NTNU

Norwegian University of
Science and Technology

Enabling an automated monitor layer for RDF using integrity constraints

Øyvind Valen-Sendstad

Master of Science in Informatics
Submission date: December 2011
Supervisor: Agnar Aamodt, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

The Semantic Web has introduced several innovations to the existing field of Integrated Operations and sparked a shift in the development of the ISO 15926 standard. Industries that want to implement ISO 15926 need to preserve a high level of data quality while opening for free data exchange with its peers. The ISO 8000 standard defines three important concepts that define data quality, namely; syntax, semantic and pragmatic data quality - which has been used as a measure. The existing tools and methods is currently not capable of performing the required level of data validation. A new approach has recently been suggested that uses a method of hybrid reasoning to validate data by applying a local Closed World Assumtion (CWA). This enables the possibility of augmenting the ISO 15926 Ontology with Integrity Constraints (IC) where data dependencies needs to be expressed. The work of this thesis shows that the IC approach is applicable to the fundamental Life-cycle dependency of ISO 15926.

0.1 Acknowledgments

This thesis has been made possible by the encouragement of my father, Magne which has introduced me for the ISO 15926 community at Det Norske Veritas (DNV). This work would not be possible without the help of Johan Klüwer and his support for my work. I would like to thank Morten Rørvik Strand for his patience when helping me with the SPARQL database when I needed it the most. And Christian Mahesh Hansen for opening my eyes for the possibilities of the Jena framework. At last I would like to thank Agnar Aamodt for his comments, interesting discussions and especially for making this thesis possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem statement

1.1.1 Original problem statement

The candidate shall examine existing tools and methods accompanied with the Semantic Web stack in order to achieve automated reasoning on the ISO 15926 architecture. Alternatively a new method can be devised. First, the method or tool should be able to perform data validation on a syntactic level. This includes verifying that the data conforms to the data fields and syntax definitions in the ontology. Second, the method or tool should be able to perform checks on the ontology for semantic level inconsistencies, circular dependencies, hierarchy inheritance violations and data type violations.

1.1.2 Changes

It was discovered that the existing reasoners and frameworks already supported the described data validation needs, and that it easily could integrated into the ISO 15926 environment. However, a new approach to data validation was discovered where a local CWA could be applied to parts of the ontology (where there are data dependencies) that can not be evaluated with the general OWA concept of the Semantic Web. In addition, studying the ISO 15926 foundation and documentation proved to take longer time than expected. This work has therefore been included as part of this thesis.

1.2 Motivation

The shift of ISO 15926 towards W3C implementations opens a new world of tools and technologies that has accelerated the implementation of ISO 15926. This development has sparked a renewed momentum from not only the oil & gas industry, but also from several other industries that understands the potential of Integrated Operations. With this growing interest there comes a concern of risks from the automation of data integration and the interoperability with collaborating and competing parties. The companies involved have huge investments in their information systems that they wish to protect while adopting to the new technology. In order to compensate for this there is a need for an underlying control mechanism that validates the data, ensuring high data quality.

1.3 Goal

Automated reasoning is a crucial part of the Semantic Web, both for verifying data correctness, preserving the information quality and processing business logic. These concepts have been throughly established in ISO 8000 that provide standard definitions for information quality. We will therefore define three levels of checking and processing according to the standard, namely syntactic, semantic and practical.

Syntactic data quality implies the need to verify that input values are correct according to the respective classes given by the ontology. This involves checking that numerical ranges are within their boundaries, strings comply to their required patterns and object references are valid. As a statement example, the relationship between a **father** and **son** has a series of constraints attached to it that needs to be evaluated. The syntactic checking will have to verify that the **father** is older than the **son**, and that none of them have a negative age.

Semantic data quality goes one step further by evaluating if two classes are equivalent and investigating if the relationships of the entities within the data is reasonable. The reasoning will involve entailing that the two persons are derived from a valid ontology class, in this case, the class **person**. The same applies to other mammals like monkeys where the reasoner needs to be able to verify that both are from the same species.

The data quality part of pragmatic is related to ISO 15926 reference logic and the incorporation of data dependencies that can not be expressed in an Ontology. (The pragmatic level also includes general measures that indirectly affects the use of ISO 15926).

This thesis seeks to answer the following questions:

1. Is there any suitable data validation tools accompanied with the Semantic Web that is suitable for ISO 15926?

2. Is it possible to apply Semantic Web tools in ISO 15926 and fulfill the ISO 8000 requirements of:

   Syntactic information quality

   Semantic information quality

   Pragmatic information quality

1.3.1 Personal goal

From my perspective the ISO 15926 standard has potential reaching far beyond the oil & gas industry. It is therefore a great opportunity to learn how it works and get first hand experience with the implementation of ISO 15926.

1.4 Thesis guide

This thesis has been written in a industrial context using experimental data provided from DNV related to a project for VNIIAES (All-Russian Scientific Research Institute for Nuclear Power Plant Operation), internally known within DNV as RAPOC. Source code is presented in indented plain text indicating that

```
        this is a source code listing
```

classes are represented in bold lower case like this, **this_is_a_class**.

individuals are represented with quotation marks like this, "instance".

properties are written in italics like this, *authorOf*

ISO 15926 originates from the European oil & gas and process industries. Therefore the background materials, examples etc. presented here reflects experiences and challenges from these industries, and in particular their challenges related to cost effectiveness, HSE requirements etc. It is also worth noting that ISO 15926 is developed by the industry as a response to industrial needs, as opposed to e.g. OWL and sematic web tools that originates from more research related areas. Since the start of the last decade the similarities between the two has however become more evident, and it is exciting to see how these two complement each other, and that OWL and Semantic Web offers an enormous potential for adding capabilities to support the take-up of ISO 15926.

The basic similarity is the use of triples and logic. Since its conception ISO 15926 has been based on the distinction between a class and individuals (as in first order logic), organizing the classes in super-subtype hierarchies, and replacing attributes by relationships. Hence the model is based on object-relationship-object. Therefore ISO 15926 is ideal for use with Semantic Web tools.

A key element in the use of semantic web tools to perform checks of data implemented according to ISO 15926 is to understand the industry requirements and how they are implemented in order to apply the appropriate checking tools, and what can be achieved with today's tools and levels of implementations. A substantial part of this thesis has been devoted to understand and describe this.

The full title of ISO 15926 is "Integration of life-cycle data for process plants including oil and gas production facilities". The two important aspects here are "integration" and "life-cycle", and I intend to show how RDF and semantic web tools can be applied to support this.

Chapter 2

Technical Background & Concepts

The introduction of computer systems in all phases of design, fabrication, operation and decommissioning of oil & gas installations from the early eighties and onwards not only introduced new opportunities, but also new challenges to the whole industry. From the mid 1980's a typical portfolio of computer systems could be as in the Figure 2.1 below which shows the systems used by Aker Engineering when designing the Snorre and Sleipner platforms. In addition to the systems there is also a network of data exchange, both within the silos, and across silos. The general picture is the same today, even if some integration has taken place.

Typically the systems can be divided by technology and applications, made of systems for scientific analysis and simulations, data bases for objects with attributes and values, 2D Computer-Aided Design (2D CAD) drafting systems (with or without associated alphanumeric data), 3D CAD systems for spatial design, management and administrative systems and office applications.

Information about one object is typically stored in many of these systems, and to have the complete picture of the status of an object one will need data from multiple systems, so the need for integration is obvious. E.g. to have the full information about a pump and its driver, e.g. an electric motor, you will need data from all groups of systems. In addition, data about an object is spread across many systems that has proprietary data definitions and different sets of applications across companies and between life-cycle stages of a plant. This causes severe problem for the interoperability between different applications, companies and life-cycle stages.

As can be seen from Figure 2.1 the complete set of documentation for oil & gas installation or a process plant is complex interconnected data that require a substantial effort to hand over. Most of the data that is to be used during operation is defined during engineering or during later design stages

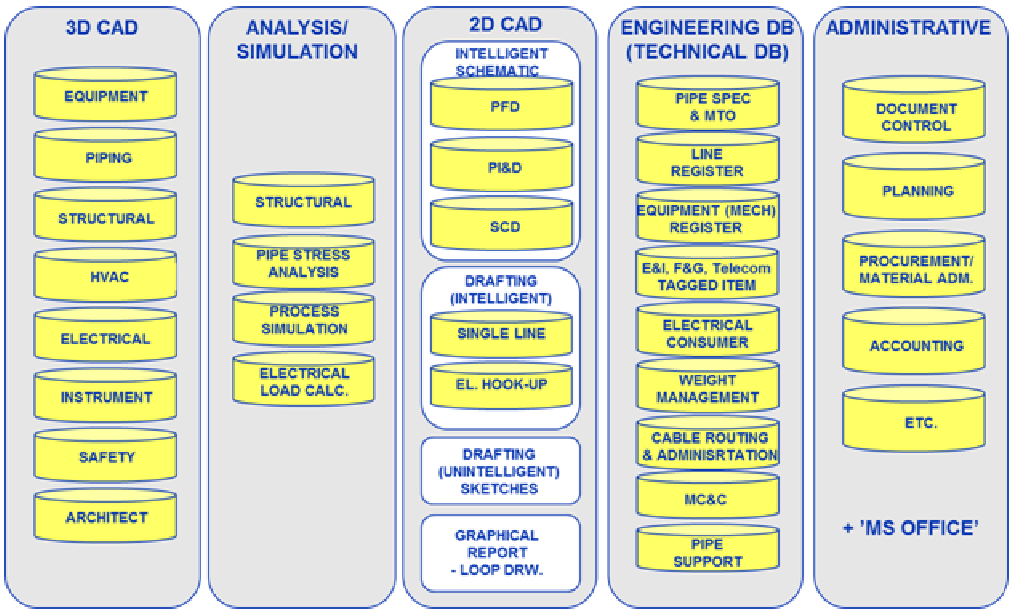

Figure 2.1: Silo perspektivet - Oil & Gas engineering systems

when modifications to the plant are designed. A key aspect of all design work is to make drawings, diagrams and sketches. During the early 1970's several companies started to offer 2D CAD design/drafting systems that would revolutionize the industry. These early systems were often a computerized version of paper-based documents, drawings and artifacts made up of proprietary formats.

Later vendors started to combine graphics and alphanumeric data by relating alphanumeric data to the graphics representing a particular object, often duplicating data that originated from other systems. This was even less standardized. Some systems attached the alphanumeric data to the graphical symbols, and later adding a 3D representation as well (like CADDS 4X) where as others defined the objects as objects in a database and related geometric representations to these, which again was used to generate the graphics. This resulted in an even more complex picture. As experience with such systems grew more and more capabilities were added to such systems, some were company specific systems and some were customizations of Commercial Off The Shelf (COTS) systems. So not even using systems from the same vendor would guarantee interoperability.

All the systems shown in Figure 2.1, as described above, were based on proprietary data models and data definitions, often poorly documented, thus making exchange of data between systems extremely difficult. This quickly proved to be a problem for the companies in the industry that wanted to share their work not only internally, but also with their contractors. The industry was at the mercy of its vendors and application developers, and in particular of the CAD vendors as in this area there were very few in-house developed systems. The CAD vendors had few incentives to help resolve this problem, so this dependence continued to grow and eventually turned out to be a serious problem. In 1979, during a two-day meeting at the Society of Manufacturing Engineers (SME) an engineer from General Electric (GE) challenged the vendors to create a neutral exchange format for 2D graphics. The following day, when the vendors confirmed that the solution was feasible, there was no way to stop the momentum of enthusiasm from the industry. As a result the Initial Graphics Exchange Specification (IGES) [15] format was developed.

The IGES initiative was an initial effort driven by the need for a common data format that could enable translations of 2D graphics between different Computer-Aided Design (CAD) systems, but it was never intended to become a standard. And it was not intended to address 3D geometry and related alpha-numeric data which had become increasingly popular and required by the industry. Therefore four years later, in 1983, a standard that addressed this, namely ISO10303, known as STandard for the Exchange of Product model data (STEP) was initiated by ISO Technical Committee 184, Sub Committee 4, Industrial Data (ISO TC184/SC4). Its aim was to provide a mechanism that would be capable of describing product data throughout the life-cycle of a product, independent from any system. This vision would involve not only the once important CAD systems, but also the Computer-Aided Manufacturing (CAM), Computer-Aided Engineering (CAE), Product data management/Engineering Data Management (PDM/EDM) and any other computer-aided systems, commonly referred to as Computer-Aided technologies (CAx). This would then cover the systems used in the oil & gas and process industries as shown on Figure 2.1.

The STEP standard proved very successful in the automobile, aircraft and space industry with respect to exchange of geometric information showing a particular design, i.e. at a snapshot in time, and the U. S. National Institute of Standards and Technology (NIST) wanted to extend the standard with this important notion, a fourth dimension, time. This would allow engineers to model the entire life-cycle of a process plant in a data model. Having such a data model would enable engineers to hand over a completed process plant to any given customer, allowing the customer to continue using the data model for maintainability, and ultimately demolition of the process plant. This was however never fully implemented in STEP.

STEP also got some take-up in the oil & gas and process industries, but only limited to some applications, snapshots of data, and within particular geographical regions. In the oil & gas industry it got more attention in Asia than in Europe. One of the key elements of STEP is the concept of an Application Protocol (AP), which is a subset of STEP dealing with a particular type of application, and at a snapshot as time never was incorporated into STEP. As described earlier, applications and their usage changes with companies and life-cycle stages, and it is therefore vital to be able to integrate data across applications. This was to become a big issue when the oil & gas and process industry proposed what was to become ISO15926 for standardization.

Towards the end of the 1980's these problems received a lot of attention, and in 1991 the European Strategic Program on Research in Information Technology (ESPRIT) commissioned a project named ProcessBase. This was an initiative aiming at establishing a data model for life-cycle information of a facility (process plant or oil & gas installation) that would suit the requirements of the industries. The results from this project were later picked up and further developed by European Process Industries STEP Technical Liaison Executive (EPISTLE), a body originally formed to coordinate activities from European industry towards STEP. Initially EPISTLE had individual companies as members, but later this changed into a situation where three national consortia were the only members: PISTEP (UK), POSC Caesar (Norway), and USPI-NL (Netherlands). Later PISTEP merged into POSC Caesar, and USPI-NL was renamed to USPI.

As described before, the documents and drawings on which the applications originally were based traditionally represented one discipline view of the data related to a plant item. Hence to get the full view of the data about a plant item information from many documents, listings and drawings had to be compiled. These drawings and documents were usually defined and maintained separately by many disciplines or groups of users, resulting in duplicated and conflicting data that cannot easily be shared either within an enterprise or with business partners of an enterprise. Data exchange and handover therefore often defaulted to printed drawings and reports using the source document layout as the format even if electronic versions were available. This is still often the case, with pdf-files and Excel or tabular files as a wide-spread documentation and exchange format. This therefore does not provide much benefit over the use of paper for documentation and exchange regarding "intelligence" other than that the preparation of the documentation might benefit from the company internal systems and how they are integrated. One could therefore not fully benefit from the opportunities provided by the introduction of computers. As an illustration of these difficulties we experienced that to be able to maintain the data after handover the owners/operators sometimes had to accept the handover of complete installations of hardware and software.

To develop methods and tools to overcome this, the "Caesar Offshore" research pro- gram initiated by DNV and co founded by the Research Council of Norway (NFR) was established in 1992. Other member companies were Statoil, Norsk Hydro, Saga, BP, Elf, Aker, Kværner and Brown and Root. The aim of this initiative was to develop neutral formats for the exchange and integration of data among companies involved in oil & gas development projects and operations [38]. In 1995 this was reorganized to become the POSC Caesar project with the same objective. In 1997 this was followed by the POSC Caesar Association (PCA), an industry association aiming at further developing ISO15926.

One particular task in the Caesar Offshore program was to propose a data model suitable for data exchange and data integration. This resulted in the selection of the EPISTLE framework and data model, which was based on work done by the ProcessBase project and by Shell International [38]. This model was also used as the basis by two similar initiatives, PISTEP in UK and USPI-NL in the Netherlands, and cooperation with these initiatives was established under the EPISTLE umbrella.

The model defined in the EPISTLE framework Version 2.0 was over the next years simplified and further developed by POSC Caesar to become the POSC Caesar Snapshot A-E data models. In 1997 the POSC Caesar Snapshot E model was proposed for standardization in ISO TC184/SC4 as the first draft of ISO 15926-2. This was later further developed with the partners in EPISTLE to become EPISTLE ECM V4.5.1, which then was standardized as ISO 15926-2:2003; the current version of the model. Even though ISO 15926 has been developed to suit the needs of the process and oil & gas industries, pilot projects have been conducted that prove that the standard can also support the needs of other industries. Such pilots includes modeling of ship propulsion systems, integration of reference data with DNVs ship application NAUTICUS, and modeling of life-cycle assessment data according to ISO 14048; Environmental management - life-cycle assessment – Data documentation format and modeling of building data. Lately it has also sparked an interest in the nuclear power plant industry.

2.1 ISO 15926

2.1.1 ISO 15926 Parts

Døpes om til ISO 15926 today og flyttes? The diagram in figure 2.11 showed one property of a class of pressure transmitter represented in the EXPRESS notation used in Part 2 of the ISO 15926 standard. At the time of writing 7 parts have been published. Part 5 has been replaced by a generic ISO TC184/SC4 standard procedure parts,

2.1.2 Industry requirements

As mentioned above the information concerning the engineering, construction and operation of process plants and oil & gas facilities is created, used and modified by many different organizations throughout a plants life. Economic, safety and environmental considerations demand that this informa-

Part	Description
1	Introduction, information concerning engineering, construction and operation of production facilities is created, used and modified by many different organizations throughout a facility's lifetime. The purpose of ISO 15926 is to facilitate integration of data to support the life-cycle activities and processes of production facilities
2	Data Model. a generic 4D model that can support all disciplines, supply chain company types and life-cycle stages, regarding information about functional requirements, physical solutions, types of objects and individual objects as well as activities
3	Reference data for geometry and topology
4, 5, 6	Part 4, is the initial, Part 5 is a procedure for how to extend Part 4 (currently an ISO standard procedure is used), and Part 6 is the procedure describing technical requirements to Part 4 and its extensions.
7	Implementation methods for the integration of distributed systems, defining an implementation architecture that is based on the W3C Recommendations for the Semantic Web
8	Implementation methods for the integration of distributed systems – Web Ontology Language (OWL) representation

Table 2.1: The published ISO 15926 parts.

tion is available to owners and operators of facilities, contractors, and regulatory bodies in a consistent, integrated form. This requirement can be satisfied by specifications that prescribe the structure and meaning of the data that is shared by organizations and disciplines involved in all stages of a plants life-cycle. The need to increase the cost efficiency of process plants is leading to business practices that depend on the efficient integration and sharing of plant information in a computer processable form. These business practices include the following.

1. Many users' needs now span more than one of the traditional information views. Safety and environment are two examples of this.

2. Concurrent engineering requires design work to progress in parallel, with the state of the design being available electronically, in computer processable form, to other engineering, planning, purchasing and logistical activities.

3. Significant cost savings are expected from standardization of component specifications. The information about these specifications is required in computer processable form for easy incorporation into plant designs and requirements.

4. In the past, hand-over of plant design information was often restricted to design drawings and paper documents. Use of this information in managing the operation and modification of the plant was restricted to manual processes, or the information had to be redefined in a format suitable to the required application. Having the plant design and equipment information in computer processable form increases the efficiency and effectiveness of the operational phase of the plant.

5. Accurate computer processable information about a plants performance throughout its lifetime is of high value, for optimizing future modifications to the plant and for designing new plants on the basis of experience with existing plants.

6. By using a consistent context for data definitions, the information used in the various aspects of the plants life-cycle can be brought together. This allows information to be integrated, shared and exchanged in a consistent, computer processable form.

In order for ISO15926 to accomplish these goals the data quality is very important, both in the context of interoperability and in the integration of existing technologies.

The scope of business activities that are supported by ISO 15926 is illustrated in Figure 2.2 below, (reprinted from ISO 15926-1 with permissions

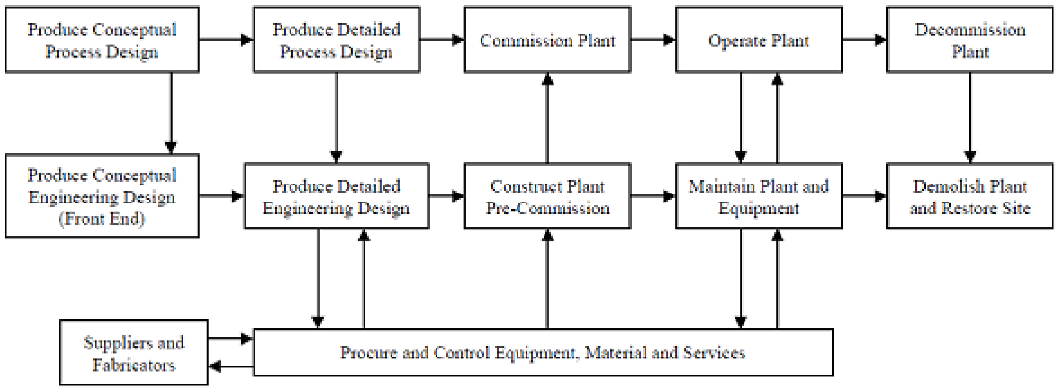
Produce Conceptual Process Design
Produce Detailed Process Design
Commission Plant
Operate Plant
Decommission Plant
Produce Conceptual Engineering Design (Front End)
Produce Detailed Engineering Design
Construct Plant Pre-Commission
Maintain Plant and Equipment
Demolish Plant and Restore Site
Suppliers and Fabricators
Procure and Control Equipment, Material and Services

Figure 2.2: Activity model of the process plant life-cycle.

from the POSC Caesar association) which shows the main activities and data flows associated with the life-cycle of a plant.

The life of process plants etc. typically is 30 years and more, and in the case of the Troll A platform it is expected that the life time will be 70 years. All the data required to operate the plant will have to be available throughout its life. Computers typically last for 3-5 years, applications will typically need an upgrade at least every 5-10 years, and database systems etc. all come in new versions. Therefore multiple migrations of data will be required over the lifetime of the plant. If this is still to be based on proprietary data definitions the consequences for both cost and data quality is huge over the lifetime of the plant. The data should therefore be defined independent of particular implementations.

2.1.3 Conceptual data model

To meet the challenges related to data integration and life-cycle data the data model specified in ISO 15926-2 [18] is a conceptual data model as described in the three schema architecture of ISO/TR 9007 [17]. The three schema architecture identifies three types of data models:

1. External model: the data structure corresponds to a view of data for a particular purpose that includes rules about the data that are appropriate to the particular purpose.

2. Conceptual data model: a neutral model that is capable of supporting any valid view that falls within its scope. Such models can only include rules for data that are universally true across its entire scope for the envisaged life of the model. As a consequence most rules or constraints arising from particular business uses of data are excluded from conceptual data models.

3. Physical model: a definition of the way data is stored. The entity data types reflect things that are important for storage and access and not the business meaning of the data.

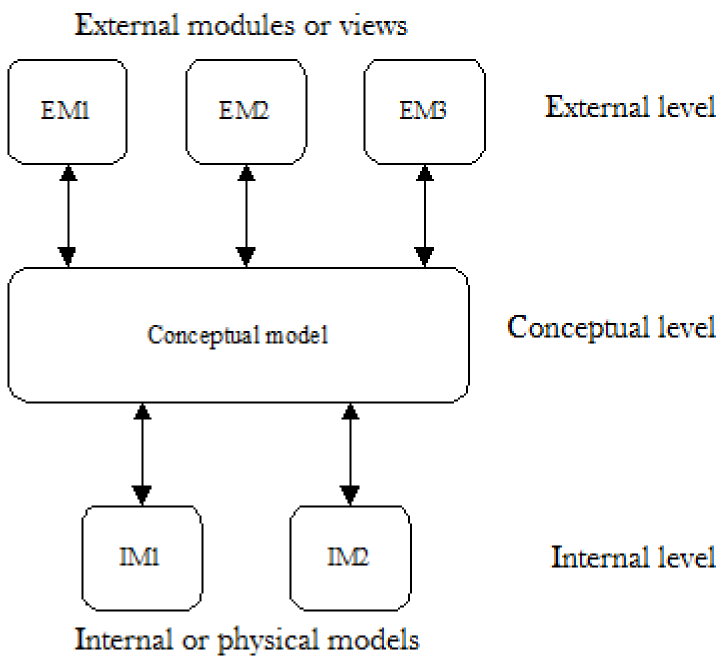

Figure 2.3: Three schema architecture from the conceptional model of ISO15926 based on ISO/TR 9007, reprinted from ISO 15926-1 with permission of POSC Caesar association.

These concepts are illustrated in Figure 2.3.

2.1.4 Overall design requirements

Based on the scope of business activities defined in Section 2.1.2, the overall design requirements that follow are:

1. Meet the data requirements for integration of facility life-cycle data.
2. Be clear and unambiguous
3. Be stable in the face of changing data requirements
4. Be flexible in the face of changing business practices
5. Be reusable by others
6. Be consistent with other models covering the same scope

7. Be able to reconcile differences with conflicting data models

Data integration means combining information derived from several independent sources into one coherent set of data that represents what is known. Because the independent sources often have overlapping scopes, combining their data requires the common things to be recognized, duplicate information to be removed, and new information represented. To succeed in the role of integration, the data model must have a context that can include all the possible data that might be wanted or required.

To be able to provide such capabilities ISO 15926 specifies a data model that defines the meaning of the life-cycle information in a single context independent of a particular view, but supporting all the views that process engineers, equipment engineers, operators, maintenance engineers and other specialists may have as described above. It is worth while recognizing that "integration" necessarily also involves exchange/import of data from external sources as there currently are no "native" ISO 15926 systems covering the need of the industry.

2.1.5 Architecture

The architecture that underlies ISO 15926 is illustrated in Figure 2.4.

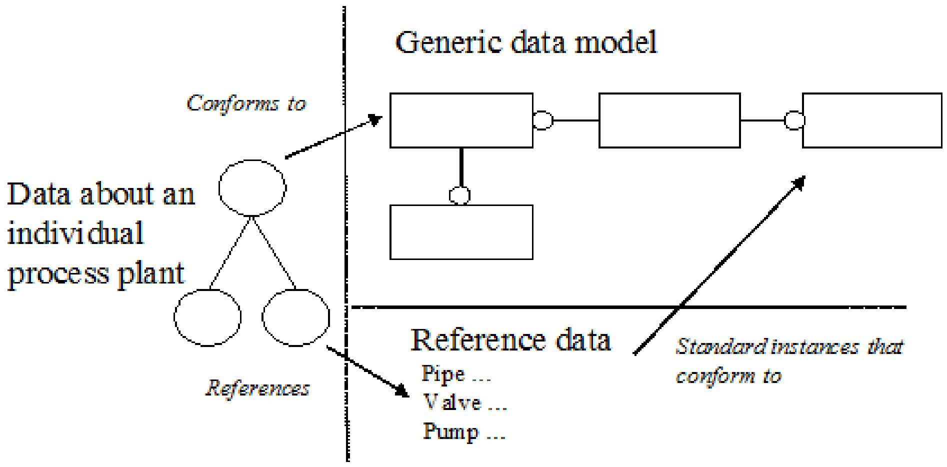

Figure 2.4: The generic data model of th ISO 15926 Architecture, reprinted from ISO 15926-1 with permission of POSC Caesar association.

By the term "generic data model" we mean that it is independent of any industry. The data model entities represent the persistent nature. Subject area (industry) terminology is added as Reference Data (RD) in an ReferenceData Library (RDL), which is an extension of the ontology defined in the data model. Therefore additional subject areas can be included by extending the RDL. To enable integration of information and to give a stable and flexible model with respect to developing and changing business practices, the model excludes all business rules that are appropriate to specific applications. Required business rules can where required be incorporated in Reference Data. Process plant life-cycle data is structured according to the data model and is divided into:

1. Data about an individual process plant or part thereof, (see left-hand side of Figure 2.4) which conforms to the data model and references the Reference Data.
2. Reference Data (RD) (bottom right-hand part of Figure 2.4), which represents information that is common to many process plants or of interest to many users.

The data model specified in ISO 15926-2 supports exchange and integration of data but does not provide sufficient specific meaning of data to enable unambiguous communication. Data about an individual process plant can be shared and exchanged only if both the sender and the receiver use the same reference data. This reference data shall be sufficient to ensure unambiguous communication between parties. Reference data is divided into the following:

1. Instances that represent reference individuals, for example the European Datum of 1950 (ED50) located at the geodetic observatory at Potsdam near Berlin is a reference individual.
2. Instances that represent reference classes.

There is a variety of RD required to meet the requirements for life-cycle data, and in order to group and allocate the responsibility for definition and maintenance of the RD with the appropriate parties a classification system for RD applicable in a process plant context has been introduced by ISO15926.

2.1.6 Data model design/EPISTLE principles

The ISO15926 data model is designed in accordance with data modeling principles developed by EPISTLE. These principles control the use of entity data types, attributes and relationships when defining a conceptual data model. Some of the effects of these principles are as follows.

1. The model entity data types are part of a universal subtype/supertype hierarchy of entity data types.
2. Entity data types are generic, representing and being named after the persistent nature of their members.

3. Attribute information is usually expressed by references to entity data types. The only mandatory attribute in ISO 15926-2 is "thing.id". Any other attribute information is expressed using relationships.

4. Relationships and activities are represented by entity data types. This is because of the need for separate metadata, e.g. from a data management and data quality perspective it is required to record who established a relationship, and what is its status. See section on data quality and ISO 8000. Note: This causes huge challenges when interfacing with external models. The question always is "what type of relationship represents a given attribute in traditional models?"

2.1.7 Some aspects related to the use of ISO 15926

By applying these design principles to industrial data the RD or ontology of ISO15926 spans from very generic concepts, or classes, down to very precisely defined classes representing catalog type data as shown in the Figure 2.5 below. A classification system for the ontology is formalized by the types of classes defined in ISO 15926-1, Reference data.

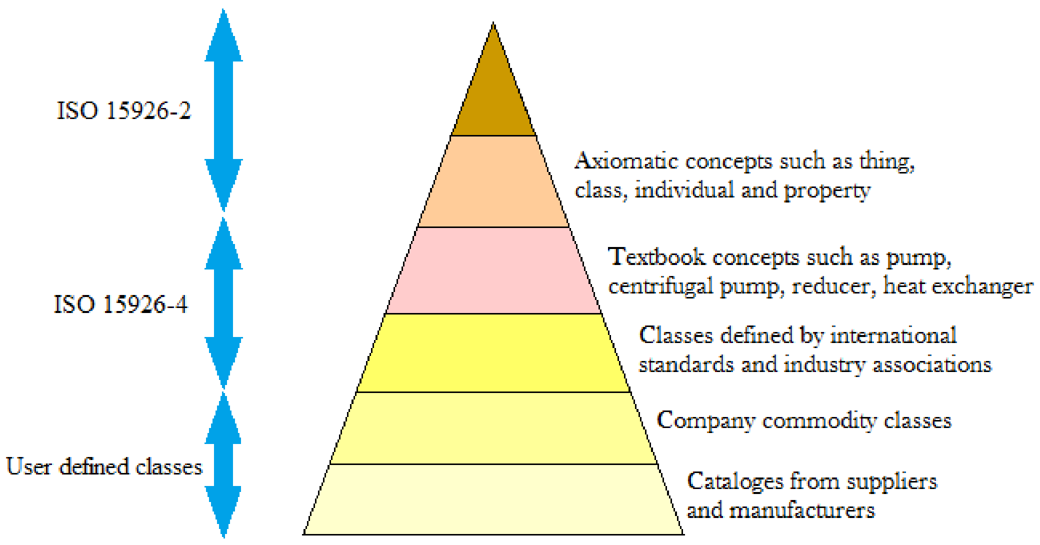
ISO 15926-2
ISO 15926-4
User defined classes
Axiomatic concepts such as thing, class, individual and property
Textbook concepts such as pump, centrifugal pump, reducer, heat exchanger
Classes defined by international standards and industry associations
Company commodity classes
Cataloges from suppliers and manufacturers

Figure 2.5: Data Catalog, reprinted from ISO 15926-1 with permission of POSC Caesar association.

The work done in the EPISTLE and ISO 15926 communities have shown that all concepts, or types of objects, used in the oil & gas and process industries can be organized into such an ontology. "Characteristics" or "attribute values" common to class members are defined once as computer processable data related to the relevant class and "inherited" down the hierarchy.

In fact all the generic concepts and product specifications developed by standardization bodies (ISO, International Electro technical Commission (IEC), American Society of Mechanical Engineers (ASME)) can be defined in an ontology that is independent of any particular project or organization. The intent is that this shall be standardized in ISO 15926-4, Reference Data. Parts of it will as a step in the standardization process be held in the POSC Caesar RDL as an "industry ontology" before being standardized by ISO. These ontologies can be referred to as the "Background ontology" for a project. As there will be many interrelated ontologies the intent is to store this as "linked data".

As these ontologies will be used by many projects and will be vital to the industry, quality becomes an issue, both technical and from a management point of view. So in addition to ISO 15926, ISO 8000 is also expected to play a vital role.

For the technical quality the use of automated checking will be vital in the definition and maintenance of these Background Ontologies, and also in the process of checking a particular project design against the background ontology.

As the ISO 15926 model is a generic entity model where most of the meaning that is of interest in a particular domain is expressed in the Reference Data. Therefore only the basic structure can be checked using generic tools. To be able to perform domain data checking the Reference Data and project data has to be represented in such a way that it can be used by reasoners. This is what we can call "computer sensible data".

To understand which standard checks that can be used with ISO 15926 and for which purpose, and for the design of new checks, it is important to understand how data is modeled in ISO 15926, and how ISO 15926 complies with and uses OWL and RDF. OWL, UML and ISO 15926 are all based on triples and can be represented in RDF. The key difference is in the use of triples. In particular it is important to understand the modelling related to

1. Properties

2. Attributes

3. Characteristics

as the way these are dealt with is significantly different than for e.g. traditional UML models. See section 2.1.8, Representation of "Attributes" using ISO 15926 regarding the representation of attributes.

2.1.8 Representation of "Attributes" using ISO 15926

Traditional "attributes" or "characteristics" can be subdivided into two main groups.

The first group is the "characteristics" or "properties" that refers to a property expressed as a number and a Unit of Measure (UoM). The name of the attribute maps to a member of ISO 15926-2 "class_of_indirect_property", whereas the value maps to a "property" that is classified by a member of ISO 15926-2 "single_property_dimension". The UoM is represented by a "scale".

The other main group is the "Attributes" where the value is a text. This is of a different nature, as in this case the text either

1. is a statement about something,
2. or identifies something else

In the first case one need to identify what the statement is about and the nature of the statement, in the second case the challenge is to identify what the text identifies, and how this is related to the thing that "has the attribute". As there are no attributes in ISO 15926 one have to determine the nature of the relationship. This is one of the most challenging parts when it comes to mapping external data on to ISO 15926, i.e. express the meaning in ISO 15926 terms.

The reason for choosing what was to become ISO 15926, i.e. the EPISTLE framework, was that it was identified that in order to achieve the goals (exchange and integration of life-cycle data), data had to be modeled independently of existing applications, and one had to separate what things are from how they are used and represented. In short one had to concern oneself with the underlying nature of things, which is one of the foundations for EPISTLE/ISO 15926.

In traditional modeling one starts from a particular view of the world, not considering other possible views. Based on this starting point one defines what are the types of objects of interest and their attributes. The most common starting point is to start from a "functional" view of the world, i.e. you start from what the thing "does" or "is", e.g. a pump, a pipe a cup or a chair, and then everything else is an attribute of that type of object. In other views of the data one might find that what is an attribute in one view is a class in another view. E.g. "steel" is often seen as the value of an attribute ("material of construction") of "steel pipe". This is in conformance with a piping designers view of the world, but in conflict with material expert's view of the world. They see this as a lump of steel with a particular shape. And if you have a piece of steel pipe you can examine it, you will then find that it possesses all the "properties" of a pipe and also those of "steel", and in the case of process design and pipe stress calculations you need both.

As described above the value of a "text attribute" normally identifies another object, and if the "properties" or "attributes" of this is of interest to you, (and it normally is in the context of process plants) you will have to define this somewhere else to be able to record data about it. You then have the same object at least twice, once as a class and once as a test string, and these needs to be coordinated. In ISO 15926 this is dealt with as 3 classes, "pipe", "steel object", and "steel pipe", where "steel pipe" is a subclass of "pipe" and "steel object". (It inherits both sets of properties as described above). This allows us to record what is known about a "thing" one place and refer to it, avoiding duplicate information.

2.1.9 Conditions for membership

One important aspect related to automated checking based on "attributes" is the fact that these can be divided into two groups, one group is the "attributes" that are used to define membership in the class ("conditions for membership"), the other group are "descriptive attributes".

As an example, the basis for membership of a ship in the class VLCC (Very Large Crude Carrier) is that it can carry 200,000–320,000 tons of crude oil. Nothing is stated about its overall length or width. So load carrying capacity is used to establish class membership, overall length is not included in the definition of class membership (unless the class definition is changed to include a maximum length, in which case we will have a subclass, e.g. VLCC with a maximum length xxx meters.) This distinction is vital when it comes to the use of automated reasoning for defining class membership.

A complicating factor is that an "attribute" does not necessarily belong in the same category over the lifetime of the object it is related to. Say if a type of valve is selected based on capabilities, but not considering its length. If you need to replace it the length suddenly becomes a criterion for selection of a new one.

Another example is to say that a car shall have an engine, and that a petrol car shall have a petrol engine and a diesel car shall have a diesel engine. In this case one will have to define the classes engine, diesel engine, petrol engine, car, diesel car and petrol car, including the relationships between them as "conditions for membership" and to express this in a "computer processable form" to be able to reason over the data. This applies both to the structure of the ontology, and to determine class membership. An ontology would typically be as shown below. In addition comes the different ignition and fuel systems that apply to diesel and petrol engines which are not shown.

2.1.10 Key ISO 15926 Terms and definitions

The following key terms and definitions apply to ISO 15926. One should in particular note the definitions of instance and individual which often leads to confusions.

2.1.10.1 *class*

A category or division of things based on one or more criteria for inclusion and exclusion. A class need not have any members (things that satisfy its criteria for membership). Note: "class_of_individual" (1st order) and "class_of_class" (2nd order) are subtypes of "class".

2.1.10.2 *conceptual data model*

A data model in the three schema architecture defined by ISO/TR 9007, in which the structure of data is represented in a form independent of any physical storage or external presentation format

2.1.10.3 *individual*

A thing that exists in space and time. In this context existence is based upon being imaginable within some consistent logic, including actual, hypothetical, planned, expected, or required individuals. A pump with serial number ABC123, Battersea Power Station, Sir Joseph Whitworth, and the Starship "Enterprise" are examples of individuals.

2.1.10.4 *instance*

Data that represents, in computer processable form, some real-world thing

2.1.10.5 *Reference Data (RD)*

Process plant life-cycle data that represents information about classes or individuals which are common to many process plants or of interest to many users

2.1.10.6 *Reference Data Library (RDL)*

Managed collection of reference data. Classes are organized in a super/-subtype hierarchy (taxonomy) based on rules as described above. Based on information about who is responsible for the definition of the conditions for membership the classes are grouped as e.g. "core", "standard" or "manufacturer". The "core" classes are the most generic, with few restrictions, located near the top of the taxonomy, and the "manufacturer" classes, with an often extensive set of conditions for membership, are at the bottom". Membership of individuals in classes, and super/subclass relationships are in all cases established based on the "conditions for membership".

For management purposes and to place the responsibility for defining and maintaining reference data with relevant organizations, the reference data is further subdivided into the following categories:

core class is a class that is a commonly used subdivision corresponding to terms used in common language. The conditions for membership are often not formally defined. Pipe, floor, pump, and light bulb are all core classes.

de facto class is a class corresponding to common natures that are widely recognized, but not formally agreed or defined. De facto classes may be formalized by international, national, or industry agreement. An example is a manufacturer may choose to make a product of similar specification to that of another manufacturer in order to compete for the market share by choosing to conform to some characteristics of the other product.

standard class is a class whose specification for membership is owned or controlled by a standardization body and is publicly available. Standard classes result from the work of national, international, or industry standardization bodies and cover sizes, shapes, materials, performance, and manufacturing processes of equipment and materials. The rules for exclusion and inclusion (or conformance) are agreed by an open, consensus process and are made publicly available. A standard class may only constrain one particular aspect and often be insufficient to determine usage or full manufacturing specifications. Examples are the ASME B16.9 standard constrains the dimensions and shapes of steel butt-welding pipe fittings and the IEC 60079-1 standard which specifies constraints on electrical equipment to ensure standard degrees of explosion proofness.

manufactured product class is a class whose members are individuals produced by a manufacturing process. The members of a manufactured product class may be discrete or may be batches or continuous flows, such as process fluids. "Lightbulbs 60 W 230 V E27" is an example of a manufactured product class whose members are discrete. "BS4040 Leaded Petrol" is an example of a manufactured product class whose members are continuous. Note that a manufactured product class may correspond to a specification that has not been realized, such a product specification for which no products have yet been made.

commodity product class is a manufactured product class whose members conform to open agreed standards. Commodity product classes have sufficient characterization to indicate suitability of use. They are specializations of one or more de facto classes, standard classes, or both. The resulting specification is non-proprietary as no one organization controls it. The type of lightbulb known as 60 W 230 V E27 is a commodity product class.

proprietary class is a class whose specification for membership is owned, controlled, or protected by an organization and is not generally available outside that organization.

proprietary product class is a class that is a manufactured product class and a proprietary class. Proprietary product classes are specializations that depend on rules of inclusion and exclusion some of which are controlled in a closed way. This means that some aspects of the specification can be arbitrarily changed. Many proprietary product classes are specializations of commodity product classes, de facto classes, or both, where the additional restrictions reflect design or manufacturing details that the manufacturer uses to differentiate his product from others of the same general type. A product specification that is owned by a commercial organization, and is marketed under and protected by a registered trade name, is the basis for a proprietary product class. An example is lightbulbs 60 W 230 V E27 manufactured by Phillips are members of a proprietary product class.

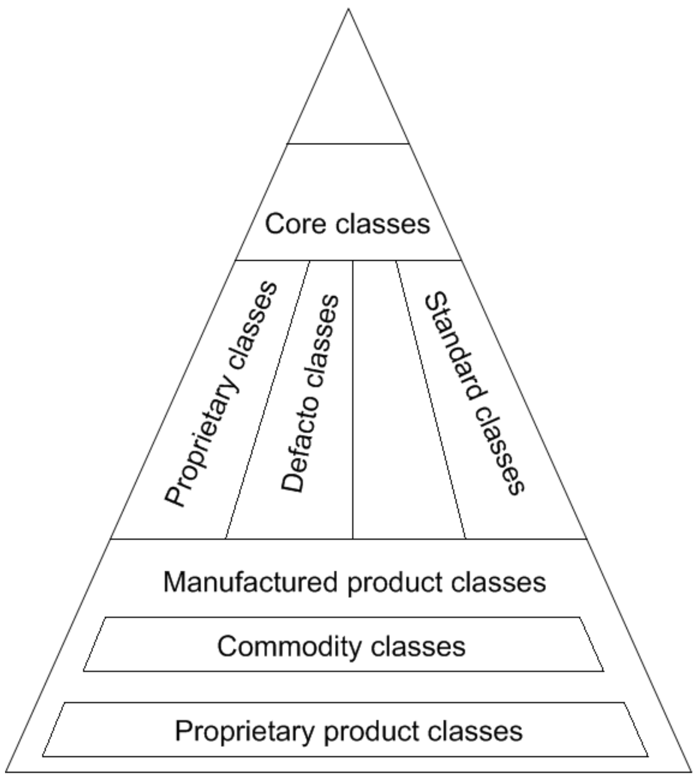

Figure 2.6: (ISO 15926-1:2004, Types of classes) - The position of a class relative to the top and base of the triangle indicates the degree of definition. Classes at the top are general and have few restrictions on membership, whereas those at the base are more specific. Classes at the base of the triangle are specializations of the ones above, and so on up the triangle.

The relationship between the different class types is illustrated in Figure 2.6

2.2 The Semantic Web

In 2005 the POSC Caesar Association (PCA), as a part of the EU-funded project CASCADE, was made aware that the World Wide Web Consortium (W3C) had developed the Web Ontology Language (OWL) knowledge representation language. They had been working on how to make information universally interpretable between all knowledge domains, while POSC Caesar project had focused on specifying a data model that would fit the process plant industries. This discovery led to a change of direction for ISO 15926 towards an interpretation that would fit the tools and services of Web 3.0 as envisioned by the inventor of the Web and founder of W3C, Tim Berners-Lee.

> "I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize.[6]"

The Web which Berners-Lee is referring to is known as the the Semantic Web, or Web 3.0. In order to achieve his vision there is a need to improve on how computers exchange the vast amount of information available on the web while staying compatible with the early web. This improvement is made possible by creating a common communication platform where the computers share their language.

> "The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation.[37]"

This shared language is a collection of concepts that computers can reference when exchanging information. They are organized in ontologies that contains a large collection of precise and hierarchically organized concepts, including their relation among each other. Ontologies enable machines to refer classes in a ontology and thereby specify the exact meaning and context from which its information relates to. So when a machine requests information over the Internet it will get a reply including a reference to the ontology used by its peer. Having such a shared reference to the definition of what the data contains is useful when systems operate in different contexts. While US systems might operate with imperial units, European systems operate with metric units. In this setting the ontologies will provide a link to the definition of the unit in question, making it easy for the peers to write a simple conversion rule. An ontology will therefore act as a knowledge base for a specific domain, like ISO 15926 which originally was designed for the oil and gas industry[1]. An interesting fact is that the ontology is titled:

> Industrial automation systems and integration - Integration of life-cycle data for process plants including oil & gas production facilities.

The developers of the standard now considers this title too narrow because the data model is so generic and its applications so wide that it can model any state information. As a result the ISO15926 has gained interest from other industries that wants to use this generic data model in their information exchange. However, the ontology has not been used or tested in a production environment so there is a reluctance to be the first one out to try out this model, there are several concerns held by the actors.

2.2.1 Potential of integrated operations

In 2007 the Norwegian Oil Industry Association (OLF) published a report estimating a moderate potential value gain of 295 billion NOK in the period 2005-2028 by using Integrated Operation on the Norwegian continental shelf [26]. This estimate is illustrated in Figure 2.7 requires a widespread implementation of Integrated Operation throughout the value chain and has an estimated cost of 24 billion NOK. 78% of the potential is attributed to an accelerated production and growth of reserves as a result of optimizations of production plants and idleness at the installations.

2.2.2 Benefits

The report in the previous section emphasizes the following success factors from the pilot project of full Integrated Operations at the Brage platform.

1. Real-time interaction between operational engineers from different activities and disciplines resulted in quick and effective interaction
2. Specialized tools with historic and real-time data enabled engineers to make better decisions
3. Enhanced digital collaboration with onshore personnel
4. Enhanced well performance due to real-time positioning of the well installations
5. Enhanced seismologic surveillance

[1]Similarly there are other domain specific ontologies like SNOMED which has been developed for the health care industry.

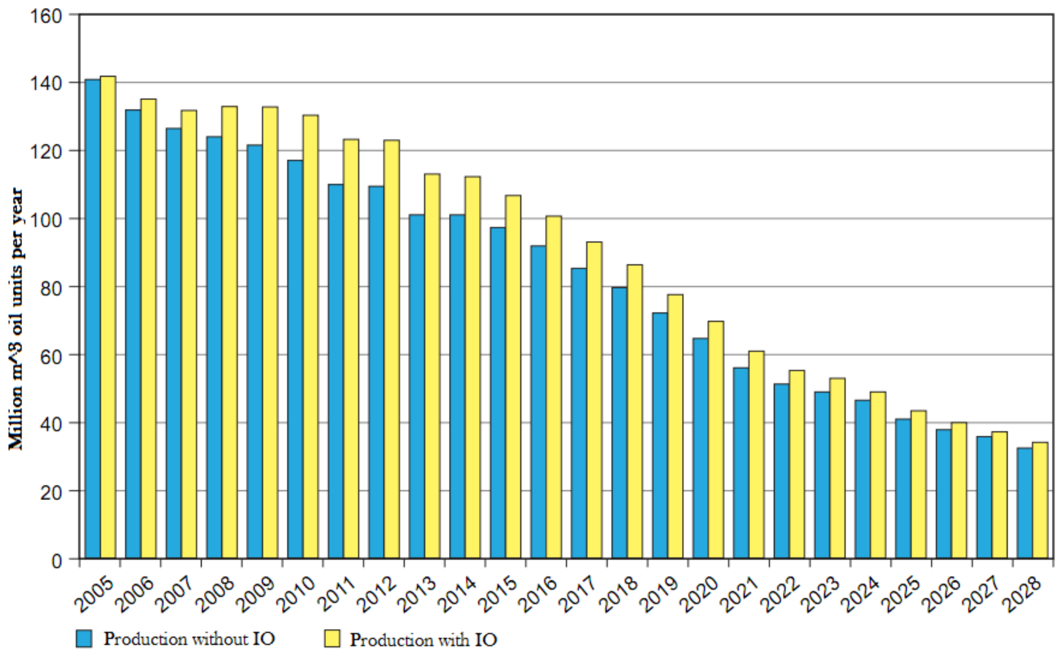

Figure 2.7: Estimated gain of implementing IO in the Norwegian continental shelf.

6. Prolonged life expectancy of the platform due to lower operating costs
7. Enhanced pressure in production wells due to better real-time control of the water injection
8. Enhanced health and safety records due to better control of operations

The Integrated Operation at the Brage platform was implemented by streamlining their operational processes without the use of ISO15926. However, operational processes are just one part of the platform life-cycle. For this reason the Integrated Operations in the High North (IOHN) project was initiated as a joint industry project in May 2008 with a duration of four years. In late 2010 IBM withdrew as supplier of the infrastructure so the project is currently in a stand still.

While there are forces in the industry that is pushing for Integrated Operation, like the OLF, there are also people who are satisfied with how things currently work. It has been said that many of the changes with IO we are seeing happening now, happened more than a decade ago in the automotive industry [14]. There can be several different reasons why the development is so slow, one is that downtime due to integration of Integrated Operation can hav huge costs. Another is that it requires a substantial effort from the workers that operate the facilities, resulting in a loss of production in a highly efficiency intensive industry. David Ottesen, CEO in Ziebel said this about the situation: The technology is here, yes. It's the determination we are lacking. The contracts are designed such that suppliers earn more when things take more time and more people are used. That is the exact opposite of what we should be striving towards." [34]

2.2.3 State of the art

In 2007 Knowledgeweb made an assessment on the current status of the Semantic Web development, acceptance and productivity level [11]. Since then there has been several new additions to the collection of tools, frameworks and services. Although the graph in Figure 2.8 is outdated it gives a good overview of the technologies that has emerged in the last couple of years. The graph shows a classic Gartner hype cycle that provides an overview of the phases a new technology goes through after its inception. One example in the graph is the Semantic annotation that Google has implemented in their search engine [1]. Another example is the semantic-mediawiki project [2] (extension of the popular mediawiki framework) which enable semantic annotations of shared concepts, making the information computer sensible so that user queries can be interpreted more intelligently. The Trust concept has also received much attention in the last years and is going through rapid development, which we will return to later. In essence, most of the technologies listed in Figure 2.8 are highly relevant today.

[1]http://www.heppresearch.com/gr4google

[2]http://semantic-mediawiki.org/

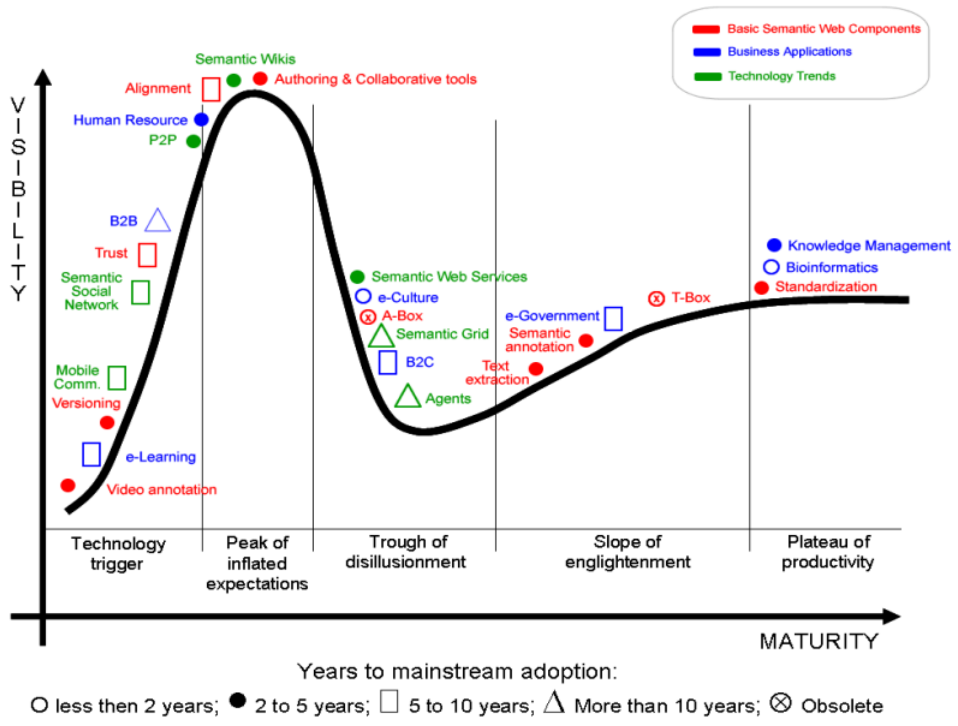

Figure 2.8: A classic Gartner hype cycle for the Semantic Web.

2.3 The Semantic Web Stack

In order for a reasoner to be able to interpret and process the information across information domains, it follows that a common platform is needed. This platform is known at the Semantic Web stack and consists of layers made up of concepts. Figure 2.9 on page 28 is an illustration of how these concepts are related. Each concept is implemented by a technology, standard or a data model capable of fulfilling the role of the concept. Some of these implementations can live their life's of their own, they are independent, meaning that they might change their direction of development in a way that disfavors the Semantic Web stack. Even if this is unlikely to happen, it is necessary to emphasize that the implementation of a concept is not of importance because, in theory, they are independent from each other. However, in practice it might prove difficult for an organization if it chooses to use a different implementation of a concept than its partners. With this in mind we will have a look at the Semantic Web stack proposed by Steve Bratt in 2007 [7].

2.3.1 URI/IRI

Starting from the bottom left of Figure 2.9 we have the *URI/IRI* block consisting of two standards that deals with resource addressing on the Web. URI's[1] are commonly known as URL's[2] that is used by the web browsers to access web pages and other Internet resources like www.example.com. We can think of a resource as an object, a "thing" we want to talk about. Resources may be authors, books, publishers, places, people, hotels, rooms, search queries, and so on, or the concept of "Ambient temperature" in figure 2.2. Every resource has a URI, a Uniform Resource Identifier [1]. The IRI's[3] enables the use of international characters including Chinese and Arabic characters that earlier wasn't possible with the characters limited from a to z. The URI/IRI block is therefore an essential part of the Internet that allows us to locate and interact with other computers.

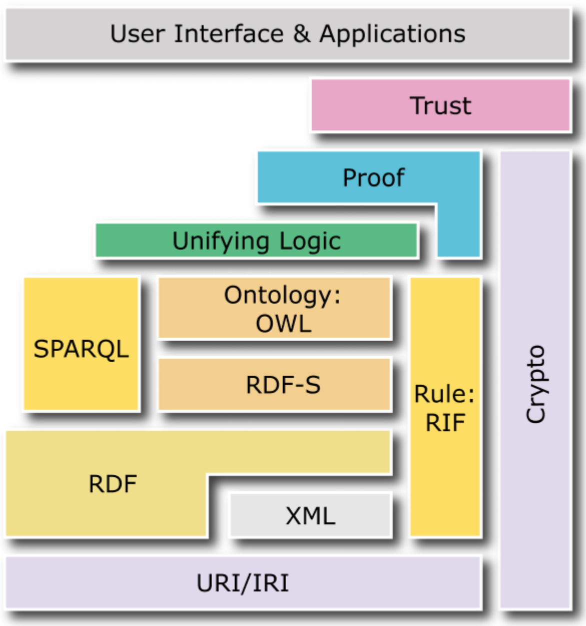

Figure 2.9: The Semantic Web Stack.

[1]Uniform Resource Identifier's is a scheme that defines the Internet addressing protocols.

[2]Uniform Resource Locator's is the set of addressing protocols that conforms to the URI scheme.

[3]Internationalized Resource Identifier's is an extension of URI from the US-ASCII character set to the Unicode/ISO-10646 character set.

2.3.2 XML

Climbing one level up in the middle of the Semantic Web stack we find the eXtensible Markup Language (XML) block. XML deals with the document structure in the information transfer. Considering that there is a variety of document formats and standards available to represent different kinds of information, they are usually optimal for their own applications. Like the mp3 file format which is suitable for acoustic information, or the doc file format which is suitable for word processing. However, there is a need for a file format that can represent any information in a common format that is understood by all. The XML file format is simply a extensible language that acts as an information carrier. The information contained in an XML document can be validated with an eXtensible Markup Languages (XML-Schemas) which is part of the XML standard. XML-Schema defines the syntax required for a XML document. This allows us to define what data types are allowed for the information contained within an XML document with a XML markup tag. E. g. when we reference a website in an XML document, we know that certain characters have special meaning and cannot be used in the URL string such as # % & * { } \ : < > ? / +. In this way XML-Schema allows us to express the rules of syntax for XML.

2.3.3 RDF

The previous paragraph described XML as a universal metalanguage for defining markup. It provides a uniform framework, and a set of tools like parsers, for interchange of data and metadata between applications. However, XML does not provide any means of talking about the semantics (meaning) of data. For example, there is no intended meaning associated with the nesting of tags; it is up to each application to interpret the data. This can be done with Resource Description Framework (RDF), which essentially is a data model. RDF is a W3C specification which describes a model on how information is represented. It is the most important part of the stack and Tim Berners-Lee has expressed that he views it as the general model of the Semantic Web [4].

Statements assert the properties of resources where a statement is an object-attribute-value triple, consisting of a resource, a property, and a value where values can either be resources or literals. There are three ways to view a statement, the first alternative is the triple notation in form of a graph. One way of viewing a single triple of linked data is by using variables (x, P, y) that can be expressed a logic formula

$$P(x,y)$$

where P is a binary predicate that describes how an object x relates to an object y. This notation is in accordance with expressions written in First Order Logic (FOL).

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"
xmlns:posccaesar="http://www.posccaesar.org/rdf/">
  <rdf:Description rdf:about="http://www.posccaesar.org/x">
    <posccaesar:P
      rdf:resource="y"/>
  </rdf:Description>
</rdf:RDF>
```

Viewing the data in a graph is useful when presenting a moderate sized collection of under 50 triples, it can however become incomprehensible with large collections. In the Artificial Intelligence (AI) community the graph perspective is also known as a Semantic Net.

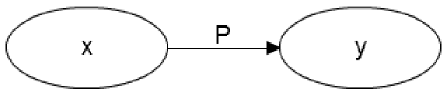

Figure 2.10: A single triple shown in W3C RDF style diagram.

This reified allows creating classes (organized in a hierarchy) and properties (with a domain/range), thus forming a primitive ontology language [1]. OWL offers a rich knowledge representation languages compared to RDF-S, which had proved too simple [6].

ISO15926 is based on the same reified structure as RDF. The following schema in Figure 2.11 shows a EXPRESS diagram of a pressure transmitter, type 3051CG. The information in the schema itself is not of interest here, but the structures and their composition is the important part of this example.

On the far left there is the name of the class in question *3051CG* with its relation to a *Temperature range* (between -45° and 85°) through a *class of* relation. The underlaying logic of this little part of the diagram is a statement of a triple where the subject(*3051CG*) has a predicate(*Indirect property*) to an object(*Temperature -45°-85°*). Starting from the *Indirect property* and going upwards we can see that the triple-pattern is repeated with the classification to the *Ambient temperature* which is giving meaning to the predicate; it is of type "Ambient temperature". This process can be repeated throughout the diagram where every box is a valid entry in the ontology it refers to. The boxes are indicating "type" (property range) and "value", e.g. (temperature range -45°- 85°). This model is similar to the ISO 15926 with the use of triples to represent metadata and data in triples with a subject, predicate and object as the example explained by Figure 2.11. This representation is desirable because it provides a data structure that enables the use of rules. The rules will in turn allow reasoning and discover new facts that is deductible from the data. The deduction of new

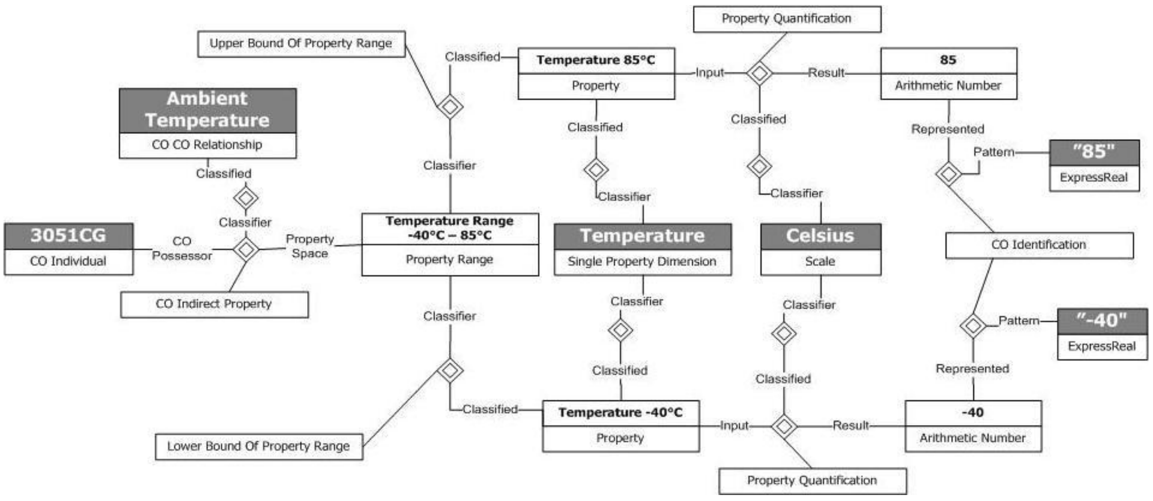

Figure 2.11: Example of a Triple.

information can be done by applying FOL, we shall however see that the use of higher order logic is required to cover industrial needs. This is an important goal of the Semantic Web, the notion that computers should be able to interpret and reason on the behalf of humans by using a set of rules.

2.3.4 RDF-S

The name RDF Schema is now widely regarded as an unfortunate choice [1]. It suggests that RDF Schema has a similar relation to RDF as XML Schema has to XML, but in fact this is not the case. XML Schema constrains the structure of XML documents, whereas RDF Schema defines the vocabulary used in RDF data models to express relationships. Resource Description Framework Schema (RDF-S) is an extensible knowledge representation language that is used to structure RDF resources and provide a vocabulary/ontology using RDF, making it a general purpose language for representing information on the web by enforcing RDF document structures. However, RDF-S has a limited ability to represent advanced knowledge structures. It can not express relationships like disjointness among classes, property characteristics symmetry and cardinality restrictions (among other).

The fundamental concepts of RDF-S are resources, properties and statements. Resources are defined by classes that represents an entity such as "Person", "Planet" and the "IP-66" standard. Properties are a special kind of resources that describe relations between resources, for example "manufactured by", "has unique id", "own by", and so on. Properties and resources are identified by URIs (and in practice by URLs). This idea of using URIs to identify "things" and the relations between them is quite important. RDF has a decentralized philosophy and allows incremental building of knowledge, and its sharing and reuse. This choice gives a worldwide, unique naming scheme. The use of such a scheme greatly reduces the name space problem that has plagued distributed data representation.

2.3.5 Ontology: OWL

OWL is a family of knowledge representation languages, namely OWL-Lite/DL/Full. It is based on RDF/XML serializations.

1. OWL-Lite - intended for users that has a simple thesauri or taxonomy-based systems. It was meant to act as a gateway for institutions that wants to try out semantic technologies without having to commit to large changes. OWL-Lite supports hierarchical classification and simple class constraints like cardinality, although limited to the values 1 and 0 (this allows setting limits on classes by for example specifying that a *person* must have exactly one *mother*. As a result it is not used very much other than for educational and for simple testing purposes. Since it is based on the DL $\mathcal{SHIF}(\mathcal{D})$ semantics it is a subset of OWL-DL [31], thus making it capable of interacting with OWL-DL applications.

2. OWL-DL - is a more complex language based on description logic that is more expressive than OWL-Lite but has some trade-offs compared to OWL-Full. It supports the users that wants maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with description logics; namely the $\mathcal{SHOIN}(\mathcal{D})$ semantics, a field of research that has studied the logics that form the formal foundation of OWL.

3. OWL-Full - is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. Its semantics is beyond $\mathcal{SROIQ}(\mathcal{D})$ and it is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations hold. Their inverses do not [27].

1. Every legal OWL Lite ontology is a legal OWL DL ontology.
2. Every legal OWL DL ontology is a legal OWL Full ontology
3. Every valid OWL Lite conclusion is a valid OWL DL conclusion
4. Every valid OWL DL conclusion is a valid OWL Full conclusion

Figure 2.12 shows the level of expressiveness with respect to knowledge representations we have reviewed until now. Ontologies will be revisited in section 4.1.

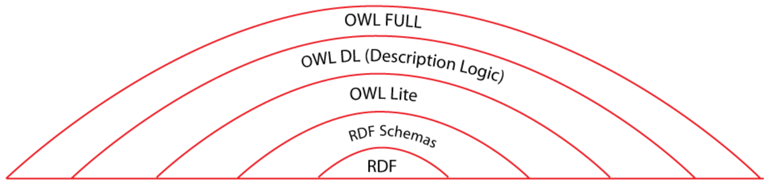

Figure 2.12: Knowledge representation languages.

2.3.6 Rule: RIF

Rule Interchange Format (RIF) is a rule interchange standard and no specific implementation. It arises from the experience from the fact that rules provide a useful method of making inferences on large and complex information sources. Its purpose is to allow systems to exchange rules that define their business logic that augments the shared information model and providing the ability to make them available through a shared format. Such rules is used by the reasoner to make inferences about information that is necessary for a business, but does not exist in the ontology. Rule languages like SWRL and RuleML implement RIF in such a way that allows an actor to publish its f. ex SWRL-rules in order for another actor to translate the rules into f. ex RuleML-rules. Several dialect are available like the RIF PR that is optimal for business rules, RIF BLD for logic programming and RIF Core which covers both dialects. The most widespread implementation is Semantic Web Rule Language (SWRL) which is a subset of the RIF Basic Logic Dialect (RIF-BLD) format. SWRL has been built as an extension of OWL and provides the capability to specify Horn-rules. It is currently pending as a proposal at W3C to be accepted as a Semantic Web rule language. Although rule support is clearly a desirable feature, there is no general commitment of implementors of OWL reasoners to a particular syntax or semantics, although SWRL rules are the most commonly used ones [16]

E. g. if an American business uses the imperial system want to connect to an European business that uses a metric system in their catalogs, there will be a conversion issue. Rules will allow them to convert values to their respective standards. A rule can be written as follows

```
    Equipment(?x)
    ∧ hasString(?x, ?system)
    ∧ swrlb:equals(?system, "metric")
    ⇒ customswrl:conversion(?x, ?system, "imperial")
```

In this rule we establish if x is a member off the class **Equipment** and determine if it has a string field named system and if its a string. Given that the conditions hold, then we can perform the conversion. When leaving out Crypto, the Rule block spans five other blocks in its height , namely the XML, RDF, RDF-S, SPARQL and Ontology blocks. It has applications within all of these because, even as an extension of OWL, it has an interpretation through RDF and RDF-S and XML in order to be serializeable and interchangeable between systems.

2.3.7 SPARQL

is a RDF query language and a data access protocol that interface with information stored in a RDF data structure. It can be represented as native RDF or any middleware that can be viewed as RDF triples. Similar to RDF it is Triple-oriented and enabled queries of conjunctions, disjunctions and optional patterns. It also have Negation As Failure (NAF) through the *NOT EXISTS* keyword added in SPARQL Protocol and RDF Query Language (SPARQL) 1.1. The following code is an example of a query on the ISO 15926 SPARQL endpoint that lists all classes that has a superclass designation containing the string "PUMP".

```
select ?superclass ?supDesignation ?subclass ?subDesignation
{
    ?superclass a p2:ClassOfClassOfIndividual .
    ?superclass RDL:hasDesignation ?supDesignation .
    FILTER (fn:contains(?supDesignation, "PUMP"))
    ?cls p2:hasSuperclass ?superclass   .
    ?cls p2:hasSubclass ?subclass .
    ?subclass RDL:hasDesignation ?subDesignation .
}
```

There are several implementations of this protocol such as, Sesame, Virtuoso, Jena TDB and Jena SDB. It is worth mentioning that SPQRQL is a highly query intensive language that does not scale very well for large implementations. In 2009 the research publication portal Elsevier announced the "Billion Triples Track" Semantic Web Challenge of 2010[1]. The goal

[1]http://challenge.semanticweb.org/

was to present an architecture that could scale up deal with at least one billion relations between instances in the dataset gathered from the open web. SPARQL has been and will probably continue to be a bottleneck in the Semantic Web. Updated information and benchmarks can be found at the W3C RDF Store Benchmarking Wiki page[1].

2.3.8 Unifying Logic

Throughout the years since the Semantic Web layer cake was introduced it has undergone several changes. During one of these changes the Unifying Logic block has been put on top of the SPARQL and Rule layers where it previously only covered the Ontology. As a result of this there has been done a fair amount of research on reasoning on the Ontology layer, some on the Rule layer, but none on the SPARQL layer. There has been little progress on this topic since then, but this is starting to pick up. The idea is that existing reasoners also can be capable of directly accessing and interpret the rules and data [5] - enabling existing DL reasoners to access the entire knowledge base. Most DL reasoners now support Ontology and, to some extent, the rule layer in this architecture. However, there are no support for SPARQL. The developers of Pellet is proposing a new approach for reasoning on SPARQL data as well, this contribution will be investigated in section 4.1.3, Reasoners

2.3.9 Proof

The Semantic Web aims to make data available as computer interpretable information that allows free interaction between systems. This expansion of information flow between systems requires improved control mechanisms that guarantees algorithm tractability, system authentication and certification, access control, version control and document signing. Very little work has been published on this area although some languages have been proposed, like the Proof Markup Language (PML) [33].

2.3.10 Trust

Trust can be understood as an extension of proof, and the two are closely related subjects that are often mentioned in the same sentence. Proof deals with mechanisms involving information exchange, while trust deals with data auditing through data provenance, networks of services and how to handle them. Issues involve handling trust by auditing information contributors of in a network of web services, discrepancies between web services, and social network analysis [23]. The semantic web has incorporated the matter of trust into the architecture, but it is not a new concept. The most

[1]http://esw.w3.org/topic/RdfStoreBenchmarking

famous example is the Google Pagerank algorithm that used positive reinforcement of trust to websites that had a high count of hyperlinks pointing to that domain, and negatively reinforcing websites that had a high count of hyperlinks pointing outwards. The matter of trust has ever since, and continue to be a highly competitive arena where businesses and other groups compete for visibility on the web that gain their interests.

2.3.11 User Interface & Applications

The social web is currently a strong force in development of user interfaces and applications in the Semantic Web. Users are starting to see the benefits of linking their data as the amount of information is growing, and this has become evident on social websites like Facebook where users can tag other people in images, include them in status updates and messages etc. There are many challenges and new ideas that the general web can benefit from this development, and especially for small factor devices. Readers that want to know more about the status, ambitions and research on this topic can have a look at the website of Visual Interfaces to the Social and Semantic Web arranged by the International Conference on Intelligent User Interfaces at Stanford[1].

2.3.12 Crypto

Security on the Internet is, and will continue to be a major concern. This topic will not be discussed further in this thesis, but interested readers can have a look at the work of Raman Pal, which in January 2011 completed an comprehensive study on the topic[2].

2.3.13 Issues not covered

This chapter could arguably have presented OWL axioms and its corresponding $\mathcal{SHIF}(\mathcal{D})$, $\mathcal{SHOIN}(\mathcal{D})$ and $\mathcal{SROIQ}(\mathcal{D})$ DL semantics. It was considered too comprehensive and contributing little to understanding of this study.

[1]http://www.smart-ui.org/events/vissw2011/

[2]http://www.scribd.com/doc/65529492/11/Cryptography-in-Semantic-Web

Chapter 3

Information quality

In the previous chapter we elaborated on the data model, technologies and data exchange of the Semantic WEB architecture. In this chapter we will start defining an information quality scale that measures level of accuracy we can achieve.

Information quality is difficult to measure because it is relative to the customer and users of the system. While some users define quality as having accurate and explicit data, others might emphasize availability and amount. There is also a matter of performance of the data services and the amount of effort it takes to be able to use the system. Information quality is therefore not a feature that applies to all users of the system, rather a subjective subconscious process that reflects the users perspective [13, p. 16]. There has been numerous attempts from different disciplines to define a universal definition of information quality, some more successful than others.

In this study we will look into ISO 8000, a standard founded in 2006 which is devoted to implementing a shared definition of information quality levels in automated computer processes. It is important to note that the ISO 8000 standard is still undergoing development and there are ongoing debates within the community on where the development is heading. Still, it will act as a guideline by referring to shared concepts and definition on information quality, and provide a scale for measuring outcomes of different tools and tests from the published work.

Later in section 4.2, Validation Tools & Software we will perform a review of the current tools and software to carry out RDF data validation and determine the best candidate.

3.1 ISO 8000

The ISO 8000 standard is aimed at quality facets of automated information exchange for the purchase of goods. It defines formats for descriptions of individuals, organizations, locations, goods and services using fifteen char-

Part	Description
1	Data quality—Part 1: Overview, principles and general requirements
100	Data quality—Part 100: Master data: Exchange of characteristic data: Overview
102	Data quality—Part 102: Master data: Exchange of characteristic data: Terminology
110	Data quality—Part 110: Master data: Exchange of characteristic data: Syntax, semantic encoding, and conformance to data specification
120	Data quality—Part 120: Master data: Exchange of characteristic data: Provenance
130	Data quality—Part 130: Master data: Exchange of characteristic data: Accuracy
140	Data quality—Part 140: Master data: Exchange of characteristic data: Completeness
150	Data quality—Part 150: Master data: Exchange of characteristic data: Quality management framework

Table 3.1: The published ISO 8000 parts.

acteristics. Table 3.1 lists the published parts, where the name reflects the location of the definitions. These characteristics can be divided into three categories, namely; Syntactic quality 3.1.2, Semantic Quality 3.1.3 and Pragmatic quality 3.1.4. It is important to note that none of these does not adhere to any specific implementations and is therefore not bound to any language.

3.1.1 ISO 15926 & ISO 8000 - Master data

ISO 8000 part 120 defines master data as data held by an organization that describes the independent and fundamental entities for an enterprise [20]. This is where ISO15926 fits in with ISO 8000 because ISO15926 is a data model of the master data, that in this thesis, is implemented with semantic technologies. The system interoperability that allows organizations to automate their information exchange is implemented through the mechanisms of ISO15926 templates. Templates are highly explicit data sheets that contain data fields for all the possible information field that a class can have.

Figure 3.1 illustrates the process of mapping field/values that enables translation from one proprietary data sheet from to another. This process of moving data from one format to another through a shared format is known as Lifting & Lowering, a concept from the artificial intelligence community [36]. It is a simplified illustration of how the field/value mapping translation of the pump example shown in Figure 2.11 could have been done between

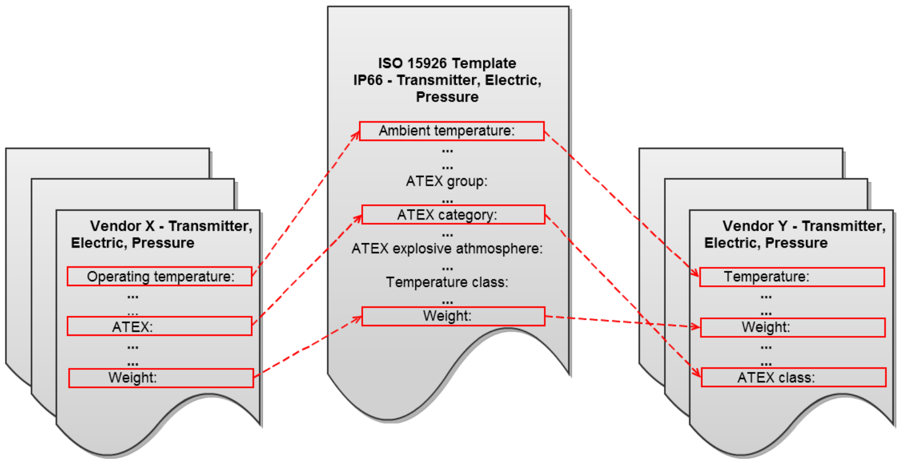

Figure 3.1: Example of data sheet mapping, the process of moving data is known as Lifting & Lowering.

two vendors. Starting from the top field of the data sheet of Vendor X to the left we have the "Operating temperature" field. It has been mapped to the corresponding template field in the ISO15926 explicit format. Similarly Vendor Y has mapped the "Temperature" field of their competing product. When engineers have mapped all the data sheet fields of their product catalog to the corresponding ISO15926 class, they will be part of a information sharing hub that allows customers to connect and find the best products that fit their needs.

3.1.2 Syntactic quality

It is treated in ISO 8000-110:2009 [19] as listed in Table 3.1. The information quality of syntax is a basic necessity in any information model and addresses the need for properly formatted data. The following characteristics on syntactic quality is therefore a definition that addresses checking of data according to its metadata

Accuracy The extent to which data has attributes that correctly represent the true value of the intended attribute of a concept of event in a specific context of use. E. g, checking that data field values corresponds to the data type assertion such as; an integer is asserted where a integer data type is expected. The same type/value rules applies to other data types such as; unsigned integers, 32-bit integers or other data types like characters or strings

Compliance The extent to which data has attributes that adhere to standards, conventions, or regulations in force and similar rules relating to data quality in a specific context of use. E. g, given a telephone number, the number sequence can be checked to verify that it complies to a telephone pattern, if such a pattern has been defined in a class. Compliance is also a matter of verifying that values conform to data ranges, e. g. that an operating temperature of an individual is within the required range specified by its class. Similarly, ontologies can (and often do) contain contradictions such as a declaration that class **A** is *sameAs* class **B**, if class **B** then have been declared as *differentFrom* - then we are able to conclude that the ontology is inconsistent

Completeness The extent to which subjects associated with an entity have values for all expected attributes and related entity instances in a specific context of use. E. g, in order for the information to be computer processable rich information is needed such as an operating temperature although it might not be explicitly declared as required. Required fields is of special importance because there is no guarantee that they have been entered correctly or fulfill the specification

Syntactic correctness is a requirement that easily can be automated with simple checks. Most publicly available reasoners offer syntactic checking on the ontology and individuals without requiring much effort, it is therefore considered to be free functionality

3.1.3 Semantic Quality

Is a measure on how the meaning of data quality is preserved on a conceptual level. This harder to accomplish because the subject is less computer sensible.

Correctness The extent to which data is used according to its intension

Consistency The extent to which data has attributes that are free from contradiction and coherent with other data in a specific context of use.

Precision The extent to which data has attributes that are exact or that provide discrimination in a specific context of use

3.1.4 Pragmatic quality

Pragmatic data quality relates to how the data is used, maintained, accessed and so forth. This level of handling data quality is more a business logic rather than strict data requirements. It has been taken into consideration of this measure of information quality because it is still important aspects on how the ISO15926 overall quality is perceived by the users.

Accessibility The extent to which data has attributes that enable it to be reached in a specific context of use, particularly by people who need supporting technology or special configuration because of some disability.

Currentness The extent to which data has attributes that are of the right age in a specific context

Credibility The extent to which data has attributes that are regarded as true and believable by users in a specific context of use

Confidentiality The extent to which data has attributes that ensure that it is accessed and interpreted only by authorized users in a specific context of use

Performance The extent to which data has attributes that can be processed and provide the expected level of performance by using the appropriate amounts and types of resources under stated conditions and in a specific context of use

Traceability The extent to which data has attributes that provide an audit trail of accesses to the data and of any changes made to the data in a specific context of use

Understandability The extent to which data (and associated metadata) has attributes that enable it to be read and easily interpreted by users and are expressed in appropriate languages, symbols, and units in a specific context of use

Availability The extent to which data has attributes that enable it to be retrieved in a specific context of use

Portability The extent to which data has attributes that enable it to be moved from one platform to another, preserving the existing quality in a specific context of use

Recoverability The extent to which data has attributes that enable it to maintain and preserve a specified level of operations and quality, even in the event of failure, in a specific context of use.

In the next chapter we will now turn to how we can model and test these criteria.

3.1.5 Provenance

In addition there is another important concept from ISO 800 worth mentioning. Provenance is a measure that describes statements about statements. This topic has been devoted special attention in ISO 8000 and should have been assessed heavier in the development of ISO15926 according to Johan Klüwer. Provenance is the data quality principle that requires a description of suppliers, customers and entities that relates to the information automation process. Say, for example, that an engineer declares that a given "pump" is a **member_of** the **hydraulic_pump** class. This constitutes a statement of the triple notion, and we would like to know who made that statement. Since **member_of** is a class we could make a perpendicular to that statement saying that **member_of** is part of another statement saying that **member_of - declaredBy** - "someEngineer". This pattern can continue on and on for any statement that has additional information attached to it. We will not discuss this further, but it is yet another reason why UML object properties needs to be classes with relations.

3.2 Related work

Martin Giese is a post-doc researcher at the department of Informatics at the University of Oslo and has been developing RDF quality criterias for applications of the ISO15926 standard. His work focuses on developing requirements for a Best practice implementation of Templates and its coherence with existing reference data.

Chapter 4

Validation of ISO 15926 data in the Semantc Web architecture

4.1 Ontologies

This chapter relates back at the discussion of shape vs material in section 2.1.8, Representation of "Attributes" using ISO 15926. Ontologies touch upon a fundamental question within metaphysics that seeks to discover the natural categorization of things. Several great philosophers have pondered the ontological question, including Descartes, Kant, Ockam and Gadamer among others. The topic of whether an object is defined by its shape or material will not be discussed further here. Still, it is reveals some differences relating to properties in how ISO 15926 is used in OWL with regard to reification. First, we will define some key terms that make up the OWL language.

Class - coincides with the definition of class in ISO 15626, see *class* in paragraph 2.1.10.1 [21]

Instances - coincides with the definition of instance in ISO 15626, see *instance* in paragraph 2.1.10.4 [21]

individual - coincides with the definition of individual in ISO 15626, see *individual* in paragraph 2.1.10.3 [21]

Property - in Owl a property is a binary relation that states relationships between individuals or from an individual to data value. Property can be further distinguished as "ObjectTypeProperty" that is defined as the relation between instances of two classes. "DataTypeProperty" is defined as the relation between instances of classes and literal values such as string, number, and date.

4.1.1 Representing ISO 15926 in OWL

As mentioned earlier ISO 15926 originated from the industry, OWL on the other hand is a language designed for general knowledge representation. From now on we will be referring to OWL whenever OWL-DL is mentioned. One of the major differences is that the OWL vocabulary only have two levels of expressing class-individual relationships. While ISO15926 has an additional level known as a meta-class[1] that is used to group a collection of classes. Lets clarify these three important levels with an example; assume that you have three pets at home, for example an eagle named sam, a cat named itchy and dog named scratchy. These three pets exist, have names, belong at your address, etc - and we refer to them as individuals of the zero order. Each pet relates to their class of species namely **eagle**, **cat** and **dog** which is of the first order. Now here comes the point, if you want to group your pets into classes like **endangered_animals** and **domestic_animals** - you will need a third level of classes. Second order classes allows you to create a meta-class that lists other classes as members across the ontology hierarchy. [2]

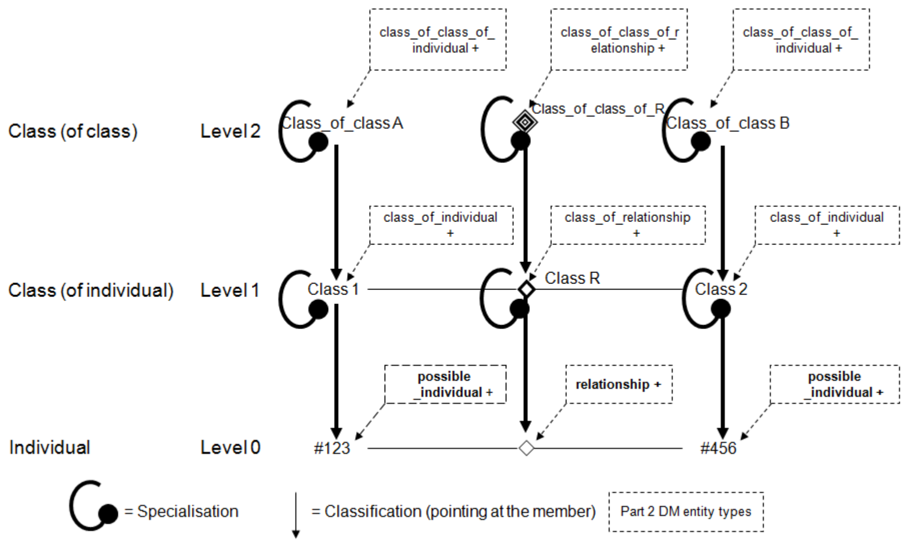

Figure 4.1: The diagram shows zero, first and second orders classes when following one of the three vertical paths.

[1]OWL-Full and OWL-DL 2 has the property *owl:sameAs* for representing meta-classes, OWL-DL 1 has none.

[2]The correct nomenclature for classes of the second order is class-of-class, turning the examples above to **class_of_endangered_animals** and **class_of_domestic_animals.**

The diagram shows zero, first and second orders classes when following one of the three vertical paths. The three horizontal paths shows possible combination of a subject-predicate-object (triple). The six rings indicate that first and second order classes can have specializations within the same order, but never between object/subject and a relation. Note that classifications are made vertically between 0 and 1, 1 and 2 - but not directly between 0 and 2. This is because individuals cannot be a class-of-class. Put in other words, you cannot have create a unique individual *my_car* from a second order class **class_of_coupes**, it has to be created of a class like **saab**, which again can be a member of **class_of_coupes**.

4.1.2 Reification

The EXPRESS data modeling language never caught the attention of the industry and is rarely used today. Still, large parts of ISO15926, and especially its part 2 were modeled in EXPRESS, which is in contrast to the Semantic Web community that based itself upon UML and later a UML profile known as ODM. As mentioned in section 2.1.7 UML has severe limitations, so rather than migrating to ODM ISO15926 has focused on mimicking the EXPRESS norm directly into an OWL translation [24]. This way of transforming the data models is important because it preserves the Triple notation of subject -¿ predicate -¿ object of the ISO15926 core data model. Looking back to the Semantic Web layer cake 2.9 we can see that OWL is based upon RDF, which is a triple notation format. As of OWL 1 reification was a ontology design pattern that was rarely used because it requires developers to make comprehensive abstractions of class relationships that had little or no application to their specific modeling requirements. As a convenience developers were able to create classes that innate properties declared within the class and avoid the issue. This approach is nevertheless not sustainable when ontologies from two different knowledge domains needs to be merged. Then a point of view problem suddenly arise, leading to the metaphysical question about whether an class is defined by its shape or material. In this sense ISO 15926 enforces a strong reification, meaning that the developers need to make classes that define both form and material, then again ISO 15926 provides this through an extensive library of definitions so that developers do not have to make reference data. As soon developers to use object properties, they will defy the expressiveness and usefulness of ISO 15926.

A reified data structure can easily be visualized as a directed graph consisting of nodes, where each node in the graph can have the role of being a subject, predicate or object. As we discussed in section 4.1 the ontology serves as a knowledge representation language that enforces the rules of class membership for any given individual during its life-cycle. An individual can go through multiple changes during its life span where relations to other individuals can be added, changed or removed. There is also a possibility that the requirements for class membership of an individual in the ontology itself changes over time, causing inconsistency between the data and its data structure, lets consider an example the from the IT-industry. In April 2009 the IT company Oracle acquired its competitor Sun, they immediately re branded the software portfolio of Sun Java systems to Oracle Java systems. With respect to an ontology such a transition will require changes because Sun will cease to exist as an individual in the database.

4.1.3 Reasoners

The Unifying Logic layer was introduced in section 2.3.8. In brief we can say that from a historical perspective there has been done a substantial amount of research on Ontology reasoning and many useful tools and ideas have been introduced in this layer because of the previous arrangement of the Semantic Web stack. Methods and algorithms that originated from the fields of mathematics and artificial intelligence was being applied to the Ontology layer to infer new knowledge and detect inconsistencies. There forward and backward chaining algorithms have two complementary strategies to traverse linked data structures, where they also have complementary strengths and weaknesses. There is also a third option that uses a combination of the two which is known as Hybrid chaining. We will briefly compare these alternatives when they are applied to linked data structures represented in the SPARQL and Ontology layer.

Forward chaining is a data driven strategy that uses modus ponens to add inferences by statements. It is a rule where an antecedent P is followed by a consequent Q in an If-then statement. When applied to linked data we can make inferences throughout a Ontology or SPARQL graph represented in RDF. Forward chaining has its name from its strategy of iterating through existing data and, adding new inferences where the data supports it.

Backward chaining is a goal driven algorithm that uses modus ponens by the same principle as with forward chaining. The difference lies in its opposite strategy of starting with the consequent and working backwards to see if is supported by the antecedent. This strategy provides a method of proving that a given goal is correct.

There are more to reasoners than just forward and backward chaining, and there have been written numerous paper on the subject [39][3][8][12], so it will not be discussed any further. What is important to learn from this is that there has been done a considerable amount of work on ontology reasoning that includes checking for circular dependencies, inconsistencies, data type validation and data range validation. In essence, the syntax data quality perspective is quite mature and is supported by most reasoners as out-of-the-box functionality.

Later, in 2007, the SPARQL and Rule layers was aligned with the Ontology layer so that all three is arranged underneath the Unifying Logic layer. This rearrangement has opened up the data (SPARQL) and business logic (Rules) to be accessed by the reasoner. The Gartner Hype cycle that was introduced early in Figure 2.8 showed that the Semantic Web hype was at its peak in the same year, there have also been a falling interest of research in the Semantic Web stack. As a result of this it seems that the development of Semantic Web tools lost some of its momentum from 2007-2009. Nevertheless, there has been progress in the academic circles that has continued this research and this is now starting to appear.

Hybrid reasoning both Forward and Backward chaining are well established algorithms within the field of artificial intelligence, but they have limited applicability on multi-layered models. While they are able to perform reasoning on the 0. order level of SPARQL data and on the 1. order level of Ontologies represented in RDF they, lack the possibility of reasoning across the levels. Recently there has been published research on the hybrid reasoning approach that combines these levels to validate multi layered models by applying the CWA [29]. Hybrid reasoning can be accomplished by the technique of punning which enables the use of a single name when referring to individuals and classes, respectively of the 0 and 1st order, potentially up n'th -1 and to n'th order. This also opens up the possibility for reasoning with 1. and 2. order classes. Pellet is currently the only reasoner that supports punning [25] with the use of OWL-DL.

Hybrid reasoning allows us to locally close parts of the ontology and SPARQL data in the same namespace and allows us to write integrity constraints, which we will return to after the following section.

4.1.4 Open vs. Closed World Assumption

Reasoners are able to infer new facts from the database because of the assumption that the information contained in the database is not the whole truth about the world, it is just a limited set of facts within a limited domain. Whatever information not contained within the database is simply unknown, and unless there are any evidence that can assert the value of a statement, it is left as an open question. This is known as the OWA and is in sharp contrast to the CWA of traditional relational databases where any assertion that cannot be proven true, is regarded as false. This assumption is closely related to the non-monotonic inference rule NAF which is the basis for all databases with a CWA [36].

The CWA is intuitive to anyone that has experience with relational databases and it is very useful for appliances that has complete knowledge about a limited domain. But in order for a relational database to be provably consistent it is required to be expressed in Horn form, and is not guaranteed otherwise. [28].

For example, the following knowledge base that *pump* is entailed to be neither electric nor hydraulic, but still

```
{Electic(pump) V Hydraulic(pump)}
```

In order for reasoners to infer new facts from a given set of explicit facts, a decision has to be made by the in advance with respect to how missing facts in a database are interpreted. SPARQL-based databases are designed and intended to be OWA, while traditional relational databases are predominately CWA. The CWA is closely related to the non-monotonic inference rule known as NAF in logic programming where data about the domain is said to be complete for the observer when what is not asserted to be true, is assumed to be false [36]. This means that when a database query returns an empty set of results, then whatever was asserted, is considered to be false.

4.1.5 Integrity constraints

In The concept of IC stems from the field of relational databases where IC helps ensuring data consistency [2]. It is an important functionality which nearly all database systems support. The method consists of checking all ICs defined in the database schema and translate the constraints to corresponding queries. During a database query these queries are executed to verify that no ICs are violated, and therefore avoiding any potential constraint violations. Such methods would be useful for evaluating data in SPARQL databases as well; modeling the ICs into the knowledge domain by using existing knowledge representation languages like ISO15926 (as with OWL [22]) then translating the IC axioms into queries and validate the ICs based on the query. This idea touches upon two concepts within the semantics of DL; namely the closed vs. the open world assumption, and the UNA. Traditional relational databases practice the CWA by asserting that any query that can not be proven true will be considered false. If we were to search a OWA database for a pump made by a manufacturer named ACME

```
:isManufacturedBy rdfs:range :Manufacturer .
:product1 :isManufacturedBy :ACME .
```

Unique Name Assumption Another important ontology concept is the UNA that stems from DL. It states that different names refer to different entities in the knowledge base. This means that we cannot uniquely identify an entity by its name. OWL does not have UNA, instead there are reserved keywords that allows the ontology engineer to specify same or distinct entities by using *owl:sameAs* and *owl:differentFrom* when implementing classes.

4.2 Validation Tools & Software

There is quite a selection of reasoners available . In the following paragraphs we will review the existing tools that later will be summarized and presented in table 4.2. As we will see, for obvious reasons Pellet is the preferred option and was also chosen.

4.2.1 On-line validators

Since 2001 the W3C has provided a validator[1] which parses documents and verifies that they conform to the syntactic requirements of RDF/XML standard. The WEB page accepts a URI to a public document or text inserted into the WEB browser. It does not check if it is in accordance with a given RDF Schema Specification - therefore it cannot validate the instances up against the ontology, only do simple syntax validation. This tool is useful for developers that wants to analyze a specific document and want to have the opportunity to visualize the data in a directed graph. There are similar tools which provide the same level of validation on different formats. Unlike the W3C validator the following supports RDF-S; University of Manchester OWL Validator[2] and Ontology Metrics[3], WonderWeb OWL Ontology Validator[4], CheckRDFa[5]. However, they do not perform any reasoning on the completeness of the data. One such validator is Vapour[6] which is lacking OWL2 support and is partially dependent on the W3C Validator. It is important to note that none of the on-line validators mentioned above have been maintained since 2007, nor do they support OWL2 or have the ability to scale for large quantities of data.

4.2.2 Jena Eyeball

The Jena Eyeball project is part of the Jena Semantic Web framework. It currently only support the OWL 1.1 API and is capable of performing consistency checks on the RDF/OWL model. The project is open sources and can be downloaded and run as an application [6]

4.2.3 ODEval

The ODEval tool is a syntactic ontology checker that was introduced in 2004. It can be used for the automatic detection of possible syntactical problems

[1]W3C RDF Validator - http://www.w3.org/RDF/Validator/
[2]Manchester OWL Validator - http://owl.cs.manchester.ac.uk/validator/
[3]Manchester OWL Ontology Metrics - http://owl.cs.manchester.ac.uk/metrics/
[4]WonderWeb OWL Ontology Validator - http://www.mygrid.org.uk/OWL/Validator
[5]CheckRDFa - http://check.rdfa.info/
[6]Vapour - http://validator.linkeddata.org/vapour
[6]http://jena.sourceforge.net/Eyeball/

Name	Active	Syn	Sem	OWL2	OWA	CWA	RIF
W3C		**x**					
Manchester OWL		**x**					
Manchester Metrics		**x**					
WonderWeb		**x**					
Check RDFa		**x**					
Vapour		**x**	**part**				
Jena Eyeball		**x**					
ODEVal		**x**					
HermIT	**x**	**x**		**x**			**x**
Fact++	**x**	**x**		**x**			**x**
Pellet ICV	**x**	**x**	**x**	**x**	**x**	**x**	**x**

Table 4.1: Current data validation software.

in ontologies, such as the existence of cycles in the inheritance tree of the ontology classes, inconsistency, incompleteness, and redundancy of classes and instances [30].

4.2.4 HermIT

Is a typical up to date OWL 2 reasoner that supports RIF. With its hypertablau algorithm it is capable of performing syntactic checks on the ontology and whether the data conforms to valid fields and ranges. It should be noted that HermIT is focused on reasoning performance and is therefore suitable for low performance environments.

4.2.5 Fact++

The Fact++ is a widely used OWL 2 DL reasoner that has been maximized for performance and portability.

4.2.6 Pellet Integrity constraint validation

Pellet is a OWL 2 DL reasoner in active development under an open source license, and is currently the only reasoner available that supports integrity constraints in OWL. It a reasoner that is among the best on performance [35], and features [32]. Integrity constraints can be written in RDF and later run through the Pellet command line interface taking the ontology, constraints and SPARQL data source as parameters. Pellet is the obvious choice of tool.

4.3 Development configuration

The software configuration consists of two parts. a development environment and a testing environment, both shown in Figure 4.2 and described in the next section. All the software that has been used is Java based, enabling a seamless porting between different machines and operating systems. An illustration of how the different components was setup can be seen in Figure 4.2.

4.3.1 Jena

The Jena framework[1] is a collection of tools that provides a Java programming interface to RDF/OWL documents. It is not an executable application, but it is a Java API that assists the handling RDF/RDF-S/OWL documents into a Java data structure that seamlessly can be created or loaded, manipulated and saved, without having to manually edit the files. In addition to this Jena provides an interface to SPARQL enabled databases. Jena comes with a built in Rule engine and it has a plug and play reasoner architecture in addition to the ones that are built-in, although they are not very useful for anything else than educational purposes. Most production environment will require up to date reasoners. There is a big downside to the Jena framework from the fact that it only supports OWL 1.1. It happens to be that Jena is starting to fall behind in the development of the Semantic Web where the OWL API is taking over.

4.3.2 OWL API

The University of Manchester is currently a very active contributer to the development of the Semantic Web. One of the open source tools they provide is the OWL API[2], which in contrast to Jena, supports OWL 2 and is currently taking over as the dominating Semantic Web framework. It is however possible to use the OWL API in conjunction with Jena by replacing the Jena RDF/OWL/XML serialization API with the OWL API and to use existing OWL 2 interfaces in Jena to manipulate the documents. This is an approach that uses the best features of both that requires a bit more configuration, but is a strong combination.

Eclipse The Eclipse[3] IDE was used to develop Java[4] code for the integration of testing Pellet[5] in the Jena API set up with the OWL API to handle

[1] http://www.openjena.org/
[2] OWL API - http://owlapi.sourceforge.net/
[3] http://www.eclipse.org/
[4] http://www.java.com/
[5] http://www.clarkparsia.com/

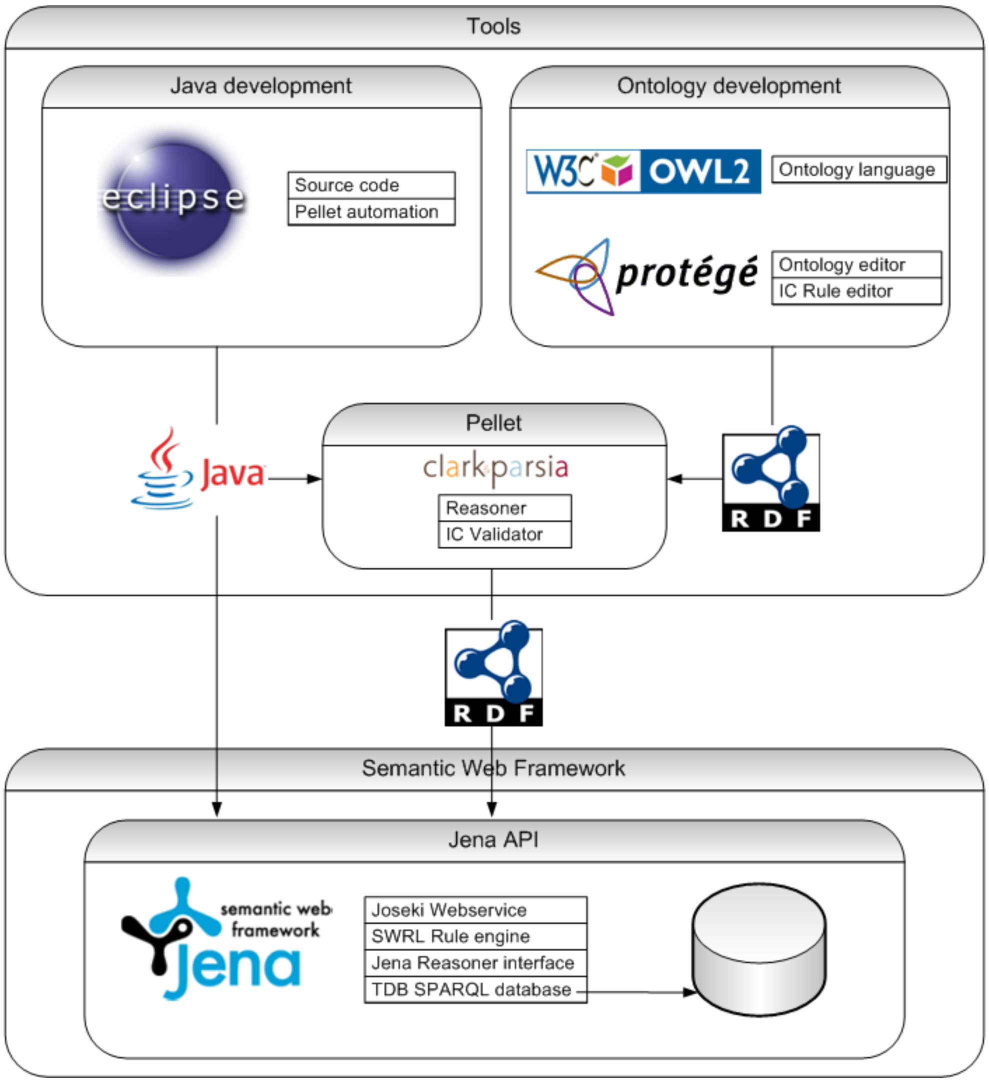

Figure 4.2: The software configuration.

RDF/XML serializations. This setup made it possible to do lookup on methods and classes provided by the Jena API and Pellet in an advanced source code editor. It also enabled easy access to Javadoc, detailed library imports and gathering application output to the console integrated in Eclipse.

Protégé Protégé[1] was used to edit the ontology and developing the Pellet integrity constraints.

[1]Protégé - http://www.protege.stanford.edu/

Brand/Type	Asus P6T Deluxe V2
Architecture	x64
Processor(CPU)	Core i7 920
Chipset	Intel X58
Ram	6 Gb
Hard drive	Corsair 64 Gb SSD
Graphics	Nvidia GeForce GTX 275
Operating system	Microsoft Windows 7
Java runtime env.	jre-19.1-b02

Table 4.2: Hardware configuration.

TDB (SPARQL) The Jena framework offers several SPARQL databases which is ideal for development environments. TDB[1] due to its high performance on single machines.

Joseki Is a simple extension to the Jetty web server that enables SPARQL queries over the http protocol.

Hardware configuration

The following machine was used in all phases of this thesis. It should be noted that the Ontology editor Protégé requires quite an amount of memory when loading large ontologies and data as with ISO 15926.

[1]TDB - http://openjena.org/TDB/

Chapter 5

Validation Test Case and Results

5.1 The Life Of An electric Motor

How the concepts of ISO 15926 can support industrial needs can be demonstrated by showing how the concepts defined in IEC 61346-4 maps onto ISO 15926. The example used is the electric motor example of IEC 61346-4 [10]. This description is based on a presentation given at the POSC Caesar "Semantic Days 2011". The objects used to represent the various aspects related to an electric motor over its life time are represented in IEC 61346-4 as shown below.

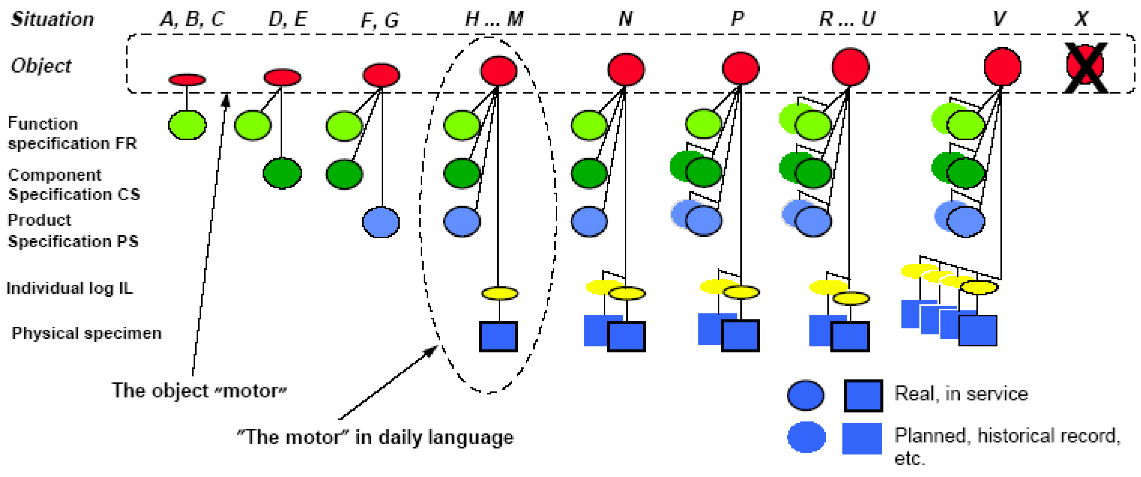

Figure 5.1: The life cycle stages of an electric motor.

"Situation" can be translated to plant life-cycle stages where "A" represents the first design stage, often called "Front End Engineering Design"

(FEED), "B" to "G" various stages of "Detail Design", and "H"-"M" various fabrication stages where "M" represents the commissioning and handover to the client. One should note that the physical items making up the plant does not appear until the stages "H"-"M". Before that we are only talking about specifications. "N"-"V" are various stages during operation, and "X" is the decommissioning and removal of the plant

The "Object" represents what is most commonly known as the "Tag" in the industry. This is the "Functional Location" in a process where a particular type of activity will take place. This is not a physical location, just a place in a schematic representation of a network of processes. It is talked about as the "Tag" even if the "Tag" is the identifier of the "Object", or "Functional location", where an activity takes place.

The "Function specification FR" is stating the requirements from the process point of view, i.e. how much under which conditions, whereas "Component specification CS" is the specification for how the functional requirements shall be met, but not stating a particular make or type. This is represented by the "Product specification PS", which represents a particular manufacturers type of products. Eventually an electric motor, the "Physical specimen" is installed and all relevant documentation is documented in the "Individual log IL". It is worth noting that all these objects exist independently and have separate life-cycles; you can replace the actual motor without changing the specification etc.

These concepts maps to the ISO 15926-2 entity types as follows. The correspondence is shown on Figure 5.2.

In ISO 15926-2 terms the "Object" (Driver at Tag xxxx) will be represented as an instance/member of "functional_physical_object" See Figure 5.2, top oval. This shall be classified by a member of "instance of "class_of_functional_object". The appropriate class here would be the class **driver**. A check to verify the completeness of a functional design is to check if all members of "functional_physical_
object" are classified by a member of "class_of_functional_object".

The "Function specification FR" is represented by an instance/member of "class_of_inanimate_physical_object". (Not shown on the diagram below) The reason for choosing a class level entity type for this is that we are not talking about a specific motor but a set of requirements that can be met by many types (classes) of motors. This is independent of the life-cycle stage and will remain so as long as an electric motor is needed. In ISO 15926 all specifications are considered to be classes. The "Component specification CS" (Electrical motor suitable for Tag xxxx) is also represented by an instance/member of "class_of_inanimate_physical_object". Here again we are not talking about a specific motor but a set of requirements that can be met by many types (classes) of motors. This set of requirements is prepared by the engineering office. This argument also holds for "Product specification CS" (Model X, Variant Y) which is a specification prepared by the manufacturer, but still a class. This is a representation in ISO 15926 terms of what can be found in manufacturer's product catalogues. This class is a subclass of the "Component specification CS" as it has fewer members.

The "Physical specimen" is an individual, "materialized_physical_object". The relationship in ISO 15926 terms between these object types and entity types is shown in the diagram below.

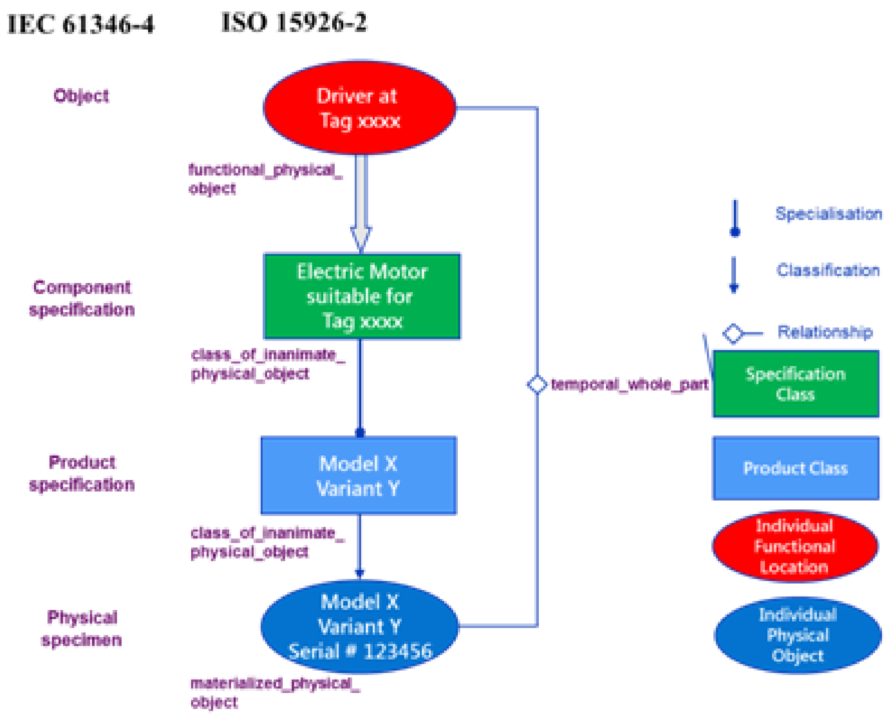
IEC 61346-4
ISO 15926-2
Object
Driver at Tag xxxx
functional_physical_object
Component specification
Electric Motor suitable for Tag xxxx
class_of_inanimate_physical_object
Product specification
Model X Variant Y
class_of_inanimate_physical_object
Physical specimen
Model X Variant Y Serial # 123456
materialized_physical_object
temporal_whole_part
Specialisation
Classification
Relationship
Specification Class
Product Class
Individual Functional Location
Individual Physical Object

Figure 5.2: The mapping between ISO 15926 and IEC 61346.

The nature of a design process is to start with a high level set of requirements defining what e.g. the plant shall be capable of, and then later specifying how this shall be achieved. The specification process is in fact a process where the "how" eventually is defined in detail by refining the initial requirements. This is a process of adding constraints that the objects making up the plant shall conform to, i.e. one version of a design will be a specialized version of the previous as there will be fewer types of objects that can be used. Therefore one version of a design can be seen as a subclass of the preceding version. Eventually one will end up specifying a manufacturer's model, or a special design.

To help in the verification of such processes logic based reasoners will be of great help. The general picture is that if one shall be able to perform such test using SW, e.g. agents, the data representing both the individuals and classes have to be represented as "computer sensible data", i.e. expressed so that a computer can use it. At least in the ISO 15926 community there is not yet sufficient data to perform such tasks on a wide scale. But even generic tests can be of help to reveal logical inconsistencies.

One aspect that is not covered in the example so far is how to deal with "preferred types". In order to limit the number of variants and to simplify maintenance, reduce types of spare parts, optimize delivery etc. companies often have a preferred set of types of products that shall be used within the company. This is information about types (classes), and not about an individual. This can be represented as classifying the types of electric motors as "preferred motor types" i.e. classifying the 1st order electric motor classes by 2nd-order electric motor type classes. Therefore, to meet the business requirements a system based on 2nd order logic is required. Therefore, to represent ISO 15926 in OWL the use of OWL DL will not be sufficient. To temporarily overcome this punning was used, but this is resolved with the current version of OWL, known as OWL 2.

During the development of the POSC Caesar RDL it has been identified that 2nd order classes in general is extremely powerful when it comes to maintaining and subdividing an ontology in addition to record industrial needs.

Quality control is a process by which an item is approved provided it meets a defined set of requirements. These requirements can as described above be represented as an ontology. Quality control is therefore to establish if a particular physical object meets the conditions for membership in a defined class; i.e. a classification process. Provided the data related both to the relevant product class and individual physical object is expressed in a "computer sensible language" establishing if a particular object is conforming to the specifications could be executed as a check if a classification is correctly established. Such check could be performed both as an item leaves the factory, and when it is received at the construction site.

Compared to the PISTEP Activity Model Situations "A"-"G corresponds to "Produce Conceptual Process Design", "Produce Detailed Process Design", "Produce Conceptual Engineering Design (Front End)", and "Produce Detailed Engineering Design" , situations "H"-"M" represents "Construct Plant Pre-Commission" up to and including "Commission Plant"., and situations "N"-"V" represents "Operate Plant" and "Maintain Plant and Equipment", whereas "X" represents "Decommission Plant" and "Demolish Plant and Restore Site". Activities "Procure and Control Equipment, Material and Services", including interaction with "Suppliers and Fabricators" takes place during "G"-"V".

5.2 Integrity constraint - Test cases

The concept of integrity constraint (IC) was invented in the field of relational databases where ICs are used to ensure data consistency [9]

When starting to build a constraint it is recommended to reflect the hierarchy of the ontology. This is not strictly necessary because there are no direct couplings between the ontology and the integrity constraints that are defined, though it is easier to maintain as the number of constraints start to grow. In the following examples we will augment the ontology with constraints that applies to various aspects of ISO15926 that spans from the general perspective down to specific implementations. For the tests we will assume that the ontology is free from inconsistencies

5.3 Life-cycle test case

In the following example we will look at the life-cycle of the general ISO15926 **pipeline** class. The reason for selecting this class is to emphasize the use of constraint inheritance, meaning that subclasses of the class we are modeling will be placed under the same constraints that has been applied to its parents. This example will therefore avoid enforcing constraints that should be implemented in specialized classes further down the hierarchy. Using inheritance in this way is a good practice that, if done at the right ontology level, will force ontology engineers to refine their ontology and make them realize whenever the constraints affects classes that perhaps should have been classified in another branch.

The diagram in Figure 5.3 is a EXPRESS diagram of the **pipeline** class. There are four important components in this example.

1. Rectangular yellow boxes that represents classes, all with the label (1)
2. Emphasized diamonds that represents classes of relations, all with the label (2)
3. Simple diamonds that represents instances of relations, all with the label (3)
4. Hash tags that represents instances of classes, all with the labeled with a # followed by a number that represents the individual

Looking into the classes we can see from the description of their label that they are subclasses of **class_of_activity**, which is a core class from the Part 2 specification. At the ontology level classes are reified with a **temporal_sequence** relationship class that links the classes as a predecessor and a successor. From these classes we can create the individuals

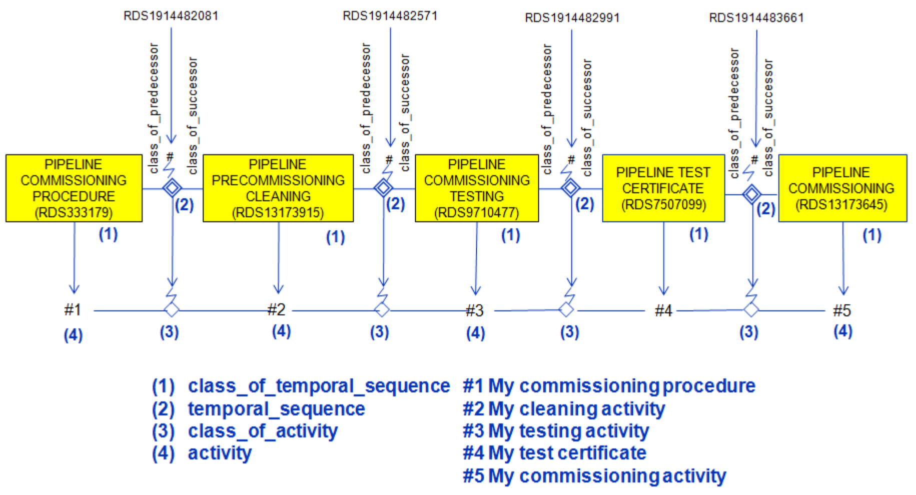
RDS1914482081
RDS1914482571
RDS1914482991
RDS1914483661
class_of_predecessor
class_of_successor
PIPELINE COMMISSIONING PROCEDURE (RDS333179)
PIPELINE PRECOMMISSIONING CLEANING (RDS13173915)
PIPELINE COMMISSIONING TESTING (RDS9710477)
PIPELINE TEST CERTIFICATE (RDS7507099)
PIPELINE COMMISSIONING (RDS13173645)
#1
#2
#3
#4
#5
(1) class_of_temporal_sequence
(2) temporal_sequence
(3) class_of_activity
(4) activity
#1 My commissioning procedure
#2 My cleaning activity
#3 My testing activity
#4 My test certificate
#5 My commissioning activity

Figure 5.3: This EXPRESS diagram shows the **pipeline** class and its relations to its life-cycles.

shown in the baseline of the diagram. The ontology provides us with domain knowledge on rules for class relationships, however, it does not give us the ability to specify dependencies of the individuals with respect to the CWA. In other words, we need to be able to specify that an individual of the class **pipeline_commissioning** depends on an individual of the class **pipeline_test_commissioning**, which again depends on the previous class and so forth - leading all the way back to the **pipeline_commissioning_procedure** class.

This dependencies have been modeled as integrity constraints that can be found in Appendix 7.2, Integrity constraints for the pipeline commissioning life-cycle. The text can be copied from the Appendix and pasted into an empty XML file, and then loaded into Protégé. The latest RDL nightly compilation is available for download from the POSC Caesar web site[1]. Then the constraints and then be run against the Pellet ICV in order to test if the dependencies of the individuals conform to the RDL.

5.4 Results

This test case shows that it is possible to apply the local CWA to the individuals by adding integrity constraints while other individual are unaffected by the constraints. Any changes to the individuals such as inserting faulty data types, data ranges etc. resulted in either a integrity constraint violation or by Pellet rejecting the individual. However, it is possible to avoid the integrity constraints by manually manipulating the class membership of

[1]http://rds.posccaesar.org/downloads/PCA-RDL.owl.zip

the individuals. This will result in a bypassing of the IC, and is expected behavior.

Regarding the pragmatic data quality perspective, the augmentation of IC will increase the credibility of the validated data. ICs will also be able to provide a better traceability of potential erroneous data that does not comply to the constraints. However, ICs does not add any value to the pragmatic data quality of portability, accessibility and so forth. Such quality measures is a matter of good implementation planning of the system to best fit its users.

The life-cycle example is a very important because nearly all classes and hence individuals are affected by it. There has been devised several similar tests for verifying that E. g; no pressurized pipe can lead into the open air, and ensuring that systems under operation do not have any warnings.

Chapter 6

Conclusions, Current & Future Work

In this thesis we have introduced a new data validation approach the Semantic Web implementation of ISO 15926. By reviewing the existing Semantic Web tools we can conclude syntactic data validation is provided by nearly all the Semantic Web tools and is to be considered out of the box functionality. Furthermore, it is important to avoid the pitfall of applying UML to the ISO 15926 class modeling from the fact that it has been shown that the OWL language allows users to specify innate object properties without applying reified principle. As a result of this there is a potential of a loss of expressiveness and precision that is one of the key features of ISO 15926.

The most important contribution is the proof of concept that shows it is possible to apply the CWA to ISO 15926 without requiring any modifications or changes to existing implementations.

It can be concluded that automated reasoning, can not replace fulfill the pragmatic data quality measures of ISO 8000. Pragmatic data quality is subjective to user experience from the choices that are made during the system implementation planning.

Studied conducted by the OLF has indicated that implementations of Integrated Operation could not only improve the utilization of existing resources, but also improve the health and safety for the workers which is one of the most valuable business assets.

6.1 Further work

As of today the Pellet ICV software is operated through a simple command line interface that can be modified in many ways. The existing program can be implemented into a web service that scans all incoming data or the source code can be imported in Jena - allowing greater control of both data and error handling.

There is no other data model that uses 2nd order classes like ISO 15926. It could be interesting to investigate if there are any benefits of applying punning and Integrated Operation on this level.

Chapter 7

Appendix

Quality Criteria for RDF representations of installations descriptions according to ISO15926 part 8

Martin Giese

criteria.tex 2209 2011-05-12 14:14:12Z martingi

1 Prerequisites

We assume the following:

- All data is originally represented as a set of "template instances" and "type assertions"
  - A template instance is an n-ary literal of the shape
    $$p(i_1, \ldots, i_n)$$
    where $i_1, \ldots, i_n$ are literals (strings, numbers, etc) or resources identifiers (URIs), and p is a template (also identified by a URI)
  - A type assertion is a literal
    $$C(i)$$
    where C is an OWL class and i is a resource identifier as before

Do we allow blank nodes in template instances?

- For every template p, there is a description giving an RDF property $R_{p,i}$ for each of the arguments of p.
- Any template instance $p(i_1, \ldots, i_n)$ is represented in RDF as a set of $n + 1$ triples

  `_:x rdf:type` p;
  `_:x` $R_{p,1}$ i_1;
  ...
  `_:x` $R_{p,n}$ i_n.

- Any type assertion $C(i)$ is represented in RDF by a triple

  i `rdf:type` C

- These representations are complemented by an ontology, including the ISO15925 part x/y/z vocabularies, and the template and meta-template ontologies, as well as a domain-specific ontology built on top of these.

2 Requirements for RDF representations in general

2.1 Separation of Information Levels

Requirement 2.1.1 *There is a clear separation of the set of triples into*

- *a vocabulary description, and*
- *instance data.*

This separation might be effected through the storage in different files, in different graphs underlying a SPARQL endpoint, etc. In any case, it must be possible to say which part a given triple belongs to.

Rationale: The main purpose of this separation is to provide a basis for the remaining criteria. The rules applying to the vocabulary definition and the instance data will be different.

Implementation: This is not a requirement that can be tested. Rather, implementations testing the other requirements will need this separation as part of their input.

Note that information both of the vocabulary description and the instance data may well be spread over several documents or data sources. The important point is that it must be clear for each triple which part it belongs to.

Requirement 2.1.2 *There is a clear separation of the vocabulary definition into*

- *an application specific vocabulary*
- *a generic vocabulary*

For the meaning of 'separation', see Req. 2.1.1.

Rationale: Like Req. 2.1.1, this is mainly to provide a basis for the remaining criteria. For instance, resources will be required to carry a sufficiently specific type, in the sense that types mentioned in the application specific vocabulary and not the generic vocabulary are considered specific enough.

Implementation: See Req. 2.1.1. Again, each part of the vocabulary may well be spread over several documents or data sources. The important point is that it must be clear for each triple (RDFS/OWL axiom) which part it belongs to.

Definition 2.1.3 *An* application class *is a class that is mentioned in the application specific vocabulary, but not in the generic vocabulary. Any class mentioned in the generic vocabulary is called a* generic class.

2.2 Literals

Requirement 2.2.1 *Any literal should be either typed (by one of the standard datatypes defined by OWL 2) or carry a language tag.*

Rationale: Any literal is either intended for machine processing or to be read by humans. In the first case, it should carry a datatype delimiting the interpretation (the datatype might by xsd:string if that is the intention). In the latter case, it is a natural language string, which should carry its language tag.

Implementation: This should be easy to implement using SPARQL queries with suitable filter expressions.

2.3 Types

Requirement 2.3.1 *Any resource mentioned should have a type, either explicitly given with `rdf:type`, or inferable, that is related to the application domain. This type should in other words be an* application class *in the sense of Def. 2.1.3.*

Rationale: All objects referred to the description of an installation should have to do directly with that installation. They must therefore have a more concrete type than being a "Thing", a "possible individual", etc. Any kind of further processing of the RDF description will require more concrete types.

Note that this requirement does not require types to be given explicitly. Types maybe inferred from an ontology and the relations in which a resource stands with other resources, e.g. domain and range reasoning.

Since it cannot be decided from the data alone what constitutes a sufficiently specific type, the separation of the vocabulary according to Req. 2.1.1 is used to decide which parts of the vocabulary are to be counted as the domain vocabulary, and which are generic terms.

Implementation: This requirement is not easy to implement, since it requires reasoning. However, if the model and accompanying ontology can be classified, the available types for all individuals can easily be checked.

The previous requirement is comparatively weak, and hard to check, but it can be seen as a minimal requirement. We can ask for more by restricting the reasoning to simple cases.

Requirement 2.3.2 *Requirement 2.3.1 is refined by requiring that a application class can be derived as type of any resource by some simple reasoning regime, e.g. RDFS entailment*

Rationale: This requirement is easy to check algorithmically, although it cannot be checked with a SPARQL query.

Implementation: RDFS reasoning (domain/range, subclass, subproperty) can be used to check for the existence of types. In fact, reasoning can be aborted for each individual, as soon as at least one type has been derived.

The latter requirement can be seen as one element in a family of requirements, indexed by the reasoning regime to be supported. At the extreme end, we can require that no reasoning is needed at all:

Requirement 2.3.3 *Requirement 2.3.1 is refined by requiring at least one application class to be explicitly given as type with an `rdf:type` triple for every resource.*

Rationale: No reasoning is needed to check this requirement. And it can be implemented without adding too much overhead to the RDF representation.

Implementation: The presence of `rdf:type` triples can probably be checked with a SPARQL query.

To ease further processing, we can add the following requirement:

Requirement 2.3.4 *Requirement 2.3.3 is refined by requiring that* all *(named) types that can be inferred for a resource are explicitly given by `rdf:type` triples.*

Rationale: This makes it easy for software that processes the RDF representation to find all instances of any given type, without requiring reasoning.

Implementation: Checking this requires reasoning again. The model needs to be classified, and for each individual i and each type C it belongs to, the presence of a triple i `rdf:type` C needs to be checked.

A certain kind of "useless" type information allowed by the previous requirements can be eliminated by the following one:

Requirement 2.3.5 *If a resource i has (can be inferred to have) type C and C is (can be inferred to be) covered by some of its sublcasses $D_1, \ldots, D_n \sqsubseteq C$, with $C \sqsubseteq D_1 \sqcup \cdots \sqcup D_n$, then there is a $k \in \{1, \ldots, n\}$ such that i has (can be inferred to have) type D_k.*

Rationale: C plays the role of an "abstract superclass": any element of C must also belong to one of the subclasses $D_1, \ldots D_n$. We require the model to say explicitly which of the subclasses this is. For instance, if "sensor" is covered by "temperature sensor", "pressure sensor", and "motion sensor", then the model should say for any concrete resource with type "sensor" which of the subclasses it belongs to.

Note: it is not clear whether this is always a desired property. For a given application, "sensor" may sufficiently concrete, and giving the actual type of sensor would not add any value. On the other hand, the ontology used to define sensors might include covering axioms for types of sensors. In such a case, this requirement should not be enforced. It only makes sense if the "granularity" of the ontology is not finer than required by the application.

Implementation: This requires reasoning in general. Given an ontology, reasoning could be used to find all sets of covering axioms as asked for in this requirement, and then generate a single SPARQL query that checks the requirement, provided type information is represented explicitly. E.g. given the sensor ontology, a SPARQL query could be written that checks that whenever a triple *i* `rdf:type :Sensor` is present, there is also a triple *i* `rdf:type :TemperatureSensor` or *i* `rdf:type :PressureSensor` or *i* `rdf:type :MotionSensor`.

The previous requirements require types of resources to be *inferable*, and require at least one of them to belong to the domain vocabulary. This is not the same as the following mostly orthogonal property:

Requirement 2.3.6 *Requirement 2.3.1 is refined by requiring that all (named) types any resource* belongs to *are explicitly given or can be inferred.*

Rationale: What this means is that anything that is e.g. a temperature sensor actually carries that type, explicitly or implicitly. For the previous requirements, it would be enough if a temperature sensor belonged to the supertype "sensor", assuming that "sensor" is part of the domain vocabulary.

Implementation: This cannot be checked mechanically, since it requires knowing what identifiers refer to.

2.4 Consistency

Requirement 2.4.1 *The RDF representation and its accompanying ontology must be consistent.*

Rationale: An inconsistent model is obviously faulty.

Implementation: The implementation requires full OWL DL reasoning.

If the model is too large to be classified, this requirement could be weakened be restricting the reasoning performed, for instance as follows:

Requirement 2.4.2 *The ontology TBox must be consistent, and no individual is asserted (or can be inferred by RDFS reasoning) to belong to two classes that are known to be disjoint from TBox reasoning.*

Rationale: For large ABoxes with reasonably sized TBoxes, this might be tractable, and catch most potential mistakes.

Note that it is hard to give a semantic explanation for such a restriction. If the model as a whole is inconsistent, then it has no interpretation, and no amount of restricted reasoning can change that.

3 Requirements for RDF representations in accordance with ISO15926 part 8

3.1 Conservative Extensions

Requirement 3.1.1 *The application specific vocabulary must be a conservative extension of the underlying ISO15926 part x/y/z, template, and meta-template vocabularies.*

Rationale: This ensures that the application specific ontology does not "modify" the meaning of the underlying ontologies.

Implementation: This requires reasoning, and that reasoning is non-standard. Not sure which implementations exist at all.

This is a nice minimal requirement, but might be hard to check.

It might be possible to relax this, by restricting the way in which the domain vocabulary can refer to the imported ontologies, e.g. only by subclassing. Such restrictions can easily become over-restrictive however.

3.2 Part 8 adherence

We want to ensure that all information is actually represented as "reified" template statements.

Requirement 3.2.1 *For any triple x y z belonging to the instance data, where y is different from* `rdf:type`*, and y is not declared by the vocabulary description to be an* `owl:AnnotationProperty`*, there is a (possibly inferred) triple x* `rdf:type` *p where p is in turn of type . . . (what? abc:Template?)*

Rationale: We want only template instances and type assertions in the RDF data. However, annotations are allowed, like for instance `rdfs:label` to give a human readable identifier to a resource.

Note: in combination with Req. 2.3.1, this means that the template types p should also be part of the application specific vocabulary.

Implementation: This can be checked with SPARQL queries generated from the domain vocabulary, provided that the types are explicitly represented. Maybe with a static SPARQL query.

Requirement 3.2.2 *For any resource or blank node x mentioned in the instance data, such that x* `rdf:type` *p for some n-ary template p with declared properties $R_{p,i}$, there is a triple x $R_{p,i}$ y_i for all $i = 1, \ldots, n$, either explicitly or implicitly, for some literal or named resource y_i.*

Rationale: Whenever a template instance is given, all arguments should be given. Not just inferred to exist.

TODO: Have to think about blank nodes for the y_i. It might make sense to have blank nodes in the template instances. But then, such blank nodes have to be kept apart from things that can be inferred to exist simply from the template axioms.

Implementation: This can be checked with SPARQL queries generated from the domain vocabulary, provided that the types are explicitly represented. Maybe with a static SPARQL query.

4 Further Thoughts

1. Many of the previous requirements can be read the "semantic" way, i.e. as requirements on inferable information, or as requirements on explicitly given information. Which to choose depends on how much reasoning we are willing to perform. In any case, it would be good to find a structure for this document which allows referring to requirements together with the implied amount of reasoning without reduplicating all the requirements.
2. We have several recurring categories of implementability:
   a) undecidable
   b) algorithm not known to us
   c) algorithm known but non-standard
   d) implemented by standard reasoning tools
   e) implementable as a set of SPARQL queries that can be computed from the domain ontology
   f) implementable as a static SPARQL query, independent of the domain vocabulary.

   Should make reference to these categories more systematic.
3. At some point, we might want to look into an axiomatisation of properties of templates, like symmetry, transitivity, etc. which may not be easy to express after the part 8-isation.

5 TODO

- Criteria for URIs, prefixes/namespaces
- No relative URIs?

7.2 Integrity constraints for the pipeline commissioning life-cycle

```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
 <!ENTITY rdl2 "http://posccaesar.org/rdl/" >
 <!ENTITY rdl "http://irm.dnv.com/rapoc/rdl/" >
 <!ENTITY cats "http://irm.dnv.com/rapoc/cats/" >
 <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
 <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
 <!ENTITY templates2 "http://irm.dnv.com/rapoc/templates/" >
 <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
 <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
 <!ENTITY templates "http://standards.iso.org/iso/ts/15926/-8/ed
     -1/tech/reference-data/templates#" >
 <!ENTITY data-model "http://standards.iso.org/iso/ts/15926/-8/
     ed-1/tech/reference-data/data-model#" >
 <!ENTITY template-model "http://standards.iso.org/iso/ts
     /15926/-8/ed-1/tech/reference-data/template-model#" >
]>


<rdf:RDF xmlns="http://www.posccaesar.org/rdfverifier.owl#"
   xml:base="http://www.posccaesar.org/rdfverifier.owl"
   xmlns:templates="http://standards.iso.org/iso/ts/15926/-8/ed
     -1/tech/reference-data/templates#"
   xmlns:cats="http://irm.dnv.com/rapoc/cats/"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:data-model="http://standards.iso.org/iso/ts/15926/-8/ed
     -1/tech/reference-data/data-model#"
   xmlns:rdl="http://irm.dnv.com/rapoc/rdl/"
   xmlns:template-model="http://standards.iso.org/iso/ts
     /15926/-8/ed-1/tech/reference-data/template-model#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:templates2="http://irm.dnv.com/rapoc/templates/"
   xmlns:rdl2="http://posccaesar.org/rdl/">
 <owl:Ontology rdf:about="http://www.posccaesar.org/rdfverifier.
     owl">
   <rdfs:label>Constraints for Univ-bench Ontology</rdfs:label>
   <rdfs:comment>This ontology provides signatures for ISO
      15926-7 templates, as given in ISO 15926-7/CD-TS.</
      rdfs:comment>
   <rdfs:comment>This ontology provides basic classes for
      representing ISO 15926-8 template signatures. It depends on
       an OWL taxonomy representation of ISO 15926-2 entity types
      .</rdfs:comment>
   <rdfs:comment>Preliminary OWL native representation of ISO
      15926-2, destined for the ISO 15926-8 representation of ISO
       15926-2 entity types. This file was provided by DNV IRM,
     based on work in the IOHN project.</rdfs:comment>
 <rdfs:comment>Some axioms from LUBM schema copied to be
     interpreted as constraints</rdfs:comment>
</owl:Ontology>



<!--
//////////////////////////////////////////////////////////////////////////////////////

//
// Annotation properties
//
//////////////////////////////////////////////////////////////////////////////////////

 -->




<!--
//////////////////////////////////////////////////////////////////////////////////////

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

 -->




<!-- http://www.posccaesar.org/rdfverifier.owl#
    classOfPredesessor -->

<owl:ObjectProperty rdf:about="http://www.posccaesar.org/
    rdfverifier.owl#classOfPredesessor">
 <rdf:type rdf:resource="&owl;TransitiveProperty"/>
 <owl:propertyDisjointWith rdf:resource="http://www.posccaesar.
     org/rdfverifier.owl#classOfSuccessor"/>
</owl:ObjectProperty>



<!-- http://www.posccaesar.org/rdfverifier.owl#classOfSuccessor
     -->

<owl:ObjectProperty rdf:about="http://www.posccaesar.org/
    rdfverifier.owl#classOfSuccessor">
 <rdf:type rdf:resource="&owl;TransitiveProperty"/>
 <owl:inverseOf rdf:resource="http://www.posccaesar.org/
     rdfverifier.owl#classOfPredesessor"/>
</owl:ObjectProperty>
```

```
<!--
///////////////////////////////////////////////////////////////////////////////////////

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

 -->




<!-- http://www.posccaesar.org/rdfverifier.owl#Activity -->

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #Activity"/>



<!-- http://www.posccaesar.org/rdfverifier.owl#
     Class_of_Activity -->

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #Class_of_Activity">
 <rdfs:subClassOf rdf:resource="http://www.posccaesar.org/
     rdfverifier.owl#Activity"/>
</owl:Class>



<!-- http://www.posccaesar.org/rdfverifier.owl#
     Class_of_Temporal_Sequence -->

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #Class_of_Temporal_Sequence"/>



<!-- http://www.posccaesar.org/rdfverifier.owl#
     Pipeline_Commissioning -->

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #Pipeline_Commissioning">
 <rdfs:subClassOf rdf:resource="http://www.posccaesar.org/
     rdfverifier.owl#Class_of_Temporal_Sequence"/>
</owl:Class>



<!-- http://www.posccaesar.org/rdfverifier.owl#
```

```
    Pipeline_Commissioning_Procedure -->

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #Pipeline_Commissioning_Procedure">
  <rdfs:subClassOf rdf:resource="http://www.posccaesar.org/
      rdfverifier.owl#Class_of_Temporal_Sequence"/>
</owl:Class>



<!-- http://www.posccaesar.org/rdfverifier.owl#
    Pipeline_Commissioning_Testing -->

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #Pipeline_Commissioning_Testing">
  <rdfs:subClassOf rdf:resource="http://www.posccaesar.org/
      rdfverifier.owl#Class_of_Temporal_Sequence"/>
</owl:Class>



<!-- http://www.posccaesar.org/rdfverifier.owl#
    Pipeline_Precommissioning_Cleaning -->

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #Pipeline_Precommissioning_Cleaning">
  <rdfs:subClassOf rdf:resource="http://www.posccaesar.org/
      rdfverifier.owl#Class_of_Temporal_Sequence"/>
</owl:Class>



<!-- http://www.posccaesar.org/rdfverifier.owl#
    Pipeline_Test_Certificate -->

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #Pipeline_Test_Certificate">
  <rdfs:subClassOf rdf:resource="http://www.posccaesar.org/
      rdfverifier.owl#Class_of_Temporal_Sequence"/>
</owl:Class>



<!-- http://www.posccaesar.org/rdfverifier.owl#RDS1914482081 --
    >

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #RDS1914482081">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.posccaesar.org/
            rdfverifier.owl#Temporal_Sequence"/>
        <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.posccaesar.org/
          rdfverifier.owl#classOfPredesessor"/>
      <owl:someValuesFrom rdf:resource="http://www.posccaesar.
          org/rdfverifier.owl#Pipeline_Commissioning_Procedure"/>
     </owl:Restriction>
     <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.posccaesar.org/
          rdfverifier.owl#classOfSuccessor"/>
      <owl:someValuesFrom rdf:resource="http://www.posccaesar.
          org/rdfverifier.owl#Pipeline_Precommissioning_Cleaning"
          />
     </owl:Restriction>
    </owl:intersectionOf>
   </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="http://www.posccaesar.org/
      rdfverifier.owl#Temporal_Sequence"/>
</owl:Class>



<!-- http://www.posccaesar.org/rdfverifier.owl#RDS1914482571 --
    >

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #RDS1914482571">
  <owl:equivalentClass>
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
     <rdf:Description rdf:about="http://www.posccaesar.org/
         rdfverifier.owl#Temporal_Sequence"/>
     <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.posccaesar.org/
          rdfverifier.owl#classOfPredesessor"/>
      <owl:someValuesFrom rdf:resource="http://www.posccaesar.
          org/rdfverifier.owl#Pipeline_Precommissioning_Cleaning"
          />
     </owl:Restriction>
     <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.posccaesar.org/
          rdfverifier.owl#classOfSuccessor"/>
      <owl:someValuesFrom rdf:resource="http://www.posccaesar.
          org/rdfverifier.owl#Pipeline_Commissioning_Testing"/>
     </owl:Restriction>
    </owl:intersectionOf>
   </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="http://www.posccaesar.org/
      rdfverifier.owl#Temporal_Sequence"/>
</owl:Class>



<!-- http://www.posccaesar.org/rdfverifier.owl#RDS1914482991 --
```

```
    >

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #RDS1914482991">
 <owl:equivalentClass>
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
    <rdf:Description rdf:about="http://www.posccaesar.org/
        rdfverifier.owl#Temporal_Sequence"/>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://www.posccaesar.org/
         rdfverifier.owl#classOfPredesessor"/>
     <owl:someValuesFrom rdf:resource="http://www.posccaesar.
         org/rdfverifier.owl#Pipeline_Commissioning_Testing"/>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://www.posccaesar.org/
         rdfverifier.owl#classOfSuccessor"/>
     <owl:someValuesFrom rdf:resource="http://www.posccaesar.
         org/rdfverifier.owl#Pipeline_Test_Certificate"/>
    </owl:Restriction>
   </owl:intersectionOf>
  </owl:Class>
 </owl:equivalentClass>
 <rdfs:subClassOf rdf:resource="http://www.posccaesar.org/
     rdfverifier.owl#Temporal_Sequence"/>
</owl:Class>



<!-- http://www.posccaesar.org/rdfverifier.owl#RDS1914483661 --
    >

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #RDS1914483661">
 <owl:equivalentClass>
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
    <rdf:Description rdf:about="http://www.posccaesar.org/
        rdfverifier.owl#Temporal_Sequence"/>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://www.posccaesar.org/
         rdfverifier.owl#classOfPredesessor"/>
     <owl:someValuesFrom rdf:resource="http://www.posccaesar.
         org/rdfverifier.owl#Pipeline_Test_Certificate"/>
    </owl:Restriction>
    <owl:Restriction>
     <owl:onProperty rdf:resource="http://www.posccaesar.org/
         rdfverifier.owl#classOfSuccessor"/>
     <owl:someValuesFrom rdf:resource="http://www.posccaesar.
         org/rdfverifier.owl#Pipeline_Commissioning"/>
    </owl:Restriction>
   </owl:intersectionOf>
  </owl:Class>
 </owl:equivalentClass>
 <rdfs:subClassOf rdf:resource="http://www.posccaesar.org/
     rdfverifier.owl#Temporal_Sequence"/>
</owl:Class>



<!-- http://www.posccaesar.org/rdfverifier.owl#
    Temporal_Sequence -->

<owl:Class rdf:about="http://www.posccaesar.org/rdfverifier.owl
    #Temporal_Sequence"/>



<!--
///////////////////////////////////////////////////////////////////////////////////////

//
// Individuals
//
///////////////////////////////////////////////////////////////////////////////////////

 -->




<!-- http://www.posccaesar.org/rdfverifier.owl#
    My_cleaning_activity -->

<owl:NamedIndividual rdf:about="http://www.posccaesar.org/
    rdfverifier.owl#My_cleaning_activity">
 <rdf:type rdf:resource="http://www.posccaesar.org/rdfverifier.
     owl#Pipeline_Precommissioning_Cleaning"/>
</owl:NamedIndividual>



<!-- http://www.posccaesar.org/rdfverifier.owl#
    My_commissioning_activity -->

<owl:NamedIndividual rdf:about="http://www.posccaesar.org/
    rdfverifier.owl#My_commissioning_activity">
 <rdf:type rdf:resource="http://www.posccaesar.org/rdfverifier.
     owl#Pipeline_Commissioning"/>
</owl:NamedIndividual>



<!-- http://www.posccaesar.org/rdfverifier.owl#
    My_commissioning_procedure -->

<owl:NamedIndividual rdf:about="http://www.posccaesar.org/
```

```
    rdfverifier.owl#My_commissioning_procedure">
 <rdf:type rdf:resource="http://www.posccaesar.org/rdfverifier.
     owl#Pipeline_Commissioning_Procedure"/>
</owl:NamedIndividual>



<!-- http://www.posccaesar.org/rdfverifier.owl#
     My_test_certificate -->

<owl:NamedIndividual rdf:about="http://www.posccaesar.org/
    rdfverifier.owl#My_test_certificate">
 <rdf:type rdf:resource="http://www.posccaesar.org/rdfverifier.
     owl#Pipeline_Test_Certificate"/>
</owl:NamedIndividual>



<!-- http://www.posccaesar.org/rdfverifier.owl#
     My_testing_activity -->

<owl:NamedIndividual rdf:about="http://www.posccaesar.org/
    rdfverifier.owl#My_testing_activity">
 <rdf:type rdf:resource="http://www.posccaesar.org/rdfverifier.
     owl#Pipeline_Commissioning_Testing"/>
</owl:NamedIndividual>



<!--
/////////////////////////////////////////////////////////////////////////////////

//
// General axioms
//
/////////////////////////////////////////////////////////////////////////////////

 -->

<rdf:Description>
 <rdf:type rdf:resource="&owl;AllDisjointClasses"/>
 <owl:members rdf:parseType="Collection">
  <rdf:Description rdf:about="http://www.posccaesar.org/
      rdfverifier.owl#Pipeline_Commissioning"/>
  <rdf:Description rdf:about="http://www.posccaesar.org/
      rdfverifier.owl#Pipeline_Commissioning_Procedure"/>
  <rdf:Description rdf:about="http://www.posccaesar.org/
      rdfverifier.owl#Pipeline_Commissioning_Testing"/>
  <rdf:Description rdf:about="http://www.posccaesar.org/
      rdfverifier.owl#Pipeline_Precommissioning_Cleaning"/>
  <rdf:Description rdf:about="http://www.posccaesar.org/
      rdfverifier.owl#Pipeline_Test_Certificate"/>
 </owl:members>
</rdf:Description>
```

```
 <rdf:Description>
  <rdf:type rdf:resource="&owl;AllDifferent"/>
  <owl:distinctMembers rdf:parseType="Collection">
   <rdf:Description rdf:about="http://www.posccaesar.org/
       rdfverifier.owl#My_cleaning_activity"/>
   <rdf:Description rdf:about="http://www.posccaesar.org/
       rdfverifier.owl#My_commissioning_activity"/>
   <rdf:Description rdf:about="http://www.posccaesar.org/
       rdfverifier.owl#My_commissioning_procedure"/>
   <rdf:Description rdf:about="http://www.posccaesar.org/
       rdfverifier.owl#My_test_certificate"/>
   <rdf:Description rdf:about="http://www.posccaesar.org/
       rdfverifier.owl#My_testing_activity"/>
  </owl:distinctMembers>
 </rdf:Description>
</rdf:RDF>
```

List of Terms

antecedent
: is a preceding event. 46

API
: Application Programming Interface is a preprogrammed library that is available for a software developer as a toolkit for performing common application tasks.. 49, 51, 83

Axiom
: An axiom (or postulate) is a proposition that is not proved or demonstrated but considered to be either self-evident, or subject to necessary decision. It is any mathematical statement that serves as a starting point from which other statements are logically derived.. 48, 82

Best practice
: is a technique or methodology that, through experience and research, has proven to reliably lead to a desired result. 42

business logic
: is the part of an application program that performs the required data processing of the business. 33, 40, 47

class-of-class
: is a 2nd order class that can group a collection of classes. 44, 45

consequent
: is a following event. 46

CWA
: The assumption that a database query can be considered false unless proven true, thus considering the information in the database to be complete.. 1, 47, 48, 59, 61

DL
: Is a family of formal knowledge representation languages that models

concepts, roles and individuals, and their relationships. The fundamental modeling concept of a DL is the Axiom - a logical statement relating roles and/or concepts.. 32, 35, 36, 48

DNV
Det Norske Veritas, a non-profit organization leading the RAPOC project.. 3, 8

EXPRESS
is a modeling language for use in engineering data exchange standards that combines the entity-attribute relationship and object modeling paradigm. 30, 45

Horn-rules
is a clause (a disjunction of literals) with at most one positive literal. 33

IC
A constraint (rule) that must remain true for a database to preserve integrity.. 48, 60

IDE
Integrated Development Environment. 51

Integrated Operation
is a system where different parties share data on a common platform. 1, 24, 25, 62

ISO
Is an standardization organization with a consortium of public and private members that develops standards within several fields. Its name is not an acronym for International Standardization Organization, but is derived from the Greek word meaning "equal".. 17

ISO 15926-1
ISO 15926 Part 1: Overview and fundamental principles. 16

ISO 15926-2
ISO 15926 Part 2: Data model. 16, 84

ISO 8000
Is an information quality standard currently undergoing development that seeks to define a common measure of data quality in automated processes.. 37, 38, 42

ISO15926
: A data model standard proposed by the International Standardization Organization and its members for information exchange.. 8, 9, 11, 15, 16, 24, 25, 30, 38–40, 42, 44, 45, 48, 58

Javadoc
: is a Java tool for generating API documentation. 52

Knowledgeweb
: is a project funded by the European Commission for promoting the transition process of Ontology technology from Academia to Industry. 26

life-cycle
: A progression through a series of differing stages of development. 3–5, 7–15, 18, 20, 24, 25, 45, 54, 55, 58

Lifting & Lowering
: The process of expanding RDF data (Lifting) from a system into a explicit ISO 15926 Part 2 and translating (Lowering) it to other systems.. 38

modus ponens
: is a valid inference drawn from a hypothetical proposition. 46

ODM
: is an Object Management Group (OMG) specification to make the concepts of Model-Driven Architecture applicable to the engineering of ontologies. 45

Ontology
: defines the terms used to describe and represent an area of knowledge. 46, 47

OWA
: The assumption that a database query can be considered true unless proven false, taking into account that the database might be incomplete.. 1, 47, 48

OWL API
: is a Java API and reference implmentation for creating, manipulating and serialising OWL Ontologies. 51

OWL-DL
: A heuristic version of the OWL knowledge representation language based on description logic, hence DL, that guarantees computational completeness.. 32, 44

OWL-Full
A complete version of the OWL knowledge representation language that make no compromises to expressiveness on the expense of computational tractability.. 32

OWL-Lite
A simple version of the OWL knowledge representation language suitable for building taxonomies.. 32

Part 2
See ISO 15926-2. 58

punning
the name comes playing with words, the technique is a method of making classes and individuals share the same namespace. 47

Reified
is an explicit statement containing a subject predicate and object without using properties. 30, 58

Semantic Net
is a network which represents semantic relations among concepts. 30

Semantic Web
is a collaborative movement led by the World Wide Web Consortium (W3C) that promotes common formats for data on the World Wide Web. 45

Template
is a set comprising of a first-order logic predicate for which a definition is stated as an axiom, a template signature and a template axiom expansion. 42

Triple
An actual, physical object that is represented in the ontology as a data element. 34, 45

UML
is a standardized general-purpose modeling language in the field of object-oriented software engineering. 45

UNA
The assumption that every URI specifies a unique object, regardless of whether they may refer to the same object.. 48

List of Abbreviations

AI	Artificial Intelligence. 30
AP	Application Protocol. 7
ASME	American Society of Mechanical Engineers. 17
CAD	Computer-Aided Design. 7
CAE	Computer-Aided Engineering. 7
CAM	Computer-Aided Manufacturing. 7
CAx	Computer-Aided technologies. 7
COTS	Commercial Off The Shelf. 6
EPISTLE	European Process Industries STEP Technical Liaison Executive. 8
ESPRIT	European Strategic Program on Research in Information Technology. 8
FOL	First Order Logic. 29, 31
GE	General Electric. 7
IEC	International Electro technical Commission. 17
IGES	Initial Graphics Exchange Specification. 7
IOHN	Integrated Operations in the High North. 25
NAF	Negation As Failure. 34, 47, 48
NFR	Research Council of Norway. 8
OLF	Norwegian Oil Industry Association. 24, 25
OWL	Web Ontology Language. 23, 30, 32–34, 36, 45, 48

PDM/EDM	Product data management/Engineering Data Management. 7
PML	Proof Markup Language. 35
RD	Reference Data. 14–16
RDF	Resource Description Framework. 29–31, 34, 45–47
RDF-S	Resource Description Framework Schema. 31, 34
RDL	ReferenceData Library. 14
RIF	Rule Interchange Format. 33, 86
RIF-BLD	RIF Basic Logic Dialect. 33
SME	Society of Manufacturing Engineers. 7
SPARQL	SPARQL Protocol and RDF Query Language. 34, 35, 46–48
STEP	STandard for the Exchange of Product model data. 7, 8
SWRL	Semantic Web Rule Language. 33
W3C	World Wide Web Consortium. 23, 33
XML	eXtensible Markup Language. 29, 34
XML-Schema	eXtensible Markup Language. 29

Bibliography

[1] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer (Cooperative Information Systems)*. The MIT Press, April 2004.

[2] E.C.C.M. Association. Electronic open technical dictionary (eotd). `http://www.eccma.org/index.php`, 2005. This is an electronic document. Date of publication: February 03, 2005. Date retrieved: May 31, 2011. Date last modified: [Date unavailable].

[3] Damyan Ognyanov Atanas Kiryakov and Dimitar Manov. Owlim - a pragmatic semantic repository for owl. Technical report, Ontotext lab, Sirma Grouc corp, 2005.

[4] Tim Berners-Lee. Semantic web road map. Technical report, World Wide Web Consortium, 2004.

[5] Tim Berners-Lee. Artificial intelligence and the semantic web. `http://www.w3.org/2006/Talks/0718-aaai-tbl/Overview.html`, 2006. This is an electronic document. Date of publication: November 30, 2006. Date retrieved: May 15, 2011. Date last modified: [Date unavailable].

[6] Tim Berners-Lee and Mark Fischetti. *Weaving the Web*. HarperSanFrancisco, 1999.

[7] Steve Bratt. Semantic web, and other technologies to watch. `http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24)`, 2006. This is an electronic document. Date of publication: November 30, 2006. Date retrieved: May 15, 2011. Date last modified: [Date unavailable].

[8] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, WWW Alt. '04, pages 74–83, New York, NY, USA, 2004. ACM.

[9] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13:377–387, June 1970.

[10] International Electrotechnical Commission. Part 4: Discussion of concepts. Technical report, International Electro technical Commission, 1998.

[11] Louis Vincent Cuel Roberta, Delteil Alexandre and RIZZI Carlo. Knowledge web technology roadmap the technology roadmap of the semantic web. Technical report, Knowledgeweb, 2007.

[12] Ian Dickinson and Michael Wooldridge. Towards practical reasoning agents for the semantic web. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, AAMAS '03, pages 827–834, New York, NY, USA, 2003. ACM.

[13] Larry P. English. *Improving data warehouse and business information quality: methods for reducing costs and increasing profits.* John Wiley & Sons, Inc., New York, NY, USA, 1999.

[14] Oeystein Fossen. Integrerte operasjoner - akselerert laering; hoeydepunkter fra paagaaende olf prosjekt, stavanger 12. juni. olfs konferanse om integrerte operasjoner 12. og 13. juni 2007. `http://www.olf.no/getfile.php/Konvertert/www.olf.no/Aktuelt/Dokumenter/OLFs%20IO-konferanse%202007%20-%20%C3%98ystein%20Fossen%2C%20Arbforskinst.pdf`, 2007. This is an electronic document. Date of publication: June 13, 2007. Date retrieved: July 15, 2011. Date last modified: [Date unavailable].

[15] Barbara L. M. Goldstein, Sharon J. Kemmerer, and Curtis H. Parks. A brief history of early product data exchange standards - nistir 6221, 1998.

[16] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml. W3c member submission, World Wide Web Consortium, 2004.

[17] International Standardization Organization (ISO). Information processing systems - concepts and terminology for the conceptual schema and the information base. Iso publication, JTC 1/SC 32, 1987.

[18] International Standardization Organization (ISO). Industrial automation systems and integration - integration of life-cycle data for process plants including oil and gas production facilities - part 2: Data model. Iso publication, ISO/TC 184/SC4, 2004.

[19] International Standardization Organization (ISO). Data quality – part 110: Master data: Exchange of characteristic data: Syntax, semantic encoding, and conformance to data specification. Iso publication, ISO, 2009.

[20] International Standardization Organization (ISO). Data quality – part 120: Master data: Exchange of characteristic data: Provenance. Iso publication, ISO, 2009.

[21] International Standardization Organization (ISO). Industrial automation systems and integration - integration of lifecycle data for process plants including oil and gas production facilities - part 8: Implementation methods for the integration of distributed systems: Owl implementation. Iso publication, ISO/TC 184/SC4/WG3 N2662, 2009.

[22] sub-committee SC 4 ISO Technical Committee TC 184, A.s.a.i. Industrial data, iso/ts 8000-100:2009 data quality - part 100: Master data (overview). Technical report, ISO, 2009.

[23] Bijan Parsia Jennifer Golbeck and James Hendler1. Trust networks on the semantic web. Technical report, University of Maryland, 2003.

[24] Martin G. Skjæveland Johan W. Klüwer and Magne Valen-Sendstad. Iso 15926 temapltes and the semantic web. Position paper, Information Risk Management, Det Norske Veritas, 2011.

[25] Y. Katz and B. Cuenca Grau. Representing qualitative spatial information in owl-dl. Technical report, In Proceedings of OWL: Experiences and Directions Workshop, 2005.

[26] KonKraft. Konkraft-rapport 2, produksjonsutviklingen paa norsk sokkel. Technical report, KonKraft, 2007.

[27] Deborah L. McGuinness and Frank van Harmelen. Owl web ontology language overview. Technical report, W3C, 2004.

[28] J. Minker. *On indefinite databases and the closed world assumption*, pages 326–333. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[29] Jeff Z. Pan Nophadol Jekjantuk, Gerd Groner and Edward Thomas. Towards hybrid reasoning for verifying and validating multilevel models. Technical report, University of Aberdeen, United Kingdom and University of Koblenz-Landau, Germany, 2010.

[30] Rafael Gonzalez-Cabero Oscar Corcho, Asuncion Gomez-Perez and M. Carmen Suarez-Figueroa. Odeval: a tool for evaluating rdf(s),

daml+oil, and owl concept taxonomies. Technical report, The Department of Artificial Intelligence in the School of Computer Science of the Universidad Politecnica de Madrid, 2004.

[31] Bijan Parsia and Evren Sirin. Pellet: An owl dl reasoner. Technical report, MINDSWAP Research Group, University of Maryland, 2004.

[32] Bijan Parsia and Evren Sirin. Pellet: A practical owl dl reasoner. Technical report, MINDSWAP Research Group, University of Maryland, 2006.

[33] eborah L. McGuinness Paulo Pinheiro da Silva and Richard Fikes. A proof markup language for semantic web services. Technical report, Knowledge Systems Laboratory, Stanford University, Stanford, CA 94305, USA, 2006.

[34] Bjoern Rasen. Time for integrated action. `http://www.npd.no/English/Emner/IO/IO-forum/11.03.08+time+for+integrated+action.htm`, 2008. This is an electronic document. Date of publication: July 7, 2008. Date retrieved: june 3, 2011. Date last modified: [Date unavailable].

[35] Boris Motik Rob Shearer and Ian Horrocks. Hermit: A highly-e?cient owl reasoner. Technical report, Oxford University Computing Laboratory, 2005.

[36] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002.

[37] James Hendler Tim Berners-Lee and Ora Lassila. The semantic web. *Scientific American*, 2001.

[38] Matthew West. Developing high quality datamodels. Technical report, Shell International Limited, 1996.

[39] Tim Finin Youyong Zou and Harry Chen. F-owl: An inference engine for semantic web. Technical report, Computer science and eletrical engineering department, University of Maryland, 2005.