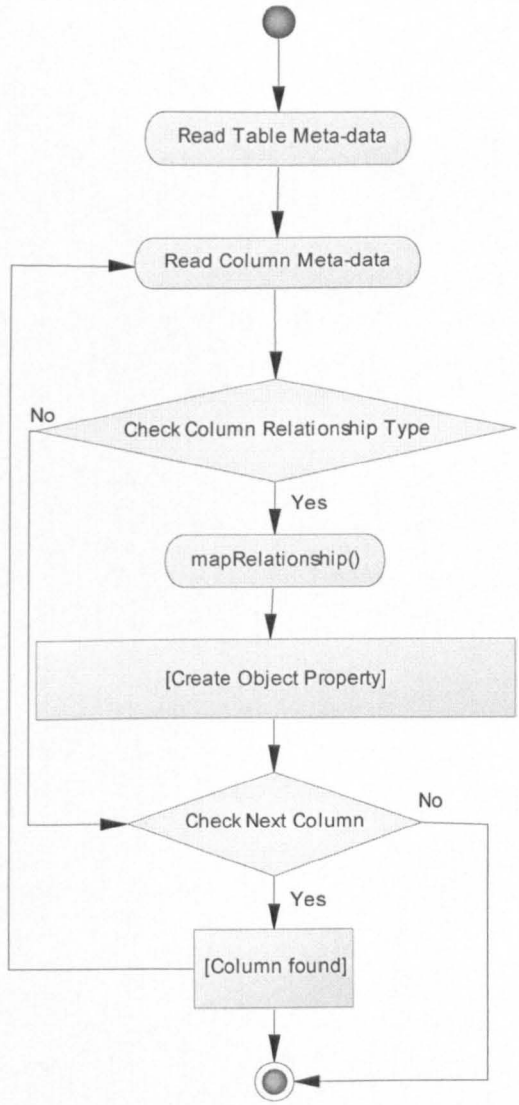


Figure 39- Mapping Column Relationships Activity Diagram

Table 20- Sample OWL Object Properties shows the output of *mapRelationship()* algorithm implementation that includes sub-algorithms like *CheckTransitiveChain()* and *CheckDisjointness()*.

Table 20- Sample OWL Object Properties

```
<!--

///////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //

///////////////////////////////////////////////////////////////////////////
     -->

    <!-- urn:spaceProj-Mgmt/hasATTACHMENTID -->
```

```
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasATTACHMENTID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:range rdf:resource="urn:spaceProj-Mgmt/ATTACHMENT"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ATTACHMENT"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOCUMENT"/>
    </owl:ObjectProperty>



    <!-- urn:spaceProj-Mgmt/hasDOCUMENTID -->

    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
        <rdf:type rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/COLLECTION"/>
        <rdfs:range rdf:resource="urn:spaceProj-Mgmt/DOCUMENT"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOCUMENT"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/NONCONF"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/REVIEW"/>
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RISK"/>
    </owl:ObjectProperty>
```

7.2 *Relational to RDF Data Population Implementation*

Once the relational schema is mapped to Ontology Schema, the next step is to map the relational data into RDF repositories. Relational data population implementation involves *mapTable(), mapColumn()* and *mapDatatype()* algorithms. The mapping application reads all the tables, columns and data type from relational database meta-data and constructs the SQL to read the data from the database. One the data is read, *mapDatatype()* algorithm is used to map SQL data types to RDF data types. Using the mapped ontological classes and properties, the actual data is populated with the corresponding RDF data type.

Table 21- Sample Mapped RDF Data

```
<ACTION rdf:about="&spaceOnt;ACTION">
        <hasACTIONDATE rdf:datatype="&xsd;dateTime">2012-09-06T00:00:00-
00:00</hasACTIONDATE>
        <hasACTIONID rdf:datatype="&xsd;double">3001</hasACTIONID>
        <hasACTIONTITLE rdf:datatype="&xsd;string">Challenger Compressor
Followup</hasACTIONTITLE>
        <hasDOCUMENTID rdf:datatype="&xsd;double">2001</hasDOCUMENTID>
        <hasACTIONSTATUS
rdf:datatype="&xsd;string">OPEN</hasACTIONSTATUS>
</ACTION>

<ACTION rdf:about="&spaceOnt;ACTION">
        <hasACTIONDATE rdf:datatype="&xsd;dateTime">2013-10-06T00:00:00-
00:00</hasACTIONDATE>
```

```
        <hasACTIONID rdf:datatype="&xsd;double">3002</hasACTIONID>
        <hasACTIONTITLE rdf:datatype="&xsd;string">Mission Satellite
Incident</hasACTIONTITLE>
        <hasDOCUMENTID rdf:datatype="&xsd;double">2002</hasDOCUMENTID>
        <hasACTIONSTATUS
rdf:datatype="&xsd;string">ARCHIEVED</hasACTIONSTATUS>
</ACTION>

<ACTION rdf:about="&spaceOnt;ACTION">
</ACTION>
<!-- urn:spaceProj-Mgmt/DOCUMENT -->

<DOCUMENT rdf:about="&spaceOnt;DOCUMENT">
        <hasCREATEDBY rdf:datatype="&xsd;string">Smith</hasCREATEDBY>
        <hasCREATEDDATE rdf:datatype="&xsd;dateTime">2011-09-06T00:00:00-
00:00</hasCREATEDDATE>
        <hasDOCUMENTID rdf:datatype="&xsd;double">2001</hasDOCUMENTID>
        <hasPROJECTID rdf:datatype="&xsd;double">1001</hasPROJECTID>
        <hasDOCUMENTTITLE rdf:datatype="&xsd;string">Challenger
Configuration</hasDOCUMENTTITLE>
</DOCUMENT>

<DOCUMENT rdf:about="&spaceOnt;DOCUMENT">
        <hasCREATEDBY rdf:datatype="&xsd;string">Louis</hasCREATEDBY>
        <hasCREATEDDATE rdf:datatype="&xsd;dateTime">2012-11-02T00:00:00-
00:00</hasCREATEDDATE>
        <hasDOCUMENTID rdf:datatype="&xsd;double">2002</hasDOCUMENTID>
        <hasPROJECTID rdf:datatype="&xsd;double">1002</hasPROJECTID>
        <hasDOCUMENTTITLE rdf:datatype="&xsd;string">Mission
Schedule</hasDOCUMENTTITLE>
</DOCUMENT>

<!-- urn:spaceProj-Mgmt/PROJECT -->

<PROJECT rdf:about="&spaceOnt;PROJECT">
        <hasPROJECTNAME rdf:datatype="&xsd;string">Challenger
Launch</hasPROJECTNAME>
        <hasPROJECTID rdf:datatype="&xsd;double">1001</hasPROJECTID>
</PROJECT>

<PROJECT rdf:about="&spaceOnt;PROJECT">
        <hasPROJECTNAME rdf:datatype="&xsd;string">Galilio Observatory
</hasPROJECTNAME>
        <hasPROJECTID rdf:datatype="&xsd;double">1002</hasPROJECTID>
</PROJECT>
```

7.3 *User Profiles and Access Rights Mapping Implementation*

User Profile and Access Rights are part of the *Domain Users Knowledge*. The mapping of the database data must cover different levels of abstraction. The more abstract the knowledge is the more important it becomes to include User Profiles and data Access Rights as part of the mapping implementation. The User Profiles and the Access Rights descriptions are specified externally in a configuration file,

which allows the mapping to be independent on the RDBMS specifics. The different access right metadata (Select, Delete, Update, Insert, Reference, etc.) are matched against the relational database objects (tables, columns, views, etc.) to define "what right a user has on a specific relational object".

The sample Oracle User Profile metadata configuration and an excerpt of the access right data are shows in Table 22- Oracle User Profile Meta-data Configuration File and Table 23- Sample Mapped Access Right Data below.

Table 22- Oracle User Profile Meta-data Configuration File

```
<Oracle_Database>
  <table>
    <name>all_tab_privs</name>
    <table_schema>table_schema</table_schema>
    <table_name>table_name</table_name>
    <privilege>privilege</privilege>
    <grantor>grantor</grantor>
    <grantee>grantee</grantee>
  </table>
</Oracle_Database>
```

Table 23- Sample Mapped Access Right Data

```
<!-- urn:spaceProj-Mgmt/DOCUMENT -->
<DOCUMENT rdf:about="&spaceOnt;DOCUMENT">
        <hasPrivilege rdf:datatype="&xsd;string">INSERT</hasPrivilege>
        <hasPrivilege rdf:datatype="&xsd;string">DELETE</hasPrivilege>
        <hasPrivilege rdf:datatype="&xsd;string">SELECT</hasPrivilege>
        <hasPrivilege rdf:datatype="&xsd;string">UPDATE</hasPrivilege>
        <grantee rdf:datatype="&xsd;string">RELATIONALONT</grantee>
</DOCUMENT>
<!-- urn:spaceProj-Mgmt/PROJECT -->

<PROJECT rdf:about="&spaceOnt;PROJECT">
        <hasPrivilege rdf:datatype="&xsd;string">INSERT</hasPrivilege>
        <hasPrivilege rdf:datatype="&xsd;string">UPDATE</hasPrivilege>
        <grantee rdf:datatype="&xsd;string">RELATIONALONT</grantee>
</PROJECT>
<!-- urn:spaceProj-Mgmt/REVIEW -->

<REVIEW rdf:about="&spaceOnt;REVIEW">
        <hasPrivilege rdf:datatype="&xsd;string">INSERT</hasPrivilege>
        <hasPrivilege rdf:datatype="&xsd;string">SELECT</hasPrivilege>
        <grantee rdf:datatype="&xsd;string">RELATIONALONT</grantee>
</REVIEW>
```

The mapped Ontology schema that describes the mapped RDF data is also used to describe the mapped privilege data. *Table 24- Sample mapped Ontology and data* shows the mapped instance data and its corresponding privilege data for each OWL class.

Table 24- Sample mapped Ontology and data

Subject	Predicate	Object
urn:spaceProj-Mgmt/ACTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/ACTION
	urn:spaceProj-Mgmt/hasACTIONTITLE	Mission Satellite Incident
	urn:spaceProj-Mgmt/hasDOCUMENTID	2.002E3^^http://www.w3.org/2001/XMLSchema#double
	urn:spaceProj-Mgmt/hasACTIONSTATUS	ARCHIEVED
	urn:spaceProj-Mgmt/hasACTIONID	3.002E3^^http://www.w3.org/2001/XMLSchema#double
	urn:spaceProj-Mgmt/hasACTIONDATE	2013-10-06T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime
urn:spaceProj-Mgmt/PROJECT	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/PROJECT
	urn:spaceProj-Mgmt/hasPROJECTNAME	Galilio Observatory
	urn:spaceProj-Mgmt/hasPROJECTID	1.002E3^^http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/REVIEW
	urn:spaceProj-Mgmt/hasACTIONID	3.002E3^^http://www.w3.org/2001/XMLSchema#double
	urn:spaceProj-Mgmt/hasREVIEWID	4.002E3^^http://www.w3.org/2001/XMLSchema#double
	urn:spaceProj-Mgmt/hasROLEID	2E0^^http://www.w3.org/2001/XMLSchema#double
	urn:spaceProj-Mgmt/hasREVIEWTITLE	Satellite Re-launch
	urn:spaceProj-Mgmt/grantee	RELATIONALONT
	urn:spaceProj-Mgmt/hasPrivilege	INSERT
	urn:spaceProj-Mgmt/hasPrivilege	SELECT

7.4 *Reification and Consolidation of Generated repository*

The result of mapping the relational database schema and data into OWL Classes and RDF repository opens the door for *Semantic Reification and Analysis*. After the mapping different semantic inference mechanisms can be applied on the generated Ontology and RDF data. The implementation shows the output of the different levels of inference capability. Some of them include specific constructs for special purpose semantic inference. The reasoners also have different levels within the same Ontology language that include common as well as unique reasoning features. These semantic inference mechanisms include open source, proprietary, third party and User-defined reasoners.

- *Jena built-in*: RDFS, OWL, OWL Micro, OWL Mini, DAML, etc.
- *Oracle Proprietary*: RDF++, OWLSIF, OWLPRIME, SKOSCORE
- *Third Party*: Pellet, Racer, FaCT++, etc.
- *User Defined Rule Reasoners*

7.4.1 RDFS Built-in Reasoner

Jena provides a number of RDFS built-in reasoners to derive additional assertions based on RDF/OWL schema and instance RDF data. In the current Jena version 2.0, there are three RDFS ontology model specifications.

1. *RDFS_MEM*: memory based specification for RDFS ontology models.
2. *RDFS_MEM_RDFS_INF*: memory based specification for RDFS ontology models with additional entailments using RDFS inference.
3. *RDFS_MEM_TRANS_INF*: memory based specification for RDFS ontology models that use the transitive reasoner for entailments.

Once the specification is chosen, the next step is to choose between the build in or configurable reasoner. Jena provides both simplified (ReasonerRegistry.getRDFSSimpleReasoner()) and standard (ReasonerRegistry.getRDFSReasoner()) reasoners. The other alternative is to configure RDFS rule reasoner as shown in *Figure 40- RDFS Reasoner Configuration* below and pass it to a reasoner factory instance by setting the different levels of reification and process modes. RDFS rule reasoner can be set at different "simple", "default", and "full" rule processing levels. It also has different parameters such as rule direction mode that could be set as "forward", "backward", or "hybrid" (default). The different RDFS processing levels enable us to perform *post-processing* inference like transitive closure on subPropertyOf and subClassOf, application of different RDFS axioms and closure rules.

```
Resource config = ModelFactory.createDefaultModel
(ReificationStyle.Standard).createResource()

.addProperty(ReasonerVocabulary.PROPsetRDFSLevel, {RDFS
processing level})

.addProperty(ReasonerVocabulary.PROPruleMode, {rule direction
mode});
reasoner =
RDFSRuleReasonerFactory.theInstance().create(config);
```

Figure 40- RDFS Reasoner Configuration

Simple Processing Level

RDFS simple processing level implements transitive closure on subPropertyOf and subClassOf relations and the implications of subPropertyOf and subClassOf. It also implements transitive closure on domain and range entailments. Simple processing level can be set by the configuration as shown in the generic syntax above or by using parameters. Its simplest processing level is reflected by the number of inferences displayed in the table screenshot below.

```
reasoner.setParameter(ReasonerVocabulary.PROPsetRDFSLevel,
                      ReasonerVocabulary.RDFS_SIMPLE);
```

Table 25- RDFS Simple Processing Level

RDFS_MEM | RDFSRuleReasoner.SIMPLE_RULES | Tuples Inferred Size=532 | Submit

RDFS_MEM

532 items found, displaying 101 to 200.

[First/Prev] 1, 2, 3, 4, 5, 6 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/hasSEVERITYID	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/SEVERITY
urn:spaceProj-Mgmt/hasSEVERITYID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasSEVERITYID	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#DatatypeProperty
urn:spaceProj-Mgmt/hasSEVERITYID	http://www.w3.org/2000/01/rdf-schema#range	urn:spaceProj-Mgmt/SEVERITY
urn:spaceProj-Mgmt/hasSEVERITYID	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/CRITICALITY
urn:spaceProj-Mgmt/hasSEVERITYID	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#ObjectProperty
urn:spaceProj-Mgmt/PREFIXES	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/hasRANKNAME	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#string
urn:spaceProj-Mgmt/hasRANKNAME	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/RANK
urn:spaceProj-Mgmt/hasRANKNAME	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#DatatypeProperty
urn:spaceProj-Mgmt/SEVERITY	http://www.w3.org/2002/07/owl#FunctionalProperty	urn:spaceProj-Mgmt/hasSEVERITYID
urn:spaceProj-Mgmt/SEVERITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class

Default Processing Level

RDFS default processing level implements all axiomatic rules. It avoids expensive check for (rdf1, rdfs4a, and rdfs4b) entailment rules as described in "full processing Level" section.

```
reasoner.setParameter(ReasonerVocabulary.PROPsetRDFSLevel,
                      ReasonerVocabulary.RDFS_DEFAULT);
```

Table 26- RDFS Default Processing Level

RDFS_MEM | RDFSRuleReasoner.DEFAULT_RULES | Tuples Inferred Size=972 | Submit

RDFS_MEM

972 items found, displaying 1 to 100.

[First/Prev] 1, 2, 3, 4, 5, 6, 7, 8 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/ROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/ROLE	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2000/01/rdf-schema#Class
http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
urn:spaceProj-Mgmt/RESOLUTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/PRIORITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	
http://www.w3.org/2000/01/rdf-schema#comment	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2000/01/rdf-schema#Literal
urn:spaceProj-Mgmt/PREFIXES	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/PREFIXES	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource
urn:spaceProj-Mgmt/SEVERITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/SEVERITY	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource
urn:spaceProj-Mgmt/TRAIL	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class

Full Processing Level

RDFS full processing level implements all RDFS axioms and closure rules with the exception of bNode and datatypes (rdfD1). It also provides type assertions for the following RDFS entailment rules.

rdfs1 (uuu aaa lll.where lll is a plain literal (with or without a language tag).)->

(_:nnn rdf:type rdfs:Literal .where _:nnn identifies a blank node allocated to lll by rule rule lg.)

rdfs4a uuu aaa xxx -> uuu rdf:type rdfs:Resource .

rdfs4b uuu aaa vvv. -> vvv rdf:type rdfs:Resource .

```
reasoner.setParameter(ReasonerVocabulary.PROPsetRDFSLevel,
                      ReasonerVocabulary.RDFS_FULL);
```

Table 27- RDFS Full Processing Level

RDFS_MEM | RDFSRuleReasoner.FULL_RULES | Tuples Inferred Size=1112 | Submit

RDFS_MEM

1,112 items found, displaying 701 to 800.

[First/Prev] 4, 5, 6, 7, 8, 9, 10, 11 [Next/Last]

Subject	Predicate	Object
_:blankNode	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Restriction
urn:spaceProj-Mgmt/JENA_G1T1_STMT	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/hasEFFECTNAME	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#string
urn:spaceProj-Mgmt/hasEFFECTNAME	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/EFFECT
urn:spaceProj-Mgmt/hasEFFECTNAME	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#DatatypeProperty
urn:spaceProj-Mgmt/INCIDENT	http://www.w3.org/2002/07/owl#FunctionalProperty	urn:spaceProj-Mgmt/hasINCIDENTID
urn:spaceProj-Mgmt/INCIDENT	http://www.w3.org/2002/07/owl#FunctionalProperty	urn:spaceProj-Mgmt/hasACTIONID
urn:spaceProj-Mgmt/INCIDENT	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/hasACTIONDATE	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#dateTime
urn:spaceProj-Mgmt/hasACTIONDATE	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/ACTION
urn:spaceProj-Mgmt/hasACTIONDATE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#DatatypeProperty
urn:spaceProj-Mgmt/hasUSERSUPERVISOR	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#string

7.4.2 OWL Built-in Reasoner

OWL reasoner is an extension of RDFS reasoner. All the entailments covered by RDFS are also inherited by OWL. Jena provides a number of OWL built-in reasoners to derive additional assertions based on an Ontology schema and instance RDF data. OWL is sub divided into three sub-languages, OWL Full, OWL DL, and OWL Lite. Jena's implementation adds another two as "Mini" and a slightly smaller "Micro" version.

OWL Full:

In the current Jena version 2.0, there are a number of OWL ontology model specifications within the three sub languages- OWL Full, OWL DL, and OWL Lite.

1. *OWL_MEM*: memory based specification for OWL ontology models.
2. *OWL_MEM_RDFS_INF*: memory based specification for RDFS ontology models with additional entailments using RDFS inference.
3. OWL_MEM_TRANS_INF: memory based specification for OWL ontology models that use the transitive reasoner for additional entailments.
4. OWL_MEM_RULE_INF: memory based specification for OWL ontology models that use OWL rules inference engine for additional entailments.
5. OWL_MEM_MICRO_RULE_INF: memory based specification for OWL ontology models that use micro OWL rules inference engine for additional entailments.
6. OWL_MEM_MINI_RULE_INF: memory based specification for OWL ontology models that use mini OWL rules inference engine for additional entailments.
7. OWL_DL_MEM: memory based specification for OWL DL ontology models.
8. OWL_DL_MEM_RDFS_INF: memory based specification for OWL DL ontology models with additional entailments using RDFS inference.
9. OWL_DL_MEM_TRANS_INF: memory based specification for OWL DL ontology models that use the transitive reasoner for additional entailments.
10. OWL_DL_MEM_RULE_INF: memory based specification for OWL DL ontology models that use OWL rules inference engine for additional entailments.
11. OWL_LITE_MEM: memory based specification for OWL Lite ontology models.

OWL built-in reasoners provide a number of OWL ontology specification models as listed above. As part of the *post-processing* phase, different OWL reasoner specifications give different specialisation of semantic inferencing capability. These specialisations enable us to infer the mapped Schema and repository using specific reasoners for different purposes. The level spans from the core OWL constructs like `intersectionOf`, `unionOf` and `hasValue` to partial constructs that exclude forward entailment from `minCardinality` and `someValuesFrom`. OWL constructs include RDFS constructs as well as OWL specific axioms like `owl:disjointWith`, `owl:sameAs`, `owl:maxCardinality`,

`owl:cardinality` etc. that empowers the reasoner to represent more complex statements. DAML (DARPA Agent Markup Language) ontology language which is partially incorporated with OWL is also supported as a legacy reasoner. DAML reasoner implementation is also included in this section to compare the reasoning capability with OWL.

OWL Micro Reasoner

OWL Micro is very close in its implementation to RDFS while including the core OWL constructs. It includes core property axioms like `intersectionOf`, `unionOf` (partial) and `hasValue`. Its performance is much higher than the rest of the Jena OWL reasoner as it excludes equality axioms and cardinality restrictions.

```
OntModelSpec spec = new
OntModelSpec(OntModelSpec.OWL_MEM_MICRO_RULE_INF);

Resource config =
ModelFactory.createDefaultModel(ReificationStyle.Standard)
                    .createResource()

.addProperty(ReasonerVocabulary.EXT_REASONER_ONT_LANG,
optionString);
Reasoner reasoner =
OWLMicroReasonerFactory.theInstance().create( config );
reasoner = reasoner.bindSchema(owlSchema);
spec.setReasoner(reasoner);
OntModel ontModel = ModelFactory.createOntologyModel(spec,
rdfData);
```

Table 28- OWL Micro Reasoner Implementation

OWL_MEM_MICRO_RULE_ | Tuples Inferred Size=1779 | Submit

OWL_MEM_MICRO_RULE_INF

1,779 items found, displaying 601 to 700.

[First/Prev] 3, 4, 5, 6, 7, 8, 9, 10 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/hasCLASSNAME	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#string
urn:spaceProj-Mgmt/hasCLASSNAME	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/CLASSIFICATION
urn:spaceProj-Mgmt/hasCLASSNAME	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#DatatypeProperty
urn:spaceProj-Mgmt/hasCATEGORYNAME	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#string
urn:spaceProj-Mgmt/hasCATEGORYNAME	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/CATEGORY
urn:spaceProj-Mgmt/hasCATEGORYNAME	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#DatatypeProperty
urn:spaceProj-Mgmt/CRITICALITY	http://www.w3.org/2002/07/owl#FunctionalProperty	urn:spaceProj-Mgmt/hasCRITICALITYID
urn:spaceProj-Mgmt/CRITICALITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
_:blankNode	http://www.w3.org/2002/07/owl#maxCardinality	1 ^^http://www.w3.org/2001/XMLSchema#nonNegativeInteger
_:blankNode	http://www.w3.org/2002/07/owl#onProperty	urn:spaceProj-Mgmt/hasTRAILID
_:blankNode	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Restriction
urn:spaceProj-Mgmt/hasTRAILID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/TRAIL

OWL Mini Reasoner

OWL Mini is a cut down version of OWL Full. It excludes forward entailment from `minCardinality` and `someValuesFrom` OWL property restrictions. As a result it avoids bNodes that result in infinite closures.

```
OntModelSpec spec = new OntModelSpec(OntModelSpec.
OWL_MEM_MINI_RULE_INF);

Resource config =
ModelFactory.createDefaultModel(ReificationStyle.Standard)
                    .createResource()

.addProperty(ReasonerVocabulary.EXT_REASONER_ONT_LANG,
optionString);
Reasoner reasoner =
OWLMiniReasonerFactory.theInstance().create( config );
```

```
reasoner = reasoner.bindSchema(owlSchema);
spec.setReasoner(reasoner);
OntModel ontModel = ModelFactory.createOntologyModel(spec,
rdfData);
```

Table 29- OWL Mini Reasoner Implementation

OWL_MEM_MINI_RULE_IN | Tuples Inferred Size=2107 | Submit

OWL_MEM_MINI_RULE_INF

2,107 items found, displaying 701 to 800.

[First/Prev] 4, 5, 6, 7, 8, 9, 10, 11 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#ObjectProperty
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/INCIDENT
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/ACTION
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#DatatypeProperty
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/PRIORITY
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#domain	urn:spaceProj-Mgmt/REVIEW
_:blankNode	http://www.w3.org/2002/07/owl#maxCardinality	1 ^^http://www.w3.org/2001/XMLSchema#nonNegativeInteger
_:blankNode	http://www.w3.org/2002/07/owl#onProperty	urn:spaceProj-Mgmt/hasRESOLUTIONID
_:blankNode	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Restriction
_:blankNode	http://www.w3.org/2002/07/owl#maxCardinality	1 ^^http://www.w3.org/2001/XMLSchema#nonNegativeInteger
_:blankNode	http://www.w3.org/2002/07/owl#onProperty	urn:spaceProj-Mgmt/hasSTATUSID
_:blankNode	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Restriction
urn:spaceProj-	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#string

OWL Default Reasoner

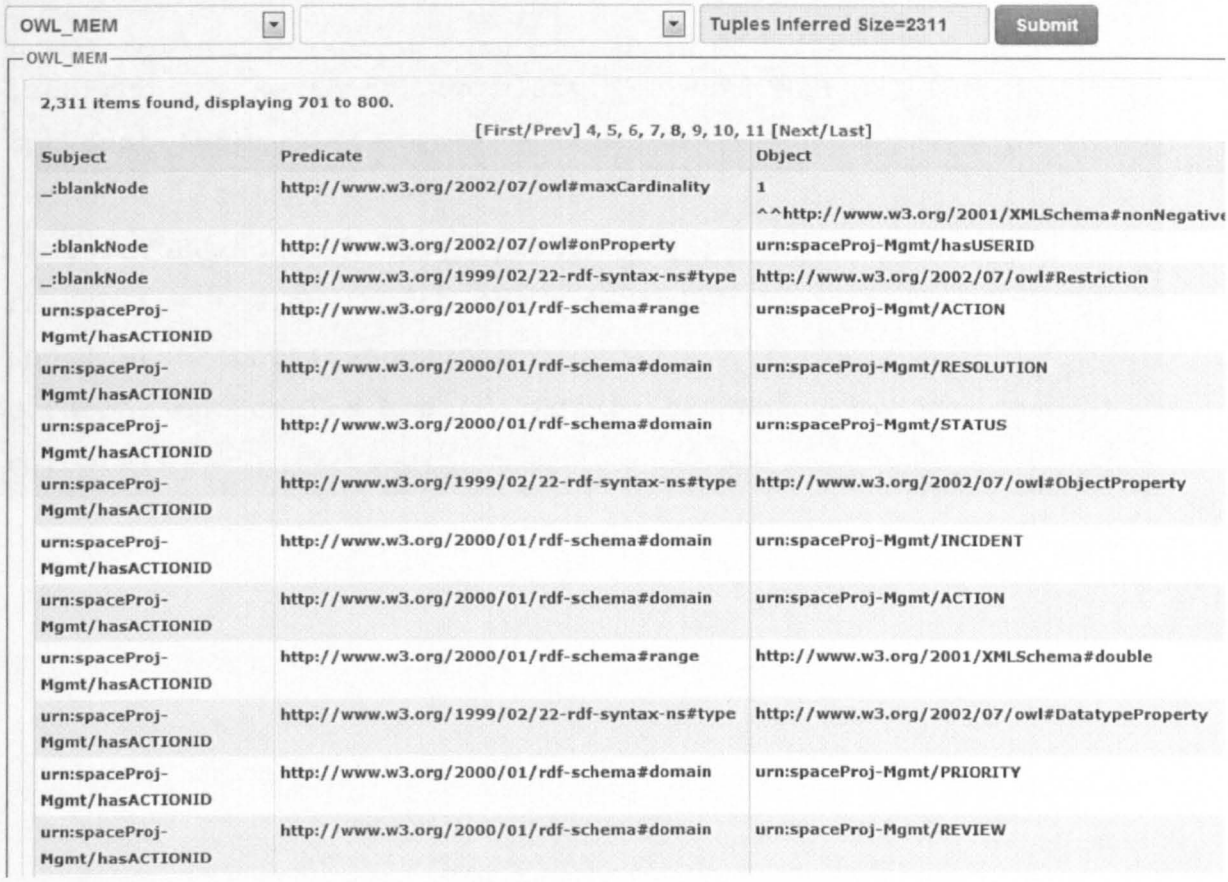
OWL Default includes most (not all) of the OWL constructs. In addition to RDFS constructs, owl:intersectionOf, owl:unionOf and owl:equivalentClass, the following properties are also supported.

{owl:disjointWith, owl:sameAs, owl:differentFrom, owl:distinctMembers, owl:someValuesFrom, owl:allValuesFrom, owl:minCardinality, owl:maxCardinality, owl:cardinality...}

```
OntModelSpec spec = new OntModelSpec(OntModelSpec.OWL_MEM);
```

```
        Reasoner reasoner =
ReasonerRegistry.getOWLReasoner();
reasoner = reasoner.bindSchema(owlSchema);
spec.setReasoner(reasoner);
OntModel ontModel = ModelFactory.createOntologyModel(spec,
rdfData);
```

Table 30- OWL Default Reasoner Implementation



DAML Reasoner

DAML is supported in Jena as a legacy reasoner. It has the following Ontology model specifications

1. DAML_MEM: memory based specification for DAML ontology models.
2. DAML_MEM_TRANS_INF: memory based specification for DAML ontology models that use the transitive reasoner for additional entailments.
3. DAML_MEM_RDFS_INF: memory based specification for DAML ontology models with additional entailments using RDFS inference.

4. DAML_MEM_RULE_INF: memory based specification for DAML ontology models that use DAML rules inference engine for additional entailments.

```
OntModelSpec spec = new OntModelSpec(OntModelSpec. DAML_MEM);

Resource config =
ModelFactory.createDefaultModel(ReificationStyle.Standard)
                    .createResource()

.addProperty(ReasonerVocabulary.EXT_REASONER_ONT_LANG,
optionString);
Reasoner reasoner =
DAMLMicroReasonerFactory.theInstance().create( config );
reasoner = reasoner.bindSchema(owlSchema);
spec.setReasoner(reasoner);
OntModel ontModel = ModelFactory.createOntologyModel(spec,
rdfData);
```

Table 31- DAML Reasoner Implementation

DAML_MEM | DAML as EXT_REASONER_ONT_LANG | Tuples Inferred Size=1352 | Submit

DAML_MEM

1,352 items found, displaying 701 to 800.

[First/Prev] 4, 5, 6, 7, 8, 9, 10, 11 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/RISK	http://www.w3.org/2002/07/owl#FunctionalProperty	urn:spaceProj-Mgmt/hasRISKID
urn:spaceProj-Mgmt/RISK	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/USERROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/USERPROFILE
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	
urn:spaceProj-Mgmt/ACTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/DOCUMENT
urn:spaceProj-Mgmt/ACTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/PROJECT
urn:spaceProj-Mgmt/PROJECT	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/DOCUMENT
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/PANEL
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/RESOLUTION
urn:spaceProj-Mgmt/ACTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/DOCUMENT
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/PROJECT
urn:spaceProj-Mgmt/ACTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/REVIEW
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/ROLE
urn:spaceProj-Mgmt/ACTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	
urn:spaceProj-Mgmt/USERROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/REVIEW
urn:spaceProj-Mgmt/USERROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/ACTION
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	
urn:spaceProj-Mgmt/USERROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/PROJECT
urn:spaceProj-Mgmt/USERROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/DOCUMENT
urn:spaceProj-Mgmt/USERROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/ROLE
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/CLASSIFICATION

7.4.3 Oracle's implementation using Jena Adapter

The current Oracle 11g Semantic implementation supports RDFS++, OWLSIF, OWL Prime and Simple Knowledge Organization System (SKOS) vocabularies. The main advantage of using Oracle is its utility to manage the models, entailments, namespaces, indexes and related objects. Using Oracle's Semantic utilities helps to use existing scalability and performance of Oracle RDBMS.

Here is a generic way of creating a model and construct a graph using SPAQRL query. The graph created takes advantage of built-in inference (`graph.performInference()`) and analysis (`graph.analyze()`, `graph.analyzeInferredGraph()`) java methods.

```
oracle = new Oracle(PropertyFileHandler.getJDBCDriver(),
PropertyFileHandler.getSemanticSchema(),
PropertyFileHandler.getSemanticPassword());
```

```
            Model model =
ModelOracleSem.createOracleSemModel(oracle, modelName);

            InputStream in =
FileManager.get().open("./SpaceOntologyExtReasoner.owl");
            model.read(in, "RDF/XML");
            queryString = " SELECT ?subject ?prop ?object
WHERE { ?subject ?prop ?object } ";
            query = QueryFactory.create(queryString);
            qexec = QueryExecutionFactory.create(query,
model);

            Attachment attachment =
Attachment.createInstance(
                    new String[]{},
              ["RDFS++", "OWLSIF", "OWLPRIME", "SKOSCORE"],
                    InferenceMaintenanceMode.NO_UPDATE,
                    QueryOptions.DEFAULT);

            GraphOracleSem graph = new
GraphOracleSem((Oracle) oracle, modelName, attachment);
            graph.analyze();
            graph.performInference();
            graph.analyzeInferredGraph();

            query = QueryFactory.create(queryString);
            qexec = QueryExecutionFactory.create(query, new
ModelOracleSem(graph));
```

RDFS++

RDFS++ includes all RDFS vocabulary constructs plus `owl:sameAs` and `owl:InverseFunctionalProperty`.

Table 32- Oracle's RDFS++ Inference Result

RDFS++ Submit

RDFS++

1,223 items found, displaying 801 to 900.

[First/Prev] 5, 6, 7, 8, 9, 10, 11, 12 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/hasCLASSNAME	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	urn:spaceProj-Mgmt/hasCLASSNAME
urn:spaceProj-Mgmt/COLLECTION	http://www.w3.org/2002/07/owl#FunctionalProperty	urn:spaceProj-Mgmt/hasCOLLECTIONID
-419eabcc:13e2cd93b63:-7bba	http://www.w3.org/2002/07/owl#onProperty	urn:spaceProj-Mgmt/hasCOLLECTIONID
urn:spaceProj-Mgmt/hasCOLLECTIONID	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	urn:spaceProj-Mgmt/hasCOLLECTIONID
urn:spaceProj-Mgmt/CRITICALITY	http://www.w3.org/2002/07/owl#FunctionalProperty	urn:spaceProj-Mgmt/hasCRITICALITYID
-419eabcc:13e2cd93b63:-7bb9	http://www.w3.org/2002/07/owl#onProperty	urn:spaceProj-Mgmt/hasCRITICALITYID
urn:spaceProj-Mgmt/hasCRITICALITYID	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	urn:spaceProj-Mgmt/hasCRITICALITYID
urn:spaceProj-Mgmt/LIKELIHOOD	http://www.w3.org/2002/07/owl#FunctionalProperty	urn:spaceProj-Mgmt/hasLIKELIHOODID
-419eabcc:13e2cd93b63:-7bb0	http://www.w3.org/2002/07/owl#onProperty	urn:spaceProj-Mgmt/hasLIKELIHOODID
urn:spaceProj-Mgmt/hasLIKELIHOODID	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	urn:spaceProj-Mgmt/hasLIKELIHOODID
urn:spaceProj-Mgmt/RISK	http://www.w3.org/2002/07/owl#FunctionalProperty	urn:spaceProj-Mgmt/hasRISKID
-419eabcc:13e2cd93b63:-7ba5	http://www.w3.org/2002/07/owl#onProperty	urn:spaceProj-Mgmt/hasRISKID
urn:spaceProj-Mgmt/hasRISKID	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	urn:spaceProj-Mgmt/hasRISKID

OWLSIF

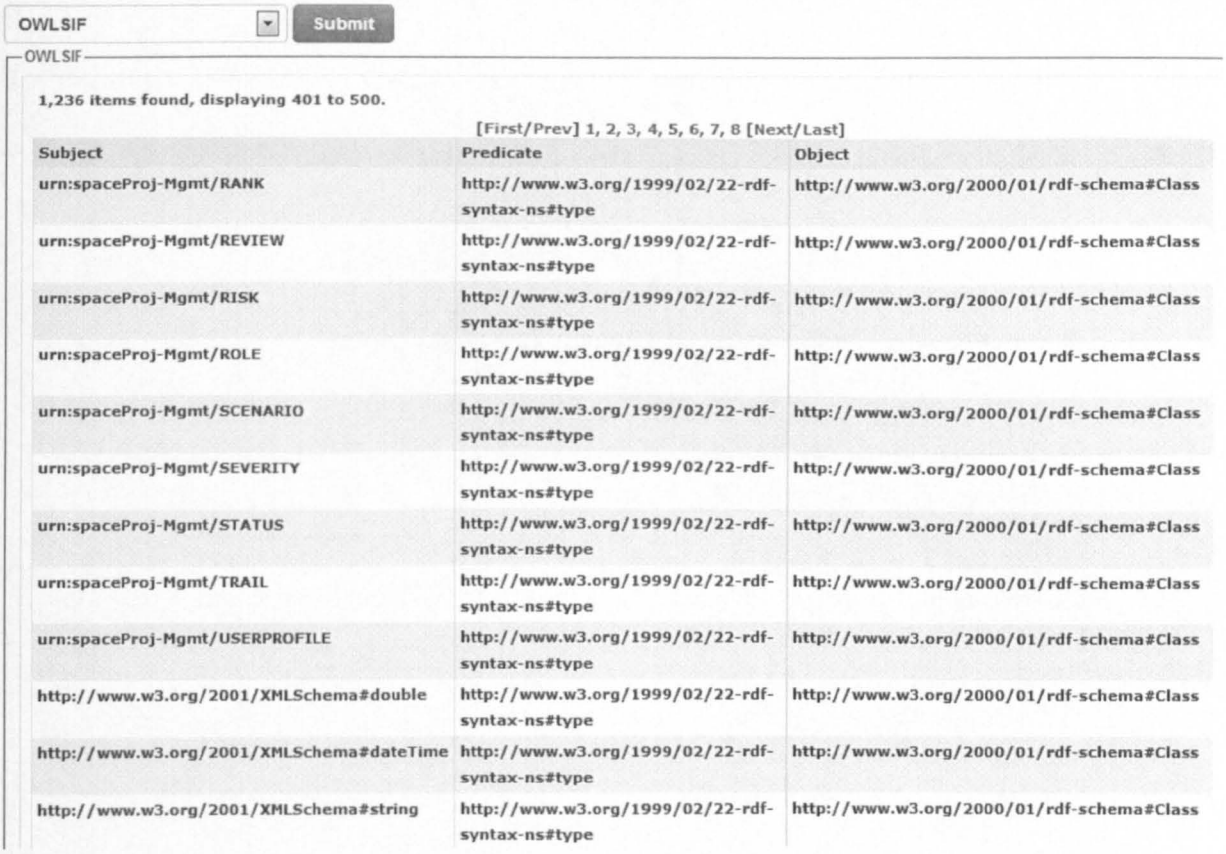
OWLSIF is OWL with IF Semantic, with the vocabulary and semantics proposed for pD* semantics in "Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary" (Horst, 2005). It includes the following OWL constructs in addition to all RDFS constructs.

```
owl:FunctionalProperty
owl:InverseFunctionalProperty
owl:SymmetricProperty
owl:TransitiveProperty
owl:sameAs
owl:inverseOf
owl:equivalentClass
owl:equivalentProperty
owl:hasValue
```

owl:someValuesFrom

owl:allValuesFrom

Table 33- Oracle's OWLSIF Inference Result



OWLSIF Submit

OWLSIF

1,236 items found, displaying 401 to 500.

[First/Prev] 1, 2, 3, 4, 5, 6, 7, 8 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/RANK	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/RISK	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/ROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/SCENARIO	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/SEVERITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/STATUS	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/TRAIL	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/USERPROFILE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
http://www.w3.org/2001/XMLSchema#double	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
http://www.w3.org/2001/XMLSchema#dateTime	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
http://www.w3.org/2001/XMLSchema#string	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class

OWLPRIME

OWL Prime includes basic RDF properties (`rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`) and the following OWL constructs.

owl:FunctionalProperty

owl:InverseFunctionalProperty

owl:SymmetricProperty

owl:TransitiveProperty

owl:sameAs

owl:inverseOf

owl:equivalentClass

owl:equivalentProperty

owl:hasValue

owl:someValuesFrom

owl:allValuesFrom

owl:differentFrom

owl:disjointWith

owl:complementOf

Table 34- Oracle's OWL Prime Inference Result

OWLPRIME Submit

OWLPRIME

1,002 items found, displaying 1 to 100.

[First/Prev] 1, 2, 3, 4, 5, 6, 7, 8 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/SpaceKnowledgeManagement	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Ontology
urn:spaceProj-Mgmt/NONCONF	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/PANEL	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/PRIORITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/RANK	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/RISK	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/ROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/SCENARIO	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/SEVERITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/STATUS	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/TRAIL	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class

Core Simple Knowledge Organization System (SKOSCORE)

SKOS Core is based on a subset of SKOS that is applicable to taxonomies, thesauri and similar control vocabularies. Around 40 SKOS-specific terms are included in the Oracle Database semantic technologies support, such as `skos:broader`, `skos:relatedMatch`, and `skos:Concept`. Over 100 SKOS axiomatic triples have been added, providing the basic coverage of SKOS

semantics. However, support is not included for the integrity conditions described in the SKOS specification (Murray, 2010).

Table 35- Oracle's SKOSCORE Inference Result

SKOSCORE Submit

SKOSCORE

1,153 items found, displaying 301 to 400.

[First/Prev] 1, 2, 3, 4, 5, 6, 7, 8 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/hasSEVERITYID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasDISPOSITIONID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasPROJECTID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasDOMAINID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasEDITORIALID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasEFFECTID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasINCIDENTID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasPANELID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasPRIORITYID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasRANKID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasRESOLUTIONID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasROLEID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/hasSCENARIOID	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2001/XMLSchema#double

7.4.4 Generic user-defined Rule Reasoner

In the section below an illustration for the implementation of user-defined rule reasoner is presented using four (Review, Action, Document, Project) sample classes and their properties. The rule is an implementation of the `CheckTransitiveChain()` algorithm.

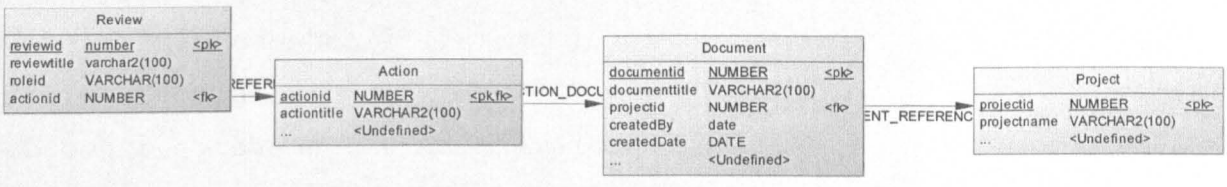


Figure 41- Sample Relational Model

The program excerpt below shows a way to apply custom rules to mapped data. The sample rule "transiveChainSubClassOf" is defined as:

For Classes x, y, z, if x is subClass of y and y is subClass of z then x is subClass of z".

The rule produces chain of classes between:

Review	and	Document
Review	and	Project
Action and	and	Project

```
{
...
...
/*Read Schema and Data*/
Model schema = FileManager.get().loadModel
("file:source/mappedSchema.rdf");
Model data = FileManager.get().loadModel
("file:source/mappedData.rdf");
/* Set rule*/
String ruleString = "[transTriple: (?A
rdfs:subClassOf ?B) (?B rdfs:subClassOf ?C) ->
(?A rdfs:subClassOf ?C)]"
                    + "[transQuad: (?A
rdfs:subClassOf ?B), (?B rdfs:subClassOf ?C),
(?C rdfs:subClassOf ?D) -> (?A rdfs:subClassOf
?D) ]";
List rules = Rule.parseRules(ruleString);

Reasoner reasoner = new
GenericRuleReasoner(rules);
/*Bind Schema*/
reasoner = reasoner.bindSchema(schema);
InfModel inf =
```

```
ModelFactory.createInfModel(reasoner, data);
......
}
```

The first three rows in *Table 36- Generic User-defined Rule Implementation* below show the new subClass triple produced as a result of the user-defined reasoner.

Table 36- Generic User-defined Rule Implementation

Custom Inferred Triples

106 items found, displaying 1 to 100.

[First/Prev] 1, 2 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/ACTION	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/PROJECT
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/PROJECT
	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/DOCUMENT
urn:spaceProj-Mgmt/USERROLE	urn:spaceProj-Mgmt/hasROLEID	2^^http://www.w3.org/2001/XMLSchema#double
	urn:spaceProj-Mgmt/hasROLENAME	Controller^^http://www.w3.org/2001/XMLSchema#string
	urn:spaceProj-Mgmt/hasROLEID	1^^http://www.w3.org/2001/XMLSchema#double
	urn:spaceProj-Mgmt/hasROLENAME	Engineer^^http://www.w3.org/2001/XMLSchema#string
	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/USERROLE
urn:spaceProj-Mgmt/SpaceKnowledgeManagement	http://www.w3.org/2002/07/owl#versionInfo	SpaceOntology, Version 1.0 , 2013
	http://www.w3.org/2000/01/rdf-schema#label	School of Computing
	http://www.w3.org/2000/01/rdf-schema#label	Knowledge Management Research Centre
	http://www.w3.org/2000/01/rdf-schema#label	London Metropolitan University
	http://www.w3.org/2000/01/rdf-schema#comment	Space Management Ontology
	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Ontology
urn:spaceProj-Mgmt/ACTION	urn:spaceProj-Mgmt/hasACTIONSTATUS	ARCHIEVED^^http://www.w3.org/2001/XMLSchema#string

7.4.5 Third Party Reasoner

Third party reasoners like Pellet, Racer, and FaCT provide a more descriptive and sometimes specific reasoner engine. This is another way of adding more inference to your existing facts. Jena provides an interface to reasoners like Pellet up to Jena 2.6.0 and more and more reasoners provide their own interface nowadays. As an example here is Pellet's reasoner implementation using Jena's interface.

```
OntModel model =
ModelFactory.createOntologyModel(PelletReasonerFactory.THE_SP
EC);
            InputStream in =
FileManager.get().open("./SpaceOntologyExtReasoner.owl");
            model.read(in, null, "RDF/XML");

            OutputStream os = new
FileOutputStream("./PelletReasoned.rdf");
            model.write(os, "RDF/XML");
            System.out.println("All classes:");
```

```
            for (Iterator<OntClass> i = model.listClasses();
i.hasNext();) {
                System.out.println("class: " +
i.next().getURI());
            }
```

Table 37- Pellet Reasoner Inferred Triples

Pellet Inferred Triples

818 items found, displaying 1 to 100.

[First/Prev] 1, 2, 3, 4, 5, 6, 7, 8 [Next/Last]

Subject	Predicate	Object
http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/ROLE	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource
	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/RESOLUTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/PRIORITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
http://www.w3.org/2000/01/rdf-schema#comment	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2000/01/rdf-schema#Literal
urn:spaceProj-Mgmt/PREFIXES	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource
	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/SEVERITY	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource
	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/TRAIL	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource

Chapter 8: Evaluation

The evaluation of mapping relational databases to ontological repositories is based on the statistical analysis of the findings and the feature comparison to other similar approaches. The implementation of the algorithms using an interactive java application shows different jena built-in, proprietary, user-defined, and third party reasoner inference outputs. A typical example would be the comparison of the *post-processing* inference output on the mapped ontological repository using pure Jena and Oracle proprietary reasoners. The second approach of evaluation is to compare the different features covered by the mapping algorithm against existing mapping implementations.

8.1 Statistical Analysis

Statistical analysis on the mapped ontological repository shows the benefit of analysing the repository by using *post-processing* phase reasoners. The comparison starts by analysing the "direct mapping" result of the relational schema and data without the use of reasoners. The mapped OWL Schema and RDF data are a result of the incremental iterative algorithm implementation. After the comparison between the source relational database and the mapped ontological repository, the next sub-sections base their comparison on the application of different group of reasoners .

The comparison of the source relational database and mapped ontological repository validates the mapping algorithm by showing the corresponding mapped representation. As shown in *Table 12- Relational Tables and Primary key(s) Sample List* and *Table 15- Relational Columns Sample List* above the number of relational tables and relational columns is 29 and 91 respectively. The number of constraints and relationships in the relational database is 56 and 29 respectively as shown in *Table 17- Relational Constraints Sample List* and *Table 19- Relational Relationships Sample List* above. The mapped ontological repository shows the same number of OWL classes as the number of relational tables which is 29.The mapping implementation is focused on the efficient representation of columns, constraints, and relationships while keeping the correlation between the relational tables and OWL classes one-to-one. For instance relational columns are mapped

as "owl:DatatypeProperty" but the comparison shows fewer columns being mapped this way. Some of the relational columns are identified as a relationship column that makes them categorised as relationships. Relationships in relational database can be mapped in more than one ways into the Ontology Schema. A relationship can be mapped as a "rdfs:subClassOf", "owl:ObjectProperty", "owl:FunctionalProperty", etc. *Table 38- Relational to Ontological mapping Comparison below* shows the comparison between the source relational database and the mapped ontological Schema using the "Entity- Relationship- Attribute-Constraint" model classification.

Table 38- Relational to Ontological mapping Comparison

	Relational Model	Ontological Model
Entities	29 tables	29 owl:Class
		14 rdfs:subClassOf
		1 owl:equivalentClass
Attributes	91 columns	56 owl:DatatypeProperty
Relationships	29 referential integrity	26 owl:ObjectProperty
		8 owl:disjointWith
		1 owl:SymmetricProperty
Constraints	28 primary keys	29 owl:FunctionalProperty
	1 check	31 owl:Restriction
		3 owl:hasValue

One of the main advantages of mapping relational databases to ontological repositories is the possibility to apply "inference" on the raw asserted data. Asserted data is the result of the first stage of mapping before any reification is applied. The mapped data is grouped into OWL schema data and RDF data, where the schema describes the RDF data. The first stage mapping produced around 480 OWL schema triples and 28 RDF data triples as shown in *Table 39- Mapped OWL Schema Triples* and *Table 40- Mapped RDF data Triples* respectively.

Table 39- Mapped OWL Schema Triples

Mapped OWL Schema Triples

480 items found, displaying 1 to 100.

[First/Prev] 1, 2, 3, 4, 5 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/SpaceKnowledgeManagement	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Ontology
urn:spaceProj-Mgmt/NONCONF	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/PANEL	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/PRIORITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/RANK	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/REVIEW	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/RISK	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/ROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/SCENARIO	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/SEVERITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/STATUS	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/TRAIL	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/USERPROFILE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/ACTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/DOCUMENT	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/ATTACHMENT	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/CATEGORY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/CLASSIFICATION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/COLLECTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/CRITICALITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/DISPOSITION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/PROJECT	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
urn:spaceProj-Mgmt/DOMAIN	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class

Table 40- Mapped RDF data Triples

Mapped RDF Data Triples

28 items found, displaying all items.

1

Subject	Predicate	Object
urn:spaceProj-Mgmt/PROJECT	urn:spaceProj-Mgmt/hasPROJECTID	1.002E3^^http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/USERROLE	urn:spaceProj-Mgmt/hasROLEID	1E0^^http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/ACTION	urn:spaceProj-Mgmt/hasDOCUMENTID	2.002E3^^http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/ACTION	urn:spaceProj-Mgmt/hasACTIONDATE	2013-10-06T00:00:00Z^^http://www.w3.org/2001/XMLSchema#dateTime
urn:spaceProj-Mgmt/REVIEW	urn:spaceProj-Mgmt/hasROLEID	2E0^^http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/USERROLE	urn:spaceProj-Mgmt/hasROLEID	
urn:spaceProj-Mgmt/REVIEW	urn:spaceProj-Mgmt/hasACTIONID	3.002E3^^http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/ACTION	urn:spaceProj-Mgmt/hasACTIONID	
urn:spaceProj-Mgmt/REVIEW	urn:spaceProj-Mgmt/hasREVIEWID	4.002E3^^http://www.w3.org/2001/XMLSchema#double
urn:spaceProj-Mgmt/ACTION	urn:spaceProj-Mgmt/hasACTIONSTATUS	ARCHIEVED
urn:spaceProj-Mgmt/USERROLE	urn:spaceProj-Mgmt/hasROLENAME	Controller
urn:spaceProj-Mgmt/USERROLE	urn:spaceProj-Mgmt/hasROLENAME	Engineer
urn:spaceProj-Mgmt/PROJECT	urn:spaceProj-Mgmt/hasPROJECTNAME	Galilio Observatory
urn:spaceProj-Mgmt/SpaceKnowledgeManagement	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Ontology
urn:spaceProj-Mgmt/REVIEW	urn:spaceProj-Mgmt/hasPrivilege	INSERT
urn:spaceProj-Mgmt/SpaceKnowledgeManagement	http://www.w3.org/2000/01/rdf-schema#label	Knowledge Management Research Centre
urn:spaceProj-Mgmt/SpaceKnowledgeManagement	http://www.w3.org/2000/01/rdf-schema#label	London Metropolitan University
urn:spaceProj-Mgmt/ACTION	urn:spaceProj-Mgmt/hasACTIONTITLE	Mission Satellite Incident
urn:spaceProj-Mgmt/REVIEW	urn:spaceProj-Mgmt/grantee	RELATIONALONT
urn:spaceProj-Mgmt/REVIEW	urn:spaceProj-Mgmt/hasREVIEWTITLE	Satellite Re-launch
urn:spaceProj-Mgmt/SpaceKnowledgeManagement	http://www.w3.org/2000/01/rdf-schema#label	School of Computing
urn:spaceProj-Mgmt/REVIEW	urn:spaceProj-Mgmt/hasPrivilege	SELECT

The rest of the section uses the original asserted triples as a base to apply different groups of reasoners (Jena built-in, Oracle proprietary, third-party) that produce

various inference output. The evaluation uses three different types of reasoners for analysis of the mapped schema and data.

1. Using Jena framework on RDFS, OWL, and Generic User-defined reasoners.
2. Using Oracle's Semantic Web implementation of RDFS++, OWLSIF, OWLPRIME, and SKOSCORE reasoners.
3. Using third party reasoners like Pellet and DAML.

8.1.1 Evaluation of pure Jena Framework Implementation

RDFS based Analysis on Mapped OWL Schema and RDF Data

There are three processing levels for RDFS. It starts with a simple RDFS rule level where it performs transitive closure on subPropertyOf, subClassOf, domain, and range entailments. The next default processing level implements all axiomatic rules except the expensive checks like rdf1, rdfs4a, and rdfs4b. The full processing level covers all RDFS axioms with the exception of bNode and datatypes. The comparison between different RDFS rule levels shows different number of results. This shows the advantage of mapping relational database to OWL schema and RDF data which makes it open to different levels of inference analysis. *Table 41-Comparison between RDFS Default and Simple Rule Levels* below shows additional 442 triples produced using RDFS default rule level processing. The capability of RDFS default rule level reasoner to perform not only transitive closure on subPropertyOf, subClassOf, domain, and range entailments but also axiomatic rules except the expensive checks like rdf1, rdfs4a, and rdfs4b has made the realisation of those additional information about the mapped schema.

Table 41- Comparison between RDFS Default and Simple Rule Levels

RDFS_MEM | RDFS_MEM | Compare

RDFSRuleReasoner.DEFAULT_RULES | RDFSRuleReasoner.SIMPLE_RULES | (Additional Parameters)

Tuples Inferred Size=980 | Tuples Inferred Size=538 | Tuples Diff size=442

Diff All List

442 items found, displaying 1 to 100.

[First/Prev] 1, 2, 3, 4, 5 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/ROLE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/ROLE	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2000/01/rdf-schema#Class
http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
urn:spaceProj-Mgmt/RESOLUTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/PRIORITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	
http://www.w3.org/2000/01/rdf-schema#comment	http://www.w3.org/2000/01/rdf-schema#range	http://www.w3.org/2000/01/rdf-schema#Literal
urn:spaceProj-Mgmt/PREFIXES	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/PREFIXES	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource
urn:spaceProj-Mgmt/SEVERITY	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/SEVERITY	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource
urn:spaceProj-Mgmt/TRAIL	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
urn:spaceProj-Mgmt/TRAIL	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/1999/02/22-rdf-syntax-ns#first	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	http://www.w3.org/1999/02/22-rdf-syntax-ns#first
http://www.w3.org/1999/02/22-rdf-syntax-ns#first	http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/1999/02/22-rdf-syntax-ns#List
http://www.w3.org/1999/02/22-rdf-syntax-ns#first	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
urn:spaceProj-Mgmt/ACTION	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Class
http://www.w3.org/1999/02/22-rdf-syntax-ns#object	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	http://www.w3.org/1999/02/22-rdf-syntax-ns#object
http://www.w3.org/1999/02/22-rdf-syntax-ns#object	http://www.w3.org/2000/01/rdf-schema#domain	http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement

Table 42- Comparison between RDFS Full and Default Rule Levels below shows another additional 144 triples produced using RDFS full rule level processing. The capability of RDFS full rule level reasoner to perform all RDFS axioms with the exception of bNode and datatypes has made the realisation of the above additional information about the mapped schema.

Table 42- Comparison between RDFS Full and Default Rule Levels

RDFS_MEM	RDFS_MEM	Compare
RDFSRuleReasoner.FULL_RULES	RDFSRuleReasoner.DEFAULT_RULES	(Additional Parameters)
Tuples Inferred Size=1124	Tuples Inferred Size=980	Tuples Diff size=144

Diff All List

143 items found, displaying 1 to 100.

[First/Prev] 1, 2 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/hasROLEID	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	urn:spaceProj-Mgmt/hasROLEID
http://www.w3.org/2001/XMLSchema#nonPositiveInteger	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/2001/XMLSchema#nonPositiveInteger	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Datatype
http://www.w3.org/2001/XMLSchema#gMonthDay	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/2001/XMLSchema#gMonthDay	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Datatype
http://www.w3.org/2000/01/rdf-schema#member	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/2000/01/rdf-schema#member	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	http://www.w3.org/2000/01/rdf-schema#member
http://www.w3.org/2000/01/rdf-schema#member	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
urn:spaceProj-Mgmt/hasREVIEWTITLE	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	urn:spaceProj-Mgmt/hasREVIEWTITLE
http://www.w3.org/2001/XMLSchema#gYearMonth	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/2001/XMLSchema#gYearMonth	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Datatype
http://www.w3.org/2001/XMLSchema#unsignedShort	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/2001/XMLSchema#unsignedShort	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Datatype
urn:spaceProj-Mgmt/hasROLENAME	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	urn:spaceProj-Mgmt/hasROLENAME
urn:spaceProj-Mgmt/hasACTIONTITLE	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	urn:spaceProj-Mgmt/hasACTIONTITLE
urn:spaceProj-Mgmt/grantee	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Resource
urn:spaceProj-Mgmt/grantee	http://www.w3.org/2000/01/rdf-schema#subPropertyOf	urn:spaceProj-Mgmt/grantee
urn:spaceProj-Mgmt/grantee	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
http://www.w3.org/2001/XMLSchema#dateTime	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2000/01/rdf-schema#Datatype

The comparison between RDF simple, default, and full rule level processing shows the advantage of mapping the relational database to ontological repositories. The result of the mapping has become open for semantic inferencing and the application of RDFS rule level reasoners has shown more and more information about our mapped schema and data. Semantic inference analysis on the original 508 triples has made us realise additional information and produced 538 (simple), 980 (default), and 1124 (full) triples as shown in *Figure 42- RDFS Rule Level Comparison* below.

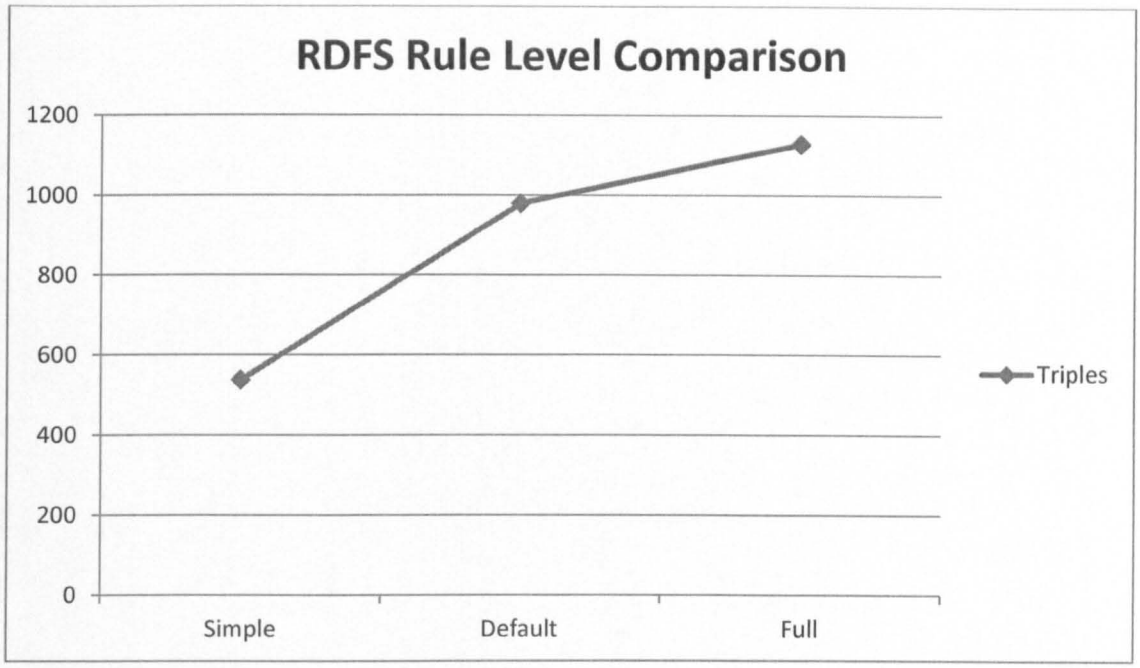


Figure 42- RDFS Rule Level Comparison

OWL based Analysis on Mapped OWL Schema and RDF Data

OWL is a more descriptive ontology language than RDFS. It has different versions that range from the simple OWL Lite to the more descriptive OWL DL up to the OWL Full. Jena provides a corresponding ontology model specification as "OWL_LITE", "OWL_DL" and "OWL" respectively. On top of the default OWL ontology models OWL_DL and OWL_Full have an option to apply rule inference engine by using OWL_DL_MEM_RULE_INF and OWL_MEM_RULE_INF specifications respectively. It also has a cut-down OWL Micro specification that includes core property axioms and significantly faster than the rest of the OWL specifications. There is also OWL Mini version between OWL Micro and OWL specifications.

Table 43- Comparison between OWL Mini and OWL Micro Reasoner below shows additional 329 triples produced using OWL Mini ontology model specification. The use of optimised rule-based reasoner by OWL Micro contributed to the fewer number of triples being produced. OWL Mini uses non-optimised rule based reasoner with subset of OWL rules.

Table 43- Comparison between OWL Mini and OWL Micro Reasoner

OWL_MEM_MINI_RULE_INF	OWL_MEM_MICRO_RULE_INF	Compare
		(Additional Parameters)
Tuples Inferred Size=2133	Tuples Inferred Size=1804	Tuples Diff size=329

Diff All List

375 items found, displaying 1 to 100.

[First/Prev] 1, 2, 3, 4 [Next/Last]

Subject	Predicate	Object
http://www.w3.org/2002/07/owl#Restriction	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/1999/02/22-rdf-syntax-ns#List	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2002/07/owl#Ontology	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2001/XMLSchema#integer	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2002/07/owl#Property	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2000/01/rdf-schema#Literal	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2001/XMLSchema#unsignedLong	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2001/XMLSchema#unsignedLong	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2001/XMLSchema#unsignedLong	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2001/XMLSchema#unsignedLong	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2001/XMLSchema#unsignedInt	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2001/XMLSchema#unsignedInt	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww
http://www.w3.org/2001/XMLSchema#unsignedInt	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://ww

Table 44- Comparison between OWL Full and OWL Mini Reasoner below shows another additional 195 triples produced using OWL Full ontology model specification. As the ontology model specification goes from mini to full, the mapped schema and data become open to a greater number of reasoning rules. These rules discover more information about the mapped schema that shows the advantage of mapping relational databases to ontological repositories.

Table 44- Comparison between OWL Full and OWL Mini Reasoner

OWL_MEM	OWL_MEM_MINI_RULE_INF	Compare
		(Additional Parameters)
Tuples Inferred Size=2328	Tuples Inferred Size=2133	Tuples Diff size=195

Diff All List

170 items found, displaying 1 to 100.

[First/Prev] 1, 2 [Next/Last]

Subject	Predicate	Object
urn:spaceProj-Mgmt/RESOLUTION	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/RE
urn:spaceProj-Mgmt/RESOLUTION	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/IN
urn:spaceProj-Mgmt/RESOLUTION	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/ST
urn:spaceProj-Mgmt/RESOLUTION	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/PR
urn:spaceProj-Mgmt/RESOLUTION	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/RO
_:blankNode	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/200
_:blankNode	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	urn:spaceProj-Mgmt/AC
_:blankNode	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/200
_:blankNode	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/PR
_:blankNode	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/ST
_:blankNode	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/IN
_:blankNode	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/RE
_:blankNode	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/RE
_:blankNode	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/RO
urn:spaceProj-Mgmt/INCIDENT	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/RE
urn:spaceProj-Mgmt/INCIDENT	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/PR
urn:spaceProj-Mgmt/INCIDENT	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/RE
urn:spaceProj-Mgmt/INCIDENT	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/ST
urn:spaceProj-Mgmt/INCIDENT	http://www.w3.org/2000/01/rdf-schema#subClassOf	urn:spaceProj-Mgmt/RO

The comparison between OWL Micro, Mini, and OWL Full ontology model specifications shows the incremental realisation of information about the mapped schema and data. The openness and machine-readability of Semantic Web architecture creates the environment to discovery more information about the mapped schema and data. The number of triples has increased from the original 508 triples to 1804 (OWL Micro), 2133 (OWL Mini), 2328 (OWL Full) as shown in *Figure 43- OWL Reasoner Inference Comparison* below.

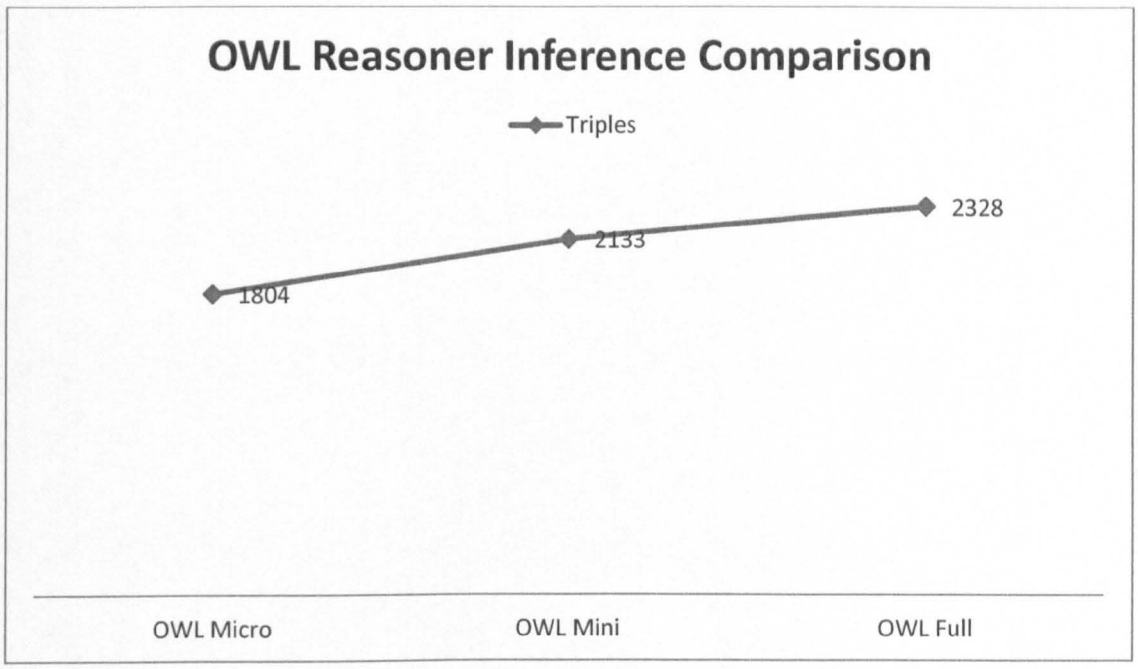


Figure 43- OWL Reasoner Inference Comparison

8.1.2 Evaluation of Oracle's Semantic Web Implementation

After the introduction of Semantic Technologies in Oracle since 10g, there is a significant increase for more ontology support further into Oracle 11g. Oracle 11g supports RDFS++, OWLSIF, OWLPRIME and SKOSCORE constructs. *Figure 44- Oracle's Built-in Inference Comparison* below shows the different number of triples that can be inferred using Oracle's Built-in reasoners.

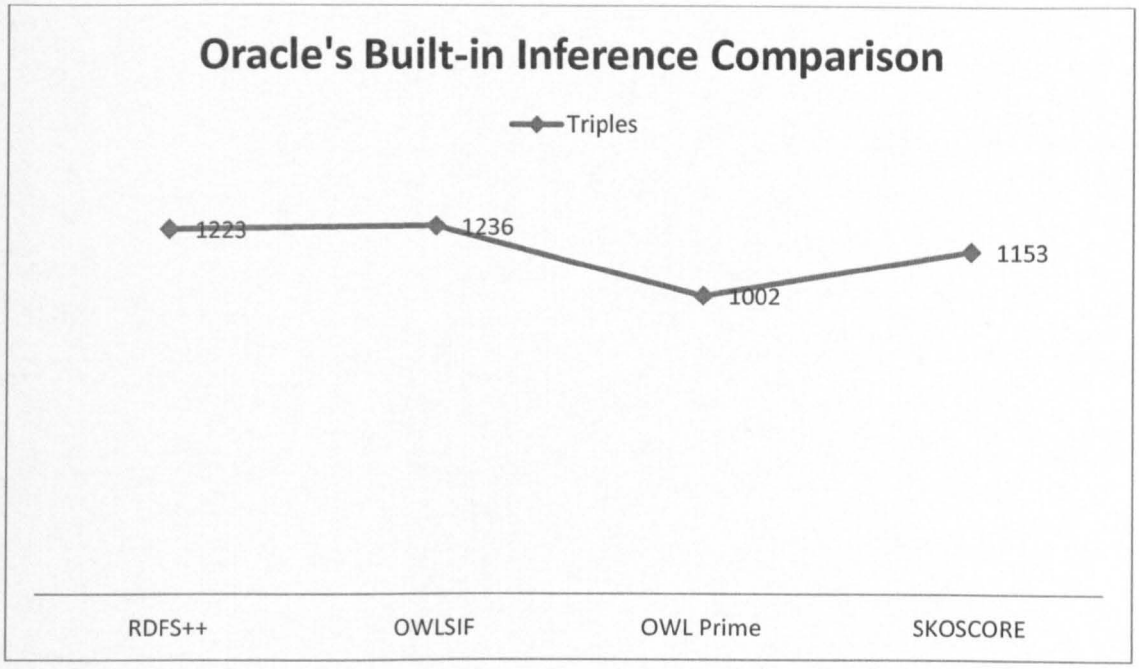


Figure 44- Oracle's Built-in Inference Comparison

The increase from 1223 triples to 1236 triples is due to the additional OWL property support by OWLSIF. In the case of the mapped OWL schema, the use of OWL properties like `owl:equivalentClass` and `owl:SymmetryProperty` contributed to the slight increase. There is a slight fall in the number of triples when using OWL Prime constructs which is attributed to OWL Prime's support to only basic RDFS properties. Core SKOS support to semantic ontology vocabularies also managed to infer a significant number of triples.

8.1.3 Evaluation of Third Party Reasoners Implementation

Using Protégé OWL API (Research, 2013) and Protégé editor a number of third party reasoners are used to validate the mapped OWL Schema. FaCT++, HermiT, and Pellet reasoners are used to check the validity of the OWL schema. *Figure 45- Protégé Graph view of Mapped OWL Schema* shows the ontology graph when loaded into Protégé editor. Using the "Reasoner" menu, a reasoner can be selected and executed to produce the validation report result as shown in *Figure 46- Third Party Reasoners Log for Mapped OWL Schema.*

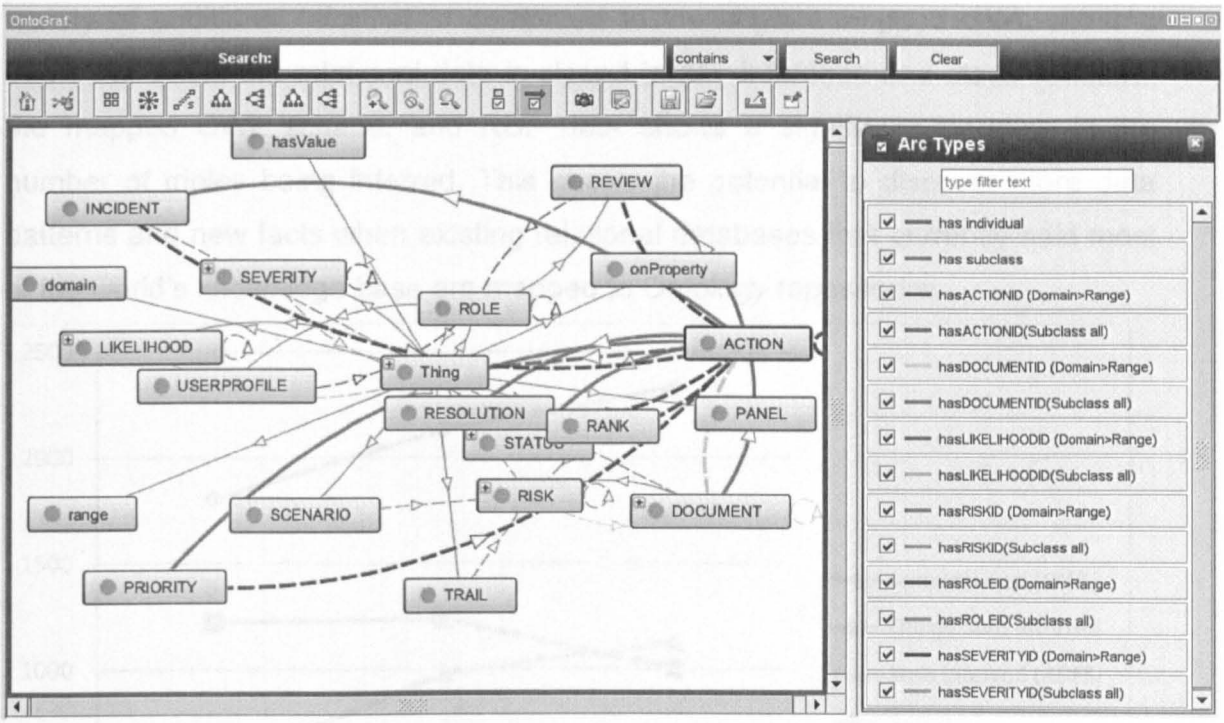


Figure 45- Protégé Graph view of Mapped OWL Schema

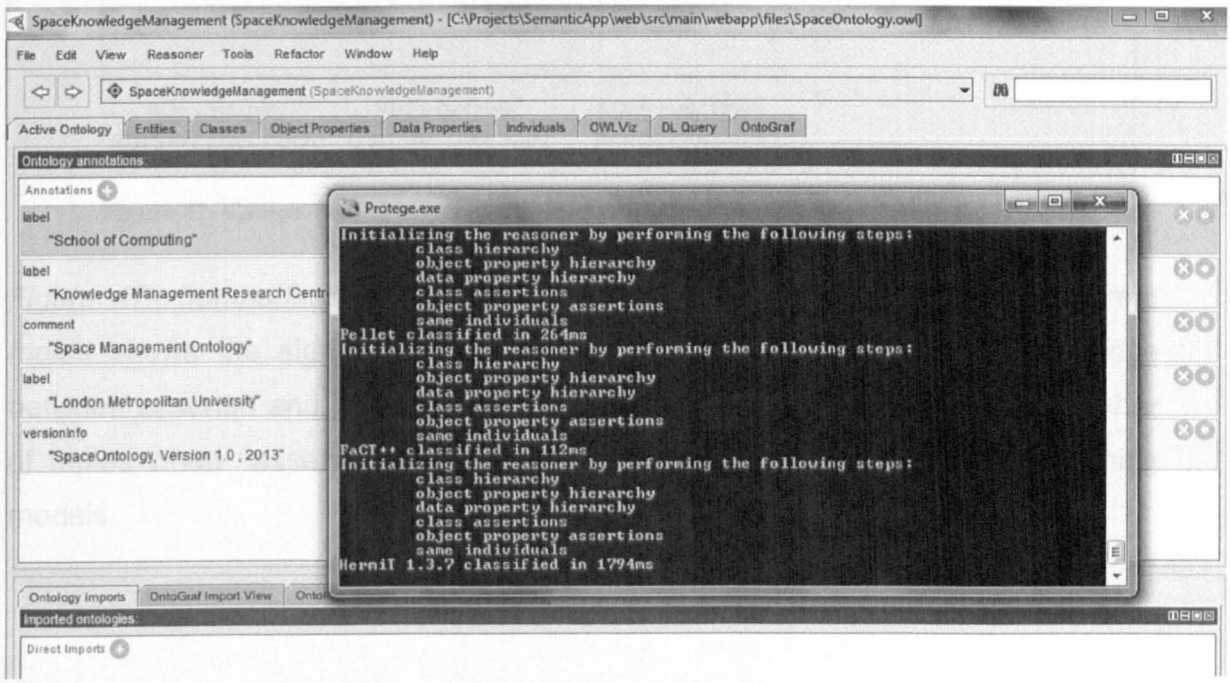


Figure 46- Third Party Reasoners Log for Mapped OWL Schema

The different evaluation results by the three main categories above (*Evaluation of pure Jena Framework Implementation, Evaluation of Oracle's Semantic Web Implementation,* and *Evaluation of Third Party Reasoners Implementation*) shows a

variety of additional information compared to the original mapped OWL schema and data. While the relational data is closed to any inference and stays constant, the mapped OWL schema and RDF data shows a significant increase in the number of triples being inferred. This shows the potential to discover more data patterns and new facts when existing relational databases that currently hold most of the world's knowledge base are mapped to Ontology repositories

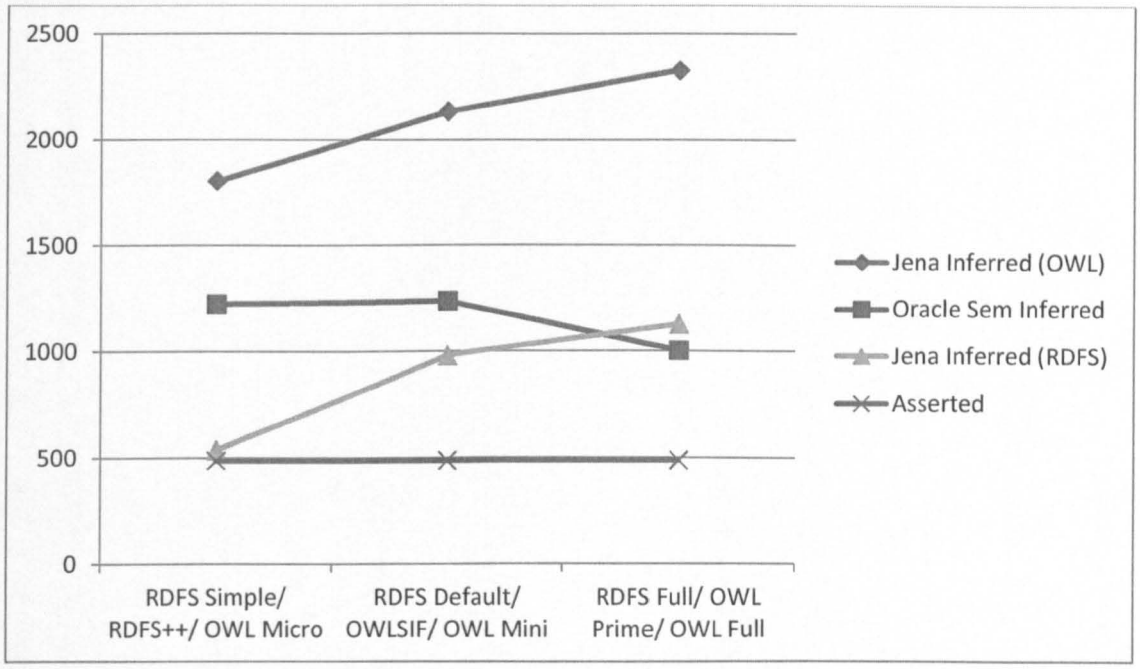


Figure 47- Comparison of Inferred (RDFS, Oracle Built-in, OWL) and Asserted triples

Figure 47- Comparison of Inferred (RDFS, Oracle Built-in, OWL) and Asserted triples shows the significant difference in the amount of inference knowledge between asserted and inferred triples. It shows the overall increase in the number of triples when asserted triples are inferred using different levels of reasoning models.

8.2 Feature Comparison

The feature comparison was done against some of the existing approaches (Direct Mapping, eD2R, R_2O, Relational.OWL, Virtuso, D2RQ, Triplify, R2RML, R3M). The criteria used are outlined by the comparison of RDB-to-RDF Mapping Languages seminal paper (Hert, Gerald, & Gall, 2011). The paper categorises the nine mapping languages into four based on their purpose as Direct Mapping, Read-only General-purpose mapping, Read-write General-purpose mapping and Special-purpose mapping. The comparison also outlined fifteen features (F1 – F15) that are compared against the new Relational Database to Semantic Web (RD2SW) implementation approach. There are additional two features (F16 & F17) contributed by this research to further extend current approach.

F1 Logical Table to Class: Other than Relational.OWL and R3M, all including the new RD2SW support mapping logical tables to ontology classes.

F2 M:N Relationships: this feature is supported implicitly by eD2R, Virtuoso, D2RQ, Triplify, R2RML, and RD2SW. The new RD2SW gives emphasis to mapping M:N relationships for a rich **Data Pattern (F16)** discovery. The only mapping language supporting explicit M:N Relationship is R3M. Direct Mapping, R2O, and Relational.OWL don't have any special support for this feature.

F3 Project Attributes: Virtuoso, Triplify, R2RML, and RD2SW support this feature using the underlying SQL queries. Direct Mapping implements the attribute to properties mapping automatically.

F4 Select Conditions: Virtuoso, Triplify, R2RML, and RD2SW also support this feature using the underlying SQL queries. R3M provides limited support while Direct Mapping and Relational.OWL do not support this feature.

F5 User-defined Instance URIs: Triplify and the new RD2SW rely on the SQL query to produce User-defined instance URIs. The new RD2SW also has a configuration setup that could be read at run time to add more User-defined

instances. Direct Mapping and Relational.OWL do not support this feature. In fact Relational.OWL only uses blank nodes for all instances.

F6 Literal to URI: All except Direct Mapping and Relational.OWL support this feature.

F7 Vocabulary Reuse: Since existing relational attributes are used in Direct Mapping and Relational.OWL, the generated ontology properties cannot be reused. The rest of the mapping languages support this feature.

F8 Transformation Functions: All except Direct Mapping and Relational.OWL support this feature. While Triplify, R2RML, and RD2SW use the functions in SQL, R3M requires a bidirectional function to maintain read-write capability.

F9 Datatypes: Except Direct Mapping, all support datatype feature.

F10 Named Graphs: While Relational.OWL, Triplify, R3M, and RD2SW do not support this feature purposely, the rest were developed before the emergence of this feature and do not have support for Named Graphs.

F11 Blank Nodes: this feature is not supported in Direct Mapping, R3M and Triplify. Relational.OWL maps all instances as blank nodes. The rest including the new RD2SW support this feature.

F12 Integrity Constraints: while Direct Mapping provides no support for this feature, R3M and the new RD2SW provide a rich combination of integrity constraint support. The rest have limited support.

F13 Static Metadata: Support for static non-schema level data is only available in D2RQ and R2RML.

F14 One Table to n Classes: Direct Mapping and Relational.OWL have no support for this feature while R_2O and R3M provide a partial and restricted support respectively. Virtuoso, D2RQ, Triplify, R2RML, and the new RD2SW use the strength of the underlying SQL query to implement this feature.

F15 Write Support: Relational.OWL and R3M provide a unidirectional and bidirectional write support respectively. The rest of the mapping languages are designed for a read-only implementation.

F16 Data Patterns: this feature relies on the extensive use of relationships (F2) and their constraints (F12) to add more RDB implicit knowledge to the ontological repository. Up to the publication of this thesis there is no mapping language that covers this feature.

F17 Data Control Languages (DCL): The other new feature of RD2SW is Data Control Language (DCL). The richer vocabularies of Semantic Ontology languages enable us to represent DCLs that are stored outside the relational schema in relational databases. Ontology languages allow the direct representation of DCLs. This meta-data gives a much more fine control over the mapped data manipulation

The results of the fifteen (F1 – F15) existing and the two new (F16 & F17) additional feature comparison can be summarised as shown in *Table 45* and *Table 46* below.

Table 45- RDB-to-RDF Mapping Languages Comparison Legend (Hert, Gerald, & Gall, 2011) and New RD2SW

Legend	Description
F1	Logical Table to Class
F2	M:N Relationships
F3	Project Attributes
F4	Select Conditions
F5	User-defined Instance URIs
F6	Literal to URI
F7	Vocabulary Reuse
F8	Transformation Functions
F9	Datatypes
F10	Named Graphs
F11	Blank Nodes
F12	Integrity Constraints
F13	Static Metadata
F14	One Table to n Classes
F15	Write Support
F16	**Data Patterns**
F17	**Data Control Languages (DCL)**

Table 46- Summary table of RDB-to-RDF Mapping Languages Comparison Legend (Hert, Gerald, & Gall, 2011) with RD2SW

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17
Direct Mapping	(✓)	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✓		
eD2R	✓	✓	✓	✓	✓	✓	✓	✓	✓	✖	✓	(✓)	✖	✓	✖		
R_2O	✓	✖	✓	✓	✓	✓	✓	✓	✓	✖	✓	(✓)	✖	(✓)	✖		
Relational.OWL	(✓)	✖	✓	✖	✖	✖	✖	✖	✓	✖	✓	(✓)	✖	✖	✓		
Virtuoso	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	(✓)	✖	✓	✖		
D2RQ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✖	✓	(✓)	✓	✓	✖		
Triplify	✓	✓	✓	✓	✓	✓	✓	✓	✓	✖	✖	(✓)	✖	✓	✖		
R2RML	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	(✓)	✓	✓	✖		
R3M	(✓)	✓	✓	(✓)	✓	✓	✓	✓	✓	✖	✖	✓	✖	✓	✓		
RD2SW	✓	✓	✓	✓	✓	✓	✓	✓	✓	✖	✓	✓	✖	✓	✖	✓	✓

Chapter 9: Conclusion and Future Work

This research is dedicated to mapping existing relational databases to ontological repositories for a richer and explicit representation of knowledge. Algorithms that use different relational models, patterns and domain-specific knowledge have been designed to achieve an exhaustive mapping implementation. The summary of the research, implementation, evaluation; and main contributions are summarised as follows.

Research Summary

Relational databases are a very efficient way of representing and storing relational models. In spite of their popularity, the lack of representing the data with richer semantics makes them inadequate candidates for representing implicit knowledge. Most of the dynamically generated web comes from relational databases (Geller, et al., 2008) and keeps growing exponentially due to the fast growing collaborative enterprise, commerce, social and other software applications. The problem here is a growing 'bank of information' that is not being delivered accurately to the end user because of the absence of a methodology for interpreting the semantics of data in relational databases. The legendary 'keyword' search on relational databases helps applications to return matching records but with no consideration of the semantics. When a search result goes to tens of hundreds of pages, finding the right result becomes almost impossible.

To overcome this problem, mapping relational databases (RDB) to ontological repositories becomes a timely solution. Semantic Web framework has standards suitable for representing ontological repositories as well as ontology languages to describe the data. Part of the Semantic Web framework is the possibility to use logical inference engines as well as domain-specific rules that enable a better interpretation of the data and discovery of patterns.

The investigation on the current approaches of mapping relational databases to ontological repositories is functionally divided into "mapping creation", "mapping implementation", and "query implementation". Mapping

creation follows an automatic, semi-automatic or manual generation of ontological repositories. Mapping creation is usually accompanied by either a new ontology that is created as part of the mapping or an existing ontology designed by the domain expert users. The challenge in using existing ontologies is to find a well-designed ontology to match the ontological repository. It is usually unlikely to find the same party that designed and developed the ontology to be also the expert in mapping the relational database to ontological repositories which makes the success of efficiently matching the repository with an existing ontology questionable. The mapping success also depends on the use of different synonym, lexical and pattern matching. The other mapping option is the use of new ontologies. This is a common approach partly due to the lack of existing ontologies as well as the challenge in matching the concepts to an existing ontology. In the case of this research a new ontology is created as part of mapping the relational database to the ontological repository. This research's mapping creation approach followed an automatic ontological repository generation while creating a new ontology as part of the mapping creation. The mapping and query implementation use a dynamic approach that uses both SQL (*pre-processing* and *in-processing*) and SPARQL (*post-processing*) for accessing the data at different phases of the mapping process.

The research also compared a number of general-purpose and special mapping methods. The comparison used a number of criteria like logical entities, relationships, attributes, vocabulary reuse, data types, blank nodes, integrity constraints, and write support features. The result of the analysis shows absence of means to map complex relationships, data patterns, and consideration of relational database elements outside the relational schema. Leaving out patterns and only mapping simple relational tables and relationships results in the abandonment of current and legacy implicit knowledge. The existing but unrepresented implicit knowledge and data patterns shall be mapped to their natural ontological repository for its explicit representation. The mapping languages concentrate on the relational schema leaving out the rest of the relational database like the Data Control Language (DCL) data. Mapping DCL data directly in the ontological

repository keeps the integrity of the repository with respect to data manipulation access control. Including user's privilege and access rights as part of the mapped repository adds more semantic to the ontological data and helps to deliver the right data to the right user with the right access privilege.

The current mapping languages have no means of accounting for all the available relational database area objects and their variant meta-data combinations. It also has no means of accounting for domain data knowledge like data-patterns (disjointness, symmetry, transitive chain, etc.) and the application domain knowledge within the application domain that can be explicitly represented using the rich semantic ontology languages. To address these issues, the mapping algorithm categorised the sources into relational database area knowledge (keys, constraints, and relationships), domain area knowledge (complex relationships and patterns), domain users knowledge (user privileges and access rights) and application domain knowledge (predicates, functions, procedures). The algorithms start from simple relational tables, columns, constraints, relationships and then to the more complex pattern checks like "transitive chain" and "disjointness". The algorithm is incremental and iterative and accounts for basic as well as complex relational database elements. The implementation and evaluation of the algorithm uses a "Space Project Management" application scenario. The application is used as a source of relational schema, the instance data, and application domain knowledge.

Summary of Implementation and Evaluation

The mapping implementation of relational databases to ontological repositories using domain-specific knowledge starts by setting up the relational database to be used as a source schema. The implementation has a *pre-processing* phase to identify keys, constraints, and relationships to create the ontological schema. At the same time the relational database constraints and relationships are used incrementally to find patterns and enrich the ontological schema. Once the ontological schema is created, the *in-processing* phase starts by mapping the relational data to RDF

repositories. The *in-processing* phase also converts the relational SQL data types to RDF data types with a valid URI, literal or blank node. After the *pre* and *in-processing* phase, the *post-processing* phase is where the semantic analysis starts.

Semantic analysis in this research uses inference engines and application domain rules. The implementation and evaluation use different Jena built-in, proprietary, user-defined, and third party reasoners to infer additional data. The implementation shows the application of these groups of reasoners on the mapped ontological repository.

The results of the reasoner output show a significant increase in the richness of the data. The comparison across the different Jena built in (RDFS, OWL, DAML), proprietary (Oracle's RDFS++, OWLSIF, OWL Prime, SKOS Core), user-defined (ex: transitive Chain), and third party (Pellet, Racer, FaCT) reasoners show the representation of the data patterns and complex relationships in addition to the first phase asserted triples output. The ontology language specifications together with the new domain-specific ontology is used to describe the mapped RDF repository to show the complex hierarchical relationships amongst the classes, properties, relations, restrictions, and axioms. The representation of patterns and complex relationships shows Semantic Web's capability of mapping implicit relational knowledge into explicit ontological statements using the different Semantic Web Ontology languages. Our approach also allowed a range of inference engine reasoning that produced a richer working ontological repository.

Main Contributions

Firstly, a detailed mapping algorithm implementation has been designed to map relational databases to ontological repositories. The algorithm covers not only the simple relational models like relations, relationships and attributes, but also patterns and domain-specific knowledge. Other than the explicit entities that can be mapped using basic ontology language expressions, this mapping algorithm implementation also accounts implicit "knowledge" hidden inside the relational model. Missing this implicit

knowledge during mapping will leave behind invaluable information that would have been explicitly represented and used in a Semantic Web environment. Using complex ontology language axioms and rule logic, this research managed to map different relational patterns and domain-specific knowledge (domain data, domain user, and application domain) into the ontological repository. A further inclusion of user privileges and access rights also gave the ontological repository more control over the data manipulation.

Secondly, experimentally testable mapping implementations have also been delivered as part of the research. The experiment comprises different verification, validation and evaluation techniques. The incremental iterative mapping implementation is verified using a log of the sub-procedures within the whole iteration. The input relational schema and data has also been validated against the ontological output of the mapping implementation. After the verification of the incrementally iterative mapping implementation and the validation of its output, the mapped ontological repository has also been evaluated using different open source, proprietary and user defined inference engines (i.e. Semantic Reasoners).

Thirdly, the mapping also produced a richer ontological repository open to standard (RDFS, OWL, DAML), proprietary (Oracle's RDFS++, OWLSIF, OWL Prime, SKOS Core), and third-party (Pellet, Racer, FaCT) reasoners that lead to the discovery of new data and patterns. The mapping also has a by-product Ontology that could be used as a "dictionary" to increase the expressiveness of the concepts and their relationships.

Finally, the research resulted in an interactive RDB to RDF mapping semantic web application that can be configured to multiple database management systems. The application utilises up-to-date Apache Jena framework to manage the ontological repository. The problem domain, existing mapping approaches, proposed solution and implementation results are summarised in detail as follows.

Semantic Web as the core motto of Web 3.0 has made a substantial progress in driving the transition from the 'web of documents' to 'web of data' that is universal,

interoperable and machine-processable. Acquisition of online databases like Freebase (online collaborative knowledge base) by Google and its 'Linked Open Data' project are signs of adopting a new knowledge-based approach to data storage and retrieval. The participation of end users in adding more meta-data through collaborative tagging as well as a collective initiative to use common meta-data schema like *schema.org* by Bing, Google and Yahoo! is an encouraging prospect to the realisation of an ever larger 'web of data'. These meta-data utilisations make the mapping of deep web relational databases to ontological repositories a more natural addition to the 'web of data'.

By following the Semantic Web standards set by World Wide Web Consortium (W3C) and making sure the hidden relational knowledge is mapped across by designing efficient and detailed algorithms, the goal of using Semantic Web to represent knowledge can be realised. The journey of mapping relational databases to Semantic Web should be more than just the superficial plain data if we want Semantic Web to be accepted as the next stage of representing 'web resources' in a standardised, unambiguous, interoperable and above all Linked-data format. The standards and the implementation results show Semantic Web as the next efficient framework of representing knowledge in the 21st century. Historical legacy and current relational databases have implicit knowledge waiting to be discovered by Semantic Web standards.

Future Work

Although the algorithm implementation managed to show the potential of using different relationship patterns, it could go further to add more patterns to explore the hidden relational database for an explicit mapping to ontological repositories. There is also more opportunity to identify inter-knowledge correlations based on the different domain data, user, and application categories. An instance such as building a domain user's profile based on the application knowledge is a typical example on how different users can inherit privileges on application transaction controls. The rest of the future work concentrates on interchanging rules and ontologies to further strengthen and reuse the mapping algorithms.

Rule Sharing and Interchange

The current mapping implementation makes use of rules to represent and interpret the semantics of the ontological repository. As part of the Semantic Web Architecture, World Wide Web Consortium (W3C) has chosen Rule Interchange Format (RIF) as a standard for sharing and interchanging rules. The current RIF working group has released different editions such as RIF Basic Logic Dialect, RIF Production Rule Dialect, and RIF Core Dialect. These RIF dialects will enrich the mapped ontological repository by adding more deductive, inductive, and reactive rules. RIF also goes further into complex reasoning like default, uncertainty, and paraconsistent reasoning. Once the ontological repository is created using different reasoning rules, using an exchangeable rule standard keeps the integrity of the Ontology repository.

Ontology Alignment

One of the future works is to address the lack of interlinks between the different ontological schemas. The algorithms and their implementation can be applied to another relational database. The expected result of the algorithm implementation can be assessed based on the ontological matching between two or more relational databases. The ontological alignment can be a hierarchical 'rdfs:subClassOf' relationship between the two relational databases. Another typical example based on the current algorithm implementation is the use of symmetrical 'rdfs:subClassOf' relationship between two classes that results in 'owl:equivalentClass' ontology alignment. A further alignment of 'owl:SymmetricProperty' could also draw a property ontology alignment. The axioms mentioned above are used in the ontological schema of the current implementation. More ontology alignment can be done to further enrich the ontology.

Published Papers

1. Title: ***"Algorithms for mapping RDB Schema to RDF for Facilitating Access to Deep Web"***. The First International Conference on Building and Exploring Web Based Environments; WEB 2013; January 27 - February 1, 2013 - Seville, Spain [Published]. URL: http://www.thinkmind.org/index.php?view=article&articleid=web_2013_2_30_40075

2. Title: ***"Mapping Space Project Management Application to Semantic Web Using Jena Adapter for Oracle"***. 8th International Workshop on Semantic Web Enabled Software Engineering, 2 to 4 December 2012; Nara, Japan [Published as a poster paper]. URL: http://swese.odsd.eu/swese2012/files/swese2012-proceeding.pdf

References

Abadi, D. J., Marcus, A., Madden, S. & Hollenbach, K., 2009. SW-Store: a vertically partitioned DBMS for Semantic Web. The International Journal on Very Large Data Bases, 18(2), pp. 385-406.

Abadi, D. J., Marcus, A., Madden, S. R. & Hollenbach, K., 2007. Using The Barton Libraries Dataset As An RDF Benchmark, MIT: Technical Report MIT-CSAIL-TR-2007-036.

Alalwan, N., Zedan, H. & Siewe, F., 2009. Generating OWL Ontology for Database Integration. Washington, DC, USA, IEEE, Proceedings of the 3rd International Conference on Advances in Semantic Processing, pp. 22-31.

Allemang, D. & Hendler, J., 2008. Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL. Boston: Morgan Kaufmann.

Auer, S. et al., 2009. Triplify – Lightweight Linked Data Publication from Relational Databases. Madrid, Spain, ACM, Proceedings of the World Wide Web Conference (WWW 2009).

Baader, F. et al., 2010. The Description Logic Handbook: Theory, Implementation and Applications. 2nd Edition ed. New York, NY, USA: Cambridge University Press.

Barrasa, J., Corcho, O. & G´omez-P´erez, A., 2003. Fund Finder: A Case Study of Database-to-Ontology Mapping. Florida, USA, Proceedings of the 2nd International Semantic Web Conference, ISWC2003.

Barrasa, J., Corcho, O. & G´omez-P´erez, A., 2004. R2O, an Extensible and Semantically based Database-to-Ontology Mapping Language. s.l., In Proceedings of the 2nd Workshop on Semantic Web and Databases(SWDB2004), pp. 1069--1070.

Benjamins, V., Contreras, J., Corcho, O. & Gómez-Pérez, A., 2002. Six Challenges for the Semantic Web. Toulouse, France, s.n.

Benslimane, S. M., Benslimane, D. & Malki, M., 2006. Acquiring OWL Ontologies from Data-Intensive Web Sites. Palo Alto, California, USA, ACM, Proceedings of the 6th International Conference on Web Engineering (ICWE), pp. 361-368.

Berners-Lee, T., 1999. Weaving the Web. s.l.:Orion Business Books.

Berners-Lee, T. & Beckett, D., 2008. Turtle - Terse RDF Triple Language -W3C Team Submission. [Online]
Available at: http://www.w3.org/TeamSubmission/turtle/

Bizer, C. & Cyganiak, R., 2007. D2RQ Lessons Learned. Cambridge, USA, s.n., pp. 25-26.

Blakeley, C., 2007. RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping), s.l.: OpenLink Software.

Boley, H. et al., n.d. EBNF for the RIF-Core. [Online]
Available at: http://www.w3.org/2005/rules/wiki/Core#EBNF_for_the_RIF-Core_Rule_Language
[Accessed 2012].

Boley, H., Kifer, M., Patranjan, P.-L. & Polleres, A., 2007. Rule Interchange on the Web. s.l., s.n., pp. 269-309.

Bruijn, J. d., 2010. RIF RDF and OWL Compatibility. [Online]
Available at: http://www.w3.org/TR/2010/REC-rif-rdf-owl-20100622/
[Accessed 2012].

Byrne, K., 2008. Having Triplets – Holding Cultural Data as RDF. Aarhus, Denmark, Proceedings of the ECDL 2008 Workshop on Information Access to Cultural Heritage.

Carroll, J. J. et al., 2004. Jena: Implementing the Semantic Web Recommendations. New York, s.n., pp. 74-83.

Casanova, M. A., Barbosa, S. D. J., Breitman, K. K. & Furtado, A. L., 2012. Three Decades of Research on Database Design at PUC-Rio. Journal of Information and Data Management, 3(1), pp. 17-32.

Chau, R. & Yeh, C.-H., 2005. Enabling a Semantic Smart WWW: A Soft Computing Framework for Automatic Ontology Development. CIMCA/IAWTIC, pp. 1067-1071.

Choi, M.-Y.et al., 2010. The RDFS mapping for recursive relationship of relational data model. Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on , pp. 1-6.

Cyganiak, R., 2005. A relational algebra for SPARQL, s.l.: HP Technical Report, HPL-2005-170.

D2RQ, 2013. Accessing Relational Databases as Virtual RDF Graphs. [Online]
Available at: http://d2rq.org/
[Accessed 01 05 2013].

Daconta, M. C., Obrst, L. J. & Smith, K. T., 2003. The Semantic Web. s.l.:Wiley Publishing.

Das, S., Sundara, S. & Cyganiak, R., 2010. R2RML: RDB to RDF Mapping Language. [Online]
Available at: http://www.w3.org/2001/sw/rdb2rdf/r2rml/

Gali, A., Chen, C. X., Claypool, K. T. & Uceda-Sosa, R., 2004. From Ontology to Relational Databases. ER Workshops 2004, LNCS 3289, p. Gali.

Geller, J., Chun, S. A. & An, Y. J., 2008. Toward the Semantic Deep Web. IEEE Computer, 41(9), pp. 95-97.

Glassey, O., 2007. When Taxonomy Meets Folksonomy: Towards Hybrid Classification of Knowledge?. ESSHRA International Conference Berne, Switzerland, pp. 12-13.

Golbreich, C. & Wallace, E. K., 2009. OWL 2 Web Ontology Language: New Features and Rationale. [Online]
Available at: http://www.w3.org/TR/2009/REC-owl2-new-features-20091027/
[Accessed 2012].

Guy, M. & Tonkin, E., 2006. Folksonomies: "Tidying up tags?. D-Lib Magazine,Vol. 12, Number 1, January 2008.

Harmelen, G. A. a. F. v., 2008. A Semantic Web Primer. s.l.:The MIT Press, 2nd Edition.

Hert, M., Gerald, R. & Gall, H. C., 2011. A comparison of RDB-to-RDF Mapping Languages. Graz, Austria, ACM, pp. 25-32.

Horrocks, I., Patel-Schneider, P. F. & Harmelen, F. V., 2003. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. Journal of Web Semantics, 1(1), pp. 7-26.

Huang, J., Abadi, D. J. & Ren, K., 2011. Scalable SPARQL Querying of Large RDF Graphs. PVLDB, 4(11), pp. 1123-1134.

Jalali, V., Zhou, M. & Wu, Y., 2011. A Study of RDB-Based RDF Data Management Techniques. Wuhan, China, Web-Age Information Management - 12th International Conference.

Jena, A., 2012. Apache Jena: Jena Ontology API. [Online] Available at: http://jena.apache.org/about_jena/architecture.html [Accessed 21 12 2012].

Jena, I., 2012. Apache Jena: Reasoners and rule engines: Jena inference support. [Online] Available at: http://jena.apache.org/documentation/inference/ [Accessed 23 12 2012].

Jena, O., 2012. Jena Ontology API. [Online] Available at: http://jena.apache.org/documentation/ontology/ [Accessed 19 12 2012].

Jörgensen, C., 2007. Image Access, the Semantic Gap, and Social Tagging as a Paradigm Shift. Florida State University, s.n.

Kim, S. & Kwon, J., 2008. Folksonomy-Based Information Retrieval in Context-aware Environment. IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.11, pp. 252-257.

Kroski, E., 2005. The hive mind: Folksonomies and user-based tagging. [Online]
Available at: http://infotangle.blogsome.com/2005/12/07/the-hive-mind-folksonomies-and-user-based-tagging/
[Accessed 2011].

LibraryThing, 2013. LibraryThing | Catalog your books online. [Online] Available at: http://www.librarything.com/ [Accessed 2013].

Lopez, X., 2009. Oracle Database 11g Semantic Technologies, s.l.: Oracle.

Mathes, A., 2004. Folksonomies - Cooperative Classification and Communication Through Shared Metadata. Computer Mediated Communication - LIS590CMC.

Marx, E., Salas, P., Breitman, K. & Viterbo, J., 2012. RDB2RDF: A relational to RDF plug-in for Eclipse, s.l.: Published online in Wiley Online Library (wileyonlinelibrary.com) Softw. Pract. Exper..

Motik, B., Parsia, B. & Patel-Schneider, P. F., 2009. OWL 2 Web Ontology Language: XML Serialization. [Online]
Available at: http://www.w3.org/TR/2009/REC-owl2-xml-serialization-20091027/
[Accessed 2012].

Motik, B., Patel-Schneider, P. F. & Grau, B. C., 2009. OWL 2 Web Ontology Language: Direct Semantics. [Online]
Available at: http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/
[Accessed 2012].

Murray, C., 2010. Semantic Technologies Developer's Guide 11g Release 2 (11.2), Redwood City, CA: Oracle Corporation.

NCBO, 2012. The National Centre For Biomedical Ontology. [Online]
Available at: http://www.bioontology.org/
[Accessed 2012].

Nilsson, M., Palmér, M. & Naeve, A., 2002. Semantic web metadata for e-learning - some architectural guidelines. Proc. of the 11th World Wide Web Conference (WWW2002), Hawaii, USA, May 7-11.

Pan, Z. & Heflin, J., 2003. DLDB: Extending Relational Databases to Support Semantic Web Queries. s.l., s.n.

Perez de Laborda, C. & Conrad, S., 2005. Relational.OWL - A Data and Schema Representation Format based on OWL. Newcastle, NSW, Australia, In Proceedings of the 2nd Asia-Pacific Conference on Conceptual Modelling.

Ramakrishnan, R. & Gehrke, J., 2000. Database Management Systems (2nd Edition). s.l.:Mc-Graw Hill.

RCAHMS, 2012. Royal Commission on the Ancient and Historical Monuments of Scotland. [Online]
Available at: http://www.rcahms.gov.uk/
[Accessed 2012].

Ronkowitz, K., 2012. Is Folksonomy Taxonomy or Fauxonomy?. [Online]
Available at: http://www.serendipity35.net/index.php?/archives/11-Is-Folksonomy-Taxonomy-or-Fauxonomy.html
[Accessed 2012].

Rybiński, H., 1987. On first-order-logic databases. ACM Transactions on Database Systems (TODS), 12(3), pp. 325-349.

Sahoo, S. S. et al., 2009. A Survey of Current Approaches for Mapping of Relational Databases to RDF. [Online]
Available at: http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport
[Accessed 2012].

Schneider, M., 2009. OWL 2 Web Ontology Language: RDF-Based Semantics. [Online]
Available at: http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/
[Accessed 2013].

Seaborne, A. et al., 2008. SPARQL Update. [Online]
Available at: http://www.w3.org/Submission/SPARQL-Update/
[Accessed 01 05 2013].

Shadbolt, N., Berners-Lee, T. & Hall, W., 2006. The Semantic Web Revisited. IEEE Intelligent Systems, vol. 21, no. 3, pp. 96-101.

Sheth, A., Ramakrishnan, C. & Thomas, C., 2005. Semantics for the Semantic Web: The Implicit, the Formal and the Powerful. Int'l Journal on Semantic Web & Information Systems, pp. 1-18.

SquirrelRDF, 2013. SquirrelRDF. [Online] Available at: http://jena.sourceforge.net/SquirrelRDF/ [Accessed 06 2013].

Staab, S. & Studer, R., 2009. Handbook on Ontologies. Berlin: Springer.

Studer, R., Benjamins, V. R. & Fensel, D., 1998. Data and Knowledge Engineering. Knowledge engineering: Principles and methods, pp. 161-197.

Suchanek, F. M., Kasneci, G. & Weikum, G., 2007. Yago: A Core of Semantic Knowledge. Banff, Alberta, Canada, Proceedings of the 16th International Conference on World Wide Web, pp. 697-706.

Tirmizi, S. H., Sequeda, J. & Miranker, D., 2008. Translating SQL Applications to the Semantic Web. Heidelberg, Springer-Verlag Berlin, pp. 450-464.

Virtuoso, 2013. Mapping Relational Data to RDF with Virtuoso's RDF Views. [Online] Available at: http://virtuoso.openlinksw.com/whitepapers/relational%20rdf%20views%20mapping.html [Accessed 01 05 2013].

W3C-OWLGroup, 2010. W3C OWL Working Group, OWL 2 Web Ontology Language. [Online] Available at: http://www.w3.org/TR/owl2-overview/ [Accessed 2012].

Wilkinson, K., Sayers, C., Kuno, H. A. & Reynolds, D., 2003. Efficient RDF Storage and Retrieval in Jena2. Berlin, Germany, Proceedings of SWDB'03. The first International Workshop on Semantic Web and Databases, pp. 131-150.

Wright, A., 2009. Exploring a 'Deep Web' That Google Can't Grasp. [Online] Available at: http://www.nytimes.com/2009/02/23/technology/internet/23search.html?_r=1&partner=rss&emc=rss [Accessed 2012].

Wu, W., Doan, A., Yu, C. & Meng, W., 2005. Bootstrapping Domain Ontology for Semantic Web Services from Source Web Sites. Trondheim, Norway, s.n., pp. 11-22.

Additional Bibliography

Abadi, D. J., Marcus, A., Madden, S. & Hollenbach, K. J., 2007. Scalable Semantic Web Data Management Using Vertical Partitioning. University of Vienna, Austria, Proceedings of the 33rd International Conference on Very Large Data Bases.

Alexander, N. et al., 2008. RDF Data Model in Oracle, Nashua, USA: Oracle Corporation.

Alexander, N. & Ravada, S., 2006. RDF Object Type and Reification in the Database. Washington, DC, USA, IEEE, Proceedings of the 22nd International Conference on Data Engineering, p. 93.

Allemang, D. et al., 2005. Collaborative Ontology Visualization and Evolution. IEEE Aerospace Conference, pp. 1-10.

An, Y., Borgida, A. & Mylopoulos, J., 2006. Discovering the Semantics of Relational Tables Through Mappings. Journal on Data Semantics VII, Volume 4244, pp. 1-32.

Bail, S., Parsia, B. & Sattler, U., 2011. Extracting Finite Sets of Entailments from OWL Ontologies. Barcelona, Spain, Proceedings of the 24th International Workshop on Description Logics.

Bézivin, J. & Kurtev, I., 2005. Model-based Technology Integration with the Technical Space Concept. s.l., Springer.

Boyce, S. & Pahl, C., 2007. Developing Domain Ontologies for Course Content. Educational Technology & Society, Volume 10, pp. 275-288.

Bry, F. & Eckert, M., 2007. Twelve Theses on Reactive Rules for the Web. Schloss Dagstuhl, Germany, Internationales Begegnungs- und Forschungszentrum fuer Informatik.

Bumans, G., 2010. Mapping between Relational Databases and OWL Ontologies: an Example. Computer Science and Information Technologies, Volume 756, p. 99–117.

Burton-Jones, A., Storey, V. C., Sugumaran, V. & Purao, S., 2003. A Heuristic-Based Methodology for Semantic Augmentation of User Queries on the Web. Chicago, IL, USA, Springer, pp. 476-489.

Calvanese, D., Lenzerini, M. & Nardi, D., 1999. Unifying Class-Based Representation Formalisms. Journal Of Artificial Intelligence Research, Volume 11, pp. 199-240.Chandler, D., 2007. Semiotics: The Basics. 2 ed. s.l.:Routledge.

Chang, K. C.-C. et al., 2004. Structured Databases on the Web: Observations and Implications. ACM SIGMOD Record, 33(3), pp. 61-70.

Chen, H. et al., 2006. Towards a Semantic Web of relational databases: A practical semantic toolkit and an in-use case from traditional Chinese medicine. Berlin, Springer-Verlag Berlin, Heidelberg.

Chen, L., Shadbolt, N. R. & Goble, C. A., 2007. A Semantic Web-Based Approach to Knowledge Management for Grid Applications. IEEE Transactions on Knowledge and Data Engineering, 19(2), pp. 283-296 .

Choi, N., Song, I.-Y. & Han, H., 2006. A survey on Ontology Mapping. ACM SIGMOD Record, 35(3), pp. 34-41.

Dai, H. & Mobasher, B., 2003. A Road Map to More Effective Web Personalization: Integrating. Las Vegas, Nevada, USA, Proceedings of the International Conference on Internet.

Ding, L., Wilkinson, K., Sayers, C. & Kuno, H. A., 2003. Application-Specific Schema Design for Storing Large RDF. Sanibel Island, Florida, USA, CEUR-WS.org.

Florescu, D., Levy, A. & Mendelzon, A., 1998. Database Techniques for the World-Wide Web: A Survey. ACM SIGMOD Record, 27(3), pp. 59-74.

Gallego, M. A., Fernández, J. D., Martínez-Prieto, M. A. & Gutiérrez, C., 2011. HDT-it: Storing, Sharing and Visualizing Huge RDF Datasets. Koblenz, Germany, 10th International Semantic Web Conference (ISWC).

Grobe, M., 2009. RDF, Jena, SparQL and the 'Semantic Web'. ACM, pp. 131-138.

Haase, P., Broekstra, J., Eberhart, A. & Volz, R., 2004. A Comparison of RDF Query Languages. s.l., Springer, pp. 502-517.

Hayes, P., 2004. RDF Semantics. [Online]
Available at: http://www.w3.org/TR/rdf-mt/#Reif
[Accessed 23 12 2012].

He, B., Patel, M., Zhang, Z. & Chang, K. C.-C., 2007. Accessing the deep web. Communications of the ACM, 50(5), pp. 94-101.

Hendler, J., 2008. Web 3.0: Chicken Farms on the Semantic Web. IEEE Computer Society Press, 41(1), pp. 106-108.

Henson, S. L., 2007. Approaching the Semantic Web using Folksonomies, s.l.: Relstore/Scrumdidilyumptiou.us: White paper funded by the Andrew W. Mellon Foundation for the Folksemantic.org project.

Horridge, M. & Patel-Schneider, P. F., 2009. OWL 2 Web Ontology Language: Manchester Syntax Matthew Horridge. [Online]
Available at: http://www.w3.org/TR/2009/NOTE-owl2-manchester-syntax-20091027/
[Accessed 2012].

Horst, H. J., 2005. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. Journal of Web Semantics, 3(2-3), pp. 79-115.

Hu, C., Li, H., Zhang, X. & Zhao, C., 2008. Research and Implementation of Domain-Specific Ontology Building from Relational Database. Dunhuang, Gansu, IEEE, pp. 289 - 293.

Hutt, K., 2005. A Comparison of RDF Query Languages. Hartfort, Connecticut, Proc. of 21th Computer Science Seminar.

Hyvönen, E., Saarela, S. & Viljanen, K., 2003. Ontogator: Combining View- and Ontology-Based Search with semantic Browsing. s.l., IN PROCEEDINGS OF XML.

Jain, P. et al., 2010. Ontology Alignment for Linked Open Data. Shanghai, China, 9th International Semantic Web Conference.

Kaiya, H. & Saeki, M., 2006. Using Domain Ontology as Domain Knowledge for Requirements. s.l., IEEE Computer Society, pp. 186-195.

Kapsammer, E. et al., 2006. Lifting Metamodels to Ontologies - A Step to the Semantic Integration of Modeling Languages. s.l., In Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, pp. 528-542.

Kaviani, N., Gasevic, D., Hatala, M. & Wagner, G., 2007. Web Rule Languages to Carry Policies. Bologna, Italy, IEEE Computer Society, pp. 188-192.

Konstantinou, N., Spanos, D.-E., Stavrou, P. & Mitrou, N., 2010. Technically Approaching the Semantic Web Bottleneck. International Journal of Web Engineering and Technology, 6(1), pp. 83-111.

Kurtev, I., Bézivin, J. & Aksit, M., 2002. Technological spaces: An initial appraisal. s.l., DOA'2002 Federated Conferences, Industrial track.

Lausen, G., Meier, M. & Schmidt, M., 2008. SPARQLing constraints for RDF. s.l., Proc. EDBT'08, ACM.

Levshin, D. V., 2010. Mapping Relational Databases to the Semantic Web with Original Meaning. Int. J. Software and Informatics, 4(1), pp. 23-37.

Lima, F. & Schwabe, D., 2003. Application Modeling for the Semantic Web. Washington, DC, USA, IEEE, p. 93.

Lu, Y. et al., 2007. Annotating structured Data of the Deep Web. Istanbul, IEEE 23rd Int'l Conf. on Data Engineering.

Maedche, A. & Staab, S., 2001. Ontology learning for the Semantic Web. Intelligent Systems, IEEE, 16(2), pp. 72 - 79 .

Miller, L., Seaborne, A. & Reggiori, A., 2002. Three Implementations of SquishQL, a Simple RDF Query Language. Sardinia, Italy, First International Semantic.

Oracle, 2009. Advanced Performance and Scalability for Semantic Web Applications. [Online]
Available at: http://www.oracle.com/technetwork/database/options/semantic-tech/compressionreq-132202.pdf
[Accessed 17 12 2012].

Oren, E., 2006. An Overview of Information Management and Knowledge Work Studies. s.l., In Proceedings of the ISWC Workshop on the Semantic Desktop.

Pan, Z., Zhang, X. & Heflin, J., 2008. DLDB2: A Scalable Multi-perspective Semantic Web Repository. IEEE / WIC / ACM International Conference on Web Intelligence, Volume 1, pp. 489-495.

Ratiu, D., 2009. Challenges for Domain Knowledge Driven Program Analysis. s.l., 3rd Workshop on FAMIX and Moose in Reengineering.

Research, S. C. f. B. I., 2013. The Protégé Ontology Editor and Knowledge Acquisition System. [Online]
Available at: http://protege.stanford.edu

Santoso, H. A., Haw, S.-C. & Abdul-Mehdi, Z., 2011. Ontology extraction from relational database: Concept hierarchy as background knowledge. Knowledge-Based Systems, 24(3), p. 457–464.

Sayers, C. & Wilkinson, K., 2003. A Pragmatic Approach to Storing and Distributing RDF in Context Using Snippets, Palo Alto: HP Laboratories, Enterprise Systems and Data Management Laboratory.

Schmidt, M. et al., 2008. An Experimental Comparison of RDF Management Approaches in a SPARQL Benchmark Scenario. Karlsruhe, Germany, International Semantic Web Conference, pp. 82-97.

Spanos, D.-E., Stavrou, P. & Mitrou, N., 2010. Bringing Relational Databases into the Semantic Web: A Survey. IOS Press Journal.

Stevens, R. & Stevens, M., 2008. A Family History Knowledge Base Using OWL 2. Karlsruhe, Germany, Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions.

Suraweera, P., Mitrovic, A. & Martin, B., 2005. The use of ontologies in ITS domain knowledge authoring. Maceio, E-learning SWEL'04.

Theoharis, Y., Christophides, V. & Karvounarakis, G., 2005. Benchmarking Database Representations of RDF/S Stores. Galway, Ireland, Springer.

Turkmen, F., Kagal, L. & Crispo, B., 2009. Interoperable Access Control Policies: A XACML and RIF Demonstration. s.l., Massachusetts Institute of Technology (MIT).

Watkins, E. R. & Nicole, D. A., 2006. Named Graphs as a Mechanism for Reasoning About Provenance. Harbin, China, Springer, pp. 943-948.

Wilkinson, K., 2006. Jena Property Table Design, Palo Alto, California USA: Hewlett-Packard Laboratories.

Wilkinson, K. et al., 2003. Supporting Scalable, Persistent Semantic Web Applications. IEEE Data Eng. Bull., 26(4), pp. 33-39.

Xu, Z., Zhang, S. & Dong, Y., 2006. Mapping between Relational Database Schema and OWL Ontology for Deep Annotation. Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, pp. 548-552.

Yee, K.-P., Swearingen, K., Li, K. & Hearst, M., 2003. Faceted metadata for image search and browsing. Florida, USA, Proceedings of the 2003 Conference on Human Factors in Computing.

Zhou, S., Ling, H., Han, M. & Zhang, H., 2010. Ontology Generator from Relational Database Based on Jena. Computer and Information Science, 3(2).

Zouaq, A. & Nkambou, R., 2009. Evaluating the Generation of Domain Ontologies in the Knowledge Puzzle Project. IEEE Transactions on Knowledge and Data Engineering, 21(11), pp. 1559-1572.

Appendix A

Oracle RDF Database Tablespace, Schema Users and Grant Script

```
/*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* Relational and RDF Tablespaces                               */
/*==============================================================*/
```

Tablespace

```
CREATE TABLESPACE relational_users datafile
'relational_users01.dbf'
        size 128M reuse autoextend on next 64M
        maxsize unlimited segment space management auto;


CREATE TABLESPACE rdf_users datafile 'rdf_tablespace.dbf'
        size 128M reuse autoextend on next 64M
        maxsize unlimited segment space management auto;
```

Relational Schema Users

```
CREATE USER document IDENTIFIED BY <password-for-document>
                        DEFAULT TABLESPACE relational_users;

CREATE USER review IDENTIFIED BY <password-for-review>
                        DEFAULT TABLESPACE relational_users;

CREATE USER action IDENTIFIED BY <password-for-action>
                        DEFAULT TABLESPACE relational_users;

CREATE USER risk IDENTIFIED BY <password-for-risk>
                        DEFAULT TABLESPACE relational_users;

CREATE USER nonConformance IDENTIFIED BY <password-for-
nonConformance>
                        DEFAULT TABLESPACE relational_users;

CREATE USER role IDENTIFIED BY <password-for-role>
                        DEFAULT TABLESPACE relational_users;

CREATE USER Profile IDENTIFIED BY <password-for-Profile>
                        DEFAULT TABLESPACE relational_users;
```

Grant RELATIONAL USERS

```
GRANT connect, resource, create table TO [Document, Review,
Action, Risk, NonConformance, Role, Profile];

GRANT REFERENCES on [Document, Review, Action, Risk,
NonConformance, Role, Profile] to [Document, Review, Action, Risk,
NonConformance, Role, Profile];
```

RDF Network

```
EXECUTE sem_apis.create_sem_network('RDF_USERS');
```

RDF Users

```
CREATE USER rdfusr IDENTIFIED BY <password-for-rdfusr> DEFAULT
TABLESPACE rdf_users;
```

Grant RDF USERS

```
GRANT connect, resource TO rdfusr;
```

Appendix B

Oracle RDF Table, Model and Index Script

```
/*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* RDF Table, Model and Index                                   */
/*==============================================================*/
```

Oracle RDF Table

```
CREATE TABLE role_rdf_data (id NUMBER, triple    SDO_RDF_TRIPLE_S)
tablespace rdf_users;

CREATE TABLE user_rdf_data (id NUMBER, triple    SDO_RDF_TRIPLE_S)
tablespace rdf_users;

CREATE TABLE project_rdf_data (id NUMBER, triple SDO_RDF_TRIPLE_S)
tablespace rdf_users;

CREATE TABLE profile_rdf_data (id NUMBER,   triple
SDO_RDF_TRIPLE_S) tablespace rdf_users;

CREATE TABLE document_rdf_data (id NUMBER, triple
SDO_RDF_TRIPLE_S) tablespace rdf_users;

CREATE TABLE review_rdf_data (id NUMBER, triple SDO_RDF_TRIPLE_S)
tablespace rdf_users;

CREATE TABLE action_rdf_data (id NUMBER, triple SDO_RDF_TRIPLE_S)
tablespace rdf_users;

CREATE TABLE risk_rdf_data (id NUMBER, triple SDO_RDF_TRIPLE_S)
tablespace rdf_users;

CREATE TABLE nonconformance_rdf_data (id NUMBER, triple
SDO_RDF_TRIPLE_S) tablespace rdf_users;
```

Oracle RDF Model

```
EXECUTE SEM_APIS.CREATE_SEM_MODEL('role', 'role_rdf_data',
'triple', 'rdf_users');

EXECUTE SEM_APIS.CREATE_SEM_MODEL('user', 'user_rdf_data',
'triple','rdf_users');

EXECUTE SEM_APIS.CREATE_SEM_MODEL('project', 'project_rdf_data',
'triple', 'rdf_users');

EXECUTE SEM_APIS.CREATE_SEM_MODEL('PROFILE', 'profile_rdf_data',
'triple', 'rdf_users');
EXECUTE SEM_APIS.CREATE_SEM_MODEL('document', 'document_rdf_data',
'triple', 'rdf_users');

EXECUTE SEM_APIS.CREATE_SEM_MODEL('review', 'review_rdf_data',
'triple', 'rdf_users');

EXECUTE SEM_APIS.CREATE_SEM_MODEL('action', 'action_rdf_data',
'triple', 'rdf_users');

EXECUTE SEM_APIS.CREATE_SEM_MODEL('risk', 'risk_rdf_data',
'triple', 'rdf_users');

EXECUTE SEM_APIS.CREATE_SEM_MODEL('nonconformance',
'nonconformance_rdf_data', 'triple', 'rdf_users');
```

RDF Indexes

```
CREATE INDEX role_sub_idx ON role_rdf_data (triple.GET_SUBJECT());
CREATE INDEX role_prop_idx ON role_rdf_data
(triple.GET_PROPERTY());
CREATE INDEX role_obj_idx ON role_rdf_data
(TO_CHAR(triple.GET_OBJECT()));

CREATE INDEX user_sub_idx ON user_rdf_data (triple.GET_SUBJECT());
CREATE INDEX user_prop_idx ON user_rdf_data
(triple.GET_PROPERTY());
CREATE INDEX user_obj_idx ON user_rdf_data
(TO_CHAR(triple.GET_OBJECT()));

CREATE INDEX project_sub_idx ON project_rdf_data
(triple.GET_SUBJECT());
CREATE INDEX project_prop_idx ON project_rdf_data
(triple.GET_PROPERTY());
CREATE INDEX project_obj_idx ON project_rdf_data
(TO_CHAR(triple.GET_OBJECT()));

CREATE INDEX profile_sub_idx ON profile_rdf_data
(triple.GET_SUBJECT());
CREATE INDEX profile_prop_idx ON profile_rdf_data
(triple.GET_PROPERTY());
CREATE INDEX profile_obj_idx ON profile_rdf_data
(TO_CHAR(triple.GET_OBJECT()));
```

```
CREATE INDEX document_sub_idx ON document_rdf_data
(triple.GET_SUBJECT());
CREATE INDEX document_prop_idx ON document_rdf_data
(triple.GET_PROPERTY());
CREATE INDEX document_obj_idx ON document_rdf_data
(TO_CHAR(triple.GET_OBJECT()));

CREATE INDEX review_sub_idx ON review_rdf_data
(triple.GET_SUBJECT());
CREATE INDEX review_prop_idx ON review_rdf_data
(triple.GET_PROPERTY());
CREATE INDEX review_obj_idx ON review_rdf_data
(TO_CHAR(triple.GET_OBJECT()));

CREATE INDEX action_sub_idx ON action_rdf_data
(triple.GET_SUBJECT());
CREATE INDEX action_prop_idx ON action_rdf_data
(triple.GET_PROPERTY());
CREATE INDEX action_obj_idx ON action_rdf_data
(TO_CHAR(triple.GET_OBJECT()));

CREATE INDEX risk_sub_idx ON risk_rdf_data(triple.GET_SUBJECT());
CREATE INDEX risk_prop_idx ON
risk_rdf_data(triple.GET_PROPERTY());
CREATE INDEX risk_obj_idx ON risk_rdf_data
(TO_CHAR(triple.GET_OBJECT()));

CREATE INDEX nonconformance_sub_idx ON nonconformance_rdf_data
(triple.GET_SUBJECT());
CREATE INDEX nonconformance_prop_idx ON nonconformance_rdf_data
(triple.GET_PROPERTY());
CREATE INDEX nonconformance_obj_idx ON nonconformance_rdf_data
(TO_CHAR(triple.GET_OBJECT()));

/* ........REPEAT FOR THE REST OF RDF TABLES              */
```

Appendix C

Oracle Relational and Ontological Database Object Creation Scripts

```
/*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* ROLE Relational and Ontological Objects*/
/*==============================================================*/
/*ROLE*/
/*************************ROLE Start**************************/

create table ROLE  (
   ROLEID                NUMBER                          not null,
   ROLENAME             VARCHAR2(50),
   constraint PK_ROLE primary key (ROLEID)
);

*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* USER Relational and Ontological Objects*/
/*==============================================================*/
create table USERPROFILE (
   USERID                NUMBER                          not null,
   USERNAME             VARCHAR2(50),
   USERROLE            VARCHAR2(50),
   USERManager          VARCHAR2(50),
   USERSupervisor       VARCHAR2(50),
   constraint PK_USER primary key (USERID)
);
/*******************************Role End ***********************/
```

*/*Document*/*

```
/************************Document
Start*************************/
/*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* Document Relational and Ontological Objects*/
/*==============================================================*/


alter table BELONGTOCOLLECTION
   drop constraint FK_BELONGTO_REFERENCE_DOCUMENT;

alter table BELONGTOCOLLECTION
   drop constraint FK_BELONGTO_REFERENCE_COLLECTI;

alter table COLLECTION
   drop constraint FK_COLLECTON_DOCUMENT;

alter table DOCUMENT
   drop constraint FK_DOCUMENT_REFERENCE_ATTACHME;
```

```
alter table HASATTACHMENT
   drop constraint FK_HASATTAC_REFERENCE_DOCUMENT;

alter table HASATTACHMENT
   drop constraint FK_HASATTAC_REFERENCE_ATTACHME;

drop table ATTACHMENT cascade constraints;

drop table ATTACHMENTONTOLOGY cascade constraints;

drop table BELONGTOCOLLECTION cascade constraints;

drop table COLLECTION cascade constraints;

drop table COLLECTIONONTOLOGY cascade constraints;

drop table DOCUMENT cascade constraints;

drop table DOCUMENTONTOLOGY cascade constraints;

drop table HASATTACHMENT cascade constraints;

/*==============================================================*/
/* Table: ATTACHMENT                                            */
/*==============================================================*/
create table ATTACHMENT  (
   ATTACHMENTID         NUMBER                          not null,
   ATTACHMENTTITLE      VARCHAR2(50),
   constraint PK_ATTACHMENT primary key (ATTACHMENTID)
);

/*==============================================================*/
/* Table: ATTACHMENTONTOLOGY                                    */
/*==============================================================*/
create table ATTACHMENTONTOLOGY  (
   ATTACHMENTURI        varchar2(100)                   not null,
   constraint PK_ATTACHMENTONTOLOGY primary key (ATTACHMENTURI)
);

/*==============================================================*/
/* Table: BELONGTOCOLLECTION                                    */
/*==============================================================*/
create table BELONGTOCOLLECTION  (
   DOCUMENTURI          varchar2(100),
   COLLECTIONURI        varchar2(100)
);

/*==============================================================*/
/* Table: COLLECTION                                            */
/*==============================================================*/
create table COLLECTION  (
   COLLECTIONID         NUMBER                          not null,
   DOCUMENTID           NUMBER,
   constraint PK_COLLECTION primary key (COLLECTIONID)
);
```

```
/*==============================================================*/
/* Table: COLLECTIONONTOLOGY                                    */
/*==============================================================*/
create table COLLECTIONONTOLOGY  (
   COLLECTIONURI        varchar2(100)                  not null,
   constraint PK_COLLECTIONONTOLOGY primary key (COLLECTIONURI)
);

/*==============================================================*/
/* Table: DOCUMENT                                              */
/*==============================================================*/
create table DOCUMENT  (
   DOCUMENTID           NUMBER                         not null,
   ATTACHMENTID         NUMBER,
   DOCUMENTTITLE        VARCHAR2(50),
   CREATEDBY            VARCHAR2(16),
   CREATEDDATE          DATE,
   constraint PK_DOCUMENT primary key (DOCUMENTID)
);

/*==============================================================*/
/* Table: DOCUMENTONTOLOGY                                      */
/*==============================================================*/
create table DOCUMENTONTOLOGY  (
   DOCUMENTURI          varchar2(100)                  not null,
   constraint PK_DOCUMENTONTOLOGY primary key (DOCUMENTURI)
);

/*==============================================================*/
/* Table: HASATTACHMENT                                         */
/*==============================================================*/
create table HASATTACHMENT  (
   DOCUMENTURI          varchar2(100),
   ATTACHMENTURI        varchar2(100)
);

alter table BELONGTOCOLLECTION
   add constraint FK_BELONGTO_REFERENCE_DOCUMENT foreign key
(DOCUMENTURI)
      references DOCUMENTONTOLOGY (DOCUMENTURI);

alter table BELONGTOCOLLECTION
   add constraint FK_BELONGTO_REFERENCE_COLLECTI foreign key
(COLLECTIONURI)
      references COLLECTIONONTOLOGY (COLLECTIONURI);

alter table COLLECTION
   add constraint FK_COLLECTON_DOCUMENT foreign key (DOCUMENTID)
      references DOCUMENT (DOCUMENTID);

alter table DOCUMENT
   add constraint FK_DOCUMENT_REFERENCE_ATTACHME foreign key
(ATTACHMENTID)
      references ATTACHMENT (ATTACHMENTID);

alter table HASATTACHMENT
```

```
   add constraint FK_HASATTAC_REFERENCE_DOCUMENT foreign key
(DOCUMENTURI)
      references DOCUMENTONTOLOGY (DOCUMENTURI);

alter table HASATTACHMENT
   add constraint FK_HASATTAC_REFERENCE_ATTACHME foreign key
(ATTACHMENTURI)
      references ATTACHMENTONTOLOGY (ATTACHMENTURI);
/**************************Document End**************************/
/*Review*/
/**************************Review Start**************************/
/*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* Review Relational and Ontological Objects*/
/*==============================================================*/


alter table CLASSIFICATION
   drop constraint FK_CLASSIFI_REFERENCE_REVIEW;

alter table EDITORIAL
   drop constraint FK_EDITORIA_REFERENCE_REVIEW;

alter table HASCLASSIFICATION
   drop constraint FK_HASCLASS_REFERENCE_REVIEWON;

alter table HASCLASSIFICATION
   drop constraint FK_HASCLASS_REFERENCE_CLASSIFI;

alter table HASEDITORIAL
   drop constraint FK_HASEDITO_REFERENCE_REVIEWON;

alter table HASEDITORIAL
   drop constraint FK_HASEDITO_REFERENCE_EDITORIA;

alter table HASPANEL
   drop constraint FK_HASPANEL_REFERENCE_REVIEWON;

alter table HASPANEL
   drop constraint FK_HASPANEL_REFERENCE_PANELONT;

alter table PANEL
   drop constraint FK_PANEL_REFERENCE_REVIEW;

alter table REVIEW
   drop constraint FK_REVIEW_DOCUMENT;

alter table REVIEW
   drop constraint FK_REVIEW_PROJECT;

drop table CLASSIFICATION cascade constraints;

drop table CLASSIFICATIONONTOLOGY cascade constraints;

drop table DOCUMENT cascade constraints;
```

```
drop table EDITORIAL cascade constraints;

drop table EDITORIALONTOLOGY cascade constraints;

drop table HASCLASSIFICATION cascade constraints;

drop table HASEDITORIAL cascade constraints;

drop table HASPANEL cascade constraints;

drop table PANEL cascade constraints;

drop table PANELONTOLOGY cascade constraints;

drop table PROJECT cascade constraints;

drop table REVIEW cascade constraints;

drop table REVIEWONTOLOGY cascade constraints;

create cluster  C_CLASSIFICATION (
   CLASSID       number
);

/*==============================================================*/
/* Table: CLASSIFICATION                                        */
/*==============================================================*/
create table CLASSIFICATION  (
   CLASSID              number                          not null,
   CLASSNAME            varchar2(20),
   REVIEWID             number,
   constraint PK_CLASSIFICATION primary key (CLASSID)
);

/*==============================================================*/
/* Table: CLASSIFICATIONONTOLOGY                                */
/*==============================================================*/
create table CLASSIFICATIONONTOLOGY  (
   CLASSIFICATIONURI    varchar2(100)                   not null,
   constraint PK_CLASSIFICATIONONTOLOGY primary key
(CLASSIFICATIONURI)
);

/*==============================================================*/
/* Table: DOCUMENT                                              */
/*==============================================================*/
create table DOCUMENT  (
   DOCUMENTID           NUMBER                          not null,
   DOCUMENTTITLE        VARCHAR2(50),
   CREATEDBY            VARCHAR2(16),
   CREATEDDATE          DATE,
   constraint PK_DOCUMENT primary key (DOCUMENTID)
);

create cluster  C_EDITORIAL (
   EDITORIALID        number
```

```
);

/*==============================================================*/
/* Table: EDITORIAL                                             */
/*==============================================================*/
create table EDITORIAL  (
   EDITORIALID          number                          not null,
   EDITORIALNAME        varchar2(20),
   REVIEWID             number,
   constraint PK_EDITORIAL primary key (EDITORIALID)
);

create cluster  C_EDITORIALONTOLOGY (
   EDITORIALURI       varchar2 (100)
);

/*==============================================================*/
/* Table: EDITORIALONTOLOGY                                     */
/*==============================================================*/
create table EDITORIALONTOLOGY  (
   EDITORIALURI         varchar2(100)                   not null,
   constraint PK_EDITORIALONTOLOGY primary key (EDITORIALURI)
);

/*==============================================================*/
/* Table: HASCLASSIFICATION                                     */
/*==============================================================*/
create table HASCLASSIFICATION  (
   REVIEWURI            varchar2(100)                   not null,
   CLASSIFICATIONURI    varchar2(100)                   not null,
   constraint PK_HASCLASSIFICATION primary key (REVIEWURI,
CLASSIFICATIONURI)
);

/*==============================================================*/
/* Table: HASEDITORIAL                                          */
/*==============================================================*/
create table HASEDITORIAL  (
   REVIEWURI            varchar2(100)                   not null,
   EITORIALURI          varchar2(100)                   not null,
   constraint PK_HASEDITORIAL primary key (REVIEWURI, EITORIALURI)
);

/*==============================================================*/
/* Table: HASPANEL                                              */
/*==============================================================*/
create table HASPANEL  (
   REVIEWURI            varchar2(100)                   not null,
   PANELURI             varchar2(100)                   not null,
   constraint PK_HASPANEL primary key (REVIEWURI, PANELURI)
);

create cluster  C_PANEL (
   PANELID       number
);
```

```
/*==============================================================*/
/* Table: PANEL                                                 */
/*==============================================================*/
create table PANEL  (
   PANELID              number                          not null,
   PANELNAME            varchar2(20),
   REVIEWID             number,
   constraint PK_PANEL primary key (PANELID)
);

create cluster  C_PANELONTOLOGY (
   PANELURI           varchar2(100)
);

/*==============================================================*/
/* Table: PANELONTOLOGY                                         */
/*==============================================================*/
create table PANELONTOLOGY  (
   PANELURI             varchar2(100)                   not null,
   constraint PK_PANELONTOLOGY primary key (PANELURI)
);

/*==============================================================*/
/* Table: PROJECT                                               */
/*==============================================================*/
create table PROJECT  (
   PROJECTID            NUMBER                          not null,
   PROJECTNAME          VARCHAR2(20),
   constraint PK_PROJECT primary key (PROJECTID)
);

/*==============================================================*/
/* Table: REVIEW                                                */
/*==============================================================*/
create table REVIEW  (
   REVIEWID             number                          not null,
   REVIEWTITLE          varchar2(50),
   ROLEID               number,
   DOCUMENTID           number,
   PRO_ID               NUMBER,
   constraint PK_REVIEW primary key (REVIEWID)
);

create cluster  C_REVIEWONTOLOGY (
   REVIEWURI          varchar2(100)
);

/*==============================================================*/
/* Table: REVIEWONTOLOGY                                        */
/*==============================================================*/
create table REVIEWONTOLOGY  (
   REVIEWURI            varchar2(100)                   not null,
   constraint PK_REVIEWONTOLOGY primary key (REVIEWURI)
);

alter table CLASSIFICATION
```

```
   add constraint FK_CLASSIFI_REFERENCE_REVIEW foreign key
(REVIEWID)
      references REVIEW (REVIEWID);

alter table EDITORIAL
   add constraint FK_EDITORIA_REFERENCE_REVIEW foreign key
(REVIEWID)
      references REVIEW (REVIEWID);

alter table HASCLASSIFICATION
   add constraint FK_HASCLASS_REFERENCE_REVIEWON foreign key
(REVIEWURI)
      references REVIEWONTOLOGY (REVIEWURI);

alter table HASCLASSIFICATION
   add constraint FK_HASCLASS_REFERENCE_CLASSIFI foreign key
(CLASSIFICATIONURI)
      references CLASSIFICATIONONTOLOGY (CLASSIFICATIONURI);

alter table HASEDITORIAL
   add constraint FK_HASEDITO_REFERENCE_REVIEWON foreign key
(REVIEWURI)
      references REVIEWONTOLOGY (REVIEWURI);

alter table HASEDITORIAL
   add constraint FK_HASEDITO_REFERENCE_EDITORIA foreign key
(EITORIALURI)
      references EDITORIALONTOLOGY (EDITORIALURI);

alter table HASPANEL
   add constraint FK_HASPANEL_REFERENCE_REVIEWON foreign key
(REVIEWURI)
      references REVIEWONTOLOGY (REVIEWURI);

alter table HASPANEL
   add constraint FK_HASPANEL_REFERENCE_PANELONT foreign key
(PANELURI)
      references PANELONTOLOGY (PANELURI);

alter table PANEL
   add constraint FK_PANEL_REFERENCE_REVIEW foreign key (REVIEWID)
      references REVIEW (REVIEWID);
alter table REVIEW
   add constraint FK_REVIEW_ROLE foreign key (ROLEID)
      references ROLE.ROLE(ROLEID);

alter table REVIEW
   add constraint FK_REVIEW_DOCUMENT foreign key (REVIEWID)
      references DOCUMENT (DOCUMENTID);

alter table REVIEW
   add constraint FK_REVIEW_PROJECT foreign key (PRO_ID)
      references PROJECT (PROJECTID);
/**************************Review End**************************/
/*Action*/
```

```
/*************************Action Start*****************************/
/*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* Action Relational and Ontological Objects*/
/*==============================================================*/


alter table HASINCIDENT
   drop constraint FK_HASINCID_REFERENCE_INCIDENT;

alter table HASINCIDENT
   drop constraint FK_HASINCID_REFERENCE_ACTIONON;

alter table HASPRIORITY
   drop constraint FK_HASPRIOR_REFERENCE_ACTIONON;

alter table HASPRIORITY
   drop constraint FK_HASPRIOR_REFERENCE_PRIORITY;

alter table HASRESOLUTION
   drop constraint FK_HASRESOL_REFERENCE_ACTIONON;

alter table HASRESOLUTION
   drop constraint FK_HASRESOL_REFERENCE_RESOLUTI;

alter table HASSTATUS
   drop constraint FK_HASSTATU_REFERENCE_ACTIONON;

alter table HASSTATUS
   drop constraint FK_HASSTATU_REFERENCE_STATUSON;

alter table INCIDENT
   drop constraint FK_INCIDENT_REFERENCE_ACTION;

alter table PRIORITY
   drop constraint FK_PRIORITY_REFERENCE_ACTION;

alter table RESOLUTION
   drop constraint FK_RESOLUTI_REFERENCE_ACTION;

alter table STATUS
   drop constraint FK_STATUS_REFERENCE_ACTION;

drop table ACTION cascade constraints;

drop table ACTIONONTOLOGY cascade constraints;

drop table HASINCIDENT cascade constraints;

drop table HASPRIORITY cascade constraints;

drop table HASRESOLUTION cascade constraints;

drop table HASSTATUS cascade constraints;

drop table INCIDENT cascade constraints;
```

```
drop table INCIDENTONTOLOGY cascade constraints;

drop table PRIORITY cascade constraints;

drop table PRIORITYONTOLOGY cascade constraints;

drop table RESOLUTION cascade constraints;

drop table RESOLUTIONONTOLOGY cascade constraints;

drop table STATUS cascade constraints;

drop table STATUSONTOLOGY cascade constraints;

/*==============================================================*/
/* Table: ACTION                                                */
/*==============================================================*/
create table ACTION  (
   ACTIONID             NUMBER                          not null,
   ACTIONTITLE          VARCHAR2(50),
   PRO_ID               NUMBER,
   constraint PK_ACTION primary key (ACTIONID)
);

create cluster  C_ACTIONONTOLOGY (
   ACTIONURI          varchar2(100)
);

/*==============================================================*/
/* Table: ACTIONONTOLOGY                                        */
/*==============================================================*/
create table ACTIONONTOLOGY  (
   ACTIONURI            varchar2(100)                   not null,
   constraint PK_ACTIONONTOLOGY primary key (ACTIONURI)
);

/*==============================================================*/
/* Table: HASINCIDENT                                           */
/*==============================================================*/
create table HASINCIDENT  (
   ACTIONURI            varchar2(100),
   INCIDENTURI          varchar2(100)
);

/*==============================================================*/
/* Table: HASPRIORITY                                           */
/*==============================================================*/
create table HASPRIORITY  (
   ACTIONURI            varchar2(100),
   PRIORITYURI          varchar2(100)
);

/*==============================================================*/
/* Table: HASRESOLUTION                                         */
/*==============================================================*/
```

```
create table HASRESOLUTION  (
   ACTIONURI            varchar2(100),
   RESOLUTIONURI        varchar2(100)
);

/*==============================================================*/
/* Table: HASSTATUS                                             */
/*==============================================================*/
create table HASSTATUS  (
   ACTIONURI            varchar2(100),
   STATUSURI            varchar2(100)
);

create cluster  C_INCIDENT (
   INCIDENTID         number
);

/*==============================================================*/
/* Table: INCIDENT                                              */
/*==============================================================*/
create table INCIDENT  (
   INCIDENTID           number                          not null,
   INCIDENTNAME         varchar2(20),
   ACTIONID             NUMBER,
   constraint PK_INCIDENT primary key (INCIDENTID)
);

create cluster  C_INCIDENTONTOLOGY (
   INCIDENTURI        varchar2(100)
);

/*==============================================================*/
/* Table: INCIDENTONTOLOGY                                      */
/*==============================================================*/
create table INCIDENTONTOLOGY  (
   INCIDENTURI          varchar2(100)                   not null,
   constraint PK_INCIDENTONTOLOGY primary key (INCIDENTURI)
);

create cluster  C_PRIORITY (
   PRIORITYID         number
);

/*==============================================================*/
/* Table: PRIORITY                                              */
/*==============================================================*/
create table PRIORITY  (
   PRIORITYID           number                          not null,
   PRIORITYNAME         varchar2(20),
   ACTIONID             NUMBER,
   constraint PK_PRIORITY primary key (PRIORITYID)
);

create cluster  C_PRIORITYONTOLOGY (
   PRIORITYURI        varchar2(100)
);
```

```
/*==============================================================*/
/* Table: PRIORITYONTOLOGY                                      */
/*==============================================================*/
create table PRIORITYONTOLOGY  (
   PRIORITYURI          varchar2(100)                  not null,
   constraint PK_PRIORITYONTOLOGY primary key (PRIORITYURI)
);

create cluster  C_RESOLUTION (
   RESOLUTIONID       number
);

/*==============================================================*/
/* Table: RESOLUTION                                            */
/*==============================================================*/
create table RESOLUTION  (
   RESOLUTIONID         number                         not null,
   RESOLUTIONNAME       varchar2(20),
   ACTIONID             NUMBER,
   constraint PK_RESOLUTION primary key (RESOLUTIONID)
);

create cluster  C_RESOLUTIONONTOLOGY (
   RESOLUTIONURI       varchar2(100)
);

/*==============================================================*/
/* Table: RESOLUTIONONTOLOGY                                    */
/*==============================================================*/
create table RESOLUTIONONTOLOGY  (
   RESOLUTIONURI        varchar2(100)                  not null,
   constraint PK_RESOLUTIONONTOLOGY primary key (RESOLUTIONURI)
);

create cluster  C_STATUS (
   STATUSID           number
);

/*==============================================================*/
/* Table: STATUS                                                */
/*==============================================================*/
create table STATUS  (
   STATUSID             number                         not null,
   STATUSNAME           varchar2(20),
   ACTIONID             NUMBER,
   constraint PK_STATUS primary key (STATUSID)
);

create cluster  C_STATUSONTOLOGY (
   STATUSURI          varchar2(100)
);

/*==============================================================*/
/* Table: STATUSONTOLOGY                                        */
/*==============================================================*/
```

```
create table STATUSONTOLOGY  (
   STATUSURI            varchar2(100)                  not null,
   constraint PK_STATUSONTOLOGY primary key (STATUSURI)
);

alter table HASINCIDENT
   add constraint FK_HASINCID_REFERENCE_INCIDENT foreign key
(INCIDENTURI)
      references INCIDENTONTOLOGY (INCIDENTURI);

alter table HASINCIDENT
   add constraint FK_HASINCID_REFERENCE_ACTIONON foreign key
(ACTIONURI)
      references ACTIONONTOLOGY (ACTIONURI);

alter table HASPRIORITY
   add constraint FK_HASPRIOR_REFERENCE_ACTIONON foreign key
(ACTIONURI)
      references ACTIONONTOLOGY (ACTIONURI);

alter table HASPRIORITY
   add constraint FK_HASPRIOR_REFERENCE_PRIORITY foreign key
(PRIORITYURI)
      references PRIORITYONTOLOGY (PRIORITYURI);

alter table HASRESOLUTION
   add constraint FK_HASRESOL_REFERENCE_ACTIONON foreign key
(ACTIONURI)
      references ACTIONONTOLOGY (ACTIONURI);

alter table HASRESOLUTION
   add constraint FK_HASRESOL_REFERENCE_RESOLUTI foreign key
(RESOLUTIONURI)
      references RESOLUTIONONTOLOGY (RESOLUTIONURI);

alter table HASSTATUS
   add constraint FK_HASSTATU_REFERENCE_ACTIONON foreign key
(ACTIONURI)
      references ACTIONONTOLOGY (ACTIONURI);

alter table HASSTATUS
   add constraint FK_HASSTATU_REFERENCE_STATUSON foreign key
(STATUSURI)
      references STATUSONTOLOGY (STATUSURI);

alter table INCIDENT
   add constraint FK_INCIDENT_REFERENCE_ACTION foreign key
(ACTIONID)
      references ACTION (ACTIONID);

alter table PRIORITY
   add constraint FK_PRIORITY_REFERENCE_ACTION foreign key
(ACTIONID)
      references ACTION (ACTIONID);

alter table RESOLUTION
```

```
   add constraint FK_RESOLUTI_REFERENCE_ACTION foreign key
(ACTIONID)
      references ACTION (ACTIONID);

alter table STATUS
   add constraint FK_STATUS_REFERENCE_ACTION foreign key
(ACTIONID)
      references ACTION (ACTIONID);

/*************************Action End*******************************/
/*Risk*/

/*****************************Risk Start*************************/
/*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* Risk Relational and Ontological Objects*/
/*==============================================================*/

alter table CRITICALITY
   drop constraint FK_CRITICAL_REFERENCE_RISK;

alter table CRITICALITY
   drop constraint FK_CRITICAL_REFERENCE_SEVERITY;

alter table CRITICALITY
   drop constraint FK_CRITICAL_REFERENCE_LIKELIHO;

alter table DOMAIN
   drop constraint FK_DOMAIN_REFERENCE_RISK;

alter table HASCRITICALITY
   drop constraint FK_HASCRITI_REFERENCE_RISKONTO;

alter table HASCRITICALITY
   drop constraint FK_HASCRITI_REFERENCE_CRITICAL;

alter table HASDOMAIN
   drop constraint FK_HASDOMAI_REFERENCE_RISKONTO;

alter table HASDOMAIN
   drop constraint FK_HASDOMAI_REFERENCE_DOMAINON;

alter table HASLIKELIHOOD
   drop constraint FK_HASLIKEL_REFERENCE_CRITICAL;

alter table HASLIKELIHOOD
   drop constraint FK_HASLIKEL_REFERENCE_LIKELIHO;

alter table HASRANK
   drop constraint FK_HASRANK_REFERENCE_RISKONTO;

alter table HASRANK
   drop constraint FK_HASRANK_REFERENCE_RANKONTO;

alter table HASSEVERITY
   drop constraint FK_HASSEVER_REFERENCE_CRITICAL;
```

```
alter table HASSEVERITY
   drop constraint FK_HASSEVER_REFERENCE_SEVERITY;

alter table HASTRAIL
   drop constraint FK_HASTRAIL_REFERENCE_RISKONTO;

alter table HASTRAIL
   drop constraint FK_HASTRAIL_REFERENCE_TRAILONT;

alter table OFSCENARIO
   drop constraint FK_OFSCENAR_REFERENCE_RISKONTO;

alter table OFSCENARIO
   drop constraint FK_OFSCENAR_REFERENCE_SCENARIO;

alter table RANK
   drop constraint FK_RANK_REFERENCE_RISK;

alter table SCENARIO
   drop constraint FK_SCENARIO_REFERENCE_RISK;

alter table TRAIL
   drop constraint FK_TRAIL_REFERENCE_RISK;

drop table CRITICALITY cascade constraints;

drop table CRITICALITYONTOLOGY cascade constraints;

drop table DOMAIN cascade constraints;

drop table DOMAINONTOLOGY cascade constraints;

drop table HASCRITICALITY cascade constraints;

drop table HASDOMAIN cascade constraints;

drop table HASLIKELIHOOD cascade constraints;

drop table HASRANK cascade constraints;

drop table HASSEVERITY cascade constraints;

drop table HASTRAIL cascade constraints;

drop table LIKELIHOOD cascade constraints;

drop table LIKELIHOODONTOLOGY cascade constraints;

drop table OFSCENARIO cascade constraints;

drop table RANK cascade constraints;

drop table RANKONTOLOGY cascade constraints;

drop table RISK cascade constraints;
```

```
drop table RISKONTOLOGY cascade constraints;

drop table SCENARIO cascade constraints;

drop table SCENARIOONTOLOGY cascade constraints;

drop table SEVERITY cascade constraints;

drop table SEVERITYONTOLOGY cascade constraints;

drop table TRAIL cascade constraints;

drop table TRAILONTOLOGY cascade constraints;

create cluster  C_CRITICALITY (
   CRITICALITYID      number
);

/*==============================================================*/
/* Table: CRITICALITY                                           */
/*==============================================================*/
create table CRITICALITY  (
   CRITICALITYID        number                          not null,
   CRITICALITYNAME      varchar2(20),
   RISKID               NUMBER,
   SEVERITYID           number,
   LIKELIHOODID         number,
   constraint PK_CRITICALITY primary key (CRITICALITYID)
);

/*==============================================================*/
/* Table: CRITICALITYONTOLOGY                                   */
/*==============================================================*/
create table CRITICALITYONTOLOGY  (
   CRITICALITYURI       varchar2(100)                   not null,
   constraint PK_CRITICALITYONTOLOGY primary key (CRITICALITYURI)
);

create cluster  C_DOMAIN (
   DOMAINID           number
);

/*==============================================================*/
/* Table: DOMAIN                                                */
/*==============================================================*/
create table DOMAIN  (
   DOMAINID             number                          not null,
   DOMAINNAME           varchar2(20),
   RISKID               NUMBER,
   constraint PK_DOMAIN primary key (DOMAINID)
);

create cluster  C_DOMAINONTOLOGY (
   DOMAINURI          varchar2(100)
);
```

```
/*==============================================================*/
/* Table: DOMAINONTOLOGY                                        */
/*==============================================================*/
create table DOMAINONTOLOGY  (
   DOMAINURI            varchar2(100)                  not null,
   constraint PK_DOMAINONTOLOGY primary key (DOMAINURI)
);

/*==============================================================*/
/* Table: HASCRITICALITY                                        */
/*==============================================================*/
create table HASCRITICALITY  (
   RISKURI              varchar2(100),
   CRITICALITYURI       varchar2(100)
);

/*==============================================================*/
/* Table: HASDOMAIN                                             */
/*==============================================================*/
create table HASDOMAIN  (
   RISKURI              varchar2(100),
   DOMAINURI            varchar2(100)
);

/*==============================================================*/
/* Table: HASLIKELIHOOD                                         */
/*==============================================================*/
create table HASLIKELIHOOD  (
   CRITICALITYURI       varchar2(100),
   LIKELIHOODURI        varchar2(100)
);

/*==============================================================*/
/* Table: HASRANK                                               */
/*==============================================================*/
create table HASRANK  (
   RISKURI              varchar2(100),
   RANKURI              varchar2(100)
);

/*==============================================================*/
/* Table: HASSEVERITY                                           */
/*==============================================================*/
create table HASSEVERITY  (
   CRITICALITYURI       varchar2(100),
   SEVERITYURI          varchar2(100)
);

/*==============================================================*/
/* Table: HASTRAIL                                              */
/*==============================================================*/
create table HASTRAIL  (
   RISKURI              varchar2(100),
   TRAILURI             varchar2(100)
);
```

```
create cluster  C_LIKELIHOOD (
   LIKELIHOODID       number
);

/*==============================================================*/
/* Table: LIKELIHOOD                                            */
/*==============================================================*/
create table LIKELIHOOD  (
   LIKELIHOODID         number                          not null,
   LIKELIHOODNAME       varchar2(20),
   constraint PK_LIKELIHOOD primary key (LIKELIHOODID)
);

/*==============================================================*/
/* Table: LIKELIHOODONTOLOGY                                    */
/*==============================================================*/
create table LIKELIHOODONTOLOGY  (
   LIKELIHOODURI        varchar2(100)                   not null,
   constraint PK_LIKELIHOODONTOLOGY primary key (LIKELIHOODURI)
);

/*==============================================================*/
/* Table: OFSCENARIO                                            */
/*==============================================================*/
create table OFSCENARIO  (
   RISKURI              varchar2(100),
   SCENARIOURI          varchar2(100)
);

create cluster  C_RANK (
   RANKID       number
);

/*==============================================================*/
/* Table: RANK                                                  */
/*==============================================================*/
create table RANK  (
   RANKID               number                          not null,
   RANKNAME             varchar2(20),
   RISKID               NUMBER,
   constraint PK_RANK primary key (RANKID)
);

create cluster  C_RANKONTOLOGY (
   RANKURI      varchar2(100)
);

/*==============================================================*/
/* Table: RANKONTOLOGY                                          */
/*==============================================================*/
create table RANKONTOLOGY  (
   RANKURI              varchar2(100)                   not null,
   constraint PK_RANKONTOLOGY primary key (RANKURI)
);
```

```
/*==============================================================*/
/* Table: RISK                                                  */
/*==============================================================*/
create table RISK  (
   RISKID               NUMBER                          not null,
   RISKTITLE            VARCHAR2(50),
   PRO_ID               NUMBER,
   constraint PK_RISK primary key (RISKID)
);

create cluster  C_RISKONTOLOGY (
   RISKURI       varchar2(100)
);

/*==============================================================*/
/* Table: RISKONTOLOGY                                          */
/*==============================================================*/
create table RISKONTOLOGY  (
   RISKURI              varchar2(100)                   not null,
   constraint PK_RISKONTOLOGY primary key (RISKURI)
);

create cluster  C_SCENARIO (
   SCENARIOID          number
);

/*==============================================================*/
/* Table: SCENARIO                                              */
/*==============================================================*/
create table SCENARIO  (
   SCENARIOID           number                          not null,
   SCENARIONAME         varchar2(20),
   RISKID               NUMBER,
   constraint PK_SCENARIO primary key (SCENARIOID)
);

create cluster  C_SCENARIOONTOLOGY (
   SCENARIOURI         varchar2(100)
);

/*==============================================================*/
/* Table: SCENARIOONTOLOGY                                      */
/*==============================================================*/
create table SCENARIOONTOLOGY  (
   SCENARIOURI          varchar2(100)                   not null,
   constraint PK_SCENARIOONTOLOGY primary key (SCENARIOURI)
);

create cluster  C_SEVERITY (
   SEVERITYID          number
);

/*==============================================================*/
/* Table: SEVERITY                                              */
/*==============================================================*/
create table SEVERITY  (
```

```
   SEVERITYID           number                                  not null,
   SEVERITYNAME         varchar2(20),
   constraint PK_SEVERITY primary key (SEVERITYID)
);

/*==============================================================*/
/* Table: SEVERITYONTOLOGY                                      */
/*==============================================================*/
create table SEVERITYONTOLOGY  (
   SEVERITYURI          varchar2(100)                           not null,
   constraint PK_SEVERITYONTOLOGY primary key (SEVERITYURI)
);

create cluster  C_TRAIL (
   TRAILID      number
);

/*==============================================================*/
/* Table: TRAIL                                                 */
/*==============================================================*/
create table TRAIL  (
   TRAILID              number                                  not null,
   TRAILNAME            varchar2(20),
   RISKID               NUMBER,
   constraint PK_TRAIL primary key (TRAILID)
);

create cluster  C_TRAILONTOLOGY (
   TRAILURI           varchar2(100)
);
/*==============================================================*/
/* Table: TRAILONTOLOGY                                         */
/*==============================================================*/
create table TRAILONTOLOGY  (
   TRAILURI             varchar2(100)                           not null,
   constraint PK_TRAILONTOLOGY primary key (TRAILURI)
);

alter table CRITICALITY
   add constraint FK_CRITICAL_REFERENCE_RISK foreign key (RISKID)
      references RISK (RISKID);

alter table CRITICALITY
   add constraint FK_CRITICAL_REFERENCE_SEVERITY foreign key
(SEVERITYID)
      references SEVERITY (SEVERITYID);

alter table CRITICALITY
   add constraint FK_CRITICAL_REFERENCE_LIKELIHO foreign key
(LIKELIHOODID)
      references LIKELIHOOD (LIKELIHOODID);

alter table DOMAIN
   add constraint FK_DOMAIN_REFERENCE_RISK foreign key (RISKID)
      references RISK (RISKID);
```

```
alter table HASCRITICALITY
   add constraint FK_HASCRITI_REFERENCE_RISKONTO foreign key
(RISKURI)
      references RISKONTOLOGY (RISKURI);

alter table HASCRITICALITY
   add constraint FK_HASCRITI_REFERENCE_CRITICAL foreign key
(CRITICALITYURI)
      references CRITICALITYONTOLOGY (CRITICALITYURI);

alter table HASDOMAIN
   add constraint FK_HASDOMAI_REFERENCE_RISKONTO foreign key
(RISKURI)
      references RISKONTOLOGY (RISKURI);

alter table HASDOMAIN
   add constraint FK_HASDOMAI_REFERENCE_DOMAINON foreign key
(DOMAINURI)
      references DOMAINONTOLOGY (DOMAINURI);

alter table HASLIKELIHOOD
   add constraint FK_HASLIKEL_REFERENCE_CRITICAL foreign key
(CRITICALITYURI)
      references CRITICALITYONTOLOGY (CRITICALITYURI);

alter table HASLIKELIHOOD
   add constraint FK_HASLIKEL_REFERENCE_LIKELIHO foreign key
(LIKELIHOODURI)
      references LIKELIHOODONTOLOGY (LIKELIHOODURI);

alter table HASRANK
   add constraint FK_HASRANK_REFERENCE_RISKONTO foreign key
(RISKURI)
      references RISKONTOLOGY (RISKURI);

alter table HASRANK
   add constraint FK_HASRANK_REFERENCE_RANKONTO foreign key
(RANKURI)
      references RANKONTOLOGY (RANKURI);

alter table HASSEVERITY
   add constraint FK_HASSEVER_REFERENCE_CRITICAL foreign key
(CRITICALITYURI)
      references CRITICALITYONTOLOGY (CRITICALITYURI);

alter table HASSEVERITY
   add constraint FK_HASSEVER_REFERENCE_SEVERITY foreign key
(SEVERITYURI)
      references SEVERITYONTOLOGY (SEVERITYURI);

alter table HASTRAIL
   add constraint FK_HASTRAIL_REFERENCE_RISKONTO foreign key
(RISKURI)
      references RISKONTOLOGY (RISKURI);

alter table HASTRAIL
```

```
   add constraint FK_HASTRAIL_REFERENCE_TRAILONT foreign key
(TRAILURI)
      references TRAILONTOLOGY (TRAILURI);

alter table OFSCENARIO
   add constraint FK_OFSCENAR_REFERENCE_RISKONTO foreign key
(RISKURI)
      references RISKONTOLOGY (RISKURI);

alter table OFSCENARIO
   add constraint FK_OFSCENAR_REFERENCE_SCENARIO foreign key
(SCENARIOURI)
      references SCENARIOONTOLOGY (SCENARIOURI);

alter table RANK
   add constraint FK_RANK_REFERENCE_RISK foreign key (RISKID)
      references RISK (RISKID);

alter table SCENARIO
   add constraint FK_SCENARIO_REFERENCE_RISK foreign key (RISKID)
      references RISK (RISKID);

alter table TRAIL
   add constraint FK_TRAIL_REFERENCE_RISK foreign key (RISKID)
      references RISK (RISKID);
/*********************************Risk End**********************/
/*Non Conformance*/

/**********************Non Conformance Start******************/
/*==============================================================*/
/* DBMS name:      ORACLE Version 11g                           */
/* Non Conformance Relational and Ontological Objects*/
/*==============================================================*/


alter table CATEGORY
   drop constraint FK_CATEGORY_REFERENCE_NONCONF;

alter table DISPOSITION
   drop constraint FK_DISPOSIT_REFERENCE_NONCONF;

alter table EFFECT
   drop constraint FK_EFFECT_REFERENCE_NONCONF;

alter table MODEL
   drop constraint FK_MODEL_REFERENCE_NONCONF;

alter table NONCONF
   drop constraint FK_NONCONF_PROJECT;

alter table NONCONF
   drop constraint FK_NONCONF_DOCUMENT;

drop table CATEGORY cascade constraints;

drop table DISPOSITION cascade constraints;
```

```
drop table EFFECT cascade constraints;

drop table MODEL cascade constraints;

drop table NONCONF cascade constraints;

/*==============================================================*/
/* Table: CATEGORY                                              */
/*==============================================================*/
create table CATEGORY  (
   CATEGORYID           number                          not null,
   CATEGORYNAME         varchar2(20),
   NONCID               NUMBER,
   constraint PK_CATEGORY primary key (CATEGORYID)
);


/*==============================================================*/
/* Table: DISPOSITION                                           */
/*==============================================================*/
create table DISPOSITION  (
   DISPOSITIONID        number                          not null,
   DISPOSITIONNAME      varchar2(20),
   NONCID               NUMBER,
   constraint PK_DISPOSITION primary key (DISPOSITIONID)
);

/*==============================================================*/
/* Table: EFFECT                                                */
/*==============================================================*/
create table EFFECT  (
   EFFECTID             number                          not null,
   EFFECTNAME           varchar2(20),
   NONCID               NUMBER,
   constraint PK_EFFECT primary key (EFFECTID)
);

/*==============================================================*/
/* Table: MODEL                                                 */
/*==============================================================*/
create table MODEL  (
   MODELID              number                          not null,
   MODELNAME            varchar2(20),
   NONCID               NUMBER,
   constraint PK_MODEL primary key (MODELID)
);

/*==============================================================*/
/* Table: NONCONF
*/
/*==============================================================*/
create table NONCONF  (
   NONCID               NUMBER                          not null,
   NONCTITLE            VARCHAR2(50),
   PROJECTID            NUMBER,
```

```
   DOCUMENTID           number,
   constraint PK_NONCONF primary key (NONCID)
);

alter table CATEGORY
   add constraint FK_CATEGORY_REFERENCE_NONCONF foreign key
(NONCID)
      references NONCONF (NONCID);

alter table DISPOSITION
   add constraint FK_DISPOSIT_REFERENCE_NONCONF foreign key
(NONCID)
      references NONCONF (NONCID);

alter table EFFECT
   add constraint FK_EFFECT_REFERENCE_NONCONF foreign key (NONCID)
      references NONCONF (NONCID);

alter table MODEL
   add constraint FK_MODEL_REFERENCE_NONCONF foreign key (NONCID)
      references NONCONF (NONCID);

alter table NONCONF
   add constraint FK_NONCONF_PROJECT foreign key (PROJECTID)
      references PROJECT (PROJECTID);

alter table NONCONF
   add constraint FK_NONCONF_DOCUMENT foreign key (DOCUMENTID)
      references DOCUMENT (DOCUMENTID);
/**************************Non Conformance********************/
```

Appendix D

Test Relational Data

```
/************************ Project Data ************************/

INSERT INTO project
            (projectid, projectname
            )
     VALUES (1001, 'Challenger Launch'
            );
INSERT INTO project
            (projectid, projectname
            )
     VALUES (1002, 'Galilio Observatory '
            );

/************************ Document Data ************************/


INSERT INTO document
            (documentid, attachmentid, projectid, documenttitle,
createdby,
             createddate
            )
     VALUES (2001, NULL, 1001, 'Challenger Configuration',
'Smith',
             TO_DATE ('09/06/2011 00:00:00', 'MM/DD/YYYY
HH24:MI:SS')
            );
INSERT INTO document
            (documentid, attachmentid, projectid, documenttitle,
createdby,
             createddate
            )
     VALUES (2002, NULL, 1002, 'Mission Schedule', 'Louis',
             TO_DATE ('11/02/2012 00:00:00', 'MM/DD/YYYY
HH24:MI:SS')
            );

/************************ Action Data ************************/

INSERT INTO action
            (actionid, actiontitle, actionstatus, documentid
            )
     VALUES (3001, 'Challenger Compressor Followup', 'OPEN', 2001
            );
INSERT INTO action
            (actionid, actiontitle, actionstatus, documentid
            )
     VALUES (3002, 'Mission Satellite Incident', 'ARCHIEVED', 2002
            );
```

Test RDF Data

```
/*prompt Enter ontology_RDF_data
accept ontology_RDF_data 'Enter ontology_RDF_Data table (###_RDF_DATA)'
select a.triple.GET_SUBJECT() as Subject, a.triple.GET_PROPERTY() as
Predicate, a.triple.GET_OBJECT() as Object
from &&ontology_rdf_data a*/
/* Role Data */

insert into role_rdf_data values(1, SDO_RDF_TRIPLE_S
('role','urn:spaceProj-Mgmt/role/9001','urn:spaceProj-
Mgmt/role/name','Manager'));
insert into role_rdf_data values(2, SDO_RDF_TRIPLE_S
('role','urn:spaceProj-Mgmt/role/9002','urn:spaceProj-
Mgmt/role/name','Analyst    '));
insert into role_rdf_data values(3, SDO_RDF_TRIPLE_S
('role','urn:spaceProj-Mgmt/role/9003','urn:spaceProj-
Mgmt/role/name','Engineer'));

/* User Data */

insert into user_rdf_data values(1, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8001','urn:spaceProj-
Mgmt/user/name','Laura'));
insert into user_rdf_data values(2, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8002','urn:spaceProj-
Mgmt/user/name','John'));
insert into user_rdf_data values(2, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8003','urn:spaceProj-
Mgmt/user/name','Genny'));
insert into user_rdf_data values(2, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8004','urn:spaceProj-
Mgmt/user/name','Nick'));
insert into user_rdf_data values(2, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8005','urn:spaceProj-
Mgmt/user/name','Harry'));
insert into user_rdf_data values(3, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8006','urn:spaceProj-
Mgmt/user/name','Smith'));

insert into user_rdf_data values(1, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8001','urn:spaceProj-
Mgmt/user/role','urn:spaceProj-Mgmt/role/9003'));
insert into user_rdf_data values(2, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8002','urn:spaceProj-
Mgmt/user/role','urn:spaceProj-Mgmt/role/9003'));
insert into user_rdf_data values(2, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8003','urn:spaceProj-
Mgmt/user/role','urn:spaceProj-Mgmt/role/9002'));
insert into user_rdf_data values(2, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8004','urn:spaceProj-
Mgmt/user/role','urn:spaceProj-Mgmt/role/9002'));
insert into user_rdf_data values(2, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8005','urn:spaceProj-
Mgmt/user/role','urn:spaceProj-Mgmt/role/9002'));
```

```
insert into user_rdf_data values(3, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8006','urn:spaceProj-
Mgmt/user/role','urn:spaceProj-Mgmt/role/9001'));


insert into user_rdf_data values(4, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8006','urn:spaceProj-
Mgmt/user/managerOf','urn:spaceProj-Mgmt/user/8005'));
insert into user_rdf_data values(4, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8006','urn:spaceProj-
Mgmt/user/managerOf','urn:spaceProj-Mgmt/user/8004'));
insert into user_rdf_data values(5, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8005','urn:spaceProj-
Mgmt/user/supervisorOf','urn:spaceProj-Mgmt/user/8003'));
insert into user_rdf_data values(5, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8004','urn:spaceProj-
Mgmt/user/supervisorOf','urn:spaceProj-Mgmt/user/8002'));
insert into user_rdf_data values(6, SDO_RDF_TRIPLE_S
('user','urn:spaceProj-Mgmt/user/8002','urn:spaceProj-
Mgmt/user/supervisorOf','urn:spaceProj-Mgmt/user/8001'));

/* Project Data */

insert into project_rdf_data values(1, SDO_RDF_TRIPLE_S
('project','urn:spaceProj-Mgmt/project/1001','urn:spaceProj-
Mgmt/project/title','Challenger Launch'));
insert into project_rdf_data values(2, SDO_RDF_TRIPLE_S
('project','urn:spaceProj-Mgmt/project/1002','urn:spaceProj-
Mgmt/project/title','Galilio Observatory '));

/* Document Data */

insert into document_rdf_data values(1, SDO_RDF_TRIPLE_S
('document','urn:spaceProj-Mgmt/document/2001','urn:spaceProj-
Mgmt/document/title','Challenger Configuration'));
insert into document_rdf_data values(2, SDO_RDF_TRIPLE_S
('document','urn:spaceProj-Mgmt/document/2002','urn:spaceProj-
Mgmt/document/title','Mission Schedule '));
insert into document_rdf_data values(3, SDO_RDF_TRIPLE_S
('document','urn:spaceProj-Mgmt/document/2003','urn:spaceProj-
Mgmt/document/title','Phobos Exploration'));

insert into document_rdf_data values(4, SDO_RDF_TRIPLE_S
('document','urn:spaceProj-Mgmt/document/2001','urn:spaceProj-
Mgmt/document/createdby','urn:spaceProj-Mgmt/user/8001'));
insert into document_rdf_data values(5, SDO_RDF_TRIPLE_S
('document','urn:spaceProj-Mgmt/document/2002','urn:spaceProj-
Mgmt/document/createdby','urn:spaceProj-Mgmt/user/8002'));
insert into document_rdf_data values(6, SDO_RDF_TRIPLE_S
('document','urn:spaceProj-Mgmt/document/2003','urn:spaceProj-
Mgmt/document/createdby','urn:spaceProj-Mgmt/user/8003'));

/* Action Data */
```

```
insert into action_rdf_data values(1, SDO_RDF_TRIPLE_S
('action','urn:spaceProj-Mgmt/action/3001','urn:spaceProj-
Mgmt/action/title', 'Challenger Compressor Followup'));
insert into action_rdf_data values(2, SDO_RDF_TRIPLE_S
('action','urn:spaceProj-Mgmt/action/3002','urn:spaceProj-
Mgmt/action/title','Mission Satellite Incident '));
insert into action_rdf_data values(3, SDO_RDF_TRIPLE_S
('action','urn:spaceProj-Mgmt/action/3003','urn:spaceProj-
Mgmt/action/title','Phobos Exploration'));

insert into action_rdf_data values(4, SDO_RDF_TRIPLE_S
('action','urn:spaceProj-Mgmt/action/3001','urn:spaceProj-
Mgmt/action/document','urn:spaceProj-Mgmt/role/2001'));
insert into action_rdf_data values(5, SDO_RDF_TRIPLE_S
('action','urn:spaceProj-Mgmt/action/3002','urn:spaceProj-
Mgmt/document/role','urn:spaceProj-Mgmt/role/2002'));

insert into action_rdf_data values(6, SDO_RDF_TRIPLE_S
('action','urn:spaceProj-Mgmt/action/3001','urn:spaceProj-
Mgmt/status','OPEN^^xsd:string'));
insert into action_rdf_data values(7, SDO_RDF_TRIPLE_S
('action','urn:spaceProj-Mgmt/action/3002','urn:spaceProj-
Mgmt/references', 'ARCHIEVED^^xsd:string'));

/* Review Data*/

insert into review_rdf_data values(1, SDO_RDF_TRIPLE_S
('review','urn:spaceProj-Mgmt/review/4001','urn:spaceProj-
Mgmt/review/title','Phobos Documentation'));
insert into review_rdf_data values(2, SDO_RDF_TRIPLE_S
('review','urn:spaceProj-Mgmt/review/4002','urn:spaceProj-
Mgmt/review/title','Compressor Defect'));
insert into review_rdf_data values(3, SDO_RDF_TRIPLE_S
('review','urn:spaceProj-Mgmt/review/4003','urn:spaceProj-
Mgmt/review/title','Phobos Exploration'));

insert into review_rdf_data values(4, SDO_RDF_TRIPLE_S
('review','urn:spaceProj-Mgmt/review/4001','urn:spaceProj-
Mgmt/review/role','urn:spaceProj-Mgmt/role/9003'));
insert into review_rdf_data values(5, SDO_RDF_TRIPLE_S
('review','urn:spaceProj-Mgmt/review/4002','urn:spaceProj-
Mgmt/review/role','urn:spaceProj-Mgmt/role/9001'));
insert into review_rdf_data values(6, SDO_RDF_TRIPLE_S
('review','urn:spaceProj-Mgmt/review/4003','urn:spaceProj-
Mgmt/review/role','urn:spaceProj-Mgmt/role/9002'));

insert into review_rdf_data values(7, SDO_RDF_TRIPLE_S
('review','urn:spaceProj-Mgmt/review/4001','urn:spaceProj-
Mgmt/references','urn:spaceProj-Mgmt/document/2001'));
insert into review_rdf_data values(8, SDO_RDF_TRIPLE_S
('review','urn:spaceProj-Mgmt/review/4002','urn:spaceProj-
Mgmt/references','urn:spaceProj-Mgmt/document/2002'));
```

```
insert into review_rdf_data values(9, SDO_RDF_TRIPLE_S
('review','urn:spaceProj-Mgmt/review/4003','urn:spaceProj-
Mgmt/references','urn:spaceProj-Mgmt/document/2003'));
insert into review_rdf_data values(10, SDO_RDF_TRIPLE_S
('review','urn:spaceProj-Mgmt/review/4003','urn:spaceProj-
Mgmt/references','urn:spaceProj-Mgmt/document/2001'));
```

Appendix E

Mapped Ontology

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE rdf:RDF [
  <!ENTITY spaceOnt   "urn:spaceProj-Mgmt/" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
]>
<rdf:RDF
  xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:xsd="&xsd;"
xmlns:owl="&owl;"
  xml:base="urn:spaceProj-Mgmt/" xmlns="&spaceOnt;"
>
<owl:Ontology rdf:about="SpaceKnowledgeManagement">
  <rdfs:comment>Space Management Ontology</rdfs:comment>
  <rdfs:label>London Metropolitan University</rdfs:label>
  <rdfs:label>Knowledge Management Research Centre</rdfs:label>
  <rdfs:label>School of Computing</rdfs:label>
  <owl:versionInfo>SpaceOntology, Version 1.0 , 2013</owl:versionInfo>
</owl:Ontology>
<owl:Class rdf:about="&spaceOnt;ACTION" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasACTIONID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasACTIONID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasACTIONSTATUS" />
            <owl:hasValue rdf:datatype="&xsd;string">OPEN
                </owl:hasValue>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasACTIONSTATUS" />
```

```
            <owl:hasValue rdf:datatype="&xsd;string">CLOSED
                </owl:hasValue>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasACTIONSTATUS" />
            <owl:hasValue rdf:datatype="&xsd;string">ARCHIEVED
                </owl:hasValue>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;ATTACHMENT" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasATTACHMENTID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasATTACHMENTID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;CATEGORY" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasCATEGORYID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasCATEGORYID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;CLASSIFICATION" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasCLASSID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasCLASSID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;COLLECTION" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasCOLLECTIONID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasCOLLECTIONID" />
```

```
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
</owl:Class>
<owl:Class rdf:about="&spaceOnt;CRITICALITY" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasCRITICALITYID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasCRITICALITYID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;DISPOSITION" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasDISPOSITIONID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasDISPOSITIONID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;DOCUMENT" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasDOCUMENTID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasDOCUMENTID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/PROJECT" />
</owl:Class>
<owl:Class rdf:about="&spaceOnt;DOMAIN" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasDOMAINID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasDOMAINID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
```

```
</owl:Class>
<owl:Class rdf:about="&spaceOnt;EDITORIAL" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasEDITORIALID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasEDITORIALID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;EFFECT" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasEFFECTID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasEFFECTID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;INCIDENT" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasACTIONID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasACTIONID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasINCIDENTID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasINCIDENTID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/ACTION" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasACTIONID"
/>
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:minCardinality>
        </owl:Restriction>
```

```
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;JENA_G1T0_REIF" >
</owl:Class>
<owl:Class rdf:about="&spaceOnt;JENA_G1T1_STMT" >
</owl:Class>
<owl:Class rdf:about="&spaceOnt;JENA_GRAPH" >
</owl:Class>
<owl:Class rdf:about="&spaceOnt;JENA_LONG_LIT" >
</owl:Class>
<owl:Class rdf:about="&spaceOnt;JENA_LONG_URI" >
</owl:Class>
<owl:Class rdf:about="&spaceOnt;JENA_PREFIX" >
</owl:Class>
<owl:Class rdf:about="&spaceOnt;JENA_SYS_STMT" >
</owl:Class>
<owl:Class rdf:about="&spaceOnt;LIKELIHOOD" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasLIKELIHOODID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasLIKELIHOODID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;NONCONF" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasNONCID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasNONCID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/PROJECT" />
    <owl:SymmetricProperty rdf:about="urn:spaceProj-Mgmt/STATUS">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/STATUS" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/STATUS" />
    </owl:SymmetricProperty>
    <owl:equivalentClass>
    <owl:Class>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/STATUS" />
            <owl:hasValue rdf:resource="urn:spaceProj-Mgmt/STATUS" />
        </owl:Restriction>
    </owl:Class>
    </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;PANEL" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasPANELID"
/>
```

```
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasPANELID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;PREFIXES" >
</owl:Class>
<owl:Class rdf:about="&spaceOnt;PRIORITY" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasPRIORITYID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasPRIORITYID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/ACTION" />
</owl:Class>
<owl:Class rdf:about="&spaceOnt;PROJECT" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasPROJECTID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasPROJECTID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;RANK" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasRANKID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasRANKID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;RESOLUTION" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasACTIONID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasACTIONID"
/>
```

```
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasRESOLUTIONID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasRESOLUTIONID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/ACTION" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasACTIONID"
/>
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;REVIEW" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasREVIEWID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasREVIEWID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/ACTION" />
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/ROLE" />
</owl:Class>
<owl:Class rdf:about="&spaceOnt;RISK" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasRISKID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasRISKID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/PROJECT" />
</owl:Class>
<owl:Class rdf:about="&spaceOnt;ROLE" >
```

```
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasROLEID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasROLEID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;SCENARIO" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasSCENARIOID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasSCENARIOID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;SEVERITY" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-
Mgmt/hasSEVERITYID" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-
Mgmt/hasSEVERITYID" />
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;STATUS" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasSTATUSID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasSTATUSID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/ACTION" />
</owl:Class>
<owl:Class rdf:about="&spaceOnt;TRAIL" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasTRAILID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasTRAILID"
/>
```

```
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="&spaceOnt;USERPROFILE" >
    <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasUSERID"
/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="urn:spaceProj-Mgmt/hasUSERID"
/>
                <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/ROLE" />
</owl:Class>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONDATE">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION" />
        <rdfs:range rdf:resource="&xsd;dateTime" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONSTATUS">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONTITLE">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ACTION" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasATTACHMENTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ATTACHMENT" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCATEGORYID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CATEGORY" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasNONCID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CATEGORY" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasNONCID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CATEGORY" />
        <rdfs:range rdf:resource="&xsd;double" />
```

```
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCATEGORYNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CATEGORY" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCLASSID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CLASSIFICATION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCLASSNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CLASSIFICATION" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasREVIEWID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CLASSIFICATION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCOLLECTIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/COLLECTION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/COLLECTION" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/COLLECTION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasCRITICALITYID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CRITICALITY" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasLIKELIHOODID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CRITICALITY" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/LIKELIHOOD" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasLIKELIHOODID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CRITICALITY" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CRITICALITY" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/RISK" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CRITICALITY" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasSEVERITYID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CRITICALITY" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/SEVERITY" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasSEVERITYID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CRITICALITY" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
```

```
    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasCRITICALITYNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/CRITICALITY" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasDISPOSITIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DISPOSITION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasNONCID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DISPOSITION" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasNONCID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DISPOSITION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasDISPOSITIONNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DISPOSITION" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasATTACHMENTID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/ATTACHMENT" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasATTACHMENTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasPROJECTID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/PROJECT" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasPROJECTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCREATEDBY">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCREATEDDATE">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
        <rdfs:range rdf:resource="&xsd;dateTime" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasDOCUMENTTITLE">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasDOMAINID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOMAIN" />
        <rdfs:range rdf:resource="&xsd;double" />
```

```
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOMAIN" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/RISK" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOMAIN" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasDOMAINNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/DOMAIN" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasEDITORIALID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/EDITORIAL" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasEDITORIALNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/EDITORIAL" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasREVIEWID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/EDITORIAL" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasEFFECTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/EFFECT" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasNONCID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/EFFECT" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasNONCID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/EFFECT" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasEFFECTNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/EFFECT" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/INCIDENT" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasINCIDENTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/INCIDENT" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/INCIDENT" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/ACTION" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/INCIDENT" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasINCIDENTNAME">
```

```
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/INCIDENT" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasGRAPHID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_G1T0_REIF" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasOBJ">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_G1T0_REIF" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasPROP">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_G1T0_REIF" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasSUBJ">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_G1T0_REIF" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasGRAPHID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_G1T1_STMT" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasNAME">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_GRAPH" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCHKSUM">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_LONG_LIT" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasTAIL">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_LONG_LIT" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCHKSUM">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_LONG_URI" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasTAIL">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_LONG_URI" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasCHKSUM">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_PREFIX" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasTAIL">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_PREFIX" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasGRAPHID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/JENA_SYS_STMT" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasLIKELIHOODID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/LIKELIHOOD" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
```

```
    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasLIKELIHOODNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/LIKELIHOOD" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasNONCID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasPROJECTID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/PROJECT" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasPROJECTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasSTATUS">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/STATUS" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasSTATUS">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasNONCTITLE">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/NONCONF" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasPANELID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PANEL" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasPANELNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PANEL" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasREVIEWID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PANEL" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasURI">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PREFIXES" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasPRIORITYID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PRIORITY" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PRIORITY" />
```

```
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/ACTION" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PRIORITY" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasPRIORITYNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PRIORITY" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasPROJECTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PROJECT" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasPROJECTNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/PROJECT" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasRANKID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RANK" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RANK" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/RISK" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RANK" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasRANKNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RANK" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RESOLUTION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasRESOLUTIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RESOLUTION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
            <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RESOLUTION" />
            <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/ACTION" />
    </owl:ObjectProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RESOLUTION" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasRESOLUTIONNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RESOLUTION" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasREVIEWID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/REVIEW" />
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
```

```
<owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/REVIEW" />
        <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/ACTION" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/REVIEW" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasROLEID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/REVIEW" />
        <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/ROLE" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasROLEID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/REVIEW" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasREVIEWTITLE">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/REVIEW" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RISK" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RISK" />
        <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/DOCUMENT" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasDOCUMENTID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RISK" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasPROJECTID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RISK" />
        <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/PROJECT" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasPROJECTID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RISK" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasRISKTITLE">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/RISK" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasROLEID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ROLE" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasROLENAME">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/ROLE" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasSCENARIOID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/SCENARIO" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/SCENARIO" />
        <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/RISK" />
```

```
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/SCENARIO" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasSCENARIONAME">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/SCENARIO" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasSEVERITYID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/SEVERITY" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasSEVERITYNAME">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/SEVERITY" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasSTATUSID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/STATUS" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/STATUS" />
        <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/ACTION" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasACTIONID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/STATUS" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasSTATUSNAME">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/STATUS" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasTRAILID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/TRAIL" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/TRAIL" />
        <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/RISK" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasRISKID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/TRAIL" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasTRAILNAME">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/TRAIL" />
    <rdfs:range rdf:resource="&xsd;string" />
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasUSERID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/USERPROFILE" />
    <rdfs:range rdf:resource="&xsd;double" />
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:about="urn:spaceProj-Mgmt/hasROLEID">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/USERPROFILE" />
        <rdfs:range  rdf:resource="urn:spaceProj-Mgmt/ROLE" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasROLEID">
    <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/USERPROFILE" />
```

```
        <rdfs:range rdf:resource="&xsd;double" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasUSERMANAGER">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/USERPROFILE" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasUSERNAME">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/USERPROFILE" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-Mgmt/hasUSERROLE">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/USERPROFILE" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="urn:spaceProj-
Mgmt/hasUSERSUPERVISOR">
        <rdfs:domain rdf:resource="urn:spaceProj-Mgmt/USERPROFILE" />
        <rdfs:range rdf:resource="&xsd;string" />
    </owl:DatatypeProperty>
</rdf:RDF>
```

Appendix F

Mapped RDF Data

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE rdf:RDF [
  <!ENTITY spaceOnt   "urn:spaceProj-Mgmt/" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
]>
<rdf:RDF
  xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:xsd="&xsd;"
xmlns:owl="&owl;"
  xml:base="urn:spaceProj-Mgmt/" xmlns="&spaceOnt;"
>
<owl:Ontology rdf:about="SpaceKnowledgeManagement">
  <rdfs:comment>Space Management Ontology</rdfs:comment>
  <rdfs:label>London Metropolitan University</rdfs:label>
  <rdfs:label>Knowledge Management Research Centre</rdfs:label>
  <rdfs:label>School of Computing</rdfs:label>
  <owl:versionInfo>SpaceOntology, Version 1.0 , 2013</owl:versionInfo>
</owl:Ontology>
<!-- urn:spaceProj-Mgmt/ACTION -->


<ACTION rdf:about="&spaceOnt;ACTION">
        <hasACTIONDATE rdf:datatype="&xsd;dateTime">2013-10-06T00:00:00-
00:00</hasACTIONDATE>
        <hasACTIONID rdf:datatype="&xsd;double">3002</hasACTIONID>
        <hasACTIONTITLE rdf:datatype="&xsd;string">Mission Satellite
Incident</hasACTIONTITLE>
        <hasDOCUMENTID rdf:datatype="&xsd;double">2002</hasDOCUMENTID>
```

```
        <hasACTIONSTATUS
rdf:datatype="&xsd;string">ARCHIEVED</hasACTIONSTATUS>
</ACTION>

<ACTION rdf:about="&spaceOnt;ACTION">
</ACTION>

<!-- urn:spaceProj-Mgmt/PROJECT -->

<PROJECT rdf:about="&spaceOnt;PROJECT">
        <hasPROJECTNAME rdf:datatype="&xsd;string">Galilio Observatory
</hasPROJECTNAME>
        <hasPROJECTID rdf:datatype="&xsd;double">1002</hasPROJECTID>
</PROJECT>

<!-- urn:spaceProj-Mgmt/REVIEW -->

<REVIEW rdf:about="&spaceOnt;REVIEW">
        <hasACTIONID rdf:datatype="&xsd;double">3002</hasACTIONID>
        <hasREVIEWID rdf:datatype="&xsd;double">4002</hasREVIEWID>
        <hasROLEID rdf:datatype="&xsd;double">2</hasROLEID>
        <hasREVIEWTITLE rdf:datatype="&xsd;string">Satellite Re-
launch</hasREVIEWTITLE>
</REVIEW>

<REVIEW rdf:about="&spaceOnt;REVIEW">
        <hasPrivilege rdf:datatype="&xsd;string">SELECT</hasPrivilege>
        <hasPrivilege rdf:datatype="&xsd;string">INSERT</hasPrivilege>
        <grantee rdf:datatype="&xsd;string">RELATIONALONT</grantee>
</REVIEW>
<!-- urn:spaceProj-Mgmt/USERROLE -->

<USERROLE rdf:about="&spaceOnt;USERROLE">
        <hasROLENAME rdf:datatype="&xsd;string">Engineer</hasROLENAME>
        <hasROLEID rdf:datatype="&xsd;double">1</hasROLEID>
</USERROLE>

<USERROLE rdf:about="&spaceOnt;USERROLE">
        <hasROLENAME rdf:datatype="&xsd;string">Controller</hasROLENAME>
        <hasROLEID rdf:datatype="&xsd;double">2</hasROLEID>
</USERROLE>

<USERROLE rdf:about="&spaceOnt;USERROLE">
</USERROLE>
</rdf:RDF>
```