Semantic Web for Data integration within Oil&Gas or maritime

By
Li Kuang

Thesis submitted in Partial Fulfillment of the Requirements for the Degree Master of Technology in Information and Communication Technology

Faculty of Engineering and Science
University of Agder

Grimstad
May 2009

Abstract

The Semantic Web technology has become quite popular recently. The ontology-based data integration architecture is the important part of Semantic Web technology. It enables the sharing of concept with common schemas and also enables the representation of the information in machine understandable way. Therefore, the data source could be processed automatically.

This master thesis is using the Semantic Web technology for data integration within Oil&Gas or maritime industries. The Norwegian Oil Industry Association (OLF), which takes a leading role in next generation Integrated Operations, has developed an Oil&Gas ontology for data integration across multi-domains. The Oil&Gas ontology is based on the ISO15926 standard.

The master thesis devotes to clarify to what extend the Semantic Web technology and ISO15926 standard can be used together to improve the functionality of the Safety Instrument System (SIS), which is provided by the problem owner. This master thesis introduces a data integration framework according to the implementation methodology of ISO15926 standard and the architecture of Ontology based data integration. The framework uses the software Protégé [12] as a modeling tool to create a model for the Cause&Effect matrix based on the ISO15926 standard. The Jena API [33] is used to map the real-time data to the data source ontology. This project also implements a prototype using the Jena API including a querying system and reasoning system. The implementation of querying system, which gets the information intelligently, shows the improvement of data quality and accessibility. And the reasoning of the ontology shows the ability of automatic processing of the real time data, which has obviously improved the software functionality.

Preface

This thesis is proposed by Origo Engineering AS. First of all, I would like to thank for the high level guidance of my technical supervisor Andreas Prinz. He has given me some good advices when I went lost in the wrong direction.

And I would also like to thank my supervisor Trond Friisø and his colleague Trond K. Nilsen from Origo, who helps me to understand the problem of the thesis clearly and gives good suggestions all the time. At the same time I want to thank Origo for providing me the raw data.

Finally, I would like to thank Terje Gjøsæter, who helped me a lot with language usage in the report writing. And thanks for the effort of my thesis contact Jan Pettersen Nytun.

Grimstad, May 2009
Li Kuang

Table of contents

List of figures

List of tables

1. Introduction

Chapter 1.1 introduces the background of the project, including the domain we work with. Chapter 1.2 introduces the history of data integration. It explains why we need data integration, and why we need semantic data integration. Chapter 1.3 gives the outline of the rest part of the project and the report.

1.1 Background

The Oil&Gas industry does more and more rely on the information and communication technology. It improves the efficiency and safety of the industry. There are large amounts of data being collected and optimizing the utilization of these could bring great benefit to the economy and environment protection. Currently, most of the data sharing in the Oil&Gas industry is in XML format, which provide well-formed rules for data representation. However, XML is still not well enough, since it does not contain any semantemes of the data. Therefore, the RDF and OWL schemas are introduced for knowledge representation. These are the basic elements of the Semantic Web. The Semantic Web is known as the extension of the current web. *"It facilitates navigation and meaningful use of digital resources by automatic process. Searching, requesting, execution, and payment for services can be accomplished without the need of human interactions."[3]* The Semantic Web is ontology based, which enables the reasoning of information.

OLF is a professional body and employer's association for oil and supplier companies. It is the head organization of developing Integration Operations (IO) for the industry. OLF's IO project is responsible for providing standards, integrated solutions, and technologies for supporting operational decisions of the onshore control centers for offshore installations. It has developed IO generation 1, and plans to implement generation 2. *"The aim of the first generation (IO G1) is to integrate processes and ability to support offshore operations. The aim of the second generation (IO G2) is to help operators utilize the vendors' competences and services more efficiently than today [3]."* A challenge for IO G2 is data collection across disciplines and dissimilar data systems. Semantic Web is assumed to play a key role in data integration for integrated operations together with ISO 15926, SOA/Web Services is also assumed to be an important element. OLF have facilitated the development of an Oil & Gas Ontology (a defined terminology for oil exploration and production) to enable this data integration.

1.2 History of Data integration

Data integration is an old research topic, but that does not mean that it is not valuable to research. Due to the requirement of more and more large scale and deep data integration, there are many new and complex problems arising, which lead to lots of technologies being developed, like semantic technology. To better understand the data integration, let's start from the beginning.

The first question could be why do we need data integration? Actually, there are two reasons: First, facilitate data access of heterogeneous data sources in a single access point. Second, data from complementation information systems need to be combined to gain a comprehensive basis [1]. In fact, many applications can gain advantages of integrated information. For examples, CRM (Customer relationship management) can improve custom service by integrated custom information and service information. Integrated information enables transactions and service over network for e-commerce and e-business [1].

Data integration deals with the data transparency problem of distributed systems. That means it has to make the users think they are accessing a single information system with homogeneous data structures. But actually the data is physically distributed over heterogeneous data sources. In this way all the data has to be represented with the same standard.

Data integration has evolved from structural to semantic integration. Traditional integration is based on relational and functional data model that integrate with one single global schema [1]. With the development of Internet and web applications, mediator and agent systems have become popular in the data integration. However, providing explicit and precise semantics is the critical problem of data integration. In the requirement of integration with heterogeneous data sources, the one single global schema and mediator or agent system is not possible to fulfill the needs. Therefore, the ontology is introduced for providing explicit, formal, conceptualized definition of the data source. Compared with the former integration, the ontology based data integration reduced the semantic ambiguous by providing shared understanding. For example, the same syntax may have different meaning in two databases, but in the ontology, it will give more complete definition to each syntax to avoid the ambiguousness.

One ontology approach is only suitable for the integration within a single domain. It requires all the data sources mapping to the common ontology. As the multi-domain data integration the single ontology will have limited abilities to provide precise meaning of the data. Therefore, multi-ontology approach (e.g. ISO15926) is introduced. Multi-ontology approach divides ontologies into a hierarchy. The top level (upper) ontology is a highly abstract data model that provides meta-concept and meta-data for the lower ontology.

1.4 Report outline

The rest of this thesis is organized as follow:
Chapter 2 states the problem, delimitates the problem from different aspects, and presents the scenarios
Chapter 3 analyzes the principles of Semantic Web technology and the ISO15926 standard.
Chapter 4 shows the design specification of the data integration framework
Chapter 5 shows the mapping implementation both from Cause&Effect matrix to ISO15926 and from real-time data to data source ontology
Chapter 6 shows the prototype implementation of the querying and reasoning system
Chapter 7 shows the proving of the concept of reasoning and testing of querying.
Chapter 8 discusses the possibility of automatic mapping from data source ontology to domain

ontology
Chapter 9 gives the conclusion of the work, and point out the future works.

2. Problem description

The Chapter 2.1 gives the problem statement of this project. It describes the current problem of Origo Engineering AS and the goal they want to achieve. The Chapter 2.2 delimitates the problem from different aspects including data integration methodology (Chapter 2.2.1), goal of data integration (Chapter 2.2.2), research problem (Chapter 2.2.3), data integration environment (Chapter 2.2.4), level of data integration (Chapter 2.2.5), and semantic web technology in data integration (Chapter 2.2.6). The Chapter 2.3 gives the roles that can benefit from data integration, and scenarios of data integration. It also analyzes the Cause&Effect matrix in Chapter 2.3.1.

2.1 Problem statement

Origo Engineering AS is a company that provides safety systems for the customers. Such as Fire&Gas, Emergency shutdown and Process shutdown systems, which are often used in the drilling/well maintains. Figure.1 shows an example of a safety system named Safety Instrumented Systems (SIS). The functions of SIS are to discover and prevent situations that can escalate into larger accidents. The different systems are independent, and together they form a chain of barriers to prevent accidents [6].

SIS is passive during normal operation and it has to be verified regularly that they actually will work on demand. This could be done by explicit full-scale tests. However full-scale test is time consuming. Alternatively, logged data from unplanned shut-downs could be used to verify activated functions. Each system in the SIS has a real-time database that stores logged data. In order to verify the functions, it is necessary to collect data from various sources. These data sources are mostly heterogeneous. Therefore, data integration is needed to get better functions.

Figure.1 shows an overview of the SIS (Safety Instrument System). According to [7], it contains the following systems:

- IMS (Information Management System)
    - (1) Long term storage of alarms and events
    - (2) Trend data storage
    - (3) Long term storage of selected measurement values
    - (4) Alarm analysis
    - (5) Administrative tasks
- PCS (Process Control System)
- PSD (Process shutdown)
    - (1) Process protection
    - (2) Equipment protection
- F&G (Fire & gas)

(1) Alarm and annunciation
(2) Fire fighting

- ESD (Emergency shutdown system)
  (1) Blow down and flare/vent
  (2) Ignition source control
  (3) Process segregation

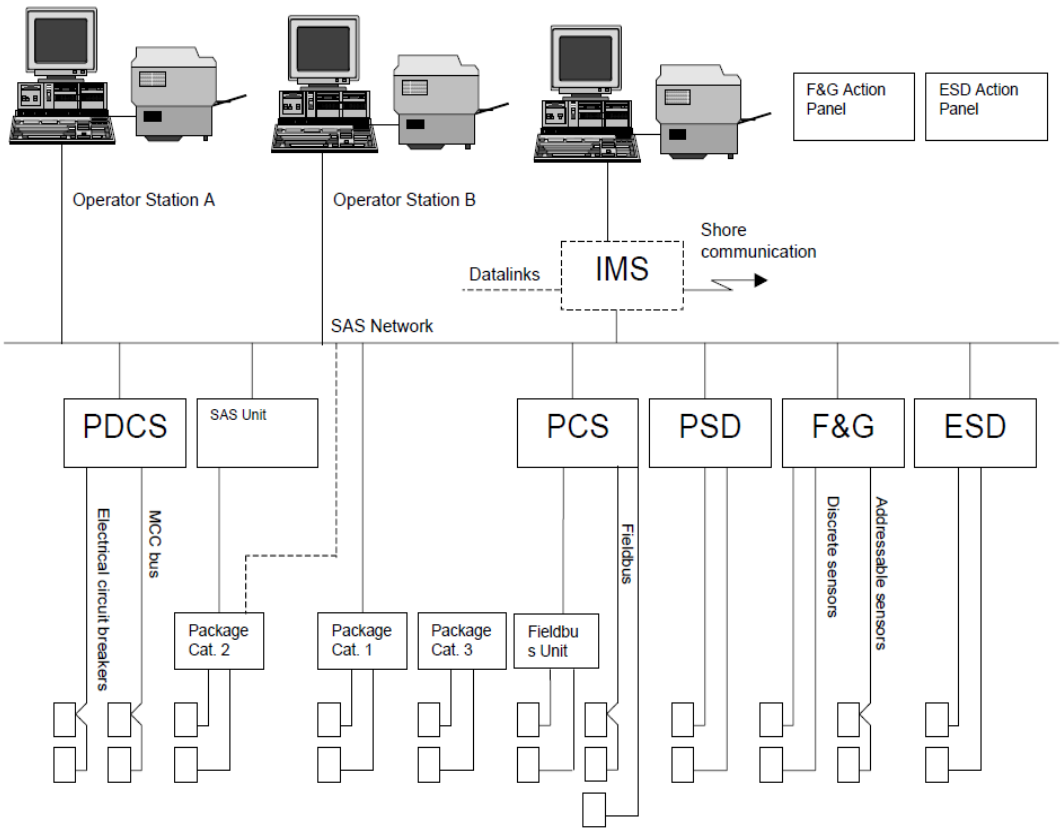


Figure 1 System overview of SIS cited from [7]

Origo has developed a prototype of a tool for online analysis of Safety Instrumented Systems (SIS) in operation. This tool will collect data from various sources, analyze them and report the goodness of the SIS in operation. However, this tool has been developed without using the Oil & Gas Ontology. To prepare for a role within IO G2 Origo want to supply this kind of information with other oil and supplier companies. Therefore, Origo want to clarify that to what extend the semantic web, the Oil and Gas ontology, and ISO 15926 can be used to optimize the current system they already have. As we mentioned in Chapter 1.1 the data integration of IO generation 2 is based on Oil&Gas ontology which is the part of ISO15926. And usage of the ontology is the basic building blocks of Semantic Web.

As it shown in Figure.2, Origo Engineering AS collects data from sensor network. These data should be integrated based on the ISO standard. Therefore, the incorporation between heterogeneous data sources could be achieved.

This project is devoted to verify if the framework for data integration based on the Oil&Gas ontology is suitable for useing in Origo Engineering AS, to clarify how the Oil & Gas Ontology could be incorporated. Based on this framework and the prototype that Origo developed, this project should suggest a solution for data integration in an IO context. And also implement a prototype and demonstrate the use of it. A test case will be developed by Origo for use in the test of the implementation.

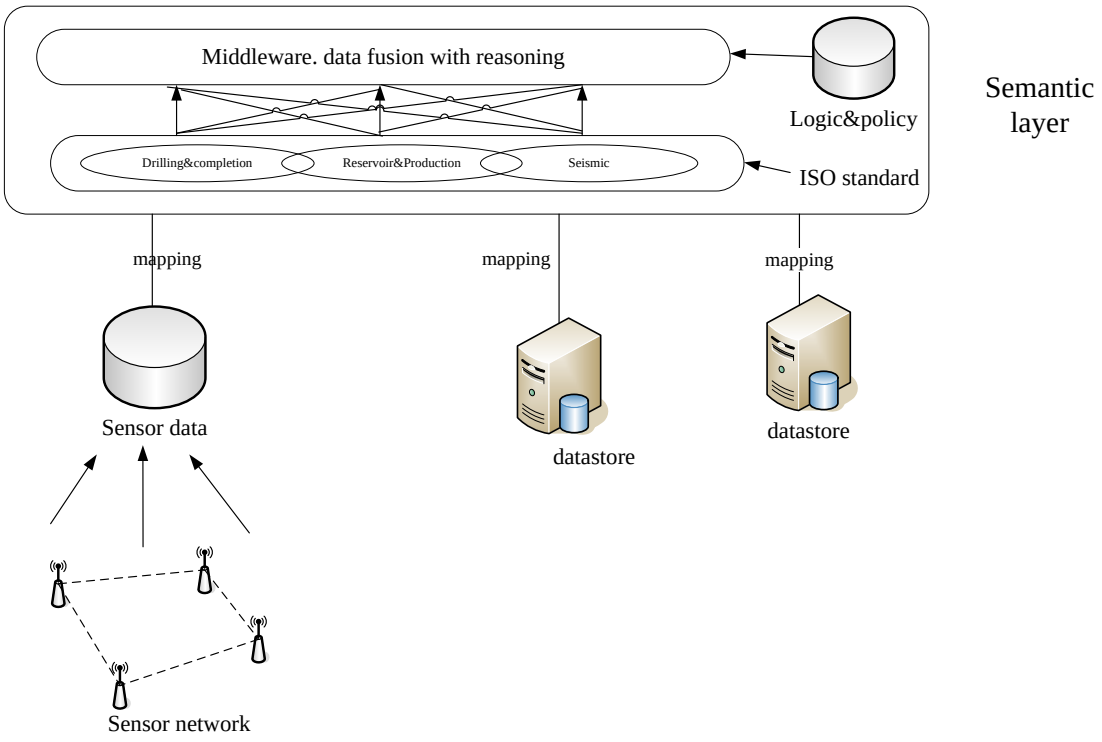


Figure 2 Architecture for the integrated information framework, cited from [3]

2.2 Problem delimitations

2.2.1 Data integration structure

As pointed out in [8], the conceptually data integration structure is arranged in four different layers: data source, data source ontology, domain ontology, and view. The structure has a lot of advantages. The system is flexible: it is better to react on changing, since the changes of one layer will not affect other layers. And the system is extensible: it is easy to add a new data source with new schemas into the system. As it shows on Figure.3, it compares the conceptual layering presents in [8] and practical layering that is used in this project, similar layering can also be found at [5]:

- Data sources: the data sources layer stores the raw data. Most of the times the data stores in the relational database, such as MySQL and Oracle. On this project, the data source is the real time data from the SIS system.

- Data source ontology: the “data source ontology” is not real ontology, since it does not represents a shared conceptualization of a domain [8]. It is the schemas of the data sources, such as Cause and Effect Matrix in this project.
- Domain ontology: the domain ontology is the real ontology, which provides the terminology and taxonomy for the domain. *“It describes the shared conceptualization of the domain at hand. It is a reinterpretation of the data described in the data-source ontologies and thus gives these data a shared semantics” [8]*. This project we have the Oil&Gas ontology as domain ontology.
- View: this layer could use the common user interface to query for the information. The semantic querying and reasoning system is defined by the software engineer who is familiar with the domain ontology and developing tools of ontology.

Each layer is connected to another layer by mapping. These mappings are exactly the objectives of this project. We need to clarify to what extend the four layers could provide better functionality. There are three mappings:

- From data sources to data source ontology can be mapped automatically.
- From data source ontology to domain ontology are usually manually created. Although, I find some papers [9] and [10] that try to research on the automatic mapping, it has been found not suitable for this project because of the complexity of the domain ontology.
- From domain ontology to view should be defined manually. According to the need, we specify which kind of information that we need to query. We also need to specify the methods to reasoning the queries, so that the queries of the users can be understood by the computers.

The detailed design of each mapping above can be found at the design specification part of this report (Chapter 4).

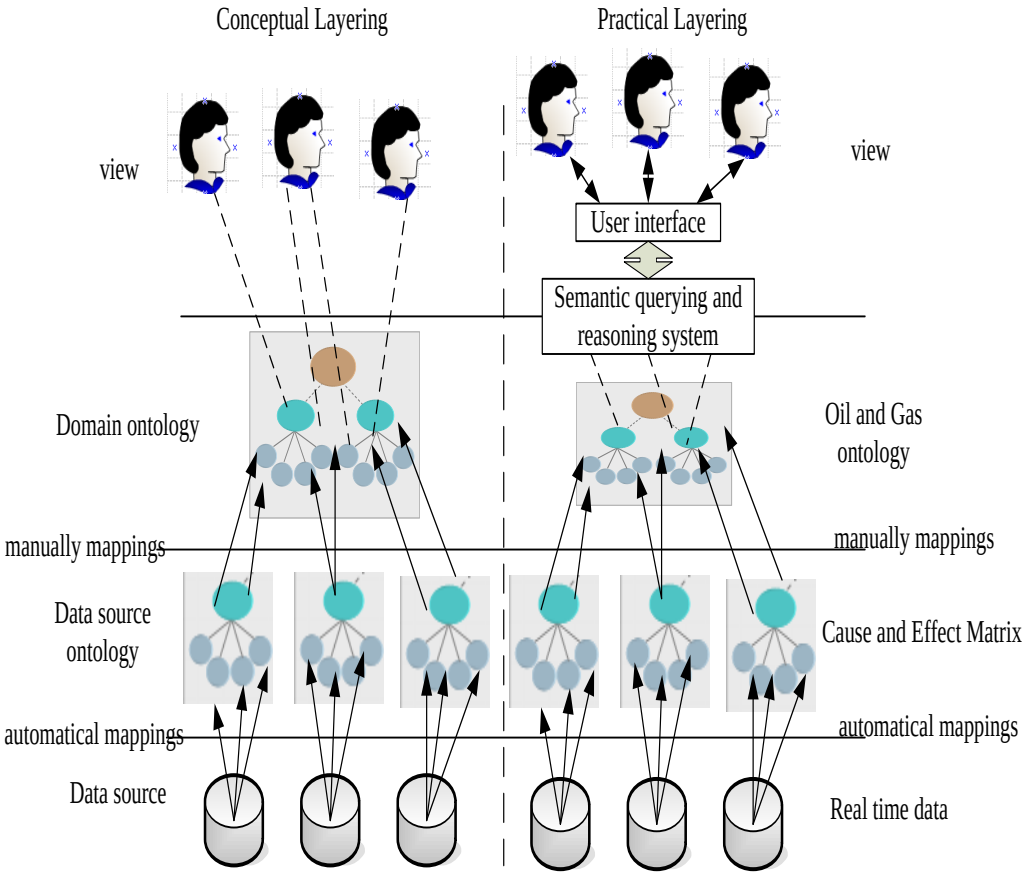


Figure 3 Comparison of the Conceptual Layering and Practical Layering of the ontology

2.2.2 Advantages of data integration

As [3] concludes: "*Ontology based Data integration provides:*

1. *improved data quality and accessibility*
2. *significant cost reduction with change of software*
3. *increased flexibility with organizational changes*
4. *improved software functionality"*

The four aspects can be used to evaluate the quality of data integration. There are all the goals we want to achieve in this project. However, the 2 and 3 are not easy to verify as we cannot change the software and organizational currently. Therefore, we developed a prototype called semantic querying and reasoning system to prove the 1 and 4. The implementation of querying system, which gets the information intelligently, shows the improvement of data quality and accessibility. And the reasoning of the ontology shows the ability of automatic processing of the real time data, which will obviously improve the software functionality.

2.2.3 Importance of the research on the Ontology

The Semantic Web technology is young. It grows more complex as the domain scales up. The

ISO15926 is a standard that could be used across domains. Due to the large scales of the domain, there are lots of challenges of the ISO15926. As it is noted in [3], "*Research is needed to ensure ontology that:*

- *provides meaning to data collected from sensors and agents and thus turns data into information*
- *provides a framework for unambiguous exchange of information*
- *data integration from different domains*
- *supplies a logical structure that can be used to make deductions about the state of the system on the basis of the data collected"*

This report presents the research work on the theoretical background (Chapter 3). It researches on the Semantic Web technology on the layering view. It finds out the methodology of representing the knowledge information with XML, RDF, and OWL. And how to inferring and reasoning the knowledge based on the ontology we defined. The Semantic Web provides the ability to turn data into information and unambiguously represent the information. The reasoning ability shows how to make deductions about the state on the basis of the data collected. Also the reasoning will be proved in the prototype.

In Chapter 3 it also investigates the ISO15926 standard on the layering view. It finds out how does the ISO15926 support data integration across domain? How does it provide unambiguous representation of the data? How does the data turn into information? What kind of information? How does it reach the robust and complete?

2.2.4 Environment of data integration

Every data source has its own structure and semantics. It is theoretically not possible to solve all the problems of heterogeneous data sources. Therefore, different kinds of integration may depend on the specific requirement of the customer. As it was concluded in [2], the particular integration task depends on: *"*
(1) The architectural view of an information system.
(2) The content and functionality of the component systems.
(3) The kind of information that is managed by component systems (alphanumeric data, multimedia data; structured, semi-structured, unstructured data):
(4) Requirements concerning autonomy of component systems,
(5) Intended use of the integrated information system (read-only or write access),
(6) Performance requirements,
(7) The available resources (time, money, human resources, know-how, etc.)"

As for (1), the architectural of the SIS system has been introduced in the Chapter 2.1. For (2), the function and content of the component system is introduced in the scenarios (Chapter 2.3). For (3), the kind of information is included in the Cause&Effect matrix. It is introduced in Chapter 2.3.1. For (4) the autonomy of component system is implemented through the reasoning of the ontology. The reasoning is implemented in Chapter 5.1.5 Voting implementation, and verified in the Chapter

7.1 reasoning verification. For (5), the intended use of the integrated information system, we focuses on the optimization of querying of the information we need. It means the integrated system is read-only. The (6) and (7) are not considered in this project, because of time and resource limitation.

2.2.5 Level of the data integration

According to [1], beside the specific requirement, several kinds of heterogeneity typically have to be considered. These include differences in: "*(1) hardware and operating systems, (2) data management software, (3) data models, schemas, and data semantics, (4) middleware, (5) User interfaces, and (6) business rules and integrity constraints.*" In this project, we will not consider too much about the hardware and middleware heterogeneity. Since most of the software is Java based, which means that they are platform independent, so we don't need to deal with the operating system heterogeneity. We use common ontology language OWL and RDF to represent and share information, therefore the data management software heterogeneity is not a problem. In fact, we mainly focus on the (3) data model, schemas, and data semantics and (6) business rules and integrity constraints. We have also developed a common user interface using JSP.

In [2] the data integration is divided into levels: (1) manual level (2) user interface level (3) application level (4) middleware level (5) data access level (6) data storage level. In manual level the user has to combine the information manually. That requires the user to be familiar with different kinds of user interfaces and query language. Moreover the user has to be a domain expert. On the user interface level the users utilizes the common user interface. However, the information integration still has to be done manually, since the data structure is still heterogeneous. The application level uses the programming to encapsulate the heterogeneous data. It is useful when the amount of data format is small. As the amount of data increases the application will be complex and slow. The middleware share the responsibility of applications. The data access level provides global applications that can access the virtual data for the physically distributed system. This project we do the data integration in the data storage level. We map the meta-data to a new format with semantic definition.

2.2.6 Semantic Web technology used in this project

XML, RDF, and OWL are the basic elements of the Semantic Web technology. Currently, most of the information is shared, transferred and stored in the XML format. However, the XML document does not provide semantic meaning for the data source. Therefore, to lift the XML document to RDF or OWL is the only way we can find to introduce semantic concept into the data source. As we know the POSC Caesar Association (PCA) [24], which is a global, nonprofit member organization that devotes in improvement of international standard in Oil&Gas industry for interactive of data [3], has done a lot of job by mapping Oil & Gas ontology into OWL. Therefore, this project will use the OWL file that PCA provided as the basis. The details of XML, RDF, and OWL will be analyzed in the theoretical background (Chapter 3).

Figure.4 shows a concrete data integration structure of this project. The Sensor network collects data in the Oil&Gas industry and store in the Origo database in XML format. The XML file represents the syntax of the metadata. In order to get the semantic of the metadata, you have to use the Tag name to check the Cause & Effect standard manually. The problem will delimited to mapping the real-time XML documents to an OWL instance, and the Cause & Effect standard into OWL. The OWL is based on the Oil&Gas standard. And I will implement a prototype, which support querying the data model through OWL and reasoning based on OWL. The ISO15926 part2 and ISO15926 part4 is available as meta-model for the user defined ontology.

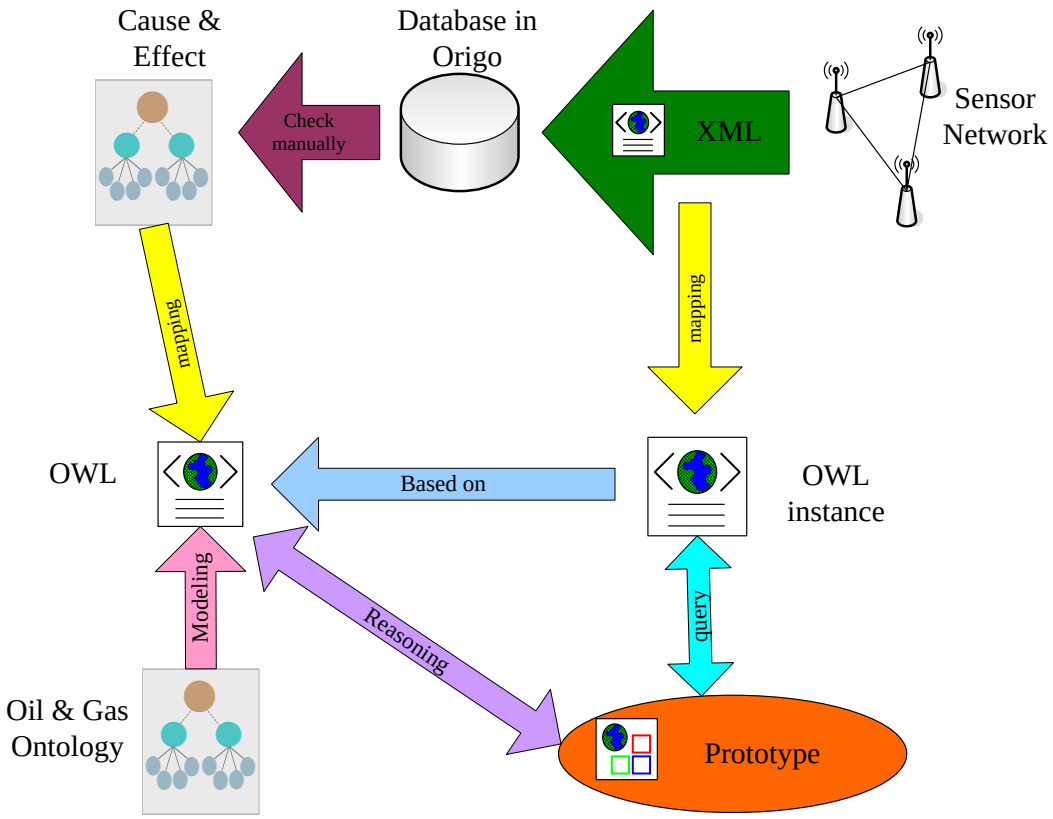


Figure 4 Concrete data integration structure

2.3 Roles and Scenarios

2.3.1 Analysis of Cause&Effect matrix

Before introducing the mapping approach it is necessary to analyst the Cause&Effect matrix first. As it shows in Figure.5, it is the Cause&Effect chart for “SEACABLE TRANSFORMATOR ROOM AREA-NORTH” that is located in area U51-2. It is divided into CauseTag and EffectTag, they matches by the “X” and “&” symbol. “X” means direct match, while “&” means that an intersection of the CauseTag and the EffectTag match. CauseTag contains elements “Description”, “Voting”, “From”, “Input Type”, “Note”, and “Tag Number”. Likewise, EffectTag contains elements “Tag Number”, “Output Type”, “Action” and “Note”. The elements will be described

below:

- Tag Number: There are two kinds of tags CauseTag and EffectTag. Every Tag Number is unique in the whole system. The tag name contains information, for example the tag "U51_DG001" put the area information U51 in the tag number.

- Description: Simple description of the CauseTag. It gives the semantic of the tag. Such as "Single Gas Low" in figure.3 it means a single sensor has detected that there is a gas concentration above the low alarm limit in the area. The description in bold type are the classification of the tags, it points out which system the tag belongs to. For example, the tag "U51_DG004" belongs to the "Gas detection Ventilation" system.

- Voting: The voting means that the status of the candidate is evaluated according to the status of the voter. There are three kinds of voting type in this table: "1ooN", "2ooN", "NooN". "1ooN" means that the voting tag is true if any of the voters is true. Similarly, "NooN" means the voting tag is true if and only if all the voters are true. The CauseTag who contains the voting type is the candidate of that voting. Voters are the CauseTags that located above the candidate CauseTag in the table. For example, the tag "O87C_U51_2_SGL002" has a voting type "1ooN", than it is a candidate of this voting. And the voters of this voting are tag "U51_DG004", "U51_DG005", and "U51_DG006". As it shows in table.1, only on the $C_3^1 = 3$ (N=3) conditions the candidate tag "O87C_U51_2_SGL002" is true. Otherwise, it will be false.

O87C_U51_2_SGL002	**U51_DG004**	**U51_DG005**	**U51_DG006**
true	true	false	false
true	false	true	false
true	false	false	true

Table 1 state table of "1ooN" voting

For the candidate tag "O87C_U51_2_CGL002" which has a voting type of "2ooN", it also has $C_3^2 = 3$ (N=3) conditions that will be true showing in table.2.

O87C_U51_2_CGL002	**U51_DG004**	**U51_DG005**	**U51_DG006**
true	true	true	false
true	true	false	true
true	false	true	true

Table 2 state table of "2ooN"voting

- From: "F&G" means the tag information is collected from the sensor in the Fire&Gas area. The "Voting" means the tag is coming from the voting system.

- Input Type and Output Type: The chart contain data type: "INT", "DI", "AI", "Loop 0", "Loop 1", "Bus", "DO". The "INT" is integer, "DI" is digital input, "AI" is analog input, and

"DO" is digital output. Loop means that several detectors are connected in a loop.

- Action: Simple description of the effect action caused by the activity in the Fire&Gas system. There are some classifications of the actions on top of the chart, such as "CAP MARIX", "Fire Protection". "CAP MARIX" is the critical alarm panel.

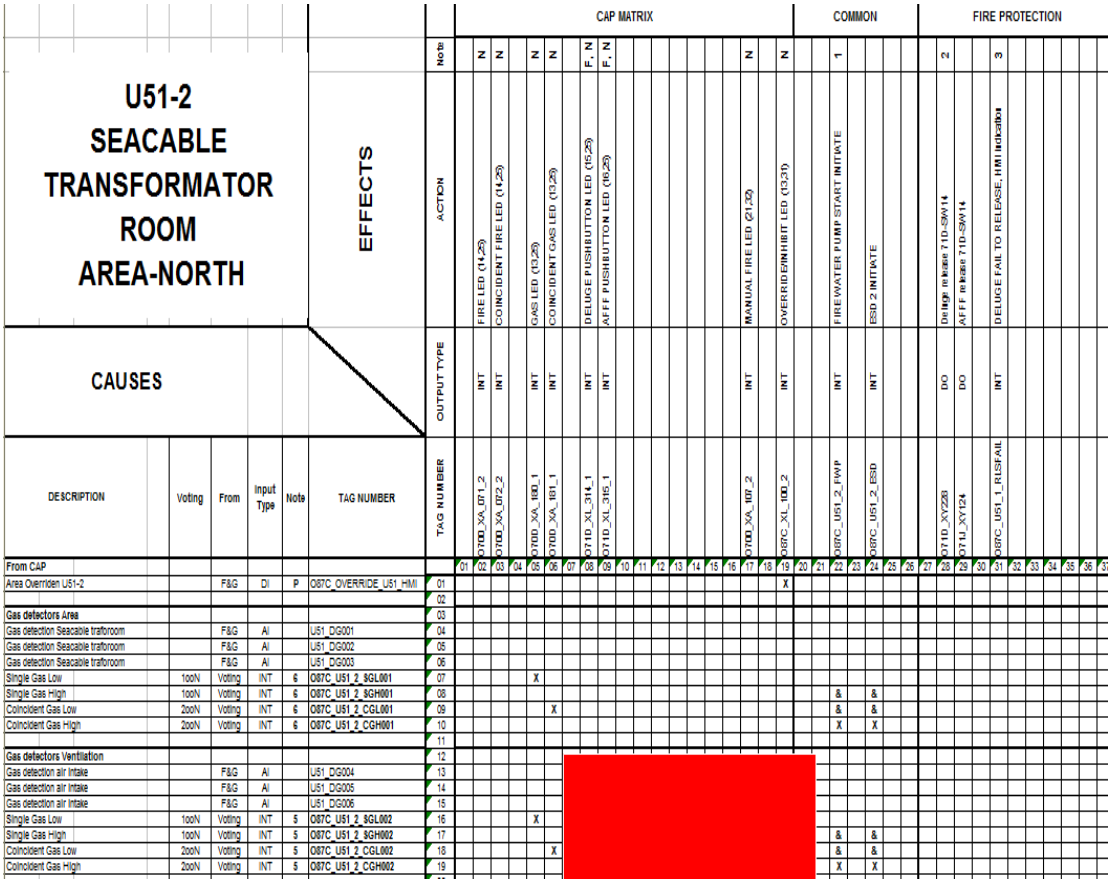


Figure 5 Cause&Effect sheet of U51-2

- Note: The note element indicates that there are some extra restrictions or information attached to the tag. There are two kinds of notes: general note and specific note. General note is the notes for all the Cause&Effect chart. Specific notes are only for the chart that contains them. For example, the tag "O87C_U51_2_CGL002" has the note 5, which is "Low alarm limit to be 5%. High alarm limit to be 10% LEL"

2.3.2 Roles in the Semantic data integration

By introducing semantic data integration based on ISO15926 within the SIS system showed above, the following roles can get some advantages.

- External developer/ System integrator:

External developer or system integrator could understand the information provided by other systems without a domain expert, because of the sharing of the same Oil&Gas ontology. The data based on ISO15926 standard are extensible and reusable. The data does not need to be modified before it is reused.

- Internal developer

  The semantic web technology enables the information to be understood by the computer. Therefore, the internal software developer could improve the current system. The current system in F&G uses manual searching or simple matching to get the real time information. That is not intelligent enough, and usually takes long time because of the redundant information. The knowledge representation of the data supports intelligent querying of the real-time data. For example, given an area name "U51-2", the query engine could get all the states of the sensors at that time. The reasoning could also bring automation of the process control. For example, in the Cause&Effect matrix of U51-2, if the two voting tags U51_DG004 and U51_DG005 actually were above alarm level, the reasoning engine will infer that the "Confirmed Tag" O87C_U51_2_CG002 is set to high. So as the Effect tag is set to high as well.

- Safety person

  The safety person is responsible for quality control and testing of the system. For example, the safety person in F&G can check the feedback of the alarming. The safety person can check if the valves are actually closed after the F&G has sent an ESD initial signal to ESD system. And the safety person from ESD system can check if the ESD receives the alarming information in time. The ontology can provide explicitly definition of the information, which is critical for the safety person. A ambiguous information in the integrated system may lead to disaster.

- Control center

The semantic data integration enables transformation from the real-time data to useful information as soon as possible. Therefore, the control center could make proper decision in real time. For example, the IMS system uses a control panel to control the whole system, based on all the information from F&G, ESD, PCS, and PSD

2.3.3 Scenarios of the SIS

Based on the SIS system described in problem statement, we will have the following subsections. Figure.6 shows the collaboration of the F&G, ESD and PCS systems. The sensors in area U51-2 collect safety data from off-shore. The Fire&Gas system processes the real time data according to the Cause&Effect matrix. If some accident (such as a fire) happens in area U51-2, the F&G system will send a signal to initiate the ESD system. The ESD system will shut down the

Emergency Shutdown Valve immediately. The PCS system records the state of the limit switches. This can be used to calculate the actual time the valve used to close.

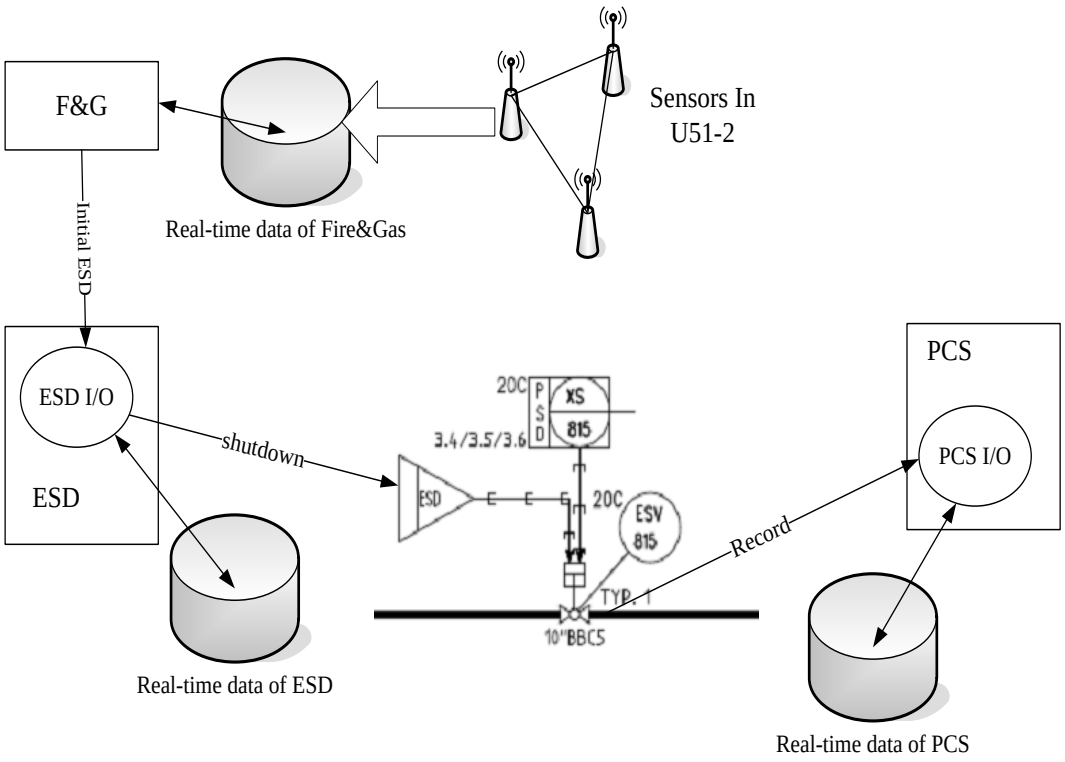


Figure 6 Collaboration of F&G, ESD and PCS

1. Assume that two detectors detect gas in U51-2. See the Cause&Effect sheet in Figure.7. From the Fire&Gas data the safety person of Fire&Gas want to verify:
   a) that two of the voting tags actually were above alarm level
   b) that the “Confirmed gas” tag, O87C_U51_2_CG002, was set to true
   c) that the intersystem tag for signaling to the ESD system, O87C_U51_2_ESD, was set to true

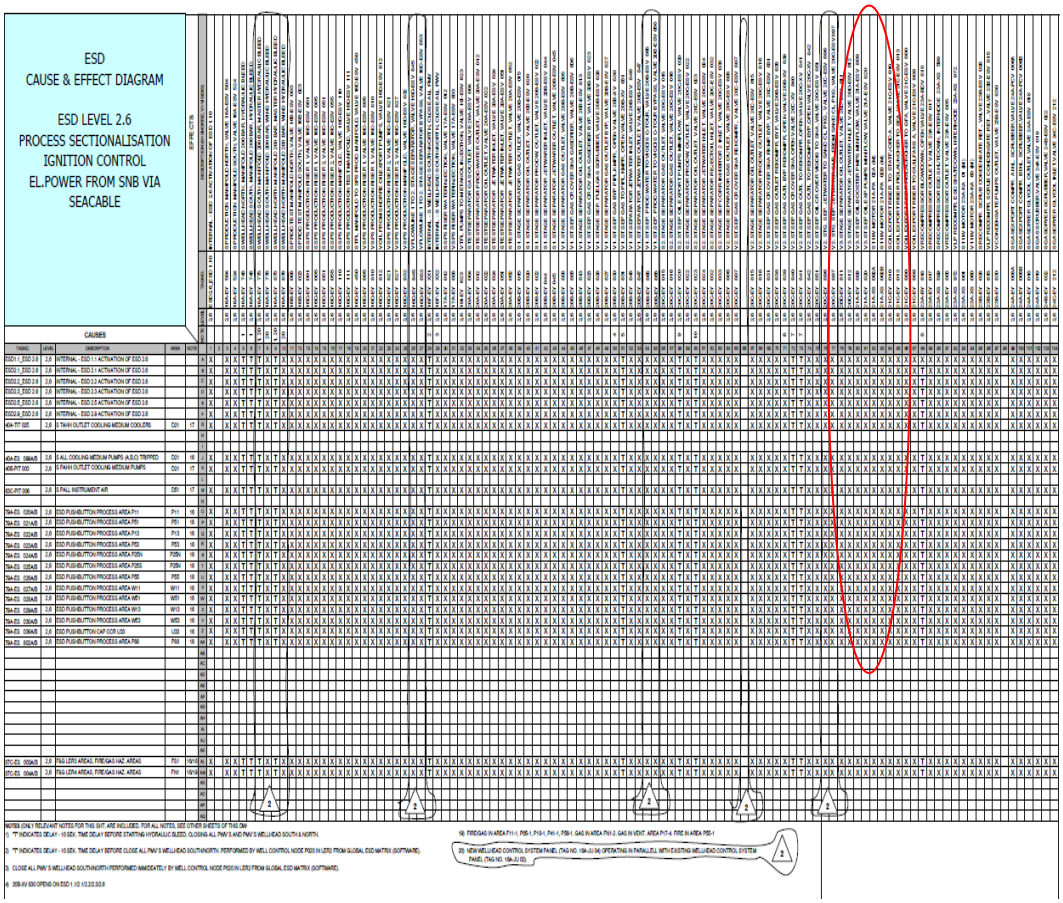


Figure 7 Cause&Effect sheet of ESD

2. From the Emergency Shutdown System Hierarchy the internal software developer of ESD need to find out which actions the ESD 2 imply. Among others, all the Emergency Shutdown Valves should be closed.

3. From the log of the ESD system the safety person would like to check that
   a) the signal from the F&G system was received, 87C-ES 003A/B (there are modifications on going at this platform so there are some inconsistencies, but for illustration it is good enough)
   b) The output signals for closing the valves are set. For illustration we pick one single valve, e.g. 20C-ESV 815. The output signal is called 20C-EY 815. This goes to a pilot valve that controls an actuator that closes the valve.

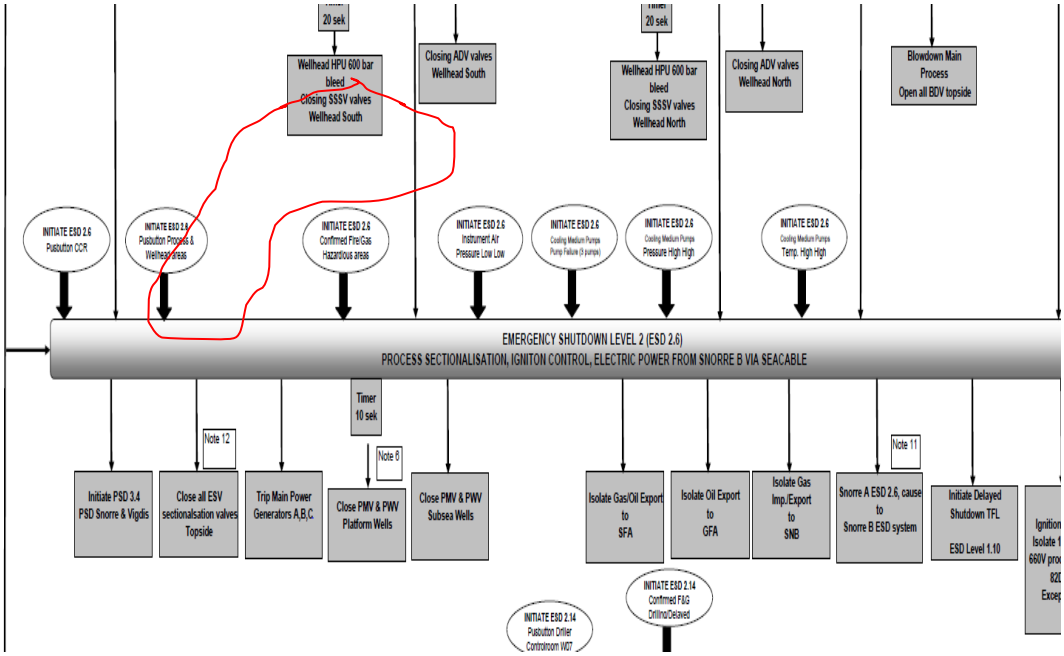


Figure 8 A part of the Emergency shutdown system hierarchy

4. A part of the Process and Instrument drawing (P&ID) is shown in figure.6. From the datasheet for the valve, the closing time is missing. From the NORSOK standard S-001, we are then guided to use 2 seconds per inch. The valve is 10'', so we assume that the closing time should be about 20 seconds. Let's say between 15 and 30 seconds. From the documentation system, we see that there are two tags related; 20C-EZSH 815 and 20C-EZSL 815, that are limit switches. The first one is indicating closed valve, whereas the other is indicating opened valve. From the PCS system the safety person of Fire&Gas would like to check:

   b) The ESV actually closed

   c) The time from the gas was detected to the valve was closed

   d) The closing time of the valve

3. Theoretical background

This chapter covers the research work of the Semantic Web technology and ISO15926 standard. The Semantic Web is a new technology and ISO15926 is work in progress. Many concepts and basic elements are necessary to introduce here. Therefore, the design and implementation part can be more easily to understand. The Chapter 3.1 introduces the concepts of Semantic Web, describes the ontology, the represents of information with XML, RDF, OWL, reasoning and inferring with OWL, and also the description logic supporting OWL reasoning. Chapter 3.2 introduces the ISO15926 standard, describes functions of each part of the standard, and how does each part related together.

3.1 Semantic Web

The inventor of Semantic Web is Tim Berners-Lee, As he said (cited from [28]): "*a goal of the Web was that, if the interaction between person and hypertext could be so intuitive that the machine-readable information space gave an accurate representation of the state of people's thoughts, interactions, and work patterns, then machine analysis could become a very powerful management tool, seeing patterns in our work and facilitating our working together through the typical problems which beset the management of large organizations*". From his words we can see that the Semantic Web is considered to be a new generation of the current web. Based on this technology, it could be possible for the user and machine to understand the content of the web. The procedure of understanding is executed automatically by the reasoning system. To reach this goal, firstly we should represent the state of people's thoughts, interactions and work patterns in an explicitly way. Therefore, we use the ontology based development, which is the significant characteristic of Semantic Web.

The resources in semantic web contain properties and values. The resources and relationships between them can be caught from statements. The statement is a simple sentence composed of subject, predicate and object. [3] e.g. consider about statement"CO2 Release U45-1", in which CO2 is the subject, Release is the predicate, and U45-1 is the object. Based on this structure this statement can be interpreted by the reasoning system, so that the integration of data could possibly be done in real-time.

Figure.9 is the famous seven layer cake, which is proposed by Tim Berners-Lee. It simply describes the hierarchy of components used in the Semantic Web. This master project will use the ontology vocabulary, RDF+RDF schema, and XML+XML schema. The ontology vocabulary is represented by OWL (Web Ontology Language), which provide a more extensive vocabulary than RDF schema. XML provides syntax for the structure of content in the XML document, while XML schema restricts the structure and element content in it. RDF is a language that expresses the data model. RDF Schema provides the vocabulary to define the properties and classes of RDF file.

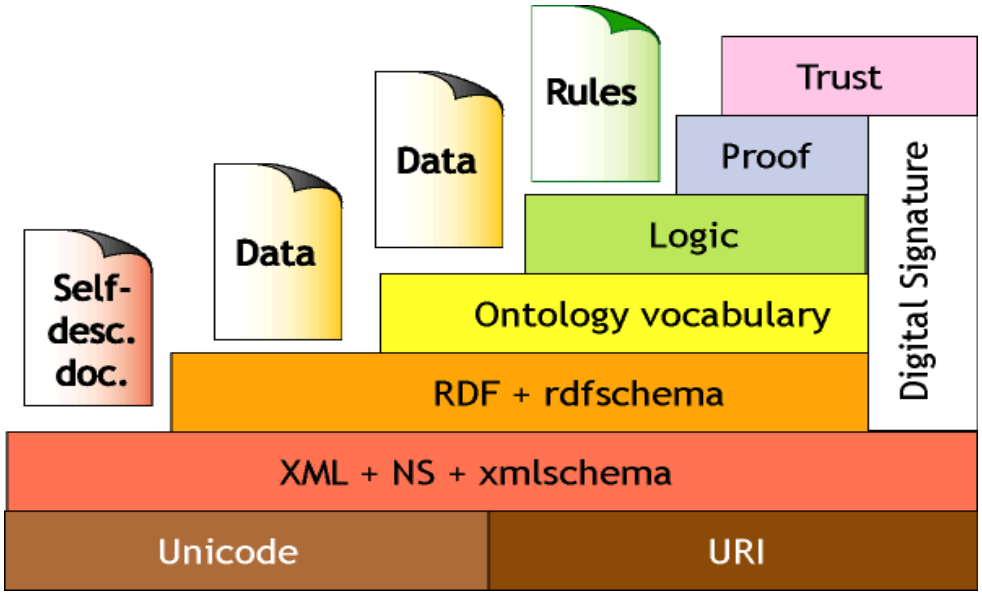


Figure 9 Seven layer cake proposed by Tim Berners-Lee

3.1.1 What is Ontology, what is it used for?

Ontology is a concept coming from philosophy. It attempts to describe the concepts of existence, relationship, taxonomy of entities. The concept is introduced to the computer science by Gruber in 1993. His famous definition is "*An ontology is an explicit and formal specification of a conceptualization of a domain of interest*" [28]. As Fensel concluded that ontology covers four aspects: explicit (unambiguous definition of relationships between entities), formal (precise mathematical description), conceptualization (abstraction aspect of entity), and share (ontology is common agreement by all the users). Correspondingly, Staab and Studer used the 4-tuple <C, R, I, A> to express the ontology, in which "C" represents set of concept, "R" represents set of relations, "I" represents the Instance, and "A" represents the axioms. For example (See figure.10) "C" can be used to represent a "*Father*" *and* "*Son*" Class. "*Jim Green"* and "*John Green*" are the instances "I" of the Class. The set "R" contains the relation "*hasSon*". The axioms can be express as *(John Green, hasSon, Jim Green).* Due to the explicit, formal, conceptualization characteristic of ontology, it has more advantages in the data integration. "*Compared with other classification schemes, such as taxonomies, thesauri, or keywords, ontologies allow more complete and more precise domain models*" [14]. There are two kinds of ontology language: Graphical notations (like Semantic network, UML, RDF) and Logic based (like Description logic, rules, and first order logic). Currently, the most commonly used language to define an ontology is the OWL (Web Ontology Language), which is a description logic. Similar description logic can be found as OIL, DAML+OIL.

3.1.2 XML + XML schemas

XML (Extensible Markup Language) is a widely used standard format for data transmission. It‘s extensible because it enable user to define mark-up element. The elements are defined by XML schemas which describe the grammar of XML languages. Compared with traditional used HTML, XML succeed in separating the data from form of expression. XML documents contain the data set in a tree structure, while XSL (Extensible Stylesheet Language) is responsible for the form of expression. XML has to be well-formed, if not it would be impossible to parse it. Moreover, the XML document is valid if all of its elements are defined in XML Schemas or DTD (DTD is an old version of XML schemas). XML Schemas defines the contents, structure and semantic of XML documents. Sharing the common XML Schemas allows for data integration. Due to the simplicity and robustness, this kind of data integration is widely used in the industry. However, the data integration in this project introduced more semantic concepts, so that it is more complex. In the following paragraph I will introduce some techniques of XML which are involved in this project.

XML namespace is an important function of XML. It is employed to prevent name conflicts of the elements by adding prefixes before the names of the elements. To define the prefix of the name, the attribute of XML xmlns (short for XML namespace) has to be set as follows: ***xmlns:prefix="URI".*** URI is short for Unified Resource Identifier that is used for identifying internet resources. In the following example the prefix is set to “part2”, the URI is set to “`http://www.15926.org/2006/02/part2#`”. OWL inherits this namespace attribute of XML. It enables one ontology to import another ontology, because the elements of an ontology can be simply reached by the namespace. In the following example is a reference class definition OWL code from the ISO15926 part 4, where it imports the ISO15926 part2 as a upper ontology. The “RDL-9697922” is defined as an instance of class “ClassOfActivity” from the ISO15926 part2.

```
xmlns:part2=http://www.15926.org/2006/02/part2#
<owl:Ontology rdf:about="">
     <owl:imports rdf:resource="http://www.15926.org/2006/02/part2"/>
</owl:Ontology>
<part2:ClassOfActivity rdf:ID="RDL-9697922">
</part2:ClassOfActivity>
```

XSD is short for (XML Schemas Definition). In the following is a sample XML schema for Cause&Effect data. There are two types of elements in XSD: Simpletype and ComplexType. A simple type element contains only text, like “CAUSEID”, “EffectID”,”NOTETYPE”. A complex type element can contain other elements or attributes, like “INTERSECTION”. The attribute definition *type =”xs:string”*define the data type of the element. The attribute definition *minOccurs=”0”* restrict the minimum number of the element to zero.

```
 <xs:schema id="CAUSEEFFECT"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
```

```
<xs:element name="INTERSECTION">
 <xs:complexType>
  <xs:sequence>
      <xs:element name="CAUSEID" type="xs:unsignedByte" minOccurs="0" />
     <xs:element name="EffectID" type="xs:unsignedByte" minOccurs="0" />
     <xs:element name="NOTETYPE" type="xs:string" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
</xs:element>
```

XSLT (XSL Transformation) is a transformation language for XML document. XSLT is part of XSL. XSL contains three parts: XSLT, XPath (XML navigation language) and XSL-FO (XML formatting language). XSLT could enable conversion from a XML tree to another tree structure. It can be used to map XML document to OWL, the implementation is done by a tool called JXML2OWL. The following code is a sample of XSLT. The basic elements of XSLT are: <xsl:template>, <xsl: for-each> and <xsl: value-of>. The element <xsl:template> is used to build template. It contains a attribute "match" that links template with XML element. The element <xsl: for-each> is used to select XML element of a specified collection of nodes. The element <xsl: value-of> can get the value of an XML element and put it to the out stream of the transformation.

```
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
    <xsl:template match="/">
      <xsl:variable name="part4Activitys0">
                  <xsl:for-each select="/CAUSEEFFECT/CAUSE/ID">
                      <ActivityId>
                          <xsl:value-of select="translate(normalize-space(.), ' ', '')"/>
                      </ActivityId>
                  </xsl:for-each>
              </xsl:variable>
     </xsl:template>
</xsl:stylesheet>
```

3.1.3 RDF+RDF schemas

RDF stands for Resource Description Framework. It is a W3C recommendation data model for representing metadata. Its data specification is based on the XML syntax and has a graph structure. It is an ontology language that can express the semantic of the data. The RDF documents can be understood by the computer. The resources of RDF are organized as statements. The statements are in the triple form <subject, predicate, object>. RDF schemas are extensions of RDF that provide user defined elements for knowledge representation. Frequently used elements of RDF Schemas are listed below:

rdfs:range of rdf:property. It defines the range of the property. (object in the triple)

rdfs:domain of rdf:property. It defines the domain of the property (subject in the triple)
rdf:type It defines an instance has a type of a class
rdfs:subClassOf It defines a class is a subclass of another class
rdfs:subPropertyOf it defines a property is a sub property of another property

Given the following example in figure.10, there are four classes “Father”, ”Child”, ”Son” and “Daughter”, in which “Son” and “Daughter” are subclasses of “Child”. And there are three properties: “hasChild”, “hasSon” and “hasDaughter”, in which “hadSon” and “hasDaughter” should be the sub property of “hasChild”. Moreover, there are two instances: “John Green” and “Jim Green”. The triples would be like (John Green, rdf:type, Father) and (Jim Green, rdf:type, Son). If the domain of property “hasSon” is set to class “Father”, and range is set to class “Son”, Then we can get the triple (John Green, hasSon, Jim Green).

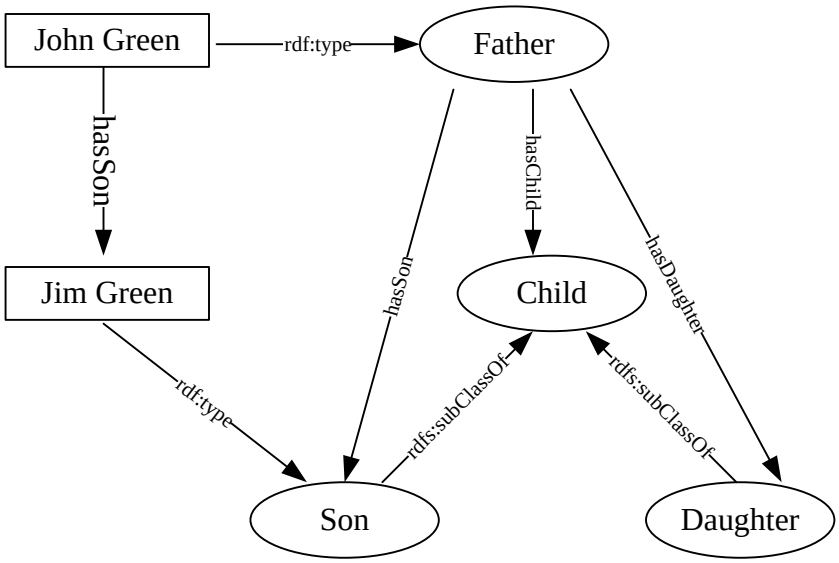


Figure 10 RDF graph example

The semantics of RDF is given by RDF Model Theory (MT). MT defines the relationships between syntax and interpretations. The interpretation is the methodology of how the machine can understand the meaning of the elements of RDF schemas, like rdf: type and rdfs:subClassOf; The property rdfs:subClassOf and rdfs:subPropertyOf should be transitive. That means:

(1) **if** *(A, rdfs:subClassOf, B)* **and** *(B, rdfs:subClassOf, C)*
then *(A, rdfs:subClassOf, C)*
(2) **if** *(A, rdf:type B)* **and** *(B, rdfs:subClassOf, C)*
then *(A, rdfs:type, C)*

3.1.3.1 Distinguish is-a and part-of relations

In RDF syntax, the is-a relation is denoted as rdf:subClassOf, and the part-of relation is denoted as

rdf:type. There two relations are easy to confuse. Is-a relationship defines a class that is an extension of another class. For example, ‘male’ is-a ‘human’, and ‘human’ is-a ‘animal’. In contrast, the part-of relationship denotes the part and whole relations between the things. For example, 'bark', 'trunk' and 'limb' are part-of ‘tree’. In the figure.11 it shows the linguistic representations of the is-a and part-of relationships.

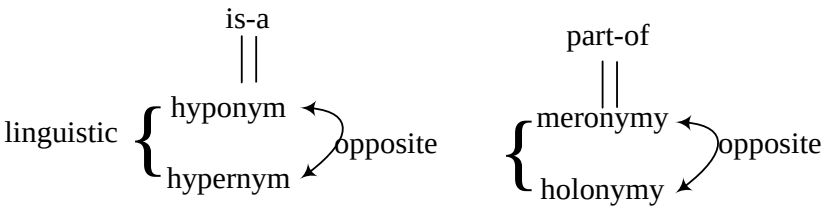


Figure 11 linguistic representation

3.1.3.2 SPARQL

Sparql is a RDF query language. It is a W3C standard that is being considered as a component of semantic Web. It can be used to execute query over large amount of data source, which is stored in a RDF format. *“SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions.”[25]* There is a Java plug-in developed under Jena API called Jena ARQ, which supports SPARQL.

The following code is a query of SPARQL. As you can see, the SPARQL support the URI definition as PREFIX, like ont: <http://protege.com/Ontology#>. The variables are presented by a symbol “?” or a prefix. The query will search for the set of results that match the triple pattern.

```
PREFIX ont: <http://protege.com/Ontology#>

SELECT ?Friend
WHERE {
   ?x ont:hasFriend ?Friend ;
}
```

3.1.4 OWL (Web Ontology Language)

OWL is an ontology representation language maintained by Word Wide Web Constitution. It provides more vocabularies than RDF Schemas to give semantic representation of the information. We will list some vocabularies that are used in this project in the following paragraph. OWL is used in the situation where the information should be processed and understood by machines. Actually, OWL is a logic based language. Its semantic is giving by the logic definition, such as description logic. And the description logic could be translated to first order logic. The logic based

semantics makes it support reasoning by the computer. There are some open source semantic reasoners like DIG, Pellet, Racer, and Fact++. In this project Pellet [17] and Racer [34] are used as reasoner.

OWL has three sublanguages that are: OWL-Lite, OWL-full, and OWL-DL. For building an OWL ontology, it is necessary to choose a proper sublanguage. According to [12], the OWL-Lite is the simplest sublanguage. It is suitable to use in the situation where simple classes hierarchy and simple constrains are needed. Obviously, it is useful for building conceptual simple hierarchy like thesauri ontology. Furthermore, the OWL-DL is based on the Description Language. The Description Language is a decidable language, so that automatic reasoning is possible. Therefore, the ontology based on OWL-DL can use semantic reasoner tools to check consistency and build class hierarchy automatically. At last, the OWL-full is the most expressive language. It has different semantic with OWL-Lite and OWL-DL. And it's compatible with RDF Schemas. It's used on the situation that OWL-Lite and OWL-DL is not sufficient to represent the semantics. The disadvantage is that it is not possible for the computer to automatically reason over all the semantics of OWL-Full until now. In this project we would like to choose the OWL-Full as ontology description language because of the complex structure of the ISO15926 standard. The ISO15926 standard requires a highly expressive language and need support for the concept of meta-classes.

Considering the syntax of OWL, there are two kinds of syntax representation: Abstract syntax and RDF/XML syntax. RDF/XML syntax is based on both RDF and XML syntax, so that it can be an extension of RDF and accessed by RDF applications. The abstract syntax is a specific language that is easier to read and write. It is more similar and related to the description logic language. In this thesis project, the ontology definitions based on ISO15926 are using the RDF/XML syntax to support the RDF applications like Sparql query. However, to express the concept more clearly, we will borrow the abstract syntax of OWL to illustrate the concept in the report writing. The example of both syntaxes can be found at following webpage.
(Abstract syntax): http://www.cs.man.ac.uk/~horrocks/ISWC2003/Tutorial/people+pets.abs
(RDF/XML syntax): http://www.cs.man.ac.uk/~horrocks/ISWC2003/Tutorial/people+pets.owl.rdf

3.1.4.1 Syntax of OWL-Full

OWL-Lite, OWL-DL, and OWL-Full has different syntax. As we will use OWL-Full in this project, we will introduce and explain some Syntax of OWL-full that used in this project in this chapter. As the W3C specification in [13] the Syntax can be basically classified into 5 types: Syntax of Classes, Properties, Individuals, Datatypes, and Annotations. The following chapters will introduce each of them separately.

(1) Syntax of Classes

As shown in Figure.12, Syntax of Classes can be divided into class description and class axioms. Class description syntax is used to build an OWL-class, and specify the structure and axioms of

the OWL-class. The class axioms list the axioms that can be defined in the OWL-class. There are three kinds of class descriptions: Enumerations, Property Restriction, and AND, OR, NOT logic description. The property restriction means: given a defined class A, it contains two aspects: the class A has a property B and the property B has some value constraints and cardinality constraints. The AND, OR, NOT is a mathematic equivalence of the OWL-syntax. For example, if a class A is defined as (class B, owl:intersectionOf , class C), that means class A is a collection of individuals that both is a type of class B and class C. Likewise, if a class A is defined as (class B, owl: uinionOf , class C), that means class A is a collection of individuals that either a type of class B or class C.

Basic Class Axioms are rdf:subClassOf, owl:equivelentClass and owl:disjointWith. The rdf:subclassOf is extended from RDF syntax described above. Class A owl:equivelentClass to Class B, if all members of A belongs to B and vice versa. Likewise, Class A is owl:disjointWith Class B, if class A and class B has no common member. For more details and examples of the Syntax described above please look at [13].

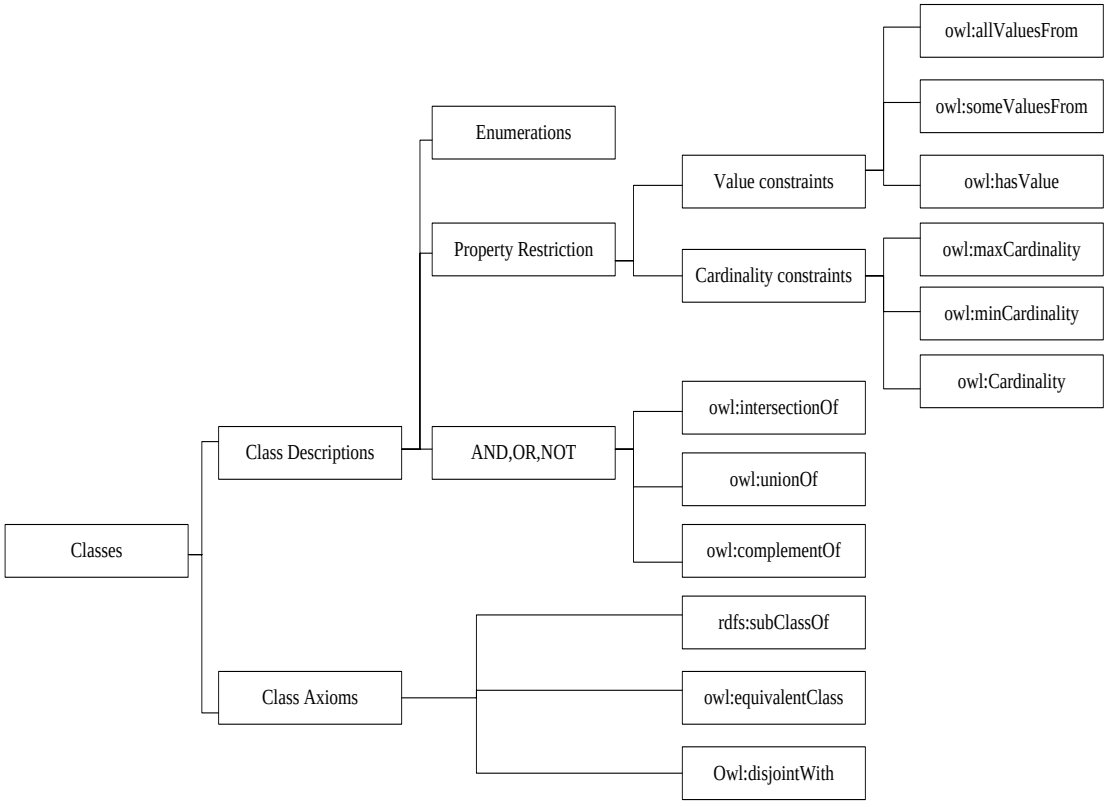
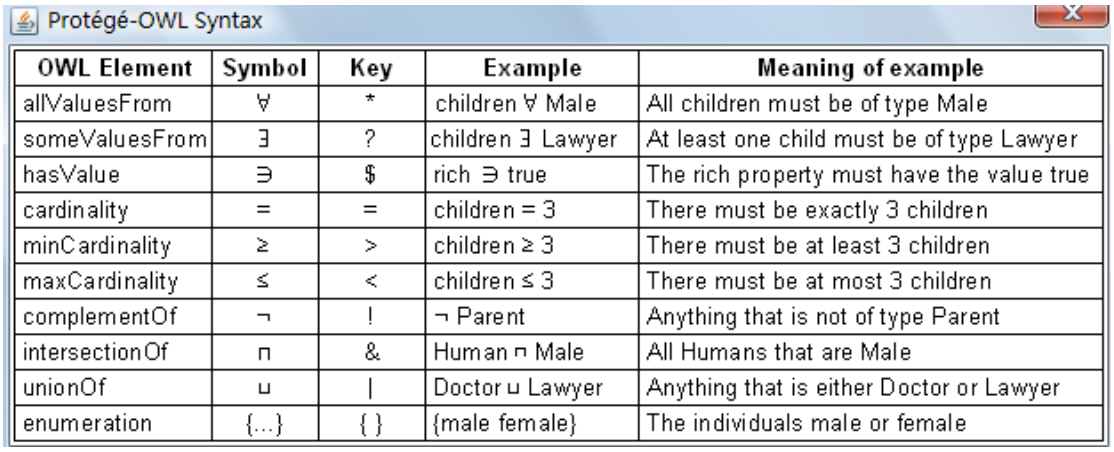


Figure 12 Classification of Syntax of classes

The Figure.13 is a snapshot cut from the software Protégé. It lists the most frequently used syntaxes of OWL. And it gives the mathematic symbol that has the same meaning of the syntax. Moreover, it gives some examples that make it easy to understand.



OWL Element	Symbol	Key	Example	Meaning of example
allValuesFrom	∀	*	children ∀ Male	All children must be of type Male
someValuesFrom	∃	?	children ∃ Lawyer	At least one child must be of type Lawyer
hasValue	∋	$	rich ∋ true	The rich property must have the value true
cardinality	=	=	children = 3	There must be exactly 3 children
minCardinality	≥	>	children ≥ 3	There must be at least 3 children
maxCardinality	≤	<	children ≤ 3	There must be at most 3 children
complementOf	¬	!	¬ Parent	Anything that is not of type Parent
intersectionOf	⊓	&	Human ⊓ Male	All Humans that are Male
unionOf	⊔	|	Doctor ⊔ Lawyer	Anything that is either Doctor or Lawyer
enumeration	{...}	{ }	{male female}	The individuals male or female

Figure 13 Protégé-OWL syntax

(2) Syntaxes of Properties

Syntaxes of properties are listed in table.3. Some of the syntaxes are inherited from RDF, such as rdfs:subPropertyOf , rdfs:domain, and rdfs:range, which will not be described here.

Classification	Syntax	Description
Relations to other properties	owl:equivalentProperty	Properties that has the same "values" but may denote different concept
	owl: inverseOf	If the axioms (*P1 owl:inverseOf P2),* than for every axiom *(x P1 y)* that exist *(y p2 x) and* vice versa
Global cardinality restrictions on properties	owl:FunctionalProperty	If P is a functional property, than for each instance x it has unique value y in axiom *(x P y)*
	owl:InverseFunctionalProperty	If P is a functional property, than for each instance y it has unique value x in axiom *(x P y)*
Logical characteristics of properties	owl:TransitiveProperty	If P is a transitive property, given axioms *(x P y)* and *(y P z)*, than it has *(x P z)*
	owl:SymmetricProperty	If P is a symmetic property, given axioms *(x P y)* , than it also has *(y P x)*
Property classification	owl:objectProperty	properties that relate instance to instance, like hasSon, may have the axiom (Father hasSon Son)
	owl:datatypeProperty	Properties that relate instance to data type, like hasTime, relate a instance to data type dateTime.

Table 3 Syntax of properties

(3) Other syntax of OWL

Considering the syntax of Individuals, there are two aspects: individual definition, and identifier. Individual can be defined as either named individual or anonymous individual. And there are three ways to define identifier individual, that are owl:sameAs, owl:differentFrom and owl:AllDifferent. See [13]. Syntax of owl Datatypes is inherited from RDF and XML. Reasoning with data type is

supported in OWL.

Syntaxes of Annotations are listed as follow: owl:versionInfo , rdfs:label, rdfs:comment, rdfs:seeAlso and rdfs:isDefinedBy. Semantic annotation is a important concept in the semantic web. It enables the semantic search that is different from traditional search engines. The full-featured text search engine software Lucene is often used to implement the semantic search. And in the Jena API there is a plug-in that supports Lucene.

owl:import syntax enable the ontology to import another ontology as references. In the ISO15926 standard the ISO15926-part4 has to import ISO15926-part2. And ISO15926-part7 has to import both ISO15926-part2 and ISO15926-part4. And so on the user defined ontology based on ISO15926 has to import all above.

3.1.4.2 Reasoning with OWL

Reasoning is a concept coming from philosophy. As it is defined in [15] "*Reasoning is the cognitive process of looking for reasons for beliefs, conclusions, actions or feelings*" Through the reasoning process, people can distinguish and recognize things. In philosophy the reasoning is divided into deductive reasoning and inductive reasoning. Formal logic is typical deduction reasoning that by given premises the conclusion can be drawn through logic inference. For example:

Premise 1: All the lions are animals
Premise 2: Simba is a lion
Conclusion: Simba is an animal

Relatively, in the RDFS this relation can be defined as follow:

Axioms 1: (lions, rdfs: subClassOf, animals)
Axioms 2: (Simba, rdf: type, lions)
Implicit Axioms: (Simba , rdf:type, animals)

This implicit semantic of RDFS is defined in the Model Theory, which can be interpreted automatically by the machine. Compare to deductive reasoning the inductive reasoning is inferring of the things will happen in the future based on the current situations, inductive statistics, or common sense. It cannot guarantee that the conclusion is 100% true if the premises are true. The mathematical induction is the typical inductive reasoning.

In the Semantic Web technology, all the reasoning could be consider as deductive reasoning. It is defined as: reasoning is the evaluation of ontologies according to their specification. As it noted in [16] *"Reasoning is the essential background technology for knowledge representation."* The reasoning has three advantages for Semantic Web: (1) ontology management, (2) inferencing, (3) query answering. Reasoning for inconsistence is a kind of ontology management, and the consistency is the important requirement for the ontology. So we will discuss it in the following paragraph. The inferencing is also the critical concept of the ontology building. We will present some examples to illustrate the inferencing with OWL. The query answering means the when the users input queries in the user interface, the applications will analyze the query based on reasoning of ontology and return the inferred answer as result to the user.

3.1.4.2.1 Reasoning with inconsistency

As it mentioned in [29], there are 3 important characteristics of the Semantic Web: scalability, distribution, and joint author-ship. All of these may lead to inconsistencies. Sacrificing the expressivity of the languages could avoid some inconsistency. However, for most applications the current languages like OWL are too expressive to implement. Actually, there are two ways to deal with inconsistency. One is find out the inconsistency and fix it. The reasoner can be used to check inconsistency, such as Pellet, Fact++, RacerPro and Jena support incomplete consistency checking for OWL-DL. Another is to tolerate the inconsistency by applying a nonstandard reasoning method to acquire meaningful answers. The inconsistency we consider about here is the logical theory inconsistency. A logic theory is inconsistent if it contains a contradiction: Both A and $\neg A$ are true in the theory. As [29] concludes there are many causes of inconsistency, we will discuss them will examples as follow:

(1) inconsistency by mis-representation of default

The knowledge engineer may define the fish as follow. It sounds reasonable if there are not any special animal does not fit with the definition.

$fish \sqsubseteq animal \cap liveInWater$ (Fish is an animal that live in water)

$animal \cap liveInWater \sqsubseteq fish$ (Animal that live in water is fish)

For example, when you want to extend the ontology as follow:

$whale \sqsubseteq animal \cap liveInWater$ (Whale is an animal and whale lives in water)

$whale \sqsubseteq \neg fish$ (Whale is not a fish)

Then the ontology will be in consistent. Because from the $animal \cap liveInWater \sqsubseteq fish$ and $whale \sqsubseteq animal \cap liveInWater$, the implicit assert can get that “whale is a fish”. However, it has defined that “whale is not a fish”, so that we have the contradiction. Therefore, in this case we cannot generally define that $animal \cap liveInWater \sqsubseteq fish$.

(2) inconsistency by polysemy

Ploysemy represents the words that have multiple meaning. In the paper [29], it gives an example of a “marriedWoman” which both mean a woman who has a husband and a woman had a husband but has divorced now.

$MarriedWoman \sqsubseteq Woman$ (A married woman is a woman)

$MarriedWoman \sqsubseteq \neg Divorcee$ (A married woman is not a divorcee)

$Divorcee \sqsubseteq HadHusband \cap \neg HasHusband$ (A divorcee had a husband and has no husband)

$HasHusband \sqsubseteq MarriedWoman$ (HasHusband means married)

$HadHusband \sqsubseteq MarriedWoman$ (HadHusband means married)

Figure.14 use a Venn chart to represent the ontology defined above. It would be easy to notice that the ontology is inconsistency with the implicit assertion: $Divorcee \sqsubseteq \neg Divorcee$ That is obviously a contradiction.

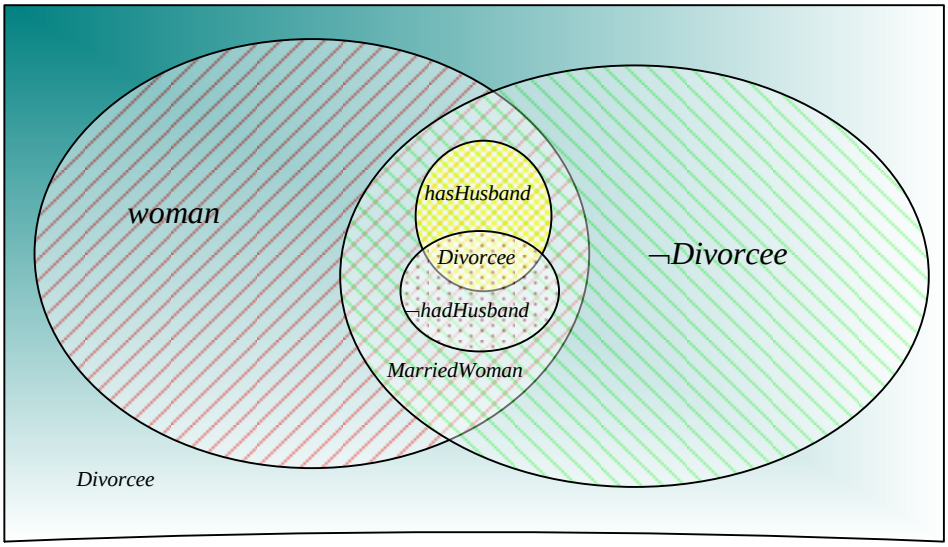


Figure 14 Venn chart of the ontology

(3) Inconsistency through migration from another formalism

Inconsistency may occur when an ontology is migrated from other data sources [29]. This should be taken into special consideration when the data integration is conducted. The translation of the ontology should be strictly analyzed. Take the famous paradox of the court for example, the ancient Greece philosopher Protagoras who is learned in law, has a pupil, Euathlus. They made a deal: if and only if Euathlus wins his first court case, the Euathlus would pay the tuitions to Protagoras. However, the Euathlus seems do not want to receive any case at all, so that he won't have to pay the tuitions. Finally, Protagoras decides to take Euathlus to the court.

Euathlus argued that if he won the case, according to the law, he doesn't need to pay the tuition. And if he lost the case, according to the deal, he doesn't need to pay the tuition, because he didn't win a case.

However, Protagoras argued if he lost the case, according to the deal, he should get the tuition back, because Euathlus had won his first case. Else if he won the case according to the law, he should also get the tuition back.

Can you judge who should win this case? If we consider the deal as an ontology, and the law as another ontology, obviously, there is a contradiction between these two ontologies. Let us use the if-then logic to represent both of them.

(1) The deal:

if *Euathlus win a case,* ***then*** *Euathlus should pay the tuition*

if *Euathlus lose a case,* ***then*** *Euathlus don't need to pay the tuition*

(2) The law:

if *Euathlus win a case,* ***then*** *Euathlus don't need to pay the tuition*

if *Euathlus lose a case,* ***then*** *Euathlus should pay the tuition*

As we know for the sentence: if a then b, "a" is the hypothesis and "b" is the conclusion. More specifically "a" is the sufficient condition for "b", and "b" is the necessary condition for "a". The ontology will be as follows:

$$\text{EuathlusWinACase} \sqsubseteq \text{EuathlusPayTuition} \quad (1)$$

$$\neg\text{EuathlusWinACase} \sqsubseteq \neg\text{EuathlusPayTuition} \quad (2)$$

$$\text{EuathlusWinACase} \sqsubseteq \neg\text{EuathlusPayTuition} \quad (3)$$

$$\neg\text{EuathlusWinACase} \sqsubseteq \text{EuathlusPayTuition} \quad (4)$$

The formula (1) and (2) are defined according to the deal between Protagoras and Euathlus, and formula (3) and (4) are defined according to the law. In the argument of Protagoras, he used the formula (1) and (4), so that no matter Euathlus win or lose the case, he will pay the tuition fee. To the contract, in the argument of Euathlus, he used the formula (2) and (3). However, as a knowledge engineer's point of view, this is obviously an inconsistency of the ontology caused by migration from the formalism of the law to the formalism of the deal. If you look at formula (1) and (3), the concept EuathlusWinACase is unsatisfiable, because (1) and (3) can be combined as:

$$\text{EuathlusWinACase} \sqsubseteq \text{EuathlusPayTuition} \cap \neg\text{EuathlusPayTuition} = \varnothing$$

Likewise, the concept for formula (2) and (4) $\neg\text{EuathlusWinACase}$ is unsatisfiable as well. The problem is which formalism you want to follow as standard to make the decision. In this case a better reasoning tool is required to make more advanced choices. In [29] it proposed a methodology for reasoning with inconsistent ontologies.

(4) inconsistency caused by multiply sources

"*When a large ontology specification is generated from multiple sources, in particular when these sources are created by several authors, inconsistencies easily occur" [29]* The ISO15926 ontology is a huge ontology specification with multiple sources in multiple domain. Therefore, the goal of consistency is a big challenge for data integration based on ISO15926. Although, ISO15926 has a centralized global upper ontology, the lower level ontology still has to cope with the heterogeneous data sources. A lot of works need to be done to build standardized templates for each specific domain.

3.1.4.2.2 Inference with OWL

As it noted in [17] the inference with OWL are divided into two kinds:

Classification: Inference with classes to create the complete class hierarchy based on the subclasses relations assertions. The queries of getting subclass or super class can be answered through the class hierarchy.

Realization: Inference with the instances to find out the direct and indirect classed that the instances belongs to. The realization step should be done after the classification, because the types of instances are defined according to the class hierarchy.

The following code is writing in OWL abstract syntax which is used to illustrate classification of classes. For the meaning of intersectionOf and someValuesFrom please check the OWL syntax in figure.5. It's worth to mention that the axiom (a, partial, b) means "a" is sufficient condition of "b". And the axiom (a, complete, b) means "a" is sufficient and necessary condition of "b" and vice versa. From the classes definition we can get the following information:

A Java programmer is a person who programs with Java.

A Programmer is a person who programs with programming language.

Java is a kind of programming language.

From the assertions above the computer can infer that Java programmer must be a Programmer.

```
Class(a:Java+Programmer complete intersectionOf(a:person
   restriction(a:programsWith someValuesFrom (a:Java))))
Class(a:Programmer complete intersectionOf(a:person
   restriction(a: programsWith someValuesFrom (a: ProgrammingLanguage))))
Class(a:Java partial a:ProgrammingLanguage)
```

The following code is writing in OWL abstract syntax which is used to illustrate the realization of individuals. From the classes and individual definition we can get the following information:

F-22 is a kind of thing.

Tom is a man who drinks cappuccino and drives F-22.

Cappuccino is a kind of Coffee

coffeeLoverMan are the man who drinks Coffee, the man who drinks coffee are coffeeLoverMan

coffeeLoverMan drives car.

From the assertions above, the computer can infer that, Tom drinks coffee, so he is a coffeeLoverMan. And coffeeLoverMan drives car, so Tom can only drives car. So that F-22 here is a car not a plane.

```
Individual(a:F-22 type(owl:Thing))

 Individual(a:Tom type(a:male)
  value(a:drinks a: Cappuccino)
  value(a:drives a: F-22))

 Individual(a: Cappuccino type(a:Coffee))

 Class(a: coffeeLoverMan complete
  intersectionOf(a:man restriction(a:drinks someValuesFrom(a:Coffee))))
 Class(a: coffeeLoverMan partial restriction(a:drives allValuesFrom (a:Car)))
```

3.1.5 Description Logic

As it defines in [18], the description logic is knowledge representation languages. "*It is used for logical reconstruction of representation tool like frames, Object-Oriented and semantic data models, semantic networks, type systems, and feature logics."* High expressivity and decidability enable the description logic to describe most of the concept in the world explicitly. There are many kinds of description logics which are represented by different naming conventions. Description logics provide well defined semantics for OWL, OWL.2 supports $\mathcal{SROIQ}^{(\mathcal{D})}$ of the description logic. Separately, OWL-DL is based on, and for OWL-Lite it is $\mathcal{SHIF}^{(\mathcal{D})}$[20]. Description logic is the basis of Semantic Web. It facilitates ontology engineering, reasoning with ontology, and service description and discovery.

As it shows in figure.15 from [17] the structure of Description logic can be divided into Tbox and Abox. Tbox defines the schema of the ontology. There are the axioms of the classes, they could be description of concepts or statement of constrains, such as subclass, equivalent class, intersection or union of axioms in OWL. Abox defines the data of the schema, which means the assertion of the individuals. Such as *differentFrom*, *sameIndividualAs*, *oneOf* syntax in OWL. Knowledge base contains both Tbox and Abox to construct a complete OWL ontology.

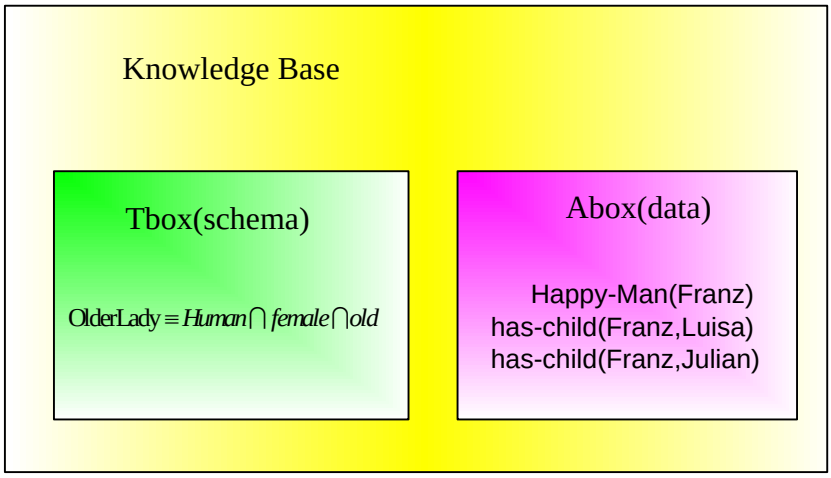


Figure 15 DL Architecture

As introduced in [19] there are some basic constructors of DL:

$$C \bigcap D, \neg C, \forall r.C, \exists r.C, (\geq n\ r), (\leq n\ r)$$

In the constructors the "C" and "D" represent classes, "r" represents property, and "n" represents cardinality. The OWL class constructors and relative DL syntax can be found in Figure.16. DL syntax also can support OWL axioms as Figure.16 cited from [20]. Relations like subclass, disjoint class and equivalent class of classes; classify the same and different individuals; sub property, equivalent, inverse, functional, transitive, reverse functional properties all can be expressed by DL syntax. As it noted in [20], the following equivalence is obviously true:

$C \equiv D$ *iff* $\text{both } C \sqsubseteq D \text{ and } D \sqsubseteq C$

$C \equiv D \Leftrightarrow \forall x.C(x) \leftrightarrow D(x)$ (The statement "C and D are equivalent class" equals to the statement "for any instance x type of C, x is also type of D and vise verse")

$C \sqsubseteq D \Leftrightarrow \forall x.C(x) \rightarrow D(x)$ (The statement "C is the subclass of D" equals to the statement "for any instance x type of C, x is also type of D")

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	$\text{Human} \sqsubseteq \text{Animal} \sqcap \text{Biped}$
equivalentClass	$C_1 \equiv C_2$	$\text{Man} \equiv \text{Human} \sqcap \text{Male}$
disjointWith	$C_1 \sqsubseteq \neg C_2$	$\text{Male} \sqsubseteq \neg\text{Female}$
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	$\{\text{President\_Bush}\} \equiv \{\text{G\_W\_Bush}\}$
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	$\{\text{john}\} \sqsubseteq \neg\{\text{peter}\}$
subPropertyOf	$P_1 \sqsubseteq P_2$	$\text{hasDaughter} \sqsubseteq \text{hasChild}$
equivalentProperty	$P_1 \equiv P_2$	$\text{cost} \equiv \text{price}$
inverseOf	$P_1 \equiv P_2^-$	$\text{hasChild} \equiv \text{hasParent}^-$
transitiveProperty	$P^+ \sqsubseteq P$	$\text{ancestor}^+ \sqsubseteq \text{ancestor}$
functionalProperty	$\top \sqsubseteq \leqslant 1P$	$\top \sqsubseteq \leqslant 1\text{hasMother}$
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1P^-$	$\top \sqsubseteq \leqslant 1\text{hasSSN}^-$

Figure 16 OWL as DL: Axioms. From [20]

Mapping OWL to equivalent DL enables the reasoning of OWL. The semantic of DL defines by the interpretations: $I = (\Delta^I, \bullet^I)$, where Δ^I is the domain, and $\bullet^I$ is the interpretation function that maps concept (class) A, role (property) R and individual i [20].

3.2 ISO15926 Standard

ISO15926 is an international standard with the title: "Industrial automation systems and integration—Integration of life-cycle data for process plants including Oil&Gas production facilities". It's used for data integration, sharing, exchange, and hand-over between computer systems [22]. The goal of ISO15926 is to enable data integration for process plants, in order to reduce the redundant and inconsistent information in sharing data between different companies or organizations. ISO15926 is extremely complete and robust, which differentiates it from other standards. ISO15926 is introduced because of the requirement for a common terminology for a huge number of heterogeneous data sources. Also this standard has to be as stable as possible, at least no substantial changes for decades. The scope of ISO15926 nearly covers the whole process plant industry, including Oil&Gas industry. It contains 7 parts; each of them is published separately.

Part 1 is an introduction document to the ISO15926 which gives an overview and describes the fundamental principles of the standard. Part 2 defines a generic, conceptual data model for representation of life-cycle of a process plant. Part 3 defines ontology for geometry and topology

based on concept of ISO 10303-42 and ISO 10303-104. Part 4 specify reference data that represents information in a certain domain. The reference data library is commonly used for all the users. Part 5 specifies the procedures to be followed by a registration authority for reference data. [22] Part 6 is the methodology for the development and validation of reference data [24], it defines the abstract syntax of the reference data. Part 7 Implementation methods for the integration of distributed systems [24]. Part 1, 2, 4 has published as ISO standard, while Part 6, 7 is still under development. In the thesis we mostly used Part 2, 4, 7, so we will give more details of them in the following chapter.

3.2.1 ISO15926 Part 2: Data model

ISO15926 Part 2 is an upper ontology, *"which define top-level concepts such as physical objects, activities, mere logical and topological relations from which more specific classes and relations can be defined" [26]* Similar upper ontology can be found as SUMO, Sowa upper ontology, Dolce, CliP. Upper ontology can be considered as similar to the meta-model concept in UML. The data model is generic, which means "atoms" can create many statements by different combinations. To construct a data integration framework based on ISO15926, the engineer must start with part2. At least some basic principles and methodology should be mentioned as follow.

Data model of Part 2 uses the 4 dimensionalism paradigm, which are 3 spaces and 1 time. Compare with 3 dimensionalism paradigm, the 4D ontology consider all the individuals as spatio-temporal extend. In such a way every individual has both spatial part and temporal part. Only on condition of both of them are equal, an individual is equal to another one. Due to the 4D characteristic, the ISO15926 can be used to represent life-cycle data for process plants.

There are four basic elements of Data model: Thing, Possible Individual, Class, and Relationship. Respectively, thing represents anything either real or abstract, Possible Individual denotes individuals with spatio-temporal extend, Class is a collection of things, and Relationship describes the relation between things. All the other elements in Part 2 are extends from these basic elements. Model diagrams in Part2 are using the EXPRESS G diagram as template. The Model diagrams contain attributes: EXPRESS entity types; EXPRESS sub typing; EXPRESS relationship; and EXPRESS G symbols. The following figure.17 is an example of model diagram in part 2; it defines the *cause_of_even*t Relationship which we will use in this project. There are several elements that have been defined as follows:

Relationship: *cause_of_event, temporal_bounding, participation, recognition, involvement_by_reference, beginning, ending*
PossibleIndividual: *activity, event, point_in_tim*e
ObjectProperty: part, caused, causer, whole, involved, involver, recognizing, recognized
The *point_in_time* class is the subclass of *event*, while the class *beginning* and *ending* are the subclass of *temporal_bounding*. The ObjectProperty *caused* has domain *cause_of_event* and range *event*, while the ObjectProperty *causer* has domain *cause_of_event* and range *activity*.

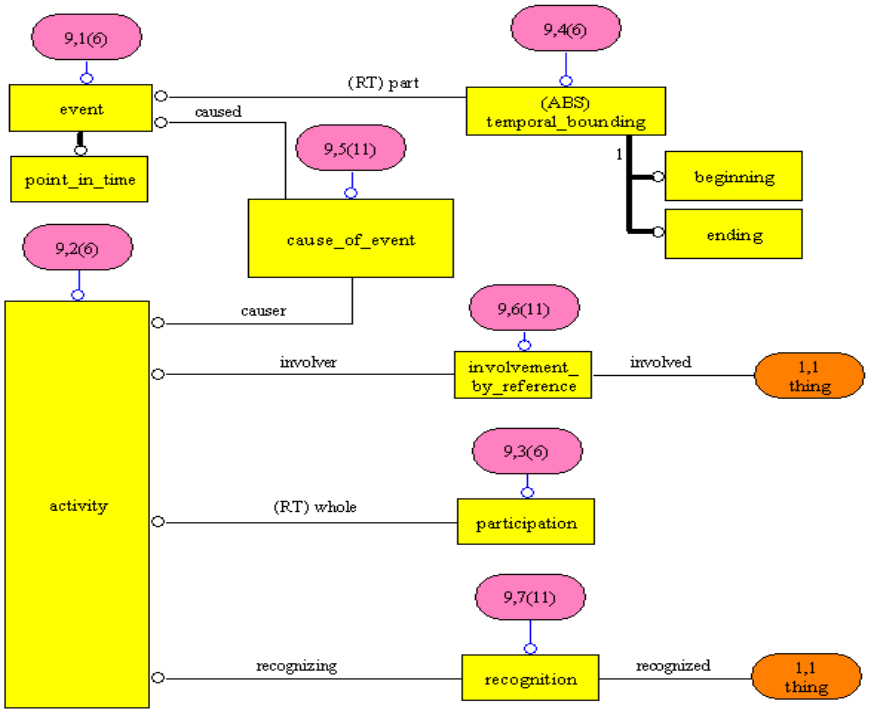


Figure 17 Model diagrams in Part 2(from [23])

3.2.2 ISO15926 Part 4: Reference Data

If you consider Part 2 as “grammar” for a sentence, the Part 4 will be the “words” [27]. Such as “firing”, “flaming” as you can see from Figure.18 as follow. Part 4 is primarily divided into some basic spreadsheets, such as activity.xls, piping.xls, property.xls, static_equipment.xls etc. The reference data is defining the terminology as a general standard used by every company and organization in a specified domain in order to provide data consistency. The reference data is arranged in a hierarchy like it shows in Figure.18. It employs the taxonomy mechanism for the “Activity”. “Activity” is a genetic meta-class from Part 2, for which the “Reacting” is the subclass of “Activity”. Likewise, the “burning” and “responding” are subclasses of “Reacting”. Based on the taxonomy defined in Part 4, it could be possible to specify the members of the terminology. For example, the “firing in U51-2” is a kind of “firing” (subclass). The members are specified in Part 7.

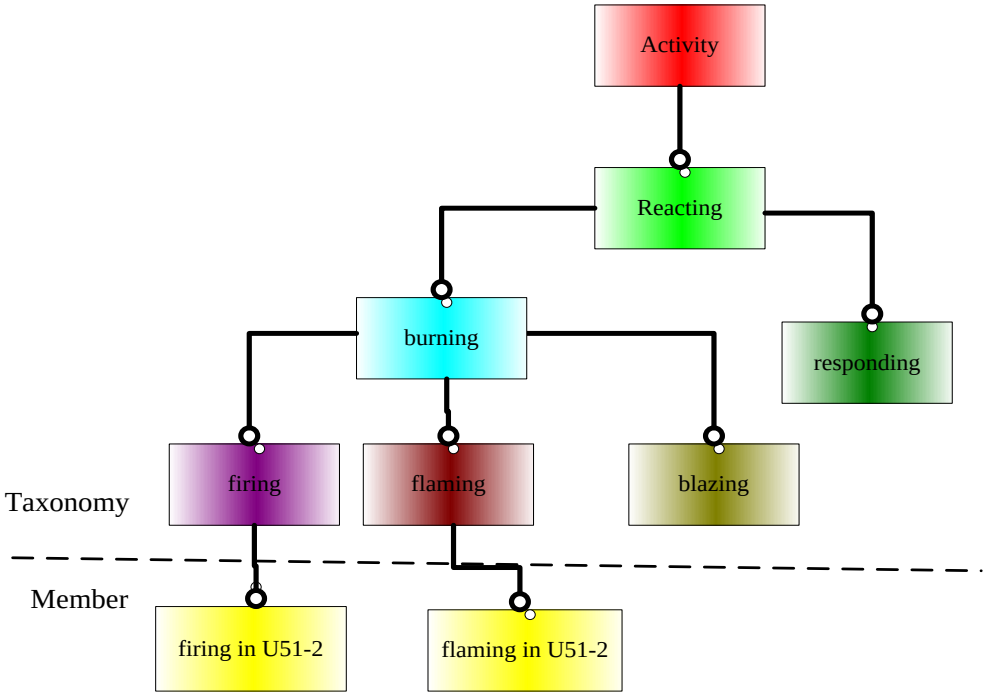


Figure 18 Reference data hierarchy

The question may arise how the Part 2 and Part 4 are related to each other, what is the modeling methodology to achieve the completeness and robustness of the ISO15926 standard? As you can see from Figure.19, it illustrates the composition of Part 2 and Part 4. In figure.19 the object with red color comes from Part 2, object with green color come from the Part 4, and the black arrow defines the relation between them. As it is described above in Part 2 there are some basic elements such as *Class*, *Relationship*, and *PossibleIndividua*l. Therefore, in Part 4 it has *Class* and *Relationship*, the *Class* of Part 4 (in green rectangle) is extended from super class in Part 2 *Class* or *PossibleIndividual*. And there are only two kinds of relationship in Part 4 that is *Classification* (green arrow) and *Specialization* (green line with round end). The Classification means something is part-of something, while Specialization means something is kind-of something.

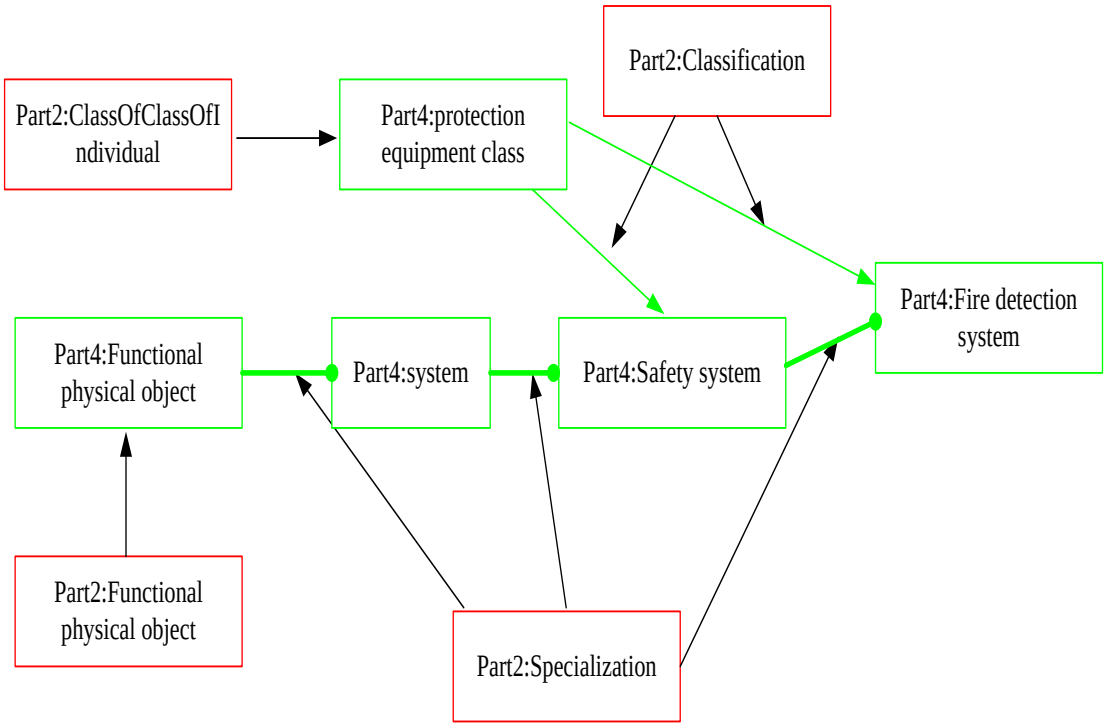


Figure 19 Part 2 related with Part 4

Figure	Type	Description
	Instance of *Part2: Specialization*	kind of a Relationship (part 2) Individual, with the objectProperty: *SuperClass* and *SubClass*
	RDFS property *rdfs:subclassOf*	objectProperty with the domain of all class from part 2 and range of all class from part 4
	Instance of *Part2: Classification*	kind of a Relationship (part 2) Individual, with the objectProperty: "*classified"* and "*classifier"*

Table 4 detail description for figure.19

To understand the *Classification* and *Specialization* relationship, the dividing levels of ISO15926 part 2 need to be introduced [24]. The basic element "Class" is the collection of things. There are some subclasses of Class, such as "*ClassOfIndividual*", "*ClassOfClass*", and "ClassOfClassOfIndividual". "*ClassOfIndividual*" is a type of "*Class*" whose members are instance of "*PossibleIndividual*". Likewise, "ClassOfClassOfIndividual" is a type "*ClassOfClass*" whose members are instances of "*ClassOfIndividual*". As you see from chart.3 below, there are three levels have been defined, so that we can get the OWL restrictions in OWL abstract language in the following code. It restricts the range of the property of each class.

Level	ClassName	Members
Level_0	*PossibleIndividual*	all individuals
Level_1	*ClassOfIndividual*	all set of individuals
Level_2	*ClassOfClassOfIndividual*	all set of set of individuals

Table 5 dividing of Levels

```
Class (part2:Classification partial
      unionOf(
        intersectionOf ( Restriction (part2:hasClassified allValuesFrom( Level_0))
                         Restriction (part2:hasClassifier allValuesFrom (Level_1)))
        intersectionOf( Restriction (part2:hasClassified allValuesFrom(Level_1))
                        Restriction(part2:hasClassifierallValuesFrom( Level_2)))))
Class( part2:Specialization partial
    unionOf(
        intersectionOf( Restriction (part2:hasSubclass    allValuesFrom (Level_0))
                         Restriction (part2:hasSuperclass allValuesFrom( Level_0)))
        intersectionOf(Restriction (part2:hasSubclass    allValuesFrom (Level_1))
                        Restriction (part2:hasSuperclass allValuesFrom( Level_1)))
        intersectionOf(Restriction (part2:hasSubclass    allValuesFrom (Level_2))
                        Restriction (part2:hasSuperclass allValuesFrom( Level_2)))))
```

3.2.3 ISO15926 Part 7: implementation methodology

Given ISO15926 Part 2 and Part 4, we still do not know how to do the data integration. Therefore, in part 7 it specifies the implementation methodology. Including the following questions: How does each part collaborate as one? How is it related to RDF and OWL? How does the data integration work in a distributed system?

3.2.3.1 ISO15926 implementation hierarchy

The Figure.20 shows the ISO15926 stack. There are many versions of this kind of stack, for example in [21] the hierarchy of process industry ontologies is a pyramid. On top of the pyramid is the ISO 15926 Part 2 that has 200 classes and properties. The layer under the ISO15926 Part 2 is the ISO15926 Part 4 that has thousands of classes. The number of classes and properties increases as it is going down the pyramid. If we come back to this Figure.20, it gives a clearer overview of the hierarchy of ISO15926 standard. On the top layer is the main technology of the Semantic Web. All the classes, properties and restrictions (ontology) in ISO15926 are written in OWL language, and all the data store (instances) are in the RDF triple. From layer 4 to layer 7 are defined in ISO15926 standard. Based on ISO 15926 Part 2 and 15926 Part 4, ISO15926 Part 7 specifies Generic Templates and Object Information Models. Layer 2 and 3 are the user layer, which includes proprietary OIMs and Document Classed (Ontology) and User data in Facades (instances).

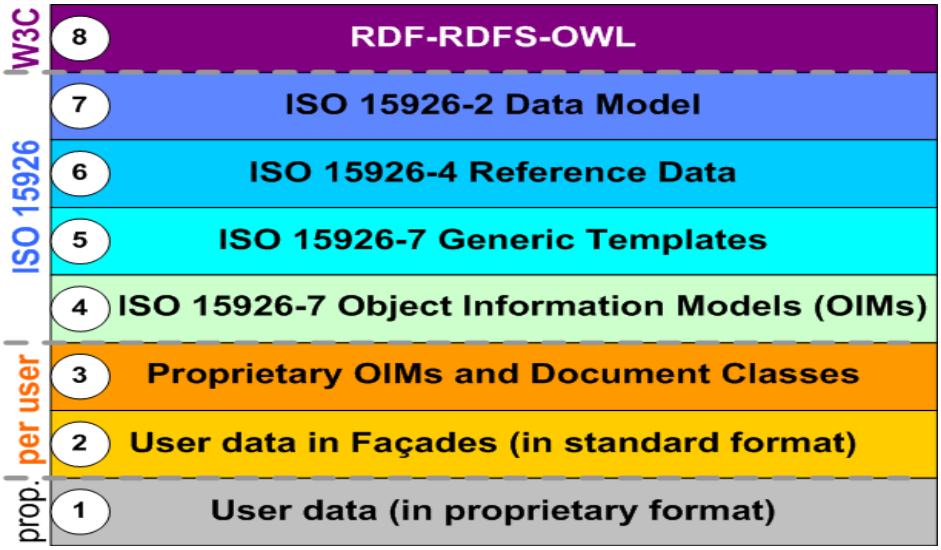


Figure 20 the ISO 15926 stack, (from [27])

3.2.3.2 Template specification

A template is a standard format for a kind of data sheet to enable common look for every user. It is a lower level model built upon ISO15926 part2. Someone use the metaphor that considers the template as lego block that you can use to build anything you like. As I understood, if we regard the part 2 as "grammar" and part 4 as "words", than the template is the "phrase" of a sentence. The template is a generic model. And there are specified templates for each field of industry. For example, there are templates for "pumps"," piping", in the first step of data integration you have to find out which templates fit for you.

There are two kinds of templates in ISO15926-7 that are Shorthand Template (ST) and Longhand Template (LT). "*A Longhand Template is a collector of ISO 15926-2 entity data types that together capture the representation of the information one wants to exchange."[27]* LT gives a full definition of information that is based on the ISO15926-2 data model. However, it would be a waste of resources for storing and processing many objects we do not need at all. Therefore that Shorthand Template is introduced. "*A Shorthand Template is an n-ary relationship that only points at the variant "leaves" of the graph of its companion ("isDefinedBy") Longhand Template."[27]* It is much more efficient to use shorthand template than longhand template. In the case that you may need Longhand Template you can get it via the "isDefinedBy" property. In an ideal situation, if we want to do the data integration, all the templates should be created and published by a standard organization. Since the ISO15926-7 is still under development and not fully published, on this project I will define some simple example templates according to the methodology when necessary.

Figure.21 shows an example of Longhand template specification. The LT-1002 is the name of the longhand template. It inherited the classes and object property from the ISO15926-2. "Classification", "Beginning" and "TemporalWholePart" are subclasses of the basic element "Relationship" in ISO15926-2, which defines the relation between things.

“ClassOfTemporalWholePart”, “ClassOfRepresentationOfThing”, and “ClassOfInformationRepre
sentation” are the set of individuals.

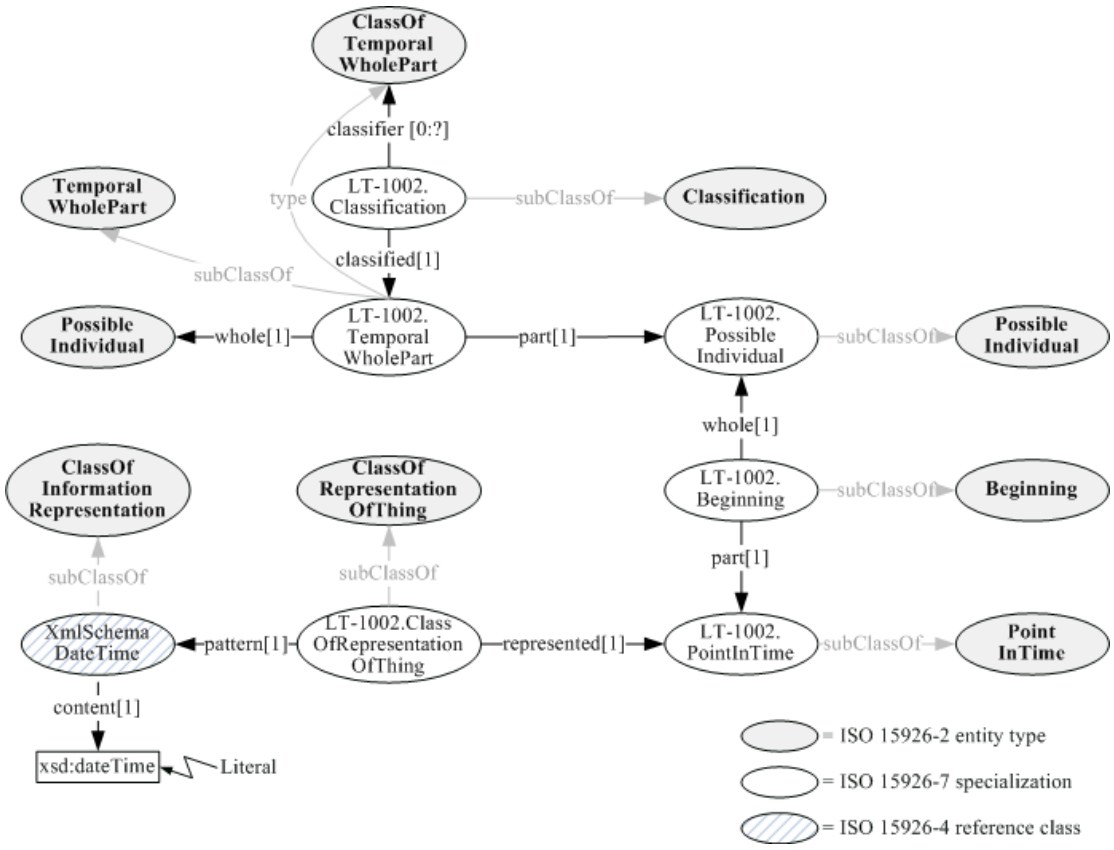


Figure 21 Longhand Template Specification, cited from [27]

Figure.22 shows an example of comparison of Longhand Template and Shorthand Template that cited from [27]. From this figure, we can clearly see the difference of Longhand and Shorthand Template. The Longhand Template gives the well-form definition of the model LT-1002 according to the ISO15926, whereas the Shorthand template turns to be developer friendly. It is easy to see that the ST-1002 is a MultidimensionalObject that has the property: temporalWhole, tempralPart, context, and beginning. The instance of ST-1002 is showing as follow.

```
<part2:MultidimensionalObject rdf:ID="ST-12321">
 <rdf:type rdf:resource="http://tpl.rdlfacade.org/data#ST-1002"/>
 <part7:temporalWhole rdf:resource="#ddf2"/>
   <part7:temporalPart rdf:resource="#dtss2"/>
   <part7:context rdf:resource="http://www.ontology.com/rdl#Cfd"/>
 <part7:beginning rdf:resource="#SDFD"/>
</part2:MultidimensionalObject>
```

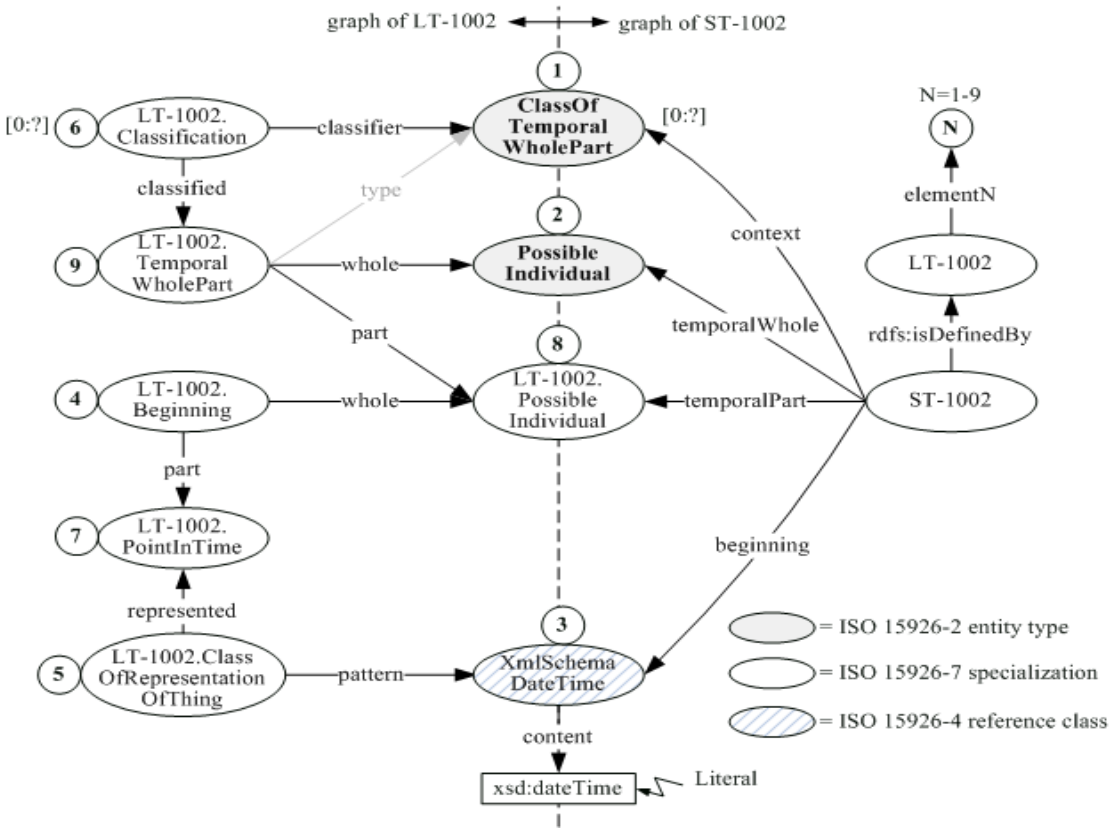


Figure 22 comparison of Longhand Template and Shorthand Template, from [27]

3.2.3.3 OIM (Object Information Models)

OIM ontology specifies the template. It is defined by domain experts, which means that each of the companies that want to use ISO15926 standard has to participate in the development of OIM. The models in OIM give more specific information of things than the template.

3.2.3.4 Data integration in a distributed system

The Façade concept is introduced to solve the problem of implementation of the above technology in a distributed system. Façade is a web server that can store triples, and supply an API (Application Programming Interface) to share information between all the Facades. The following figure.23 shows the information chain of façade. Each of the system façade hands over its messages to the Group façade, and the Group façade hands over its information to the Project façade. In this way all information in the façade can be available to the other façades. The façade support the Sparql query, the query can retrieve information from one or more façades at one time.

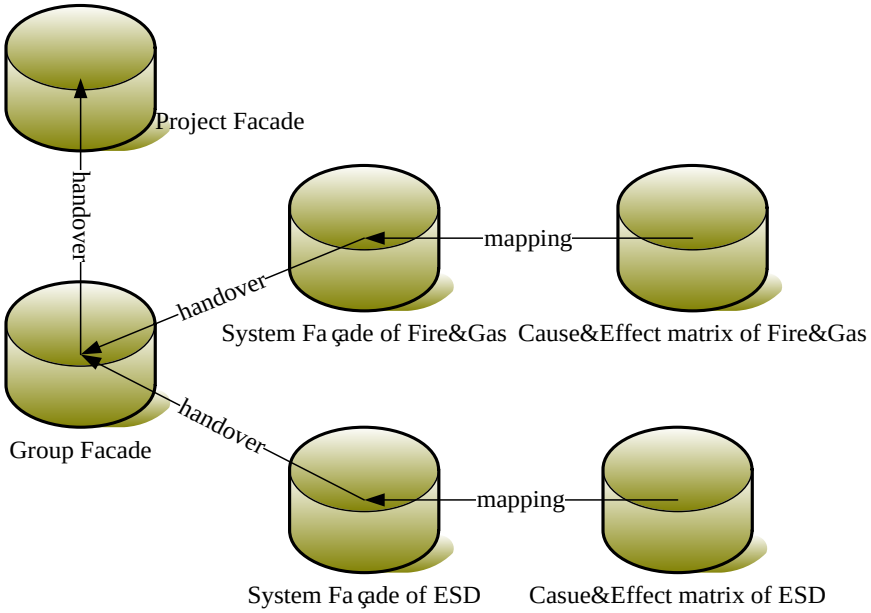


Figure 23 Information chain of Facades

4. Design specification

According to the data integration structure (in problem delimitation Chapter 2.2.1), and the implementation methodology defined in ISO16926 Part 7 (in Chapter 3.2.3), this project would like to design the data integration system like it shows in figure.24. Similar architecture of ontology based data integration architecture can be found in [4], [11], and [32]. The users of all the integrated systems share the same User Interface. The Query Engine and Reasoning engine separate the users with the data layer. It is usually developed by the integration software engineer who knows both the requirement of user and the developing tool of Semantic Web. This project will implement a prototype with the querying and reasoning system. The Cause&Effect matrix, which is the data source ontology, need to be mapping manually to the System Fa çade. As it introduced in the ISO15926 part-7 the system Fa çade are the web service that can store triples. The triples are based on the ISO15926 standard. Each system fa çade should handover its data to a group fa çade. The group fa çade stores the integrated ontology of the whole group. The Reasoning Engine could reason through the group fa çade to get the integrated meaning of the query user input. Then the Reasoning Engine sends the result back to the Querying Engine. Therefore, the querying engine could use the result from Reasoning Engine to implement semantic query of the real-time data.

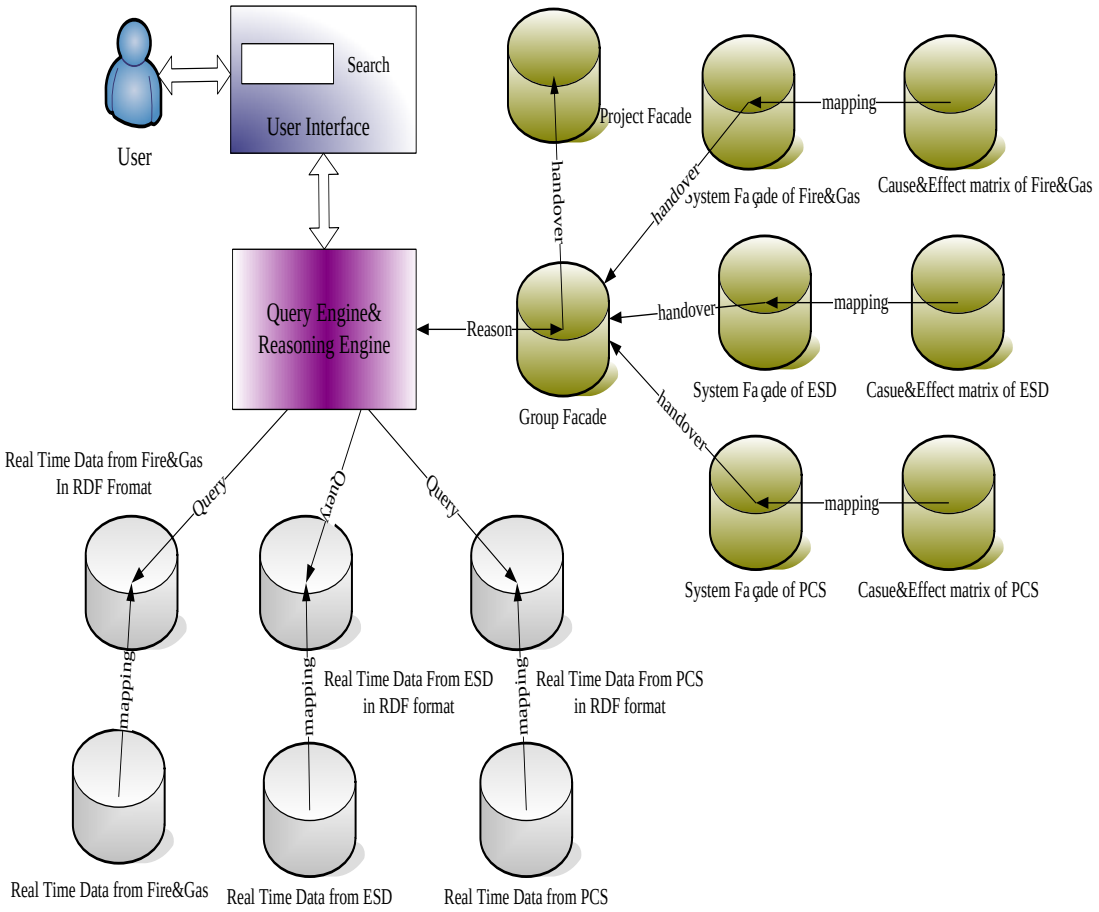


Figure 24 Data integration system architecture

There are three systems involved in the data integration. The Fire&Gas system belongs to Origo Engineering AS. Their database, Cause&Effect matrix and domain experts are available for the implementation. Therefore, this project mainly focuses on the model design and implementation of Fire&Gas system. The ESD (Emergency shutdown system) and PCS (Process Control System) are introduced for testing and verification of the data integration ability of the prototype.

The system contains the following parts:

- Mapping the Cause&Effect matrix to OWL ontology based on the ISO15926 standard. The OWL ontology is stored in the Façade database. We tried two approaches to implement this part.
  (1) Manual mapping approach: use ontology creation and modification tool Protégé to map Cause&Effect matrix to OWL ontology based on ISO15926 standard
  (2) Automatic mapping approach: Use transformation software JXML2OWL to map automatically. (Note: Although this approach has proved not suitable for this project, the automatic mapping problem is valuable for research. It will be discussed in the discussion chapter)

- Mapping the real-time data to data source ontology: depending on the format of the real-time data, there are two approaches can be used as follow:
  (1) Mapping from relational database to OWL instance: use Jena API to implement.
  (2) Mapping from XML to OWL: use JXML2OWL to implement

- Semantic Query engine and Reasoning engine implementation: it contains two parts as follow:
  (1) The query engine receives the query information from the user interface, queries the real time databases and returns the query results. The query engine is developed based on Jena API, and the RDF query language Spaqrl is used to query data.
  (2) The reasoning engine reasons the queries of the user, get the semantic information according to the ontology stored in Façade. The reasoning engine is also developed based on Jena API, and the reasoning tool Pellet is used as reasonor.

- User interface: The user interface should be user friendly. It supports keyword searching and advanced searching in different conditions. JSP is used to implement the user interface.

4.1 Mapping the Cause&Effect matrix to ISO15926 Specification

As it shows in Figure.25, the manual mapping includes the following steps:

- Analyze the ISO15926 Part 2: Figure out the functionality of classes in the Part 2, find out the top level classes and relations that could be used in this project.

- Analyze the ISO15926 Part 4: Based on the top level classes and relations, find out how are the Part 2 and Part 4 cohered as a whole, for example the classification and specification relationships. Besides, figure out the class hierarchy of the Part 4.

- Analyze the Cause&Effect matrix: This step is going on currently with the above two steps. Find out to what extent the information of Cause&Effect matrix can be expressed by the ISO15926 standard.

- Mapping the terminology in Cause&Effect matrix to Part 4: Map the terminology based on the above steps. This step needs to be iterated in order to reach an unambiguous mapping.

- Define ISO15926 Part 7 template specification and design Object Information Model: Based on the "words" given by Part 4 and "grammar" given by Part 2, it could be possible to formulate a "sentence". However, without the semantic given by the Cause&Effect matrix, the "sentence" can make no sense.

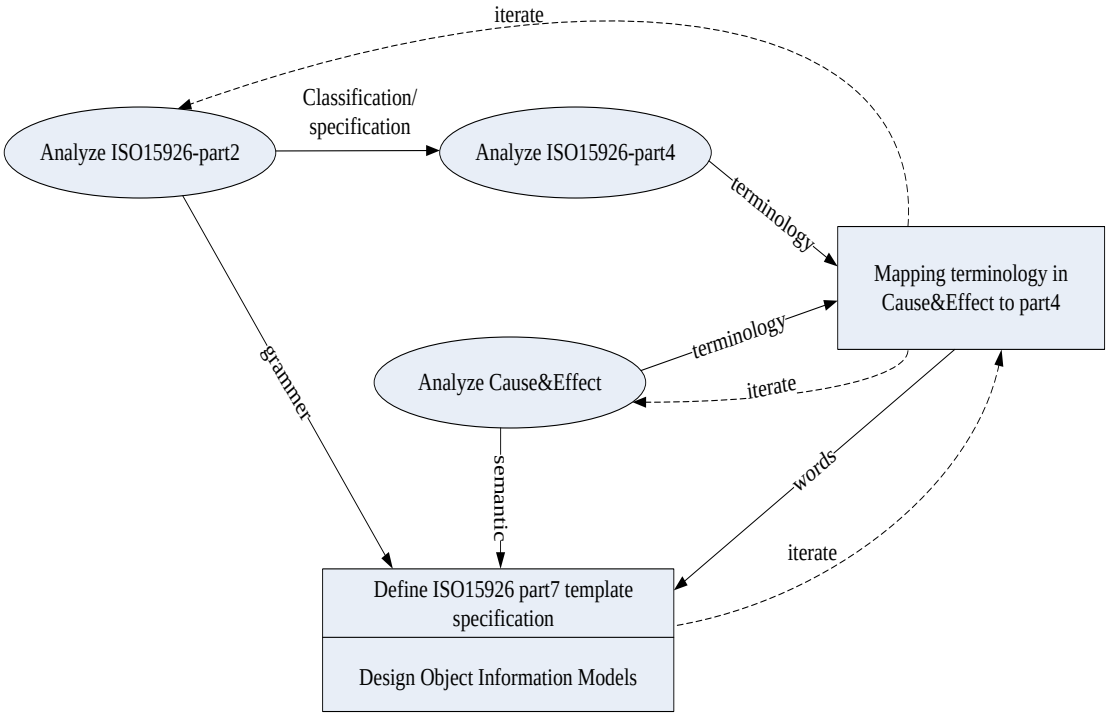


Figure 25 work flow the manually mapping

4.2 Mapping the real-time data to data source ontology Specification

As it shows in Figure.26, the XML2OWLMapping class is the main class of mapping. It gets the real time data from the database by GetDataFromDatabase class, and create ontology model from the ISO15926 Part 7 ontology database by OntologyModelCreation class. Then it maps the elements of real time data to the relative individuals of classes in the ontology, and stores it in

RDF format.

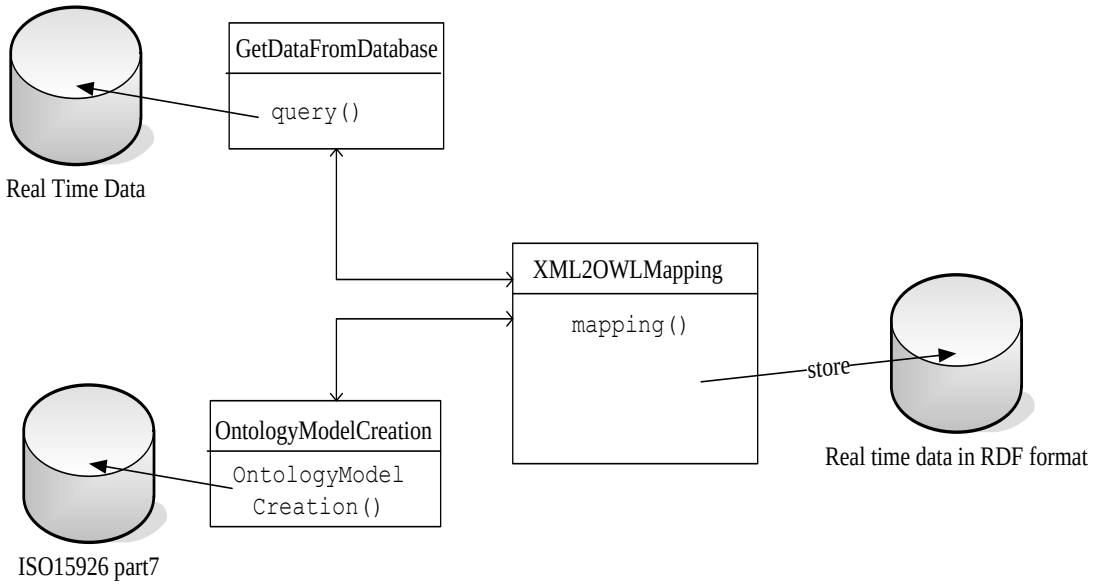


Figure 26 real-time data mapping structure

4.3 Semantic Querying and Reasoning system specification

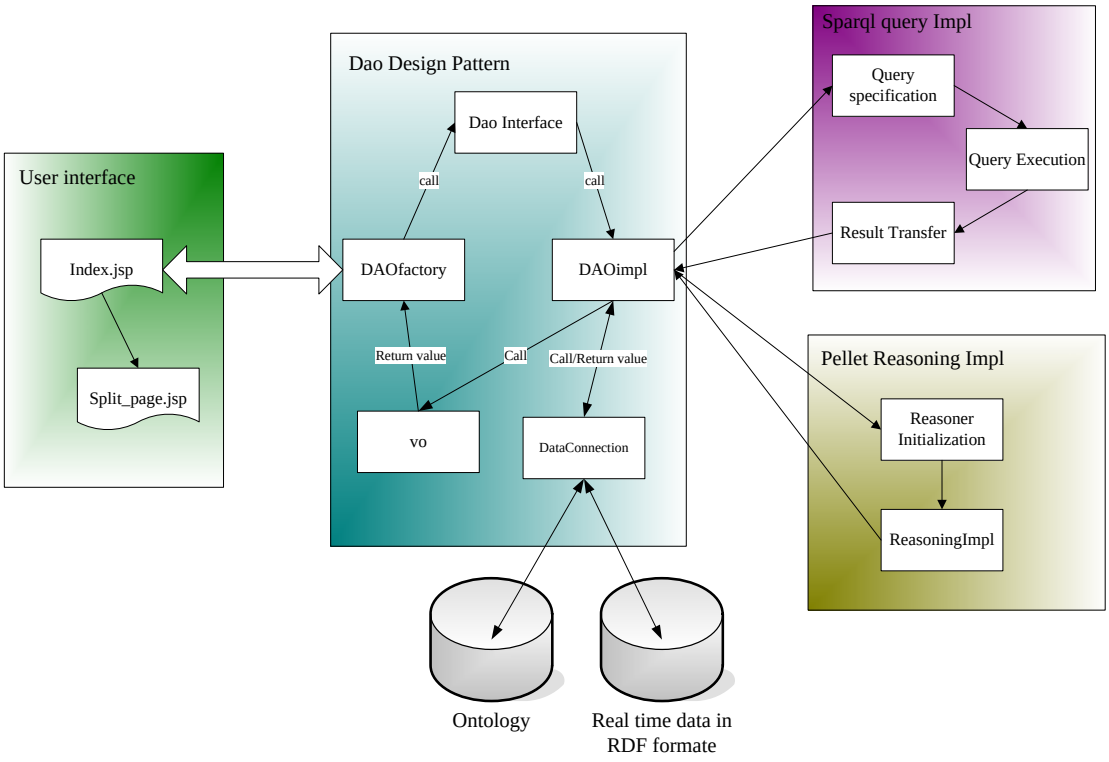


Figure 27 Semantic querying and reasoning system structure

As it shows in figure.27, this project uses the DAO (Data Access Object) design pattern to

separate the user interface, data access, and service logic. The user interface is designed by JSP. The main page is written in index.jsp file, it is responsible for interaction between the user and DAO. The Split_page.jsp file is responsible for constructing the appearance of the user interface.

The index.jsp can initialize the DAO factory class. The methods in DAOFactory class return a DAOimpl class as A DaoInterface. The DAOimpl implements the DAOInterface. Therefore, in the index.jsp it can call the methods in DAOInterface, which are implemented by DAOimpl class. The DAOimpl contains “Sparqul query Impl” part and “Pellet Reasoning Impl” part. The DataConnection class is responsible for getting data from the database. In this project it gets data from an ontology database and a real-time database in RDF format. The vo class is short for value object, it contains all the business values required by the clients.

The “Sparql query impl” part implements the querying of the ontology and real-time data. It contains three steps. First, the Query specification step specifies the Sparql query as the requirement of the clients. Second, execute the query. At last, the query result needs to be transferred into value objects.

The “Pellet Reasoing impl” part implements the reasoning of the ontology by using the Pellet Reasonor. It contains two steps. First, the reasonor needs to be initialized according to the methods in the Jena API. Second, the reasoning is implemented by finding graphs in the inferred result

5. Mapping Implementation

5.1 Mapping the Cause&Effect matrix to ISO15926

This implementation maps the Cause&Effect matrix to OWL ontology based on ISO15926 standard. We will design ontology based on ISO15926 for Cause&Effect matrix, in which the "grammar" is defined by ISO15926-2 and "word" coming from ISO15926 part-4. The software Protégéis used as model creation and modification tool. And CMapToolsCOE is used as the graph representation of the OWL ontology.

5.1.1 Hierarchy of the Models

Figure.28 shows the hierarchy of the models. The models are related by the subclass relations. Most of the classes are in the yellow color, some of them are in the red color. The classes in red color have some restrictions that are both sufficient and necessary. For example, assume the class A has sufficient and necessary restriction X, than if "a" is a instance fulfill the restriction X, than it is the member of A and if "a" is a member of A, then "a" has the restriction X. The owl:thing is the root class, all the classes in OWL should extend from it. All the classes in the Figure.28 are belonging to ISO15926 Part 2, ISO15926 Part 4, or ISO15926 Part 7. The details of them will be introduced in the following chapters. As you can see from Figure.28, it contains the following parts:

- Tag: Classification of different kinds of tags. It is defined in Part 7
- Activity: Classification of different kinds of activity. It is defined in Part 4
- Area: set of areas
- Datatype: set of datatypes
- Event: Classification of different kinds of events. It is defined in Part7
- CauseAndEffectChart: set of Cause&Effect charts
- Room: set of rooms
- PhysicalObject: set of physical objects that will be used in this project. It contains some kinds of detector, LED display and also the alarm panel.
- System: Classification of different kinds of systems. It is defined in Part 4
- Voting: Classification of different kinds of voting. It is defined in Part 7
- Note: Classification of different kinds of notes. It is defined in Part 7

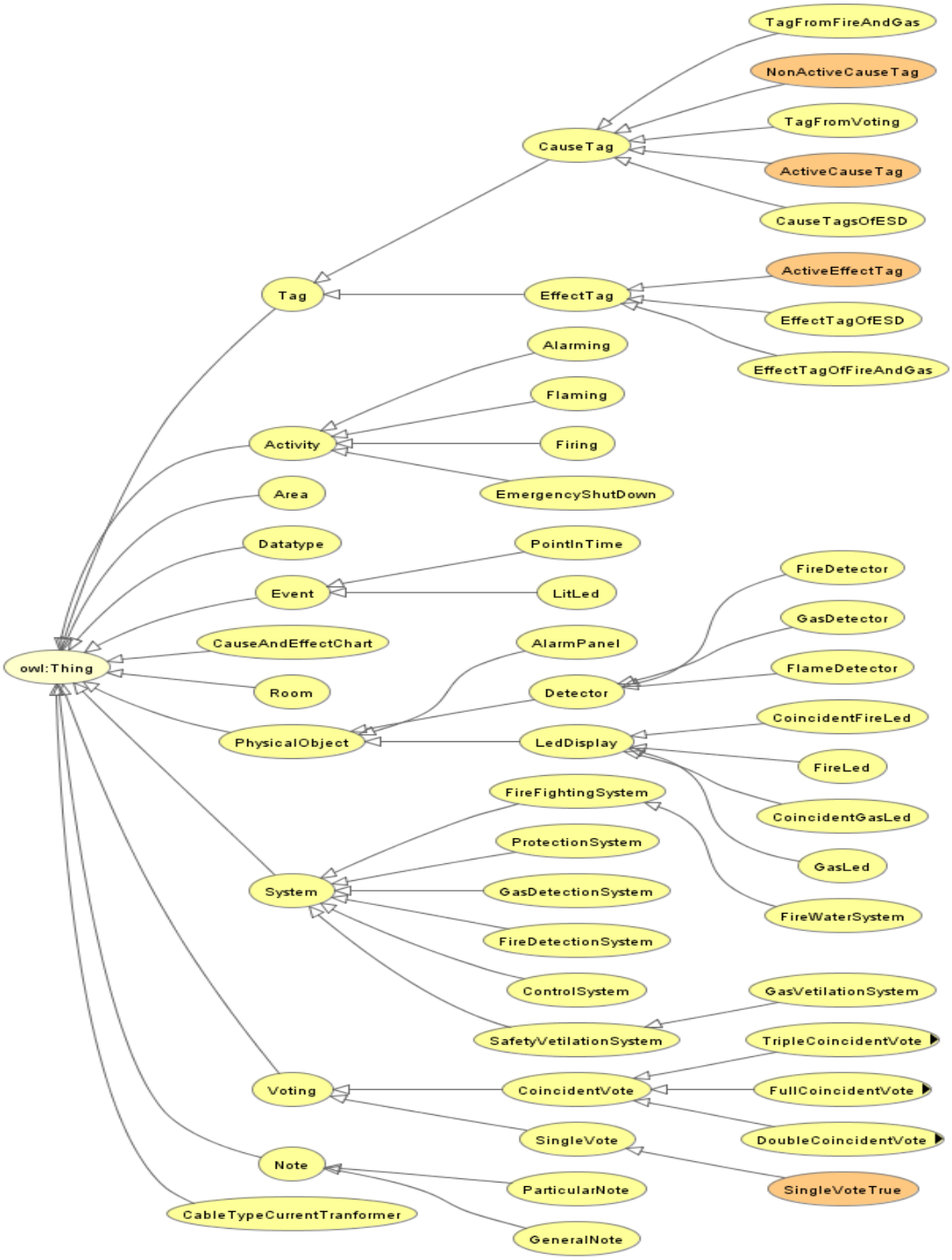


Figure 28 Class hierarchy of the models

5.1.2 “System” hierarchy

The Figure.29 shows the system hierarchy defines in the ISO15926 Part 4 [30]. In the Cause&Effect matrix of U51-2, it contains the “fire detection system”, “fire fighting system”, “gas detections system” and “safety ventilation system”. All these systems are subclass of “safety system”, which is the member of the “protection equipment class”

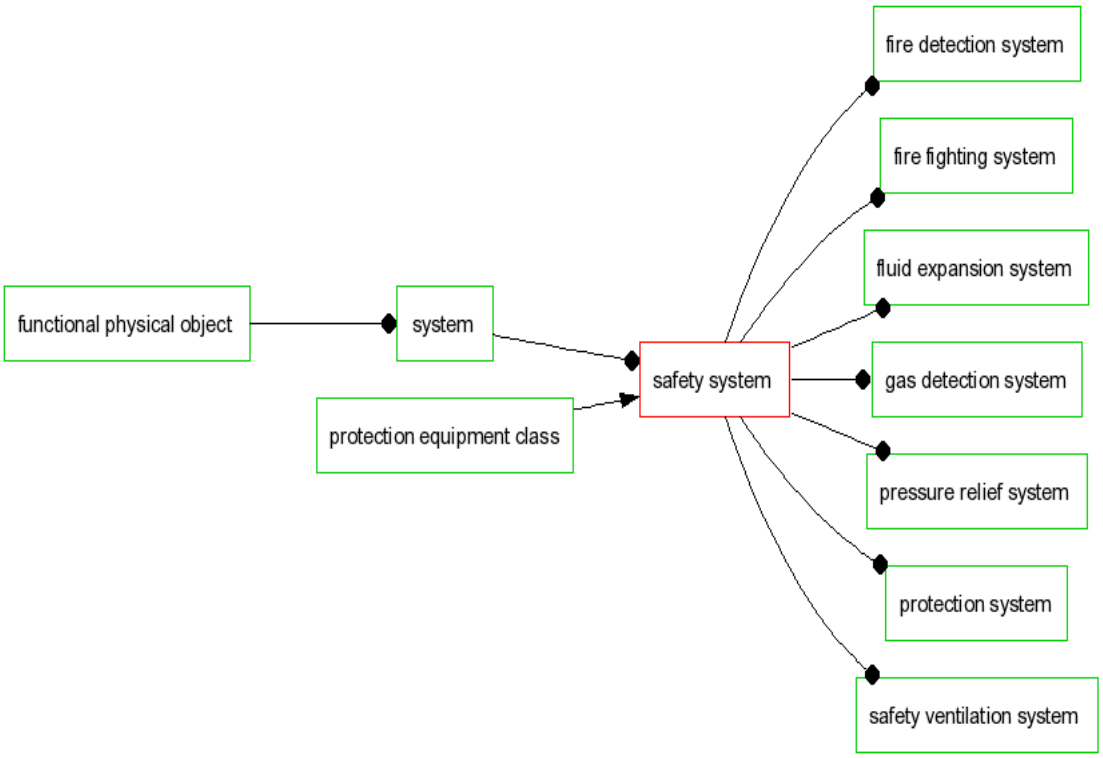


Figure 29 reference data of the “system” hierarchy, from [30]

These four systems are set into disjoint system as it shows in the following figure. Disjoint relations ship means the two classes has no common members.

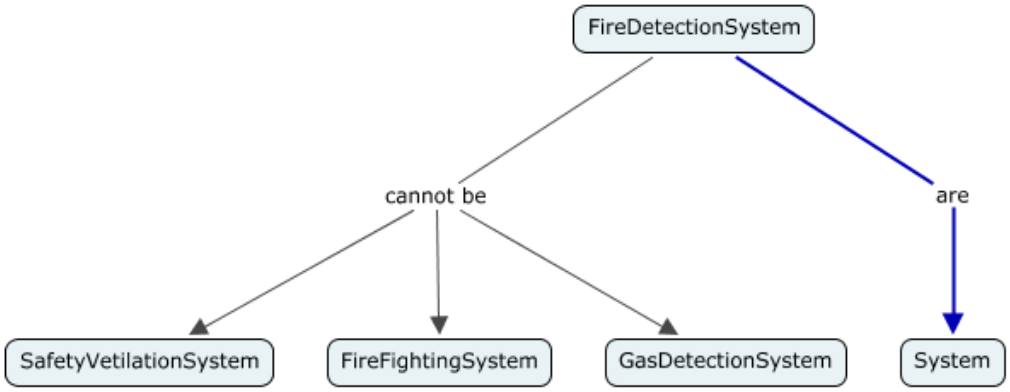


Figure 30 disjoint systems

5.1.3 Restrictions of the classes

Constrains defines the relationship, cardinality of classes. In order to give a clear view of the concept the following figures gives both graph representation and OWL abstract syntax definition. The “must be” tag in the graph representation equals to the “only” symbol in OWL abstract syntax, and also equals to the OWL element “owl:allValueFrom”. Likewise, the “can be” tag in the graph representation equals to the “some” symbol in OWL abstract syntax, and also equals to the OWL

element “owl:someValueFrom”.

Figure.31 gives the definition of “Activity” and “Room” classes. The “Activity” happens in the offshore area relates to the tag “CauseTag” by the object property “relatedTag”. There are should be some physical object involved in the activity, like “Flame detector” involves the “Flaming” activity. The activity happening may cause some event, that is defined by the “causeOfEvent” relationship. The “Room” here means the rooms in the offshore area, such as control room. Each room has a related “CauseAndEffectChart”, like the “CauseAndEffectChart” in area U51-2 is related to sea cable transformator room. The cardinality of the object property “hasChart” is constraint to exactly one. Each room contains at least one system. And the room must be located at an area.

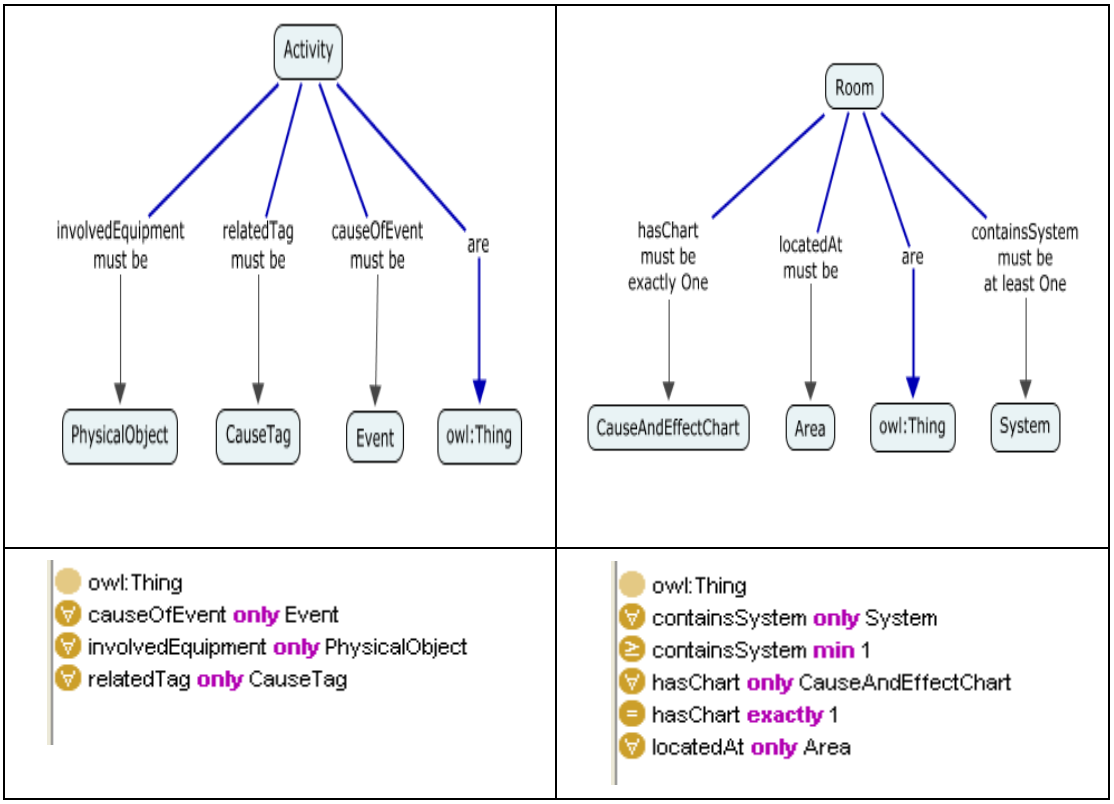


Figure 31 “Activity” and “Room” class definition

Figure.32 gives the definition of “System” and “LitLed” classes. Each system contains some tags that used for transferring information. The “Event” “LitLed” uses at least one equipments from “LedDisplay”.

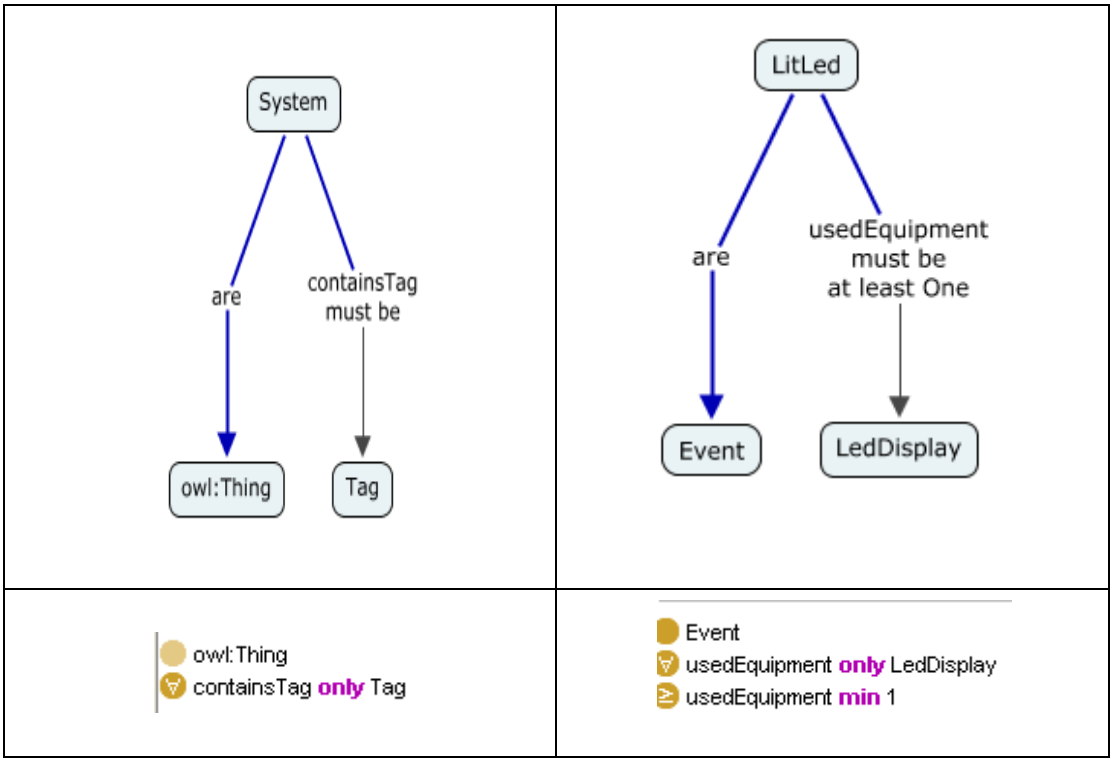


Figure 32 “System” and “LitLed” definition

Figure.33 gives the definition of “CauseAndEffectChart” and “Area” classes. Each “CauseAndEfectChart” class relates to a room. One “CauseAndEffectChart” contains at least one “CauseTag” and at least one “EffectTag”. Each area may contain some rooms.

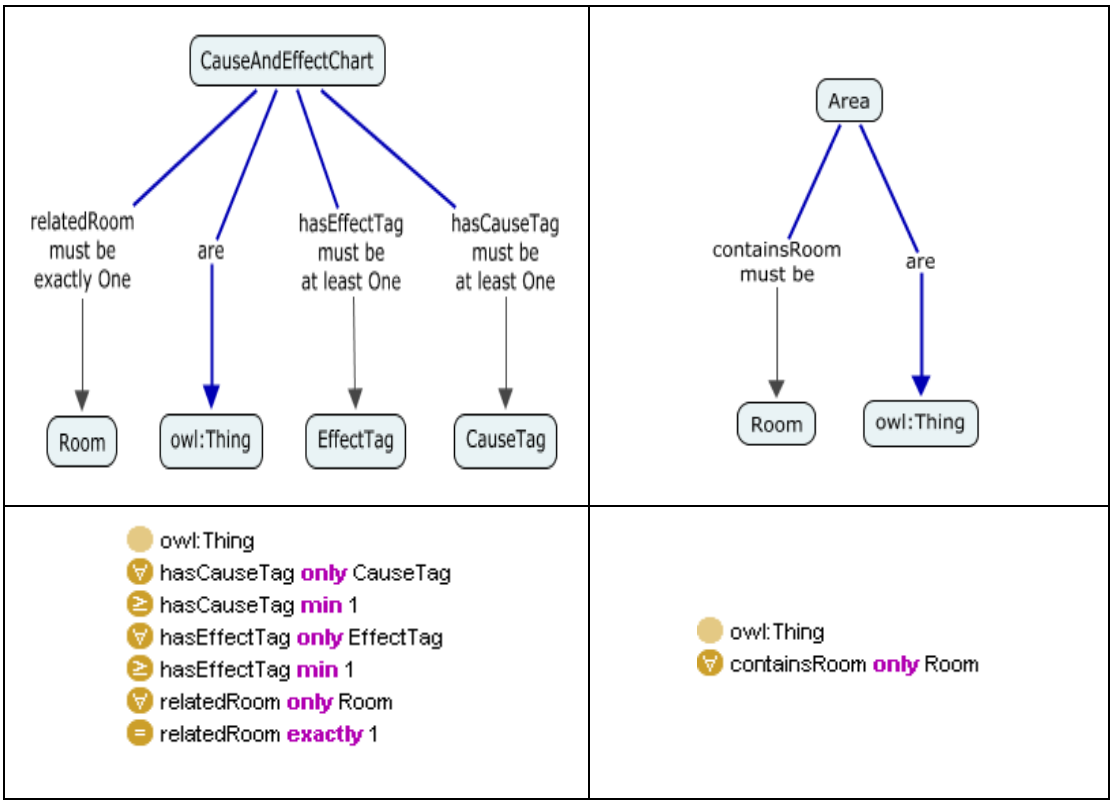


Figure 33 “CauseAndEffectChart” and “Area” class definition

5.1.4 Tag classification

Figure.34 shows the “CauseTag” and “Tag” class definition. Each tag belongs to a system by “belongToSystem” property. And a tag may have a note that gives some extra information. This definition corresponds to the “Note” element in the Cause&Effect chart. The “CauseTag” is the subclass of the class “Tag”. It inherits all the attributes from “Tag”. And it is disjoint with “EffectTag”. Each “CauseTag” is involved in an “Activity”. The “CauseTag” match to the “EffectTag” by the “causeOfEvent” property. The “CauseTag” has the input type for the input signal. As it shows in figure.34, the “CauseTag” is classified into “TagFromFireAndGas”, “TagFromVoting” and “TagFromESD” according to the “From” element of Cause&Effect chart. “TagFromFireAndGas” is the “CauseTag” which has the “F&G” symbol in the “From” element. “TagFromVoting” is the “CauseTag” which has the “Voting” symbol in the “From” element. (Please check the “CauseAndEffect” chart in Chapter 2.3.1 if you are not clear about the description above)

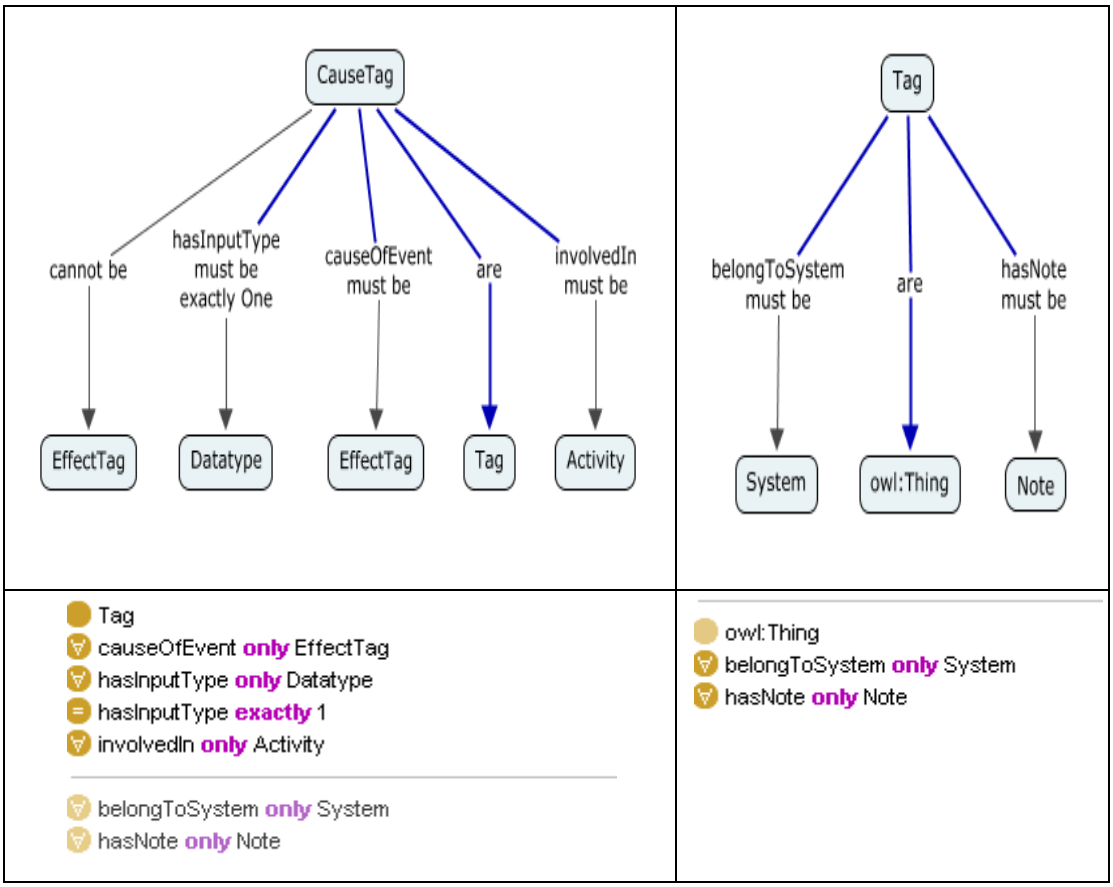


Figure 34 “CauseTag” and “Tag” class definition

Figure.35 shows the “EffectTag” and “ActiveEffectTag” definition. The definition of “EffectTag” is similar to the “CauseTag”. The differences are: the “EffectTag” relates to an “Event” rather than “Activity”, and the “EffectTag” relates with the “Datatype” class with the “hasOutputType” property. The “ActiveEffectTag” is defined as the “EffectTag” which is related to an “ActiveCauseTag”. That means if the “CauseTag” that the “EffectTag” related with is reasoning as

the type of “ActiveCauseTag” than the “EffectTag” will be reasoning as the “ActiveEffectTag”. The “ThingsWhich is define as” in the graph representation is equal to the Necessary&Sufficient definition in the OWL abstract syntax.

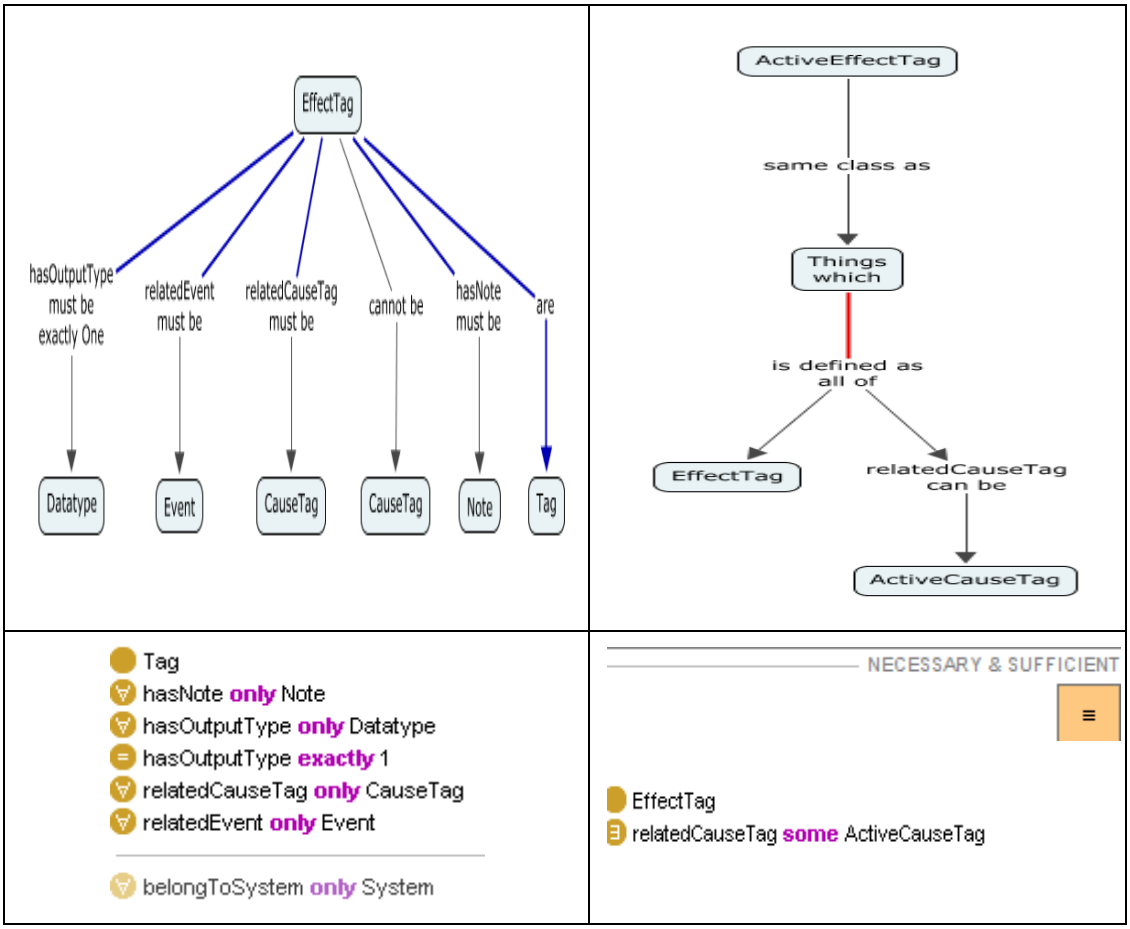


Figure 35 “EffectTag” and “ActiveEffectTag” class definition

Figure.36 shows the class definition of “ActiveCauseTag” and “NonActiveCauseTag”. It’s necessary to note that the “all of” in the graph representation is equal to the OWL syntax “owl:intersectionOf”. And “any of” is equal to “owl:unionOf”. The “ActiveCauseTag” and “NonActiveCauseTag” are disjoint class. The “ActiveCauseTag” is defined as a “CauseTag” that has state true or candidate of any kind of voting true. The voting specification will be described in the next chapter. As it shows in Figure.37, all the members of “TagFromVoting” class are candidates of one kind of voting. All the members of “TagFromFireAndGas” are voters of one kind of voting. The state of the member of class “TagFromFireAndGas” is evaluated according to the real time data collected from the sensors offshore. The state of the “TagFromVoting” is evaluated according to the voting result. Therefore, both of them can be inferred to be “ActiveCauseTag” if there are active.

The “NonActiveCauseTag” is defined as the “CauseTag” that has state false. Therefore, a member of “TagFromFireAndGas” can be evaluated to “NonActiveCauseTag” if the has state false. It does not cover the members of “TagFromVoting” because they are not real time data. It is not necessary to evaluate them to be non-active state.

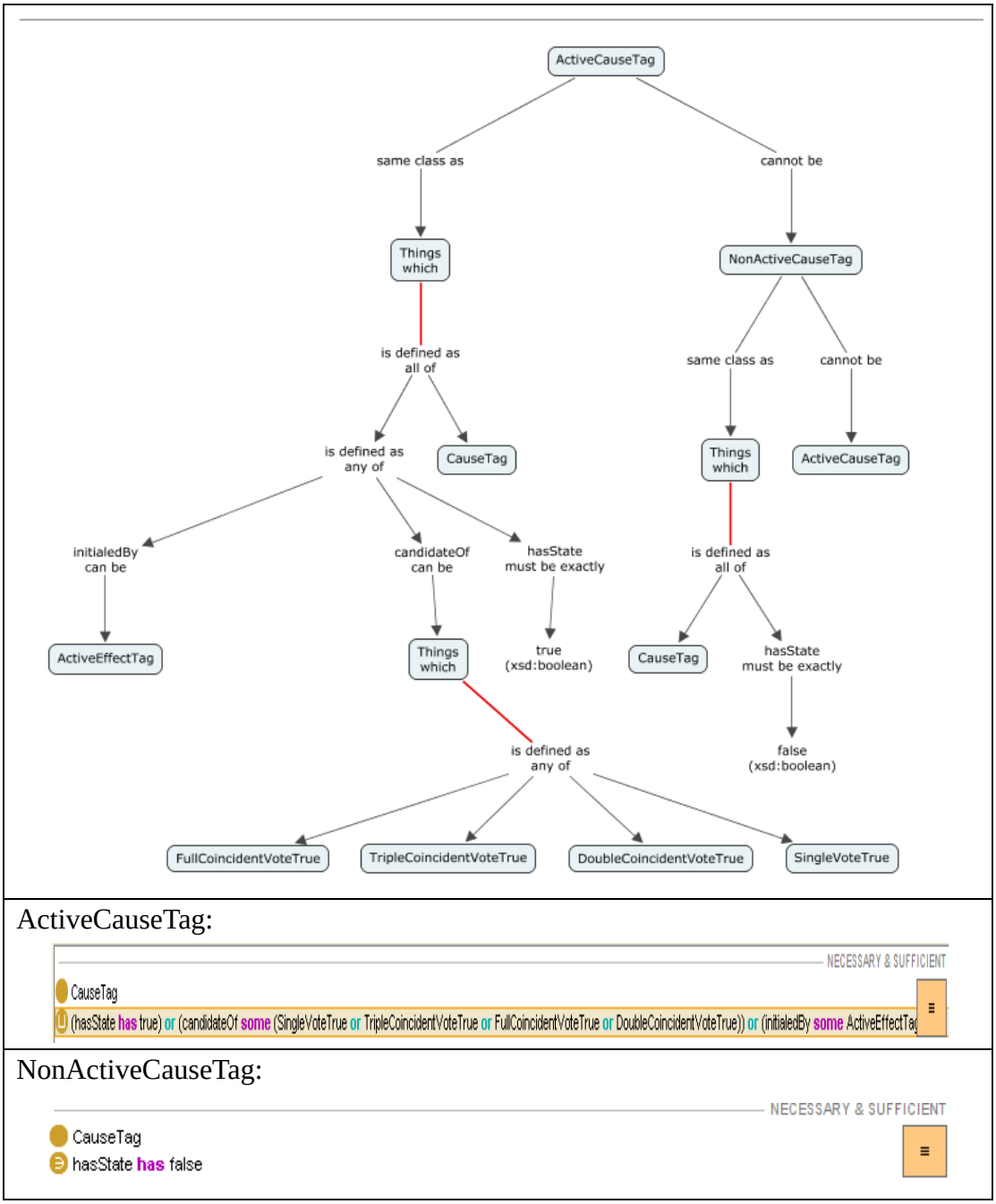


Figure 36 “ActiveCauseTag” and “NonActiveCauseTag” class definition

Figure.37 gives the class definition of “TagFromVoting” and “TagFromFireAndGas”. The restriction in light color is inherited from “CauseTag”.

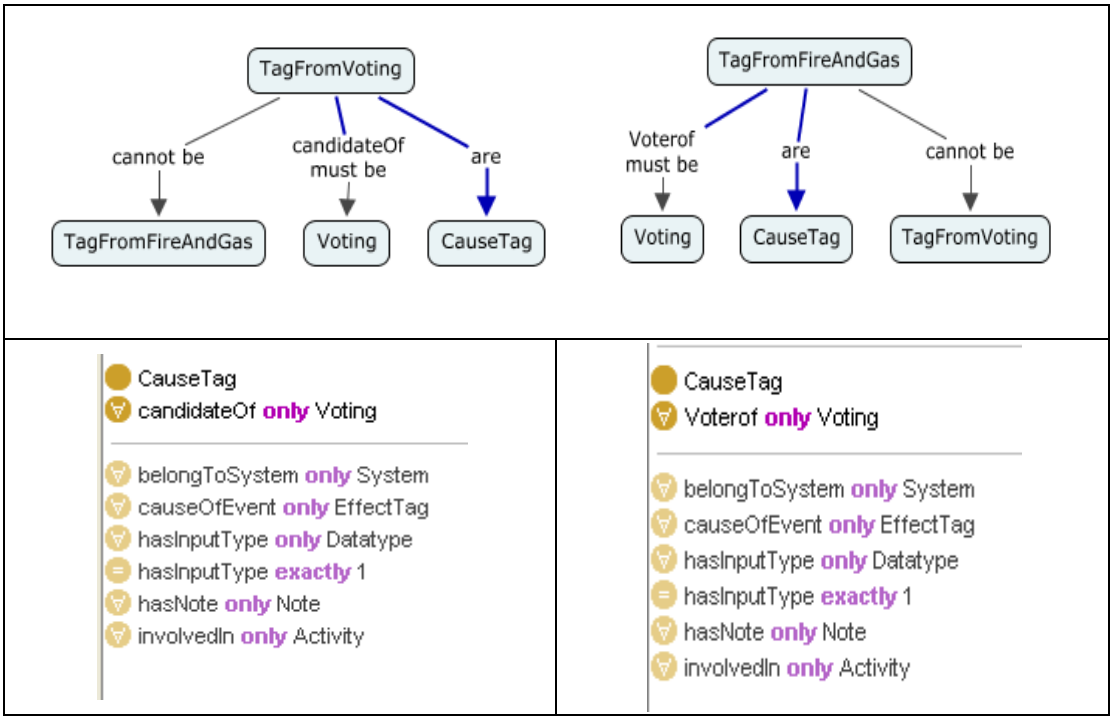


Figure 37 “TagFromVoting” and “TagFromFireAndGas” class definition

5.1.5 Voting implementation

Some cause tags from F&G will launch a voting if the tag is active. The voting will decide whether to initial a candidate or not. There are four types of voting: single vote, double vote, triple vote, and full vote, which depend on the number of active voter. The Reasoner can infer the voting result by the assertion of logic

- If a CauseTag x has state true, it will be infer as a ActiveCauseTag
- If x is a voter of a SingleVote y, then the SingleVote y will be infer as a SingleVoteTrue
- If y has a candidate Tag z, then the tag z will be infer as a ActiveCauseTag

Figure.38 gives the definition of “Voting”. The “Voting” has exactly one candidate from “TagFromVoting”, and at least one voter from “TagFromFireAndGas”.

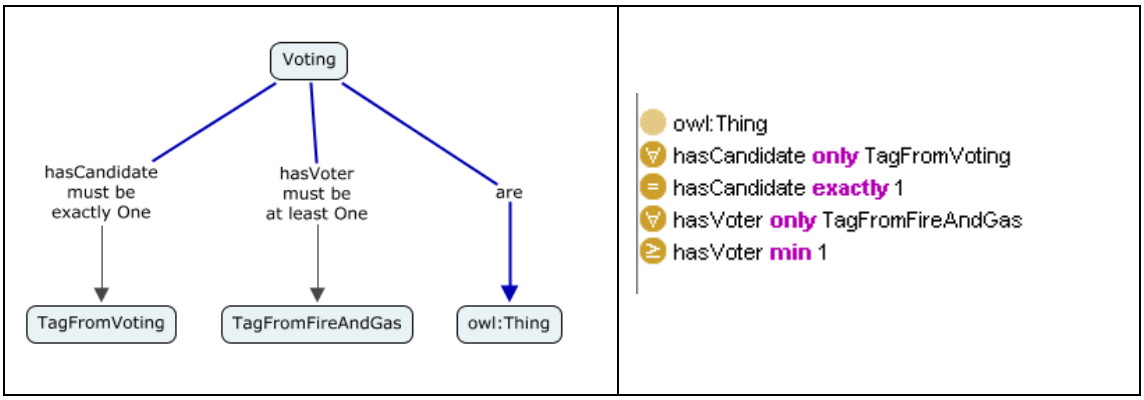


Figure 38 “Voting” class definition

Figure.39 shows the “SingleVotingTrue” class definition. It is defined as a “SingleVoting” class that has Voter that from “ActiveVotingTrue”. Any classes that are defined as “ActiveVotingTrue” class should have the property “hasActiveVoter” exactly one from “ActiveCauseTag”. The “hasActiveVoter” is the sub property of “hasVoter”. The figure also illustrates that the classes “SingleVotingTrue”, “DoubleCoincidentVotingTrue”, “TripleCoincidentVotingTrue”, and “FullCoincidentVotingTrue” are disjoint classes. They related to “1ooN”, “2ooN”, “3ooN” and “NooN” voting types in the Cause&Effect chart.

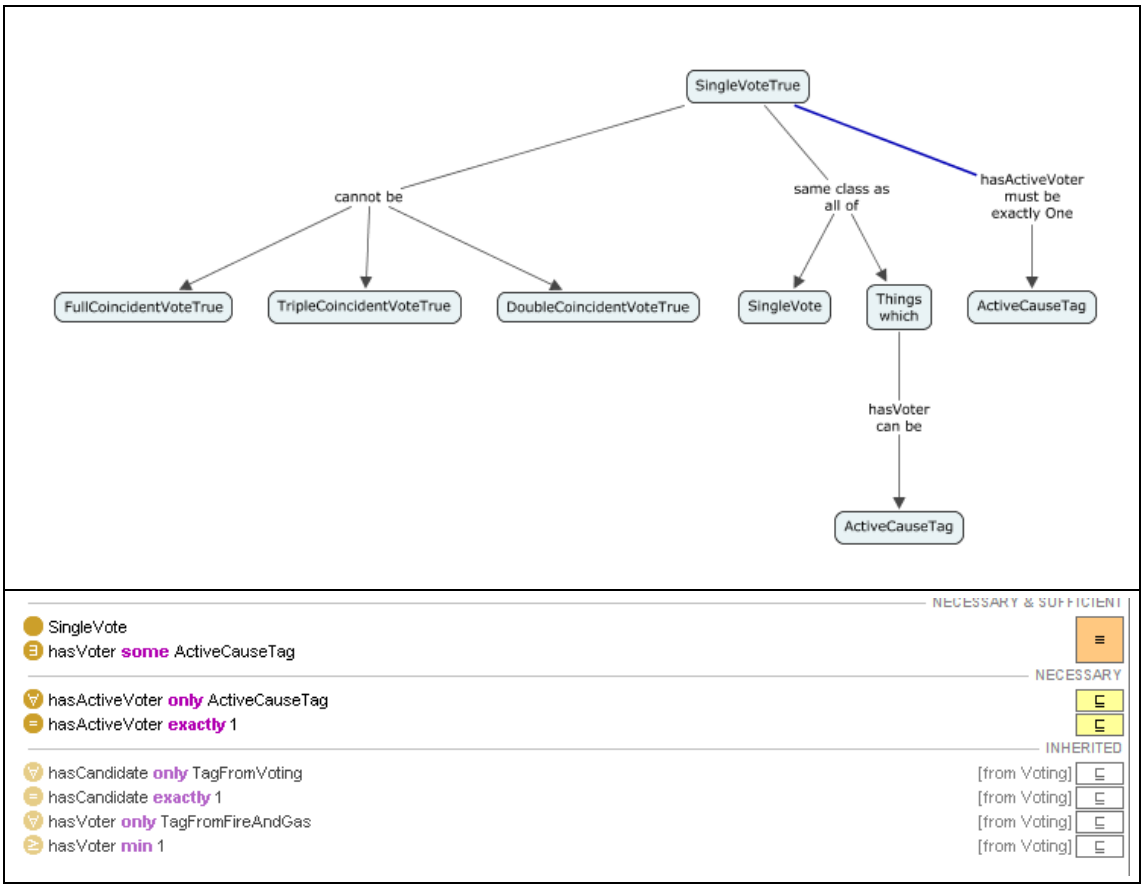


Figure 39 “SingleVotingTrue” class definition

Figure.40 shows the “DoubleCoincidentVotingTrue” class definition. As a “DoubleCoincidentVotingTrue” class it is necessarily has exactly two active voters. Assume that the “DoubleCoincidentVote” has two or three voters. It is defined as the same class as a “DoubleCoincidentVote” class that fulfills the following formula:

If the voting has two voters, than in $C_2^2 = 1$ conditions that the voting is true.

(hasVoter_1 some ActiveCauseTag ⊓ hasVoter_2 some ActiveCauseTag)

If the voting has three voters, than in $C_3^2 = 3$ conditions that the voting is true.

(hasVoter_1 some ActiveCauseTag ⊓ hasVoter_2 some ActiveCauseTag) ⊔
(hasVoter_2 some ActiveCauseTag ⊓ hasVoter_3 some ActiveCauseTag) ⊔
(hasVoter_1 some ActiveCauseTag ⊓ hasVoter_3 some ActiveCauseTag)

If the voting has four voters, than in $C_4^2 = \frac{4!}{2!(4-2)!} = 6$ conditions that the voting is true.

hasVoter_1	hasVoter_2	hasVoter_3	hasVoter_4
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0

Table 6 Conditions that double voting true

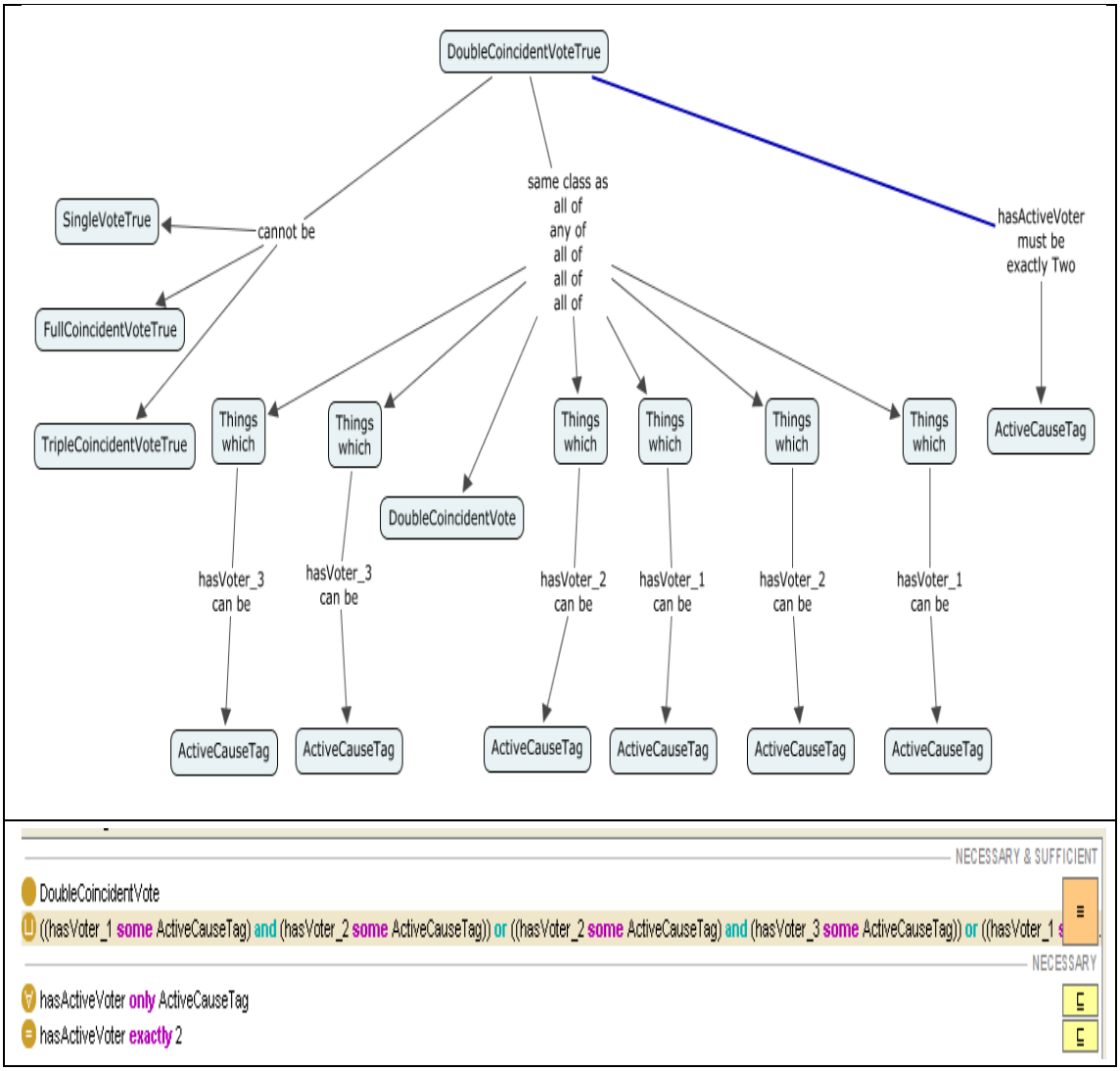


Figure 40 “DoubleCoincidentVotingTrue” class definition

The graph of “TripleCoincidentVotingTrue” and “FullCoincidentVotingTrue” are similar to the graph of “DoubleCoincidentVotingTrue” so we will not show them here. The class “TripleCoincidentVotingTrue” is defined as the same class of a “TripleCoincidentVote” that has

the property “hasAvtiveVoter” exactly 3 from “ActiveCausTag”. The “FullCoincidentVotingTrue” is defined as the same class as a “FullCoincidentVote” that “haVoter” only from “ActiveCauseTag”.

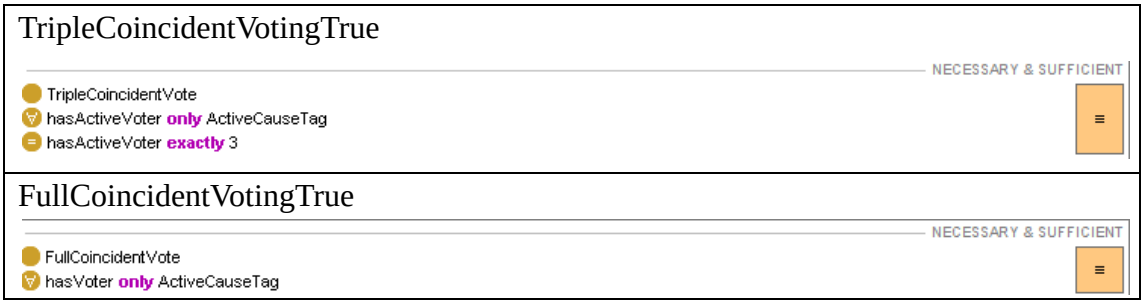


Figure 41 “TripleCoincidentVotingTrue” and “FullCoincidentVotingTrue”class definition

The figure.42 defines the restriction of the four properties: “hasVoter”, “hasActiveVoter”, “hasVotingTrue” and “Voterof”. It defines the domain and range of each property as it shows in the following chart. It also defines the “hasVoter” inverse property of “Voterof” and vice versa, and “hasActiveVoter” inverse property of “hasVotingTrue” and vice versa. The “owl:inverseProperty” means if you define the axiom (a , hasVoter, b) than it also have the axiom (b, Voterof, a). Moreover, it defines (hasActiveVoter, owl:subPropertyOf, hasVoter) and (hasVotingTrue, owl: subPropertyOf, VoterOf).

ObjectProperty	**Domain**	**Range**
hasVoter	Voting	TagFromFireAndGas
hasActiverVoter	Voting	ActiveCauseTag
Voterof	TagFromFireAndGas	Voting
hasVotingTrue	TagFromFireAndGas	SingleVoteTrue or DoubelCoincidentVoteTrue or TripleCoincidentVoteTrue or FullCoincidentVoteTrue

Table 7 Domain and range of ObjectProperty

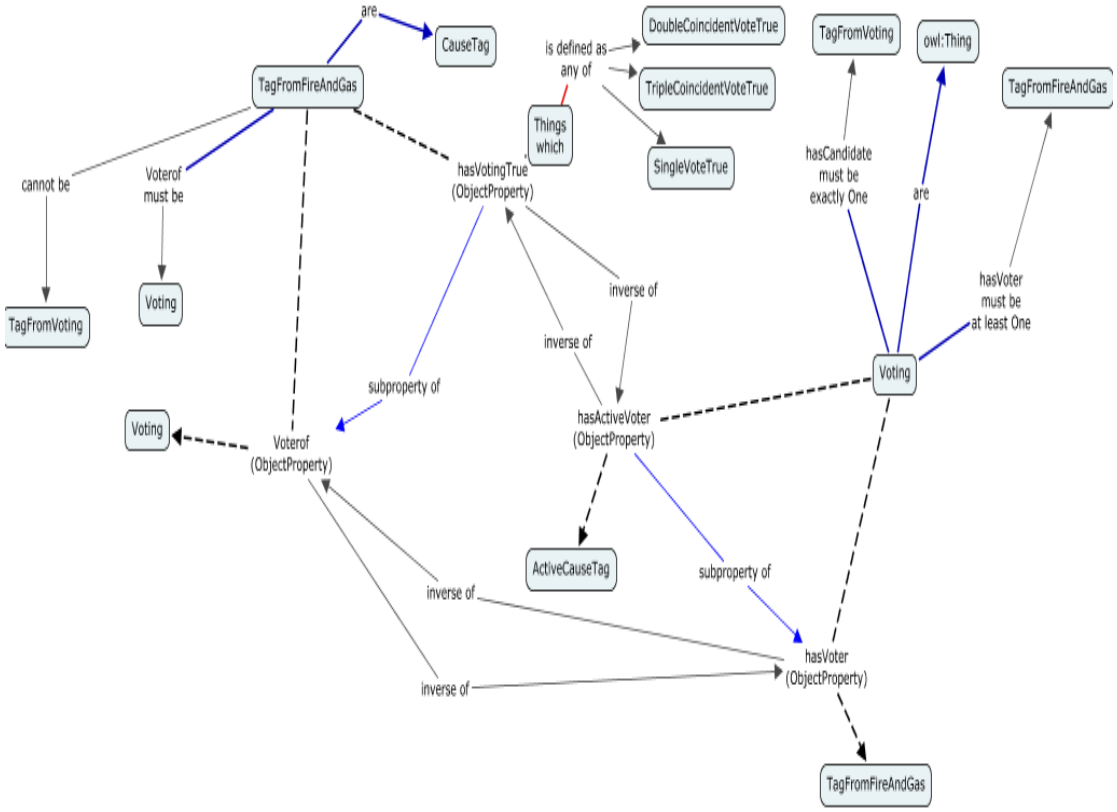


Figure 42 Property restriction

5.1.6 Instance definition

The class definition and restriction above are based on the ISO15926 standard. They could be used as shorthand template. If they get general approval by the companies, it could be used as common standard. For all the system that uses the similar Cause&Effect matrix, the template is reusable. However, the instance definition goes to more concrete level. For example, some of the tags are only used by the Origo Engineering. Obviously, this instance definition is not reusable for other situation. We defines the instance for illustration the functions of the classes. And used for the proving concept of prototype implementation.

The figure.43 shows the definition an instance “O87C_51_CF001”, which is a member of “TagFromVoting”. This tag belongs to the system “FlameInArea”. It matches with effect tags: “O71_XY228”, “O87_U51_FWP”, and “O700_XA_072_2”. It is a candidate of a voting “FullCoincdientFlameVoting”. It has input type “INT”. It involved in an activity “CoincidentFire” which is a type of “Flaming”.

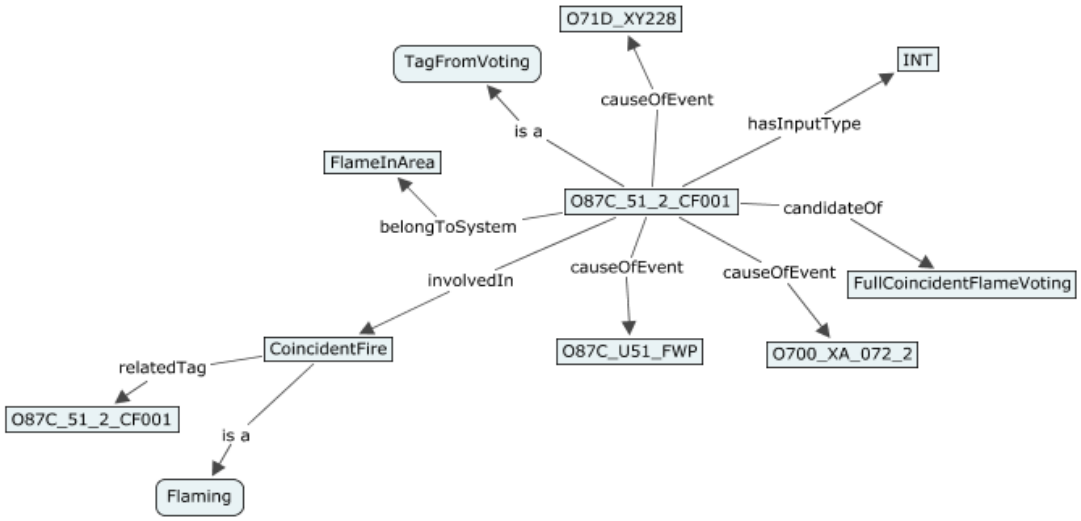


Figure 43 Instance of “TagFromVoting”

Figure.44 shows the definition of an instance of the “Room” and the “CauseAndEffectChart”. The room “SeaCableTranformatorRoomArea-North” is a type of “CableTypeCurrentTransformer” that has a “CauseAndEffectChart” that is “CauseAndEffectChart_U51_2”. The room contains a lot of systems by the “containsSystem” property. This room is located at area “U51-2”. The “CauseAndEffectChart_U51_2” has some cause tags and effect tags. And it is related to the room by “relatedRoom” property, which is the inverse property of “hasChart”.

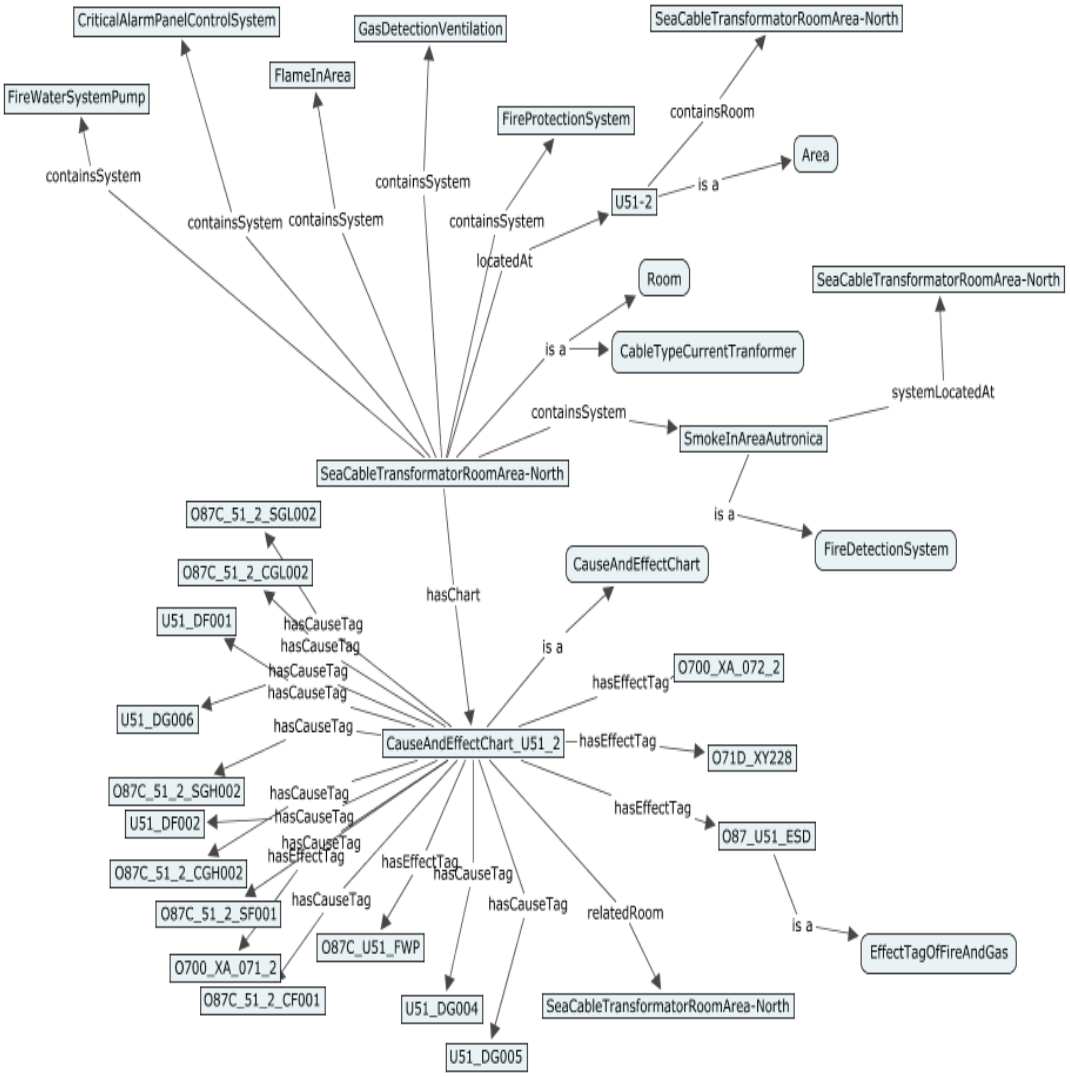


Figure 44 Instance of “Room” and “CauseAndEffecChart”

Figure.45 shows an instance of “GasDetectionSystem”. This system “GasDetectionVenidation” is located at room “SeaCableTranformatorRoomArea-North”, and it contains some cause tags.

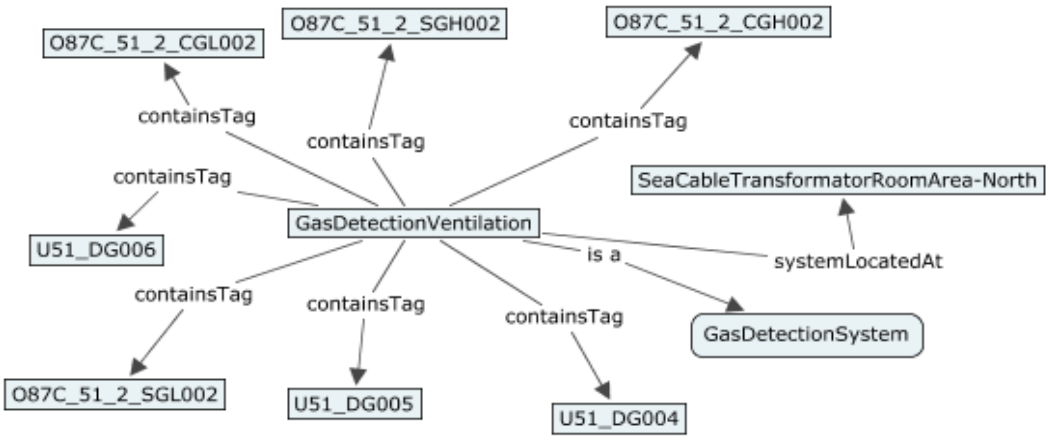


Figure 45 Instance of “GasDetectionSystem”

Figure.46 shows an instance of the class “TagFromFireAndGas”. The tag “U51_DF001” belongs to the system “FlameInArea”. It is both the voter of “SingleFlameVoting” and “FullCoincidentFlameVoting”. It has the input type “AI”, which means the analog input. It involved in the activity “FlamingDetetorUtilityHandling”. It has the DatatypeProperty “hasState” which is false. This state should be changed according to the real time data.

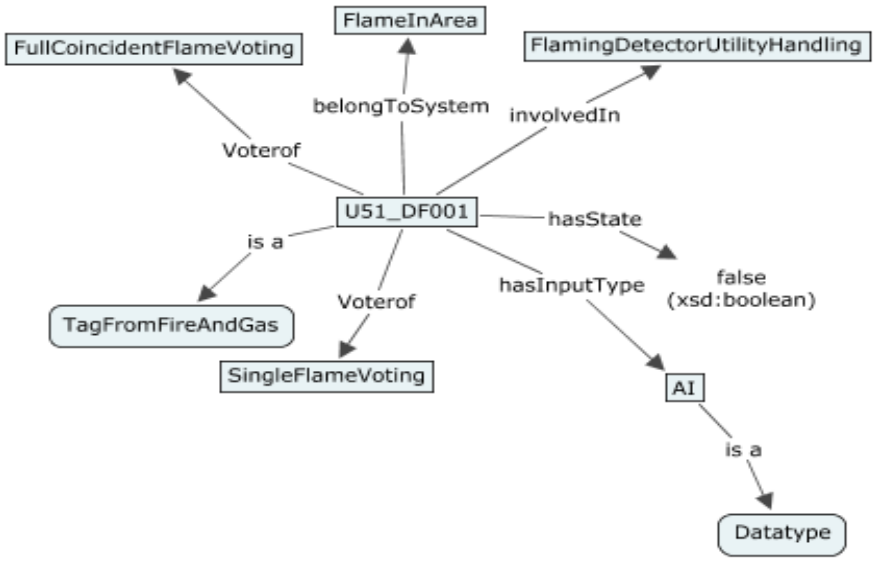


Figure 46 Instance of “TagFromFireAndGas”

Figure.47 shows the instance of “Flaming”, “EffectTagOfFireAndGas”, and “SingleVote”. The “FlamingDetectorUtilityHandling” is an activity that relates with tag “U51_DF001” and “U51_DF002”. It has involved equipment “StarEye_2000”, which is a “FlameDetector”. “O87C_U51_FWP” is an “EffectTagOfFireAndGas” that has input type “INT”, belongs to system “FireWaterSystemPump”, and has a related CauseTag “O87C_U51_2_CF001”. The “SingleFlameVoting” is type of “SngleVote” that has some voter and an candidate.

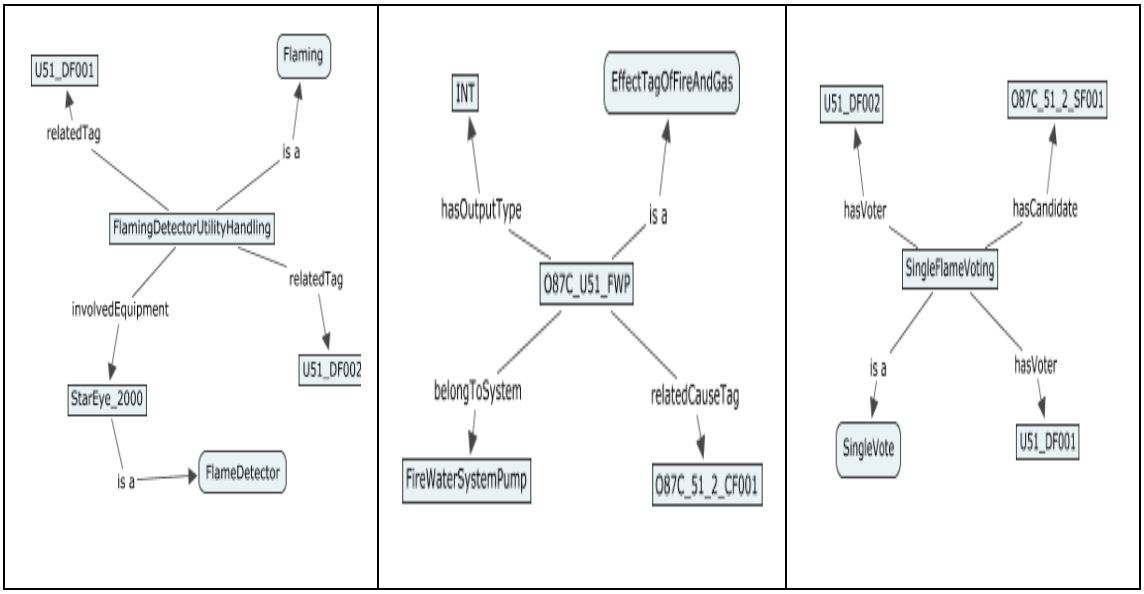


Figure 47 Instances of “Flaming”, “EffectTagOfFireAndGas”, and “SingleVote”

5.1.7 Relate Fire&Gas with ESD

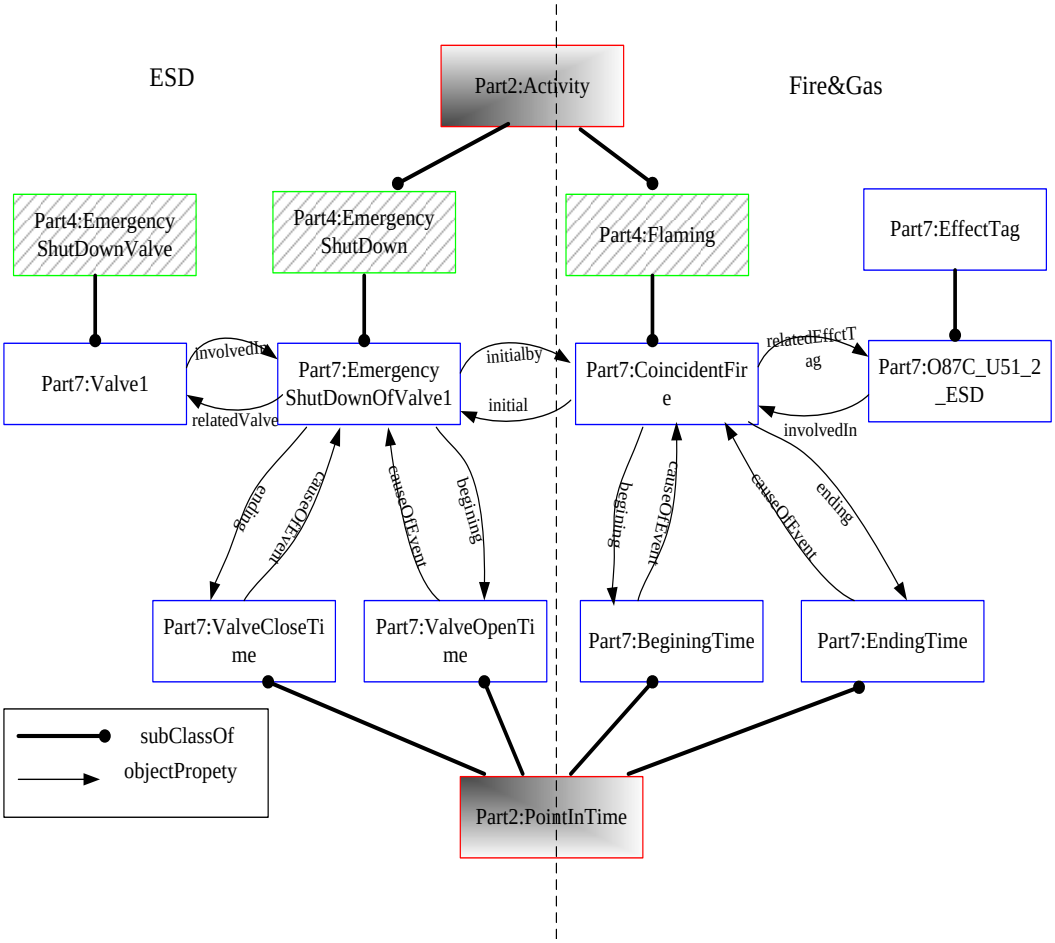


Figure 48 Sample of integration between Fire&Gas and ESD

Figure.48 shows the sample of modeling creation for data integration between Fire&Gas and ESD system. We can get the following information from the definition of the model. The upper ontology class definition and template specification can be found at ISO15926-2 [23] and ISO15926-7 [30].

- "Valve1" involved in activity: "EmergencyShutdownofValve1"
- "EmergencyShutdownofValve1" has beginning time and ending time
- "EmergencyShutdownofValve1" is initialed by activity "CoincidentFire"
- "CoincidentFire" has beginning time and ending time
- "CoincidentFire" related with a "EffectTag" named O87C_U51_2_ESD

Base on this model the use can query the status and historical activity of the "valve1" by getting all the "involedIn" activity. The user of ESD system can get the following information from the Fire&Gas system: what kind of "Flaming" is happing? It happens in which area? When does it happen? When does it end? Which valve is related to that firing? The user of Fire&Gas system also can get the following information from the ESD system: Is the valve related to the

“Flaming”closed? When the flaming is happening? When is the valve closed?

5.2 Mapping the real-time data into data source ontology

As we have manually mapped the Cause&Effect matrix to the ISO15926 standard, the real-time data need to be mapped to the data source ontology as we created above. Therefore, the data source can be accessible for the user through a more intelligent querying. The intelligent querying is queried by the user and reasoned by the data source ontology.

The mapping implementation uses the SQL to query the database, and use Jena API to map the data element to the class in the ontology and create the OWL instance. Figure.49 shows the real time data in relational database. It stores the status of the Tags within a time range. The Value is the status of the tag, “0” means false,”1” means true. The data type of Tag_Name is Varchar, From_Date and To_Date is Datetime, Value is Boolean, and Is_inhibit is Char(1).

Tag_Name	From_Date	To_Date	Value	Is_Inhibit
U51_DF001	2007-05-07 15:44:11	2007-05-07 15:44:17	0	N
U51_DF001	2007-05-07 15:44:17	2007-05-07 15:44:30	1	N
U51_DF002	2007-05-07 15:44:30	2007-05-07 15:44:40	0	N
U51_DF002	2007-05-07 15:44:50	2007-05-07 15:44:55	1	N
U51_DG003	2007-08-06 11:25:55	2007-08-06 11:25:57	1	N
U51_DG004	2007-08-06 11:25:55	2007-08-06 11:25:57	1	N

Figure 49 real time data in relational database

As it introduced in the design specification, the first step is to get the data from the relational database. This step includes: querying of the database, storing the result in the value object “Activity”, and using iterator to put the value object into the list. The next step is getting the ontology from ontology database. This step is implemented by the OntologyModelCreation class. We mainly focus on the mapping implementation in this chapter. As it shows below, for the mapping a new ontology model need to be create at first. The ontology is specified to OWL language by the “OntoModelSpec.*OWL_MEN*”. The namespace prefix need to be specified when the ontology model is initialed.

```
//Create new ontology model
OntModel realtimeData =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
//set prefix mapping
realtimeData.setNsPrefix("part7", NS1);
realtimeData.setNsPrefix("RTD", NS2);
```

The following code shows the mapping method. For each element in the real time database, a new

individual should be created with a unique name in the OWL instance document. The “oc” in the createIndividual method is the ontology class “Activity”. It means the new created individual is the instance of the “Activity”. The real time element is also the instance of the tag in the ontology. So we get the tag “oc1”as an individual, and set the new created individual “activity” to rdf:type of “oc1”. The individual “activity” inherits the property from the super class. To set the property of the individual, it needs to get the property from super class by a full URI address of the property. The full URI address includes namespace prefix and property name. The addProperty method adds the content to the specified property.

```
public void mapping(Activity act){
 // create individual for the class "Activity"
Individual activity=realtimeData.createIndividual
      (NS2+ act.getTagName()+"_"+act.getBeginingTime(),oc);
//set rdf:type of the individual
Individual oc1
=ontModel.getIndividual("http://www.owl-ontologies.com/CauseAndEff
ect1.owl#"+act.getTagName());
 activity.addRDFType(oc1);
 //set property hasBeginingTime for the individual
 Property hasBegining=ontModel.getProperty(NS1+ "hasBeginingTime");
 activity.addProperty(hasBegining, act.getBeginingTime());

//set property hasEndingTime for the individual
Property hasEnding=ontModel.getProperty(NS1+ "hasEndingTime");
 activity.addProperty(hasEnding, act.getEndingTime());

 //set property hasState for the individual
 Property hasState=ontModel.getProperty(NS1+ "hasState");
activity.addProperty(hasState, act.getStatus().toString());
   }
```

The following code is the mapping result. It is the OWL in XML/RDF format.

```
<rdf:RDF
    xmlns:RTD="http://www.owl-ontologies.com/RealTimeData.owl#"
  xmlns:part7="http://www.owl-ontologies.com/CauseAndEffect1.owl#"
  <part7:Activity
rdf:about="http://www.owl-ontologies.com/RealTimeData.owl#U51_DF002_2007-05-07
15:44:50">
    <rdf:type
rdf:resource="http://www.owl-ontologies.com/CauseAndEffect1.owl#U51_DF002"/>
    <part7:hasBeginingTime>2007-05-07 15:44:50</part7:hasBeginingTime>
    <part7:hasEndingTime>2007-05-07 15:44:55</part7:hasEndingTime>
    <part7:hasState>true</part7:hasState>
  </part7:Activity>
  <part7:Activity
rdf:about="http://www.owl-ontologies.com/RealTimeData.owl#U51_DF001_2007-05-07
15:44:17">
    <rdf:type
rdf:resource="http://www.owl-ontologies.com/CauseAndEffect1.owl#U51_DF001"/>
    <part7:hasBeginingTime>2007-05-07 15:44:17</part7:hasBeginingTime>
    <part7:hasEndingTime>2007-05-07 15:44:30</part7:hasEndingTime>
     <part7:hasState>true</part7:hasState>
    </part7:Activity>
</rdf:RDF>
```

6. Prototype implementation

6.1 Semantic reasoning implementation

Jena API is used as programmatic environment, and Pellet is used as reasoner in the semantic reasoning implementation. As it shows in figure.50, the ModelFactory is used to associate a model to a reasoner in order to get a new model. The new model has the inference data that inferred by the Pellet Reasoner. The OntModel API provides methods to find the graph in the InfGraph.

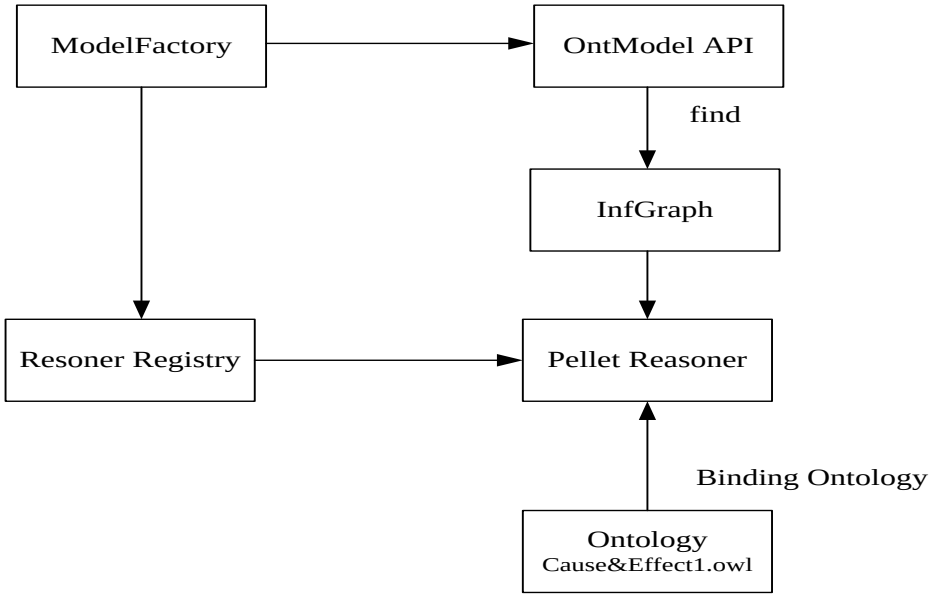


Figure 50 Jena inference methodology, cited from [31]

The following code shows the creation of a Pellet reasoner, associating it to the OntModel. And store the new created model in the infModel.

```
Reasoner pellet = new PelletReasoner();
OntModelSpec spec = new OntModelSpec(OntModelSpec.OWL_MEM);
spec.setReasoner(pellet);
infModel = ModelFactory.createOntologyModel(spec, ontModel
              .getBaseModel());
```

The following code gives an example of reasoning of the state of the tag. The state of the tag is evaluated by checking if this tag is the member of the Active Tag class. There are two kinds of Active Tag class on the model created in this project that are “ActiveCauseTag” and “ActiveEffecTag”. The “contains” method is used here to find if the infModel contains the given axioms. It returns a Boolean value, “True” means it contains the axiom, while “False” means not.

```
 Boolean status1;
 status1 = infModel.contains(individual, RDF.type, ActivecauseTag);
 Boolean status2;
status2 = infModel.contains(individual, RDF.type, ActiveEffectTag);
status =status1||status2;
```

6.2 Semantic query implementation

The semantic query system here is a simple implementation for proof of the theory. It only support querying of instance, class and instance with timestamp. The Sparql is mainly used as query language. Jena API is used for realization of the instance, and manipulating of the ontology model.

As it show in figure.51, when the user query the instance on the user interface, the Jena Reasoning system find out the class, to which the instance belong. And then match the predefined Sparql query method of the class.

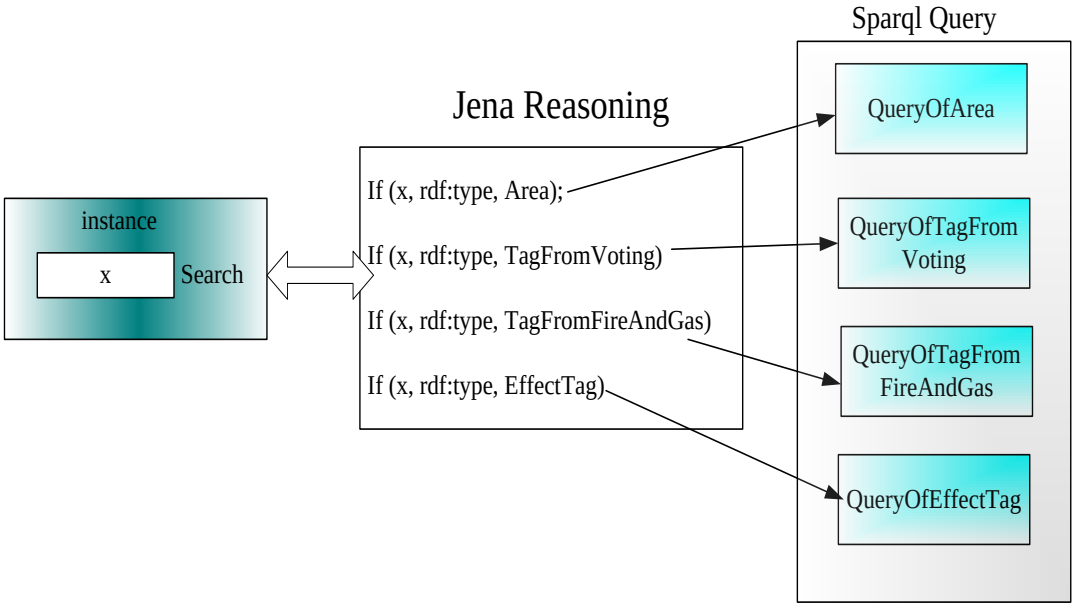


Figure 51 Query of instance

The following code is the Sparql query for the member of “TagFromVoting” class. Given a tag that is an instance of “TagFromVoting”. The query gets the Area of the tag located. And also get the EffectTag related with this tag. To find the area of the tag, it follows the searching route (tag—system—Room—Area).

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX CE: <http://www.owl-ontologies.com/CauseAndEffect1.owl#>

  SELECT  ?area ?effectTag
     WHERE {
           CE:"+causeTag +" CE:belongToSystem  _:System
            _:System CE:systemLocatedAt _:Room
            _:Room CE:locatedAt ?area
           CE:"+causeTag+" CE:causeOfEvent ?effectTag
          }
```

Figure.52 shows the query of the class. It uses the Pellet reasoner to provide inferred model of the given class, and also infer the status of the tags. The query will list all the instance of the given class. And use the method in Jena API to get the description and local name of each instance. If the instance is a kind of tag, it will also give the status of the tag.

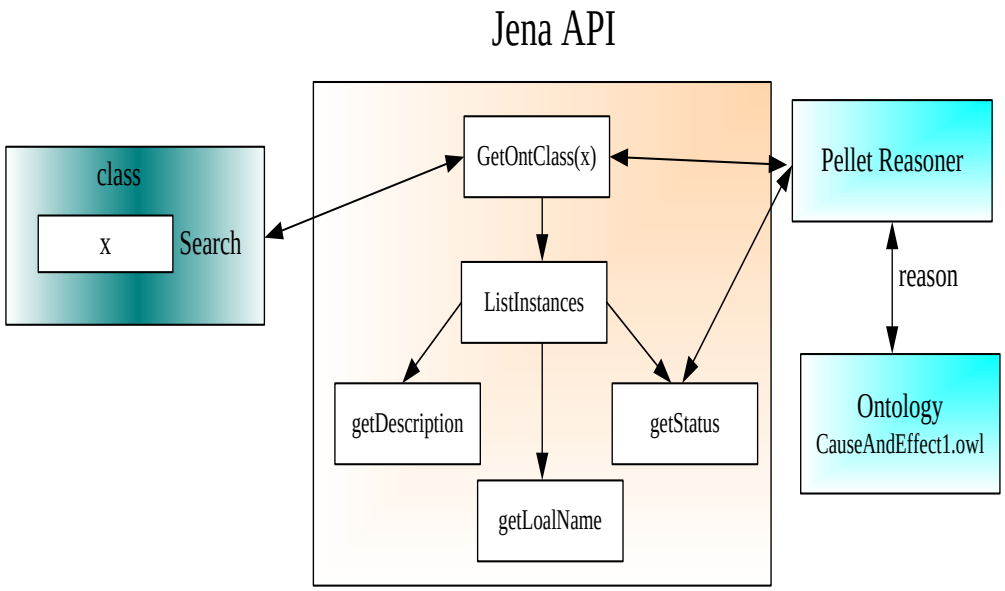


Figure 52 Query of class

7. Proof of concept

7.1 Reasoning verification

As it shows in figure.53 the Protégé provides an embedded reasoning tool for verification and testing of the OWL ontology. In the theoretical background we has introduced that there are three functions of reasoning: check consistency, classification of taxonomy, and realization of instances. In figure.53 we mark the three buttons that related to each function with red circle on the top of the Protégé use interface. The figure.53 shows the model after the reasoning step. The numbers behind the classes are the number of asserted instance and inferred instance. For example, behind the “CauseTag” there is a number “(0/13)”, which “0” is the number of asserted instance of “CauseTag”, “13” is the number of inferred instances of “CauseTag”. The window on the middle of the user interface shows both the asserted and inferred instances of each class. In the following we will verify each aspect of the OWL ontology we created.

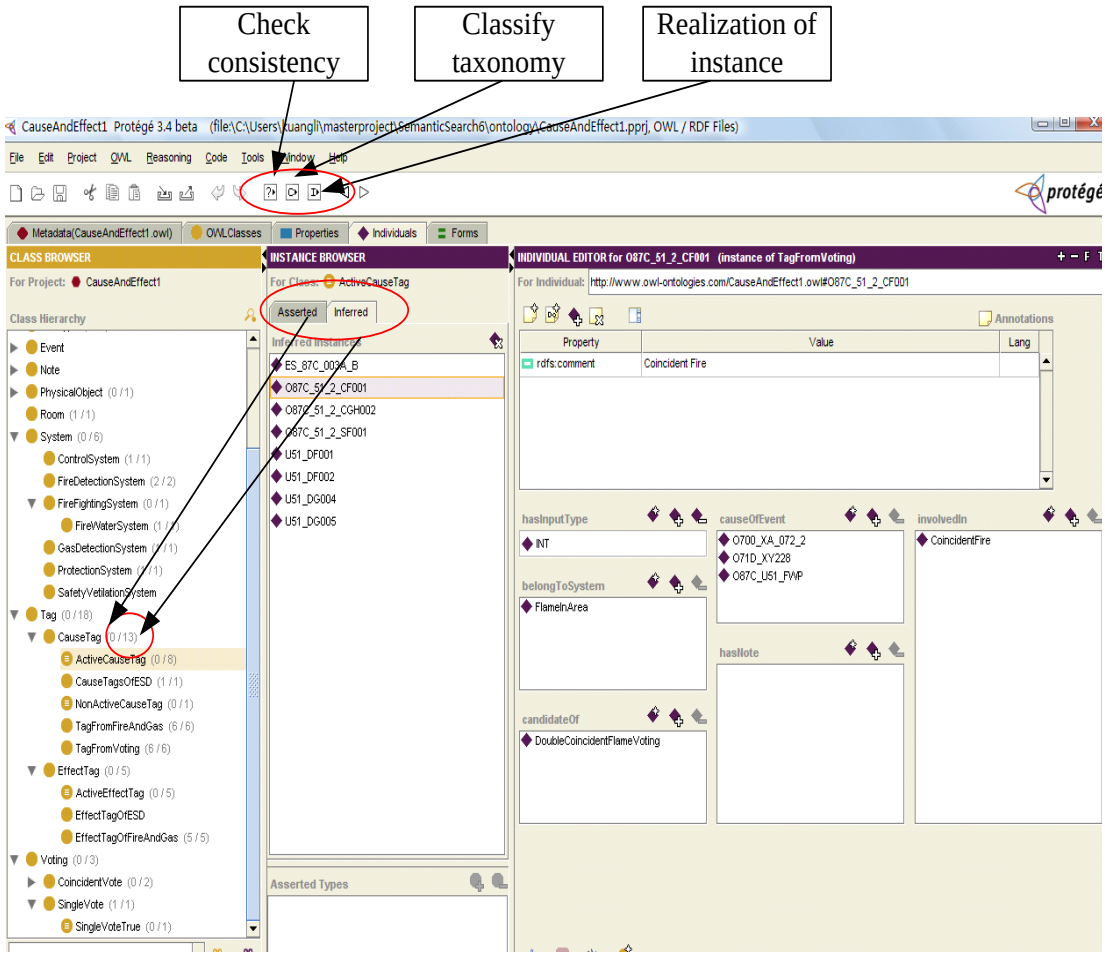


Figure 53 Protégé reasoning

7.1.1 Check consistency

Figure.54 shows the ontology we created for Cause&Effect has been checked without any inconsistency. If there is any consistency definition in the ontology, it will be marked as red color and showed on the screen. In the following paragraph we will show an example of inconsistency.

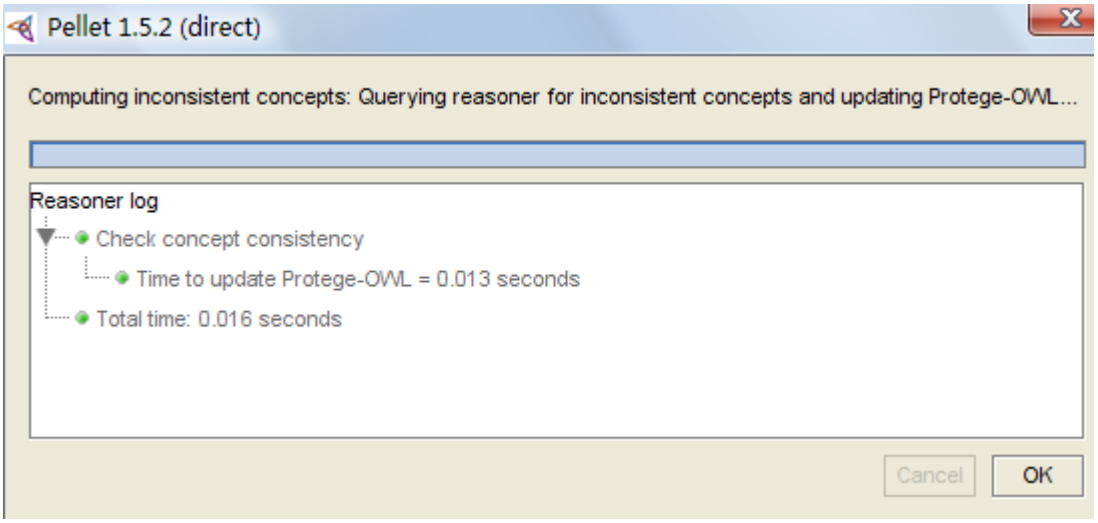


Figure 54 Check consistency

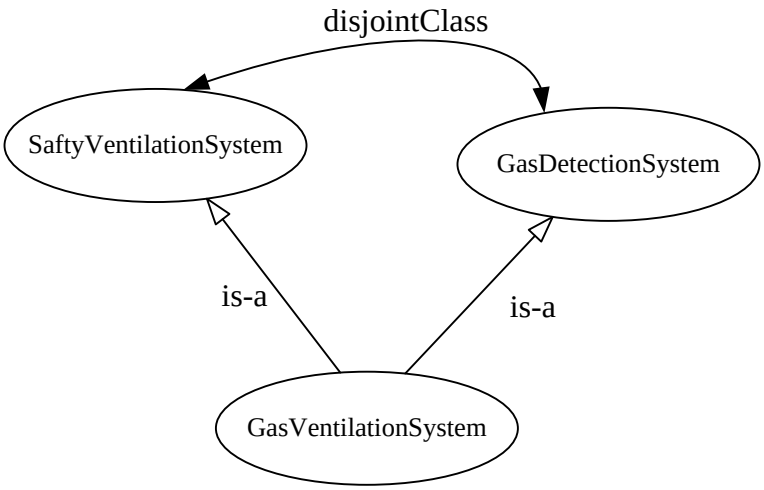


Figure 55 Inconsistency class definition

As it shows on figure.55 we define the “GasVentilationSystem” as subclass of both “SaftyVentilationSystem” and “GasDetectionSystem”. However, the “GasVentilationSystem” and “GasDectectionSystem” are disjoint class. Therefore, the class “GasVentilationSystem” is unsatisfiable. No instance can be realized as the member the “GasVentilationSystem”. This inconsistency can be found out by the Pellet reasoner as it shows in figure.56. To solve this kind of inconsistency, we can either delete the disjoint relationship or delete one of the subclass relationships.

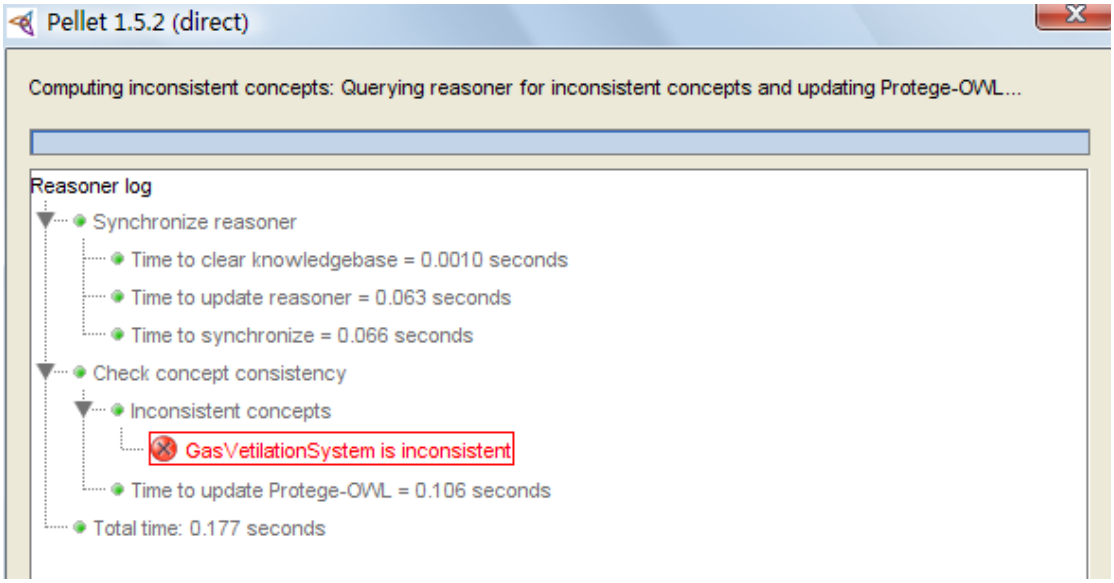


Figure 56 Inconsistency example

7.1.2 Classify taxonomy

The classify taxonomy function usually used to classify undefined class when the knowledge engineer create a new class. The reasoning system helps the engineer to modify the super type of the undefined class automatically. We will show an undefined class example as it shows in figure.57.

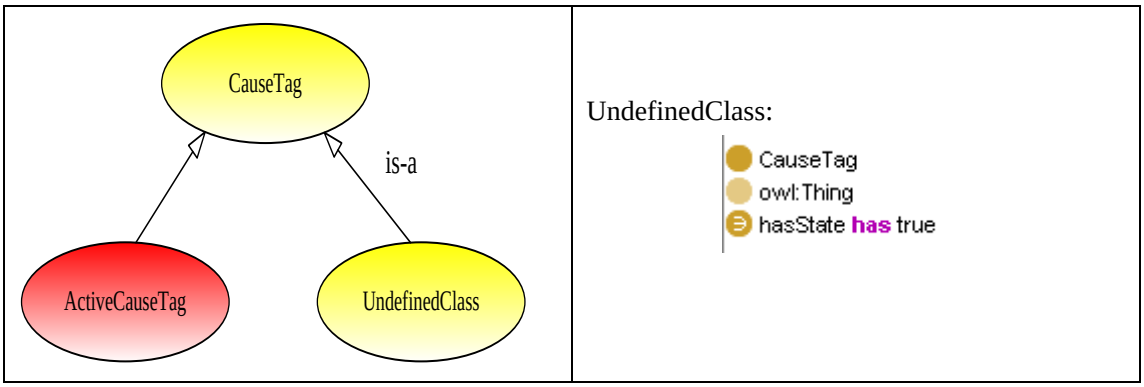


Figure 57 the “UndefinedClass” definition

As the definition of “ActiveCauseTag”, we know that the “CauseTag”, which has state true, is the “ActiveCauseTag”. Therefore, the “UndefineClass” should be subclass of “ActiveCauseTag”. After we push the “classify taxonomy” button, the following message shows out in the changed list. We get the inferred model in figure.58. The result is shows as figure.59.

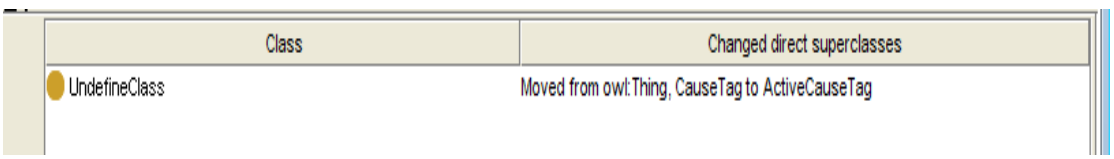


Figure 58 result of the classify taxonomy

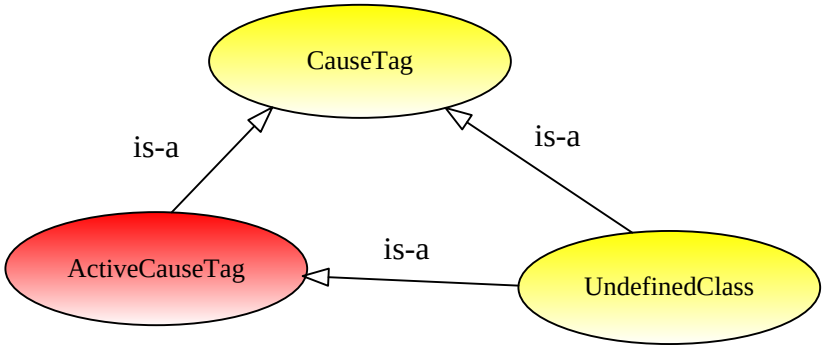


Figure 59 result of the model

7.1.3 Inferring the states of the Tag in Fire&Gas and ESD

This chapter gives an example of inferring the state of the tag automatically without human interaction. As it shows in the scenario, if we assume that some “Activity” happens in the offshore, the sensors will send the real-time data to the Fire&Gas System. The Fire&Gas system will map the real-time automatically to data source ontology. Therefore some tags in the Fire&Gas system related with the activities are setting to be true. Then through a voting system and matching of the Cause&Effect matrix, the machine finds out some “Effect Tag” should set to be true. These “Effect Tag” are related with some actions, such as the “O87_U51_ESD” in figure.60. If “O87_U51_ESD” are set to be true, it will initialize the Emergency Shutdown system automatically.

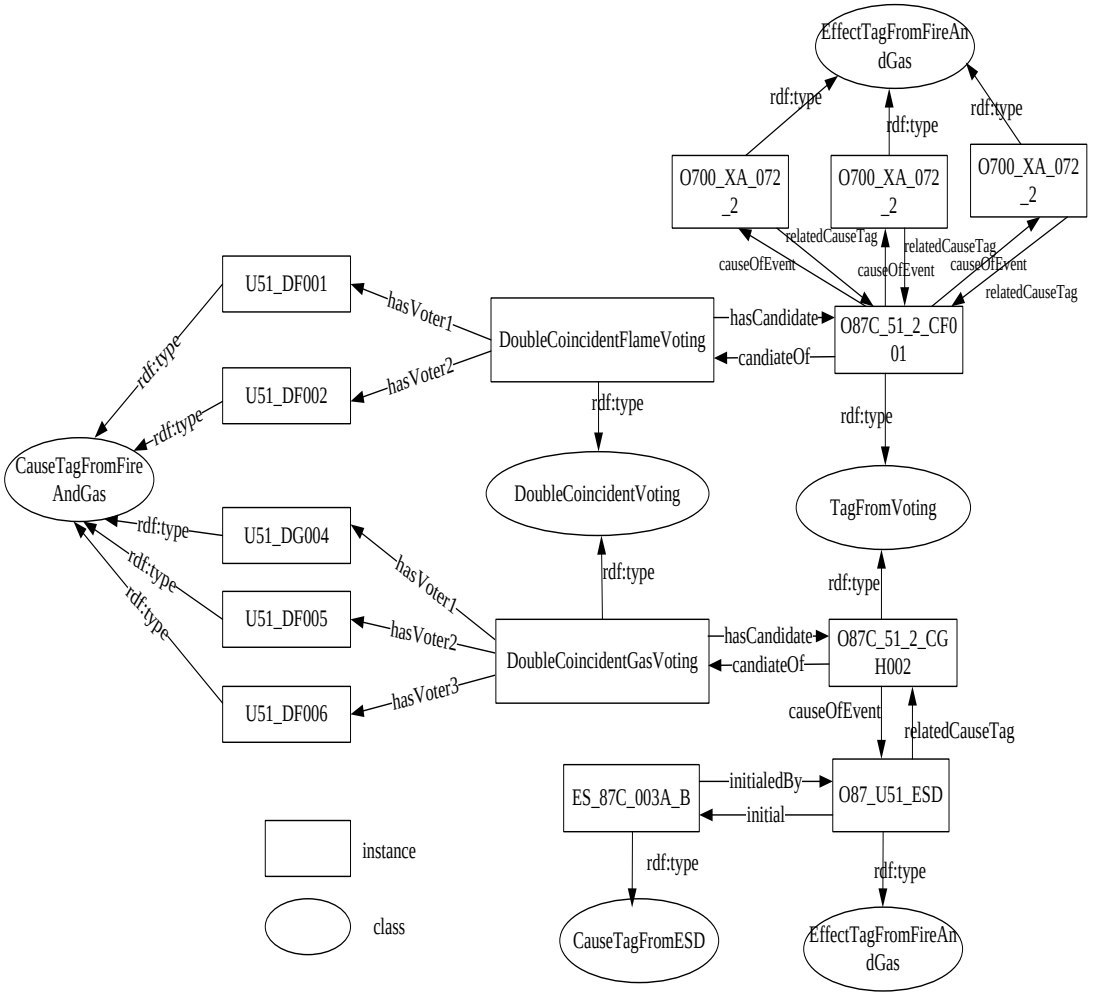


Figure 60 instances definition in the ontology

As shown in Figure.60, the instances relates to each other by the defined object property.

There are two kinds of “DoubleCoincidentVoting” that related with their voters and candidates. The “hasCandidate” and “candidateOf” are inverse properties. The instances of “TagFromVoting” relates with the “EffectTagFrommFireAndGas” by “causeAndEffet” property. The instance of “EffectTagFromFireAndGas” “O87C_51_ESD” initials the “ES_87C_003A_B”, which is the “CauseTagFromESD”.

If we set the following datatype property:

(U51_DF001, hasState, true)

(U51_DF002, hasState, true)

(U51_DG004, hasState, true)

(U51_DG005, hasState, true)

Then all the tags above will be inferred as the member of “ActiveCauseTag” as it show in figure.61.

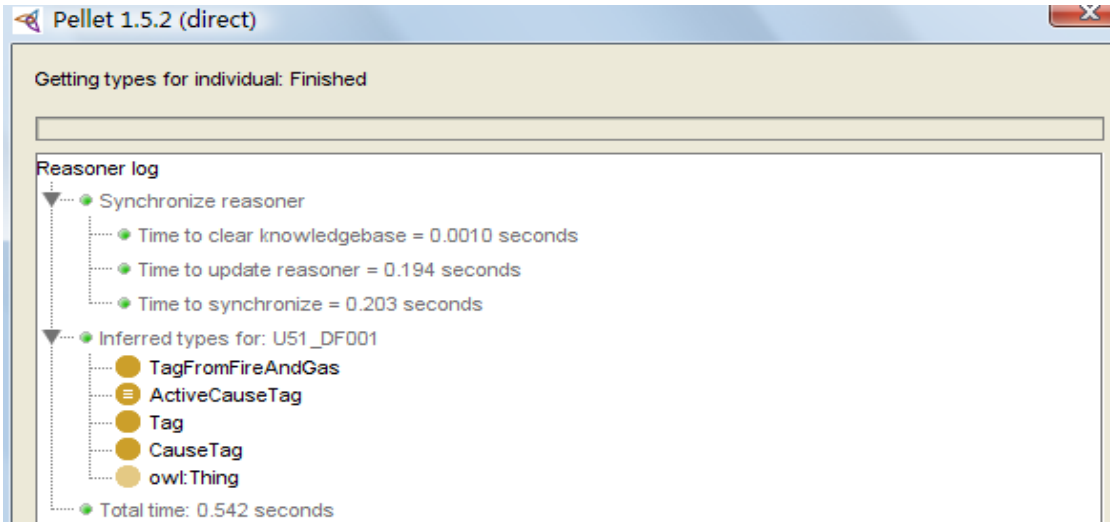


Figure 61 Compute type of U51_DF001

According to the definition of the "DoubleCoincidentVoting", if two of the voters are "ActiveCauseTag", the voting will be inferred to be true. Therefore, the "DoculeCoincidentGasVoting" and "DoculeCoincidentFlameVoting" will be inferred as the member of "DoubleCoiincidentVoteTrue"

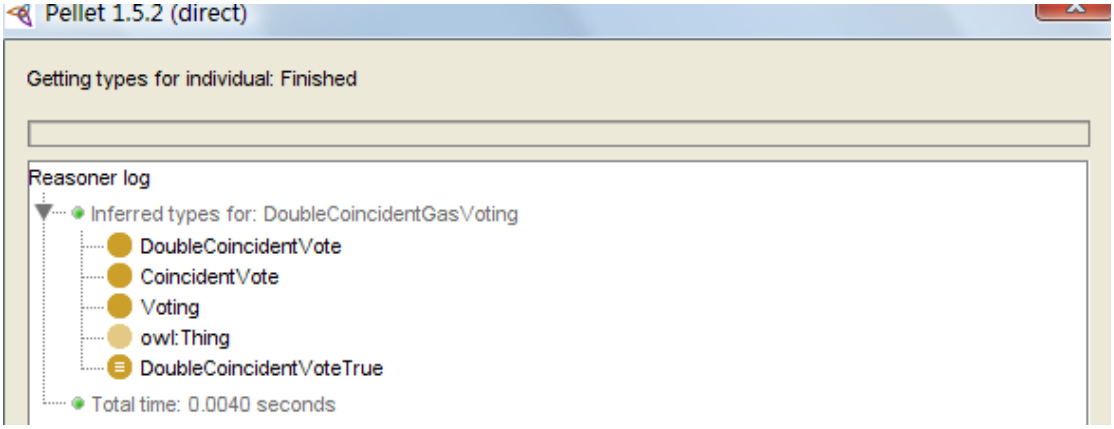


Figure 62 Compute type of "DoculeCoincidentGasVoting"

According to the definition of the "CauseTagFromVoting", the "CauseTagFromVoting" which is the candidate of a true voting will inferred to be "ActiveCauseTag". Therefore, "O87C_51_2_CGH002" and "O87C_51_2_CF001" should be inferred as the "ActiveCauseTag" as shown in Figure.63.

Figure 63 Compute type of “087C_51_2_CGH002”

Similar to the theory above, all the member of “EffectTagFromFireAndGas” will also inferred to “ActiveEffectTag”. Therefore, as the definition the “CauseTagFromESD”: If the tag, which initials the “CauseTagFromESD”, is true, than the “CauseTagFromESD” should set to be true automatically. As a consequence the Emergency shutdown could be done by the compute automatically without human interaction. As shown in Figure.64, the “ES_87C_003A_B” is inferred to be the member of “ActiveCauseTag”.

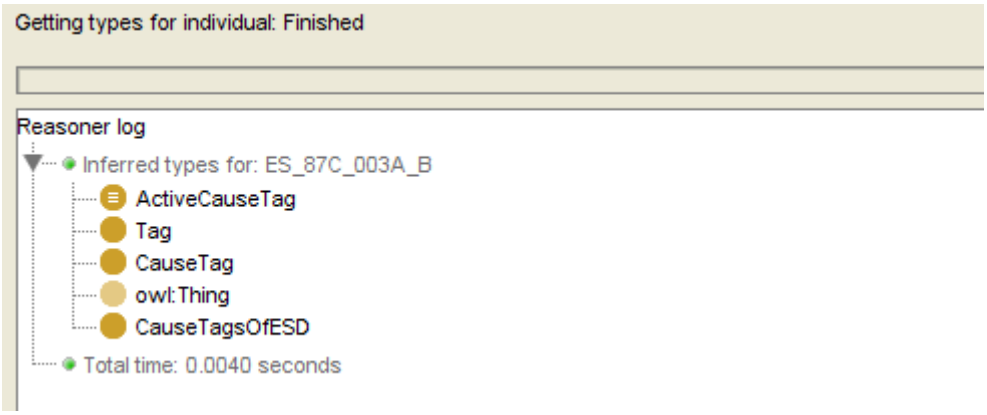


Figure 64 Compute type of “ES_87C_003A_B”

7.2 Testing of the semantic reasoning and querying system

The figure.65 shows the user interface of semantic reasoning and querying system. As you can see there are three types of things that are supported for querying: instance, class, and instance+ timestamp. Due to the time limit the query of instance+ timestamp is not fully implemented. Therefore, here we will test the query of instance and class. The user needs to select the search type and input the keyword in the text area. If not, the user interface will show error message.

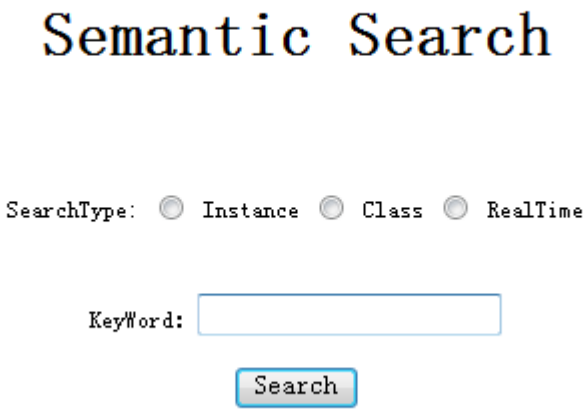


Figure 65 Semantic Search user interface

(1) Query of instance

- Given an area (“U51-2”) what are the causes and effects (i.e. Complete sheet). In this way, the external users of the system could be able to get the information they need without a domain expert.

Area	U51_2	Chart :CauseAndEffectChart_U51_2
CauseTag :U51_DG005 EffectTag :0700_XA_071_2		

Area	U51_2	Chart :CauseAndEffectChart_U51_2
CauseTag :087C_51_2_CF001 EffectTag :0700_XA_071_2		

Area	U51_2	Chart :CauseAndEffectChart_U51_2
CauseTag :087C_51_2_SF001 EffectTag :0700_XA_071_2		

Area	U51_2	Chart :CauseAndEffectChart_U51_2
CauseTag :U51_DG006 EffectTag :0700_XA_071_2		

Area	U51_2	Chart :CauseAndEffectChart_U51_2
CauseTag :U51_DG004 EffectTag :0700_XA_071_2		

Area	U51_2	Chart :CauseAndEffectChart_U51_2

Figure 66 Query result of “U51_2”

- Given a cause (tag) from Fire&Gas what are the related candidate tag of voting (tagnames+areas). This query illustrates the accessible of the data. Therefore, the internal software develop could manipulate the data source to get better function of the system.

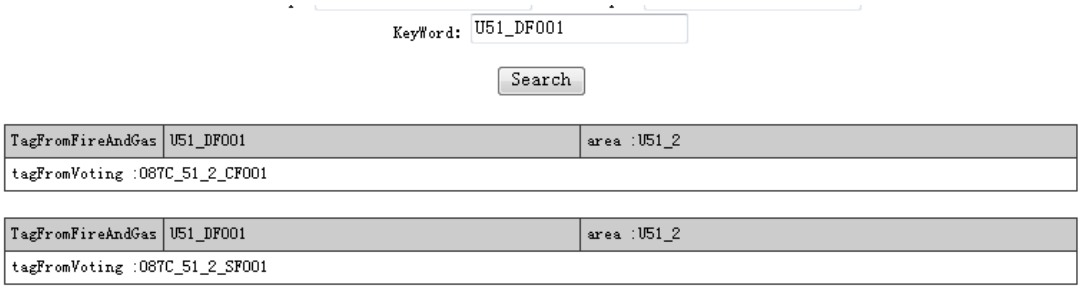


Figure 67 Query result of “U51_DF001”

- Given a cause (tag) from voting what are the possible effects (tagnames + areas)

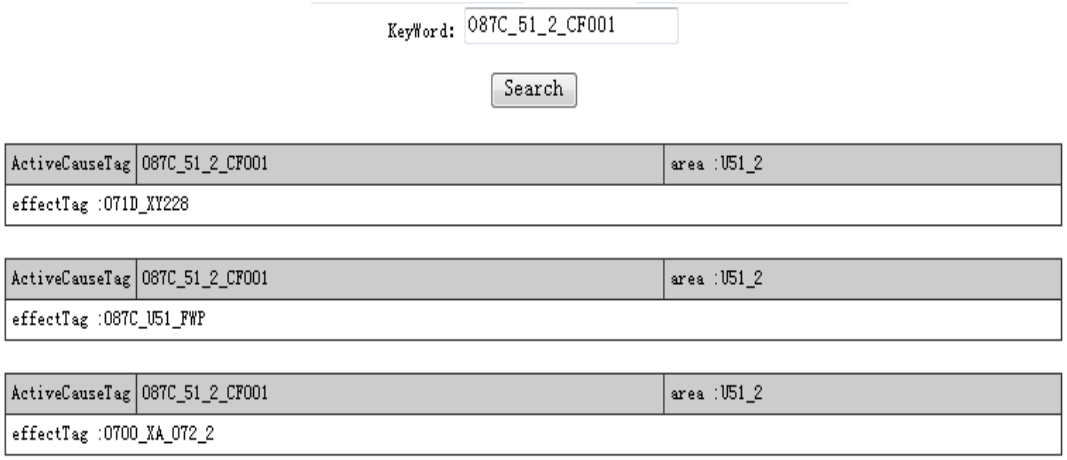


Figure 68 Query result of “O87C_51_2_CF001”

- Given an effect (tag) what are the possible Action. As it presents in the scenarios. For example, if you search the “O87C_51_2_ESD” tag, which should initial the Emergency shutdown system, it will shows out the related ESD tag and its states at some points of time. In this way, the safety person of Fire&Gas could able to verify that if the Emergency shutdown really works as it should be.

Figure 69 Query result of “O87C_51_2_ESD”

(2) Query of class

Some state of the tags is defined as it shows in chapter7.1.3. Assumes the tag “U51_DF001”, “U51_DF002”, “U51_DG004”, and “U51_DG005” are set to be true. If you query the class “Tag” in the query system the result will be show as figure.70. From this query, the internal safety person and external safety person could able to check the state all the tags at the current time. The Tag description will give the detail of what happening actually.

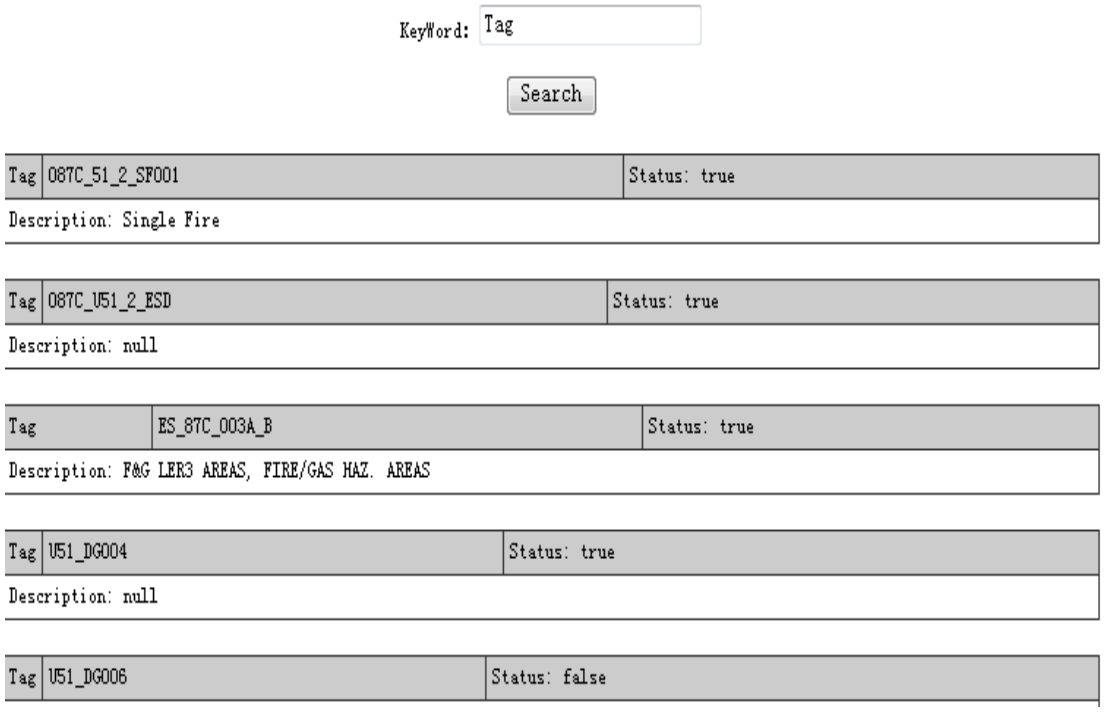


Figure 70 Query result of the “Tag” class

8. Discussion

In this chapter we would like to discuss: Is it possible to map from data source ontology to domain ontology automatically? As it is described in [8] conceptual layering of ontologies can be divided into four layers: data sources, data source ontologies, domain ontology, and view. To do the data integration, two steps are needed. First, implement mapping from data sources to data source ontology. Usually, this mapping is generated automatically by introducing some mapping pattern. Second, implement mapping from data source ontology to business ontology. This step is much more complex than the first step, automatic mapping would be difficult. However, automatic mapping is the research goal at current stage of semantic data integration.

In this project we have tried an approach as shown in Figure.71, a Cause&Effect matrix represents the data source ontology, the Oil&Gas ontology is the business ontology, and the real time data is the data sources. This approach attempts to use the tool JXML2OWL, which is a mapping tool to lift the XML to OWL, accomplish the mapping automatically. As we see in figure 71, by defining the mapping from Cause&Effect XML schemas to Oil&Gas ontology OWL, the Cause&Effect XML instance can transform to Oil&Gas ontology automatically. Similarly, by defining the mapping from Real time data XML schemas to Oil&Gas ontology OWL, the real time data XML instance can transform to real time data OWL instance automatically. In the beginning the Real time data XML instance is based on the Cause&Effect matrix. After the transformation the real time data OWL instance is based on the Oil&Gas Ontology OWL instance. The details of each mapping steps will be described in the following paragraph.

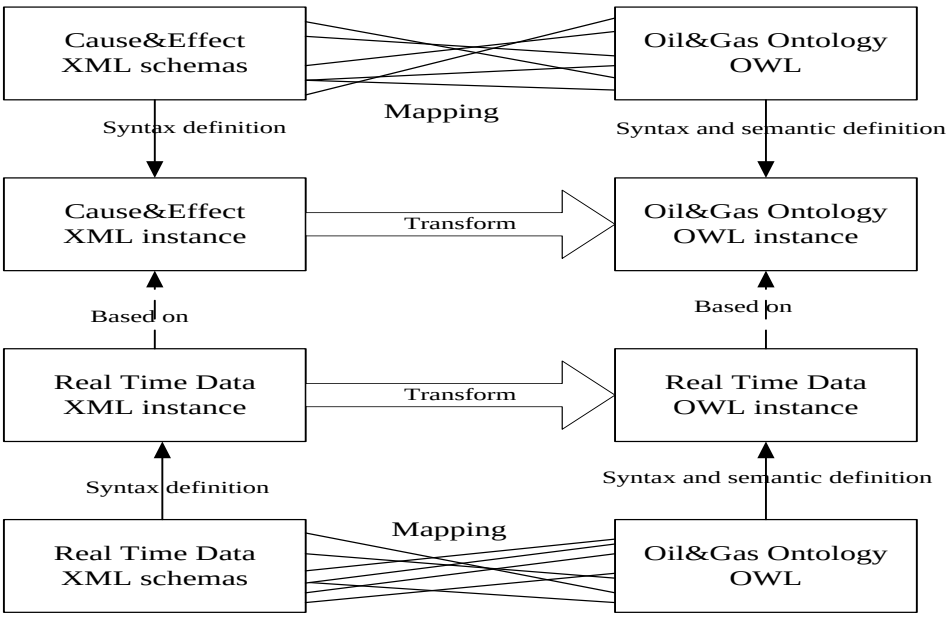


Figure 71 Automatic mapping structure

The figure.72 shows the graphical user interface of JXML2OWL Mapper. This tool enables graphically creation and modification of the mapping from XML schemas to OWL ontology utilizing the JXML2OWL API. The left side is the Cause&Effect XML schemas represented in a

tree view. On the right hand side is the Oil&Gas ontology OWL. The mapping zone is in the middle. Under the mapping zone, there are mapping links between classes and XML item XPath. And also the object and datatype mapping links is in the bottom.

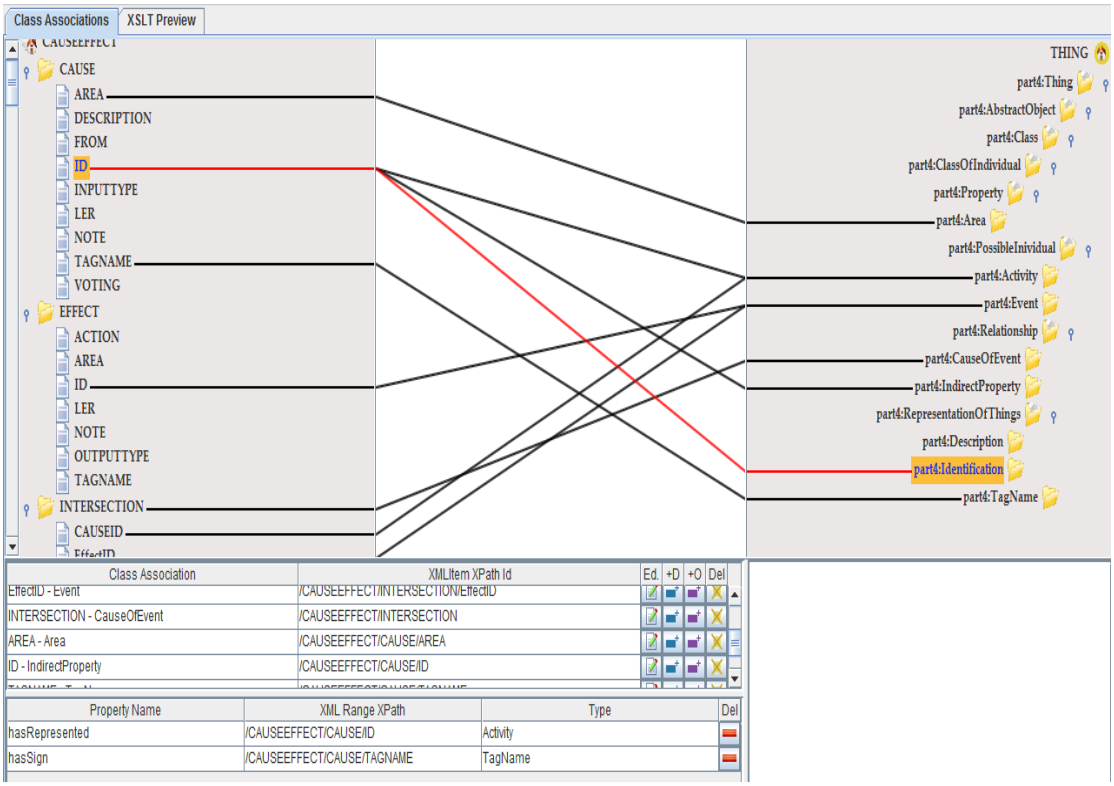


Figure 72 Mapping from Cause&Effect schemas to Oil&Gas ontology in OWL

We have got some results from this approach. Unfortunately, the result is not exactly what we need in this project. This approach is trying to retrieve semantic information from the existing XML database. It is useful when you are trying to implement a semantic query system within the homogeneous data sources. On the condition, which data integration within heterogeneous data sources, this approach has too many limitations. It simply matches the XML tree view to the OWL graph view by mapping the objectProperty and datatypeProperty of the classes. It is not possible to add some restrictions in the newly created OWL file. It's much better to manually map the local ontology to domain ontology. However, in mapping from data source to data source ontology, the JXML2OWL works well.

9. Conclusion and future work

9.1 Conclusion

As the leading industry of Norway, the Oil&Gas industry has made great effort within Information and Communication Technology. The efficiency and environment protection are the main contributions of the technology. The Safety Instrument System (SIS) is used for preventing accidents. It could save money and reduce pollution if the SIS works efficiently, that requires that we get information from the processing of the real-time data as soon as possible. The Semantic Web enables the knowledge representation of the data source, which supports reasoning and semantic querying. Therefore, the machine could understand the information. The data could be processed by the machine automatically, which can greatly improve the efficiency. Moreover, The SIS contains some subsystems. The subsystems need to work together to prevent accident. Thus, the data integration is necessary. Origo has implemented a prototype for sharing information between SIS subsystems. The prototype achieves data integration by sharing the same XML schemas, which cannot fulfill the large scale and across domain data integration. Hence, the Oil&Gas ontology is introduced to solve the problem. By applying the Oil&Gas ontology based data integration, the company could get the following advantages: *"improved data quality and accessibility*, *significant cost reduction with change of software*, *increased flexibility with organizational changes*, and *improved software functionality" [3]*

In this project, we analyzed the principles of XML, RDF, OWL, ontology, reasoning, description logic and Sparql querying of the Semantic Web technology, and ISO15926 part2 data model, part4 reference data, and part7 implementation methodology. Based on this analysis we introduced a framework of data integration. We implemented the manual mapping from Cause&Effect matrix to ISO15926 standard, and automatic mapping from real-time data to data source ontology. We also implemented a prototype of semantic querying and reasoning system. This prototype proves the concept of reasoning, and shows the improvement of data quality and accessibility by querying the information. Through all the works above, we would like draw a conclusion that the ISO15926 standard and Semantic Web are suitable to use in the SIS system. They work together could great improve the current system.

9.2 Future work

The prototype in this project is developed for proof of concept and for showing the advantages of data integration. It is not very suitable for practical use. The following work can be done to improve the query system:

- Use the Semantic Annotation for all the classes defined in the ontology, and Lucene as the query engine. Lucene supports the full text search. Jena API has developed a plug-in named

ARQ that support Lucene. Therefore, we could query by text match of the annotations, rather than only by class name and instance name.

- Complete the implementation of search through timestamps, so that the safety person could be able to get the information from offshore of any time or time range.
- Improve the search pattern for more general use. The prototype can only search the information based on predefined pattern. It is not flexible enough for different kinds of search and not extensible when the system scales up.

Reference:

[1] THREE DECADES OF DATA INTEGRATION—ALL PROBLEMS SOLVED?, Patrick Ziegler and Klaus R. Dittrich

[2] Dittrich, Klaus R. and Jonscher, Dirk (1999). All Together Now — Towards Integrating the World's Information Systems. In Masunaga, Yoshifumi and Spaccapietra, Stefano, editors, Advances in Multimedia and Databases for the New Century, pages 109–123, Kyoto, Japan, November 30 – December 2. World Scientific Press.

[3] Kari Anne Haaland Thorsen and Chunming Rong, Data Integration in Oil and Gas at Norwegian Continental Shelf, 22nd International Conference on Advanced Information Networking and Applications

[4] Guan-yu LI, Wei-li ZHANG, Huan-zhong GENG, Ontology-Based Web Data Integration Architecture Modeling and Implementation, 2007 IFIP International Conference on Network and Parallel Computing

[5] Li dong, Huang linpeng, A Framework For Ontology-based Data Integration, 2008 International Conference on Internet Computing in Science and Engineering

[6] Integrated Operation: Methodology and tools for management and control of Safety instrumented Systems, Internal document of ORIGO Engineering

[7] NORSOK STANDARD I-002, Rev.2, 2001-05-01, Safety and automation system (SAS)

[8] Jürgen Angele, Michael Gesmann. Data Integration using Semantic Technology: A use case. Proceedings of the Second International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML'06), 2006

[9] Toni Rodrigues, Pedro Rosa, Jorge Cardoso, MAPPING XML TO EXISTING OWL ONTOLOGIES

[10] Hannes Bohring* and S ¨oren Auer, Mapping XML to OWL Ontologies

[11] Tao Huang, Qingtang Liu, Sanya Liu, Shengming Wang, Yong Yang,Design and Implementation of Semantic Query System based on Ontology Context, 978-1-4244-2108-4/08 ©2008 IEEE

[12] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe. A Practical Guide to Building OWL Ontologies Using the Protege-OWL Plugin and CO-ODE Tools Edition 1.0

[13] OWL Web Ontology Language Reference W3C Recommendation 10 February 2004 [cited 2008 13. January]; Available from: http://www.w3.org/TR/owl-ref/

[14] Huhns, Michael N. and Singh, Munindar P. (1997). Agents on the Web: Ontologies for Agents.IEEE Internet Computing, 1(6):81–83.

[15] Kirwin, Christopher. 1995. 'Reasoning'. In Ted Honderich (ed.), *The Oxford Companion to Philosophy*. Oxford: Oxford University Press: p. 748

[16] Markus Krötzsch (AIFB Karlsruhe), Practical Reasoning with OWL and Rules, Half-day tutorial at the 3rd European Semantic Web Conference, ESWC 2006.

[17] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, Yarden Katz, Pellet: A Practical OWL-DL Reasoner, Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 5, No. 2. (June 2007), pp. 51-53.

[18] Alessandro Artale, Enrico Franconi, Introducing Temporal Description Logics, Temporal Representation and Reasoning, 1999. TIME-99. Proceedings. Sixth International Workshop
[19] Daniele Nardi, Ronald J. Brachman, An Introduction to Description Logics, Cambridge University Press
[20] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007.
[21] D. Leal , ISO 15926 "Life Cycle Data for Process Plant": An Overview, Oil & Gas Science and Technology – Rev. IFP, Vol. 60 (2005), No. 4, pp. 629-637
[22] ISO 15926-1 (2003) Overview and Fundamental Principles. Industrial Automation Systems and Integration - Oil&Gas, Part 1.
[23] ISO 15926-2 (2003) Data model. Industrial Automation Systems and Integration - Oil&Gas, Part 2.
[24] POSC Caesar. [cited 2009 13. March]; Available from: http://www.posccaesar.org/
[25] SPARQL Query Language for RDF. [cited 2009 13. March]; Available from: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/
[26] An Upper Ontology based on ISO 15926, Rafael Batres a *, Matthew Westb, David Lealc, David Priced, Yuji Nakaa
[27] EPISTLE, Onno Paap 2008 . [cited 2009 13. March]; Available from: http://www.infowebml.ws/
[28] Semantic Web Technologies Trends and Research in Ontology-based Systems. John Davies, Rudi Studer, Paul Warren.
[29] Zhisheng Huang, Frank van Harmelen and Annette ten Teije, Reasoning With Inconsistent Ontologies: Framework, Prototype and Experiment, 19th Joint Conference on Artificial Intelligence (IJCAI'05), 2005
[30] ISO 15926-4:2007 spreadsheets [cited 2009 1. May]; Available from: http://rds.posccaesar.org/2008/05/XML/ISO-15926-4_2007/
[31] Jena 2 Inference support [cited 2009 3. May]; Available from: http://jena.sourceforge.net/inference/
[32] Guan-yu LI, Sui-ming YU, Sha-sha DAI, Ontology-Based Query System Design and Implementation, 2007 IFIP International Conference on Network and Parallel Computing – Workshops
[33] Jena – A Semantic Web Framework for Java [cited 2009 3. May]; Available from: http://jena.sourceforge.net
[34] Renamed Abox and Concept Expression Reasoner [cited 2009 12. May] http://www.sts.tu-harburg.de/~r.f.moeller/racer/