UNIVERSITY OF OSLO
Department of Informatics

Semantic technology in the oil & gas drilling domain

Master thesis

Lars Overå
[larsove@ifi.uio.no]

June 8, 2010

Abstract

Data integration and knowledge representation in the oil and gas drilling domain are two challenges much work is focused upon. They are important real-world challenges to deal with, and the drilling domain has much to gain from better solutions than the ones that exist today. Data integration is a problem that has been known for a long time, but the existing solutions are cumbersome and expensive to use and maintain. Ontology based data integration is one approach that shows much promise and is currently gaining ground. This thesis presents both necessary theoretical background, and also domain knowledge that serves as important input. With the insight gathered from this background, an ontology for the drilling domain is created, and two use cases that are based on this ontology are presented. One of these cases is a general description of a data integration case using the QuOnto framework, and the other is a XML to RDF data conversion tool created in Java.

Acknowledgements

This thesis has from the start been dependant upon many people who have given of their time and expertise to help me complete this work. The domain knowledge that I would not have had any chance to find out on my own has been especially helpful. First I would like to thank Jens Ingvald Ornæs and Robert Ewald at NOV (National Oilwell Varco), as well as Henning Jansen previously at NOV, for valuable insight both in WITSML and in drilling operations in general. A special thanks to Jens Ingvald for helping me test "Statements" as a way of extracting domain knowledge. Also I would like to thank Nejm Saadallah at IRIS (International Research Institute of Stavanger) for much help in WITSML related issues. Then I would like to thank Inge Svensson at Baker Hughes for contributing in creation of the drilling ontology. Secondly I would like to thank David Norheim at Computas for providing insight into semantic technology. I would also like to thank Johan Klüwer at DNV (Det Norske Veritas) for help in several ontology related matters. Last I thank my adviser Arild Waaler (professor, UiO/IfI) for much valuable input throughout the entire process, and Audun Stolpe at UiO/IfI for theoretical help and general input.

Contents

Figures

Chapter 1

Introduction / problem description

1.1 Ontologies and Data Integration

This thesis will deal mostly with knowledge representation and especially data integration in oil and gas drilling operations, and present attempts at solutions and use-cases relevant for this domain. Data integration is a problem that has been known for a long time. It is an important real-world challenge to deal with, as there are great benefits to successful integration of data across computer systems, platforms and even entire organizations. Using ontologies for this purpose is a new approach, and data integration is seen as one of the fields where ontologies can be best utilized for increased efficiency.

Ontology languages in general are relatively new, and have not been applied in any great extent to real-world problems. Many of the large ontologies that exist are medical ontologies still with an academic focus. They are as such not coined specifically at solving real problems, and do not suit data integration particularly well. For ontologies to be best applied to this problem, they ought to be designed with this specific purpose in mind from the start. The thought is that ontologies will not solve any problems that have not yet been solvable, but rather handle existing problems in a more efficient and cheaper way. The robustness of ontologies is one of the properties that hopefully will prove them to be well suited for particular tasks such as data integration. The fact that maintenance today is a very large part of the total system cost, an ontology based system with less and easier maintenance should by itself prove beneficial. As ontologies are quite new and still an academic endeavor, it is not much used in the industry yet. This means that an important part of research in the field is to focus on real-world domains and problems to explore and illustrate just what the strengths of ontologies are. This thesis hopes to do exactly this with focus on the Oil and Gas drilling domain.

1.2 Oil & Gas use cases

Integrated Operations

These days, one of the things that many people in the oil and gas industry at least in Norway talk about is Integrated Operations (IO). The reason for the great interest in this two-letter abbreviated concept is the drive for more efficient and better retrieval of hydrocarbons, as well as a more streamlined overall flow of resources (including people) within the businesses, which again leads to increased efficiency through better work processes and faster handling of problems that arise. And problems are bound to occur in such a complex and difficult operation as the retrieval of hydrocarbons is. Quick and correct measures for any situation is paramount towards the goal of increased overall efficiency. This of course leads directly to increased profits which is important to any industry, although those aspects will not be focused upon in this thesis.

These points are some of the things that many people in the oil and gas industry strive to improve, but what exactly is meant by Integrated Operations and where does it fit in? The Norwegian Oil Industry Association (OLF) defines Integrated Operations as “real time data onshore from offshore fields and new integrated work processes”[1]. There are however other definitions of IO. Statoil for instance is mostly focused on moving people and resource from offshore to onshore, thus eventually having a bare minimum of people and equipment offshore. This is a narrower aim than what OLF envisions, but many of the challenges are the same. OLF focuses on two quite wide concepts in their definition: data and work processes. These two things are certainly not only relevant for the oil and gas industry, but for almost any industry for sure. Data can be anything from measured sensor values or calculated values, to information about employees and their affiliations. In between there can surely be data on just about anything, and these data sets need to be handled and stored somehow in computer systems. On top of this comes the work processes which mostly deal with people and how they solve tasks, but computer systems and interaction with them is also an important factor.

There is certainly room for improvement still in any industry in handling

[1]http://www.olf.no/getfile.php/zKonvertert/www.olf.no/Rapporter/Dokumenter/070919%20IO%20and%20Ontology%20-%20Brosjyre.pdf

data and work processes. OLF represents the oil and gas industry in Norway and made their definition based on this industry's needs, but many of the results will surely be relevant for other businesses as well. However this is not an easy task to accomplish, and considerations at different levels with varying focus is necessary. Both the handling of data and the work processes are largely about the same thing: to be able to make better decisions. There are two sides to this.

1. Better access to experts. It is mostly through good work processes that this can be accomplished. Having the right person at the right time in the right place can mean the difference between quickly solving a problem and struggling with it for hours or days.

2. Better bases for decisions. The quality of the data as well as the way it is presented are both important factors, and have significant impact of the quality of decisions. Even the most able experts will not be able to reach good decisions if information available to them is poor.

This thesis will for the most part focus on the basis for decisions, meaning the data aspect mainly concerning data structuring and data integration. I will work on providing better access to and analysis of data, as well as better quality of the data itself. How this can be realized through work processes will not be considered.

How then does "real time data onshore from offshore fields" relate to the goals of more efficient and better retrieval of hydrocarbons, as well as a more streamlined overall flow of resources? Huge amounts of data are generated both while drilling and while in production on a field. Handling this data is an important part of securing an efficient operation. Among the data are many indicators of problems and keys to solving them as well. Downhole measurements can for instance give indication of an impending blowout, which can lead to dangerous situations if not handled properly quickly enough. A major challenge in this respect is making sure the right data reach the right specialist that can analyze and decide what action to take in any given situation. Traditionally, most such decisions were taken offshore by the people on the platform while perhaps consulting experts on land. This did not happen nearly quick enough though. Today, service companies have better solutions with live video conferences every day where onshore experts talk about operations with the offshore personnel every day. This is still just the first step towards a fully integrated approach to oil and gas operations. With talk of fully automated offshore operations and all the personnel is sitting onshore, the flow of data becomes extremely important. Getting relevant data to the correct person quickly is paramount to quick decision-making when time is critical.

But Integrated Operation also visions a future where many decisions are made automatically by smart computer systems. This is obviously the quickest way to respond to time-critical situations, but it also places great requirements on the computer systems that must process the data and reach a decision. In this regard we can state two loose criteria on the data representation structure.

1. The structure of the data must be rich enough to contain relationships between different entities that humans take for granted when analyzing data. This means that implicit knowledge and context that a human domain expert has must be formalized and made explicit in the data structure. An example of this can be something as simple as the manner in which the steering wheel of a car influences the direction the car moves. Our computer system can describe a simple relationship such as "The car moves in the direction the steering wheel is turned", and this will work in very simple cases where nothing can go wrong. But as soon as the steering wheel is turned and the car does not alter direction, something is obviously wrong. We humans the know that most likely a connection has broken somewhere between the steering wheels and the tires, but for a computer system that only has knowledge about steering wheels and car directions this makes no sense. So we must here expand the structure to include the parts that can be broken and disrupt normal operation.

2. As such the precision of these structures must also be good enough so that there is no doubt what the meaning of the relationships are. If we are to expand the simple steering wheel and direction structure, it must be done in such a way that there is no doubt what the new relationships mean.

As we will see, this thesis will look at ways of structuring data such that it contains as much meaning as possible and relates to other data and entities in the domain. The actual decision-making will not be dealt with.

The main focus of this thesis is on data integration. A classic problem that arises in large businesses is that inter-department communication is lacking and they create their own data systems with their own representation models of data and information. These models may often describe more or less the same domain, but still be incompatible. This is clearly a problem when someone wants to gather information from different systems that use different models. A large manual job of aligning the data must often be done, and this is costly and time consuming. On a larger scale this problem only gets worse when dealing with different companies with different areas of interest while working in the same domain. Huge data integration efforts are necessary to get computer systems to communicate with each other. This problem can of course in principle be avoided by standardization, but things aren't always that simple as will be explained further when discussing

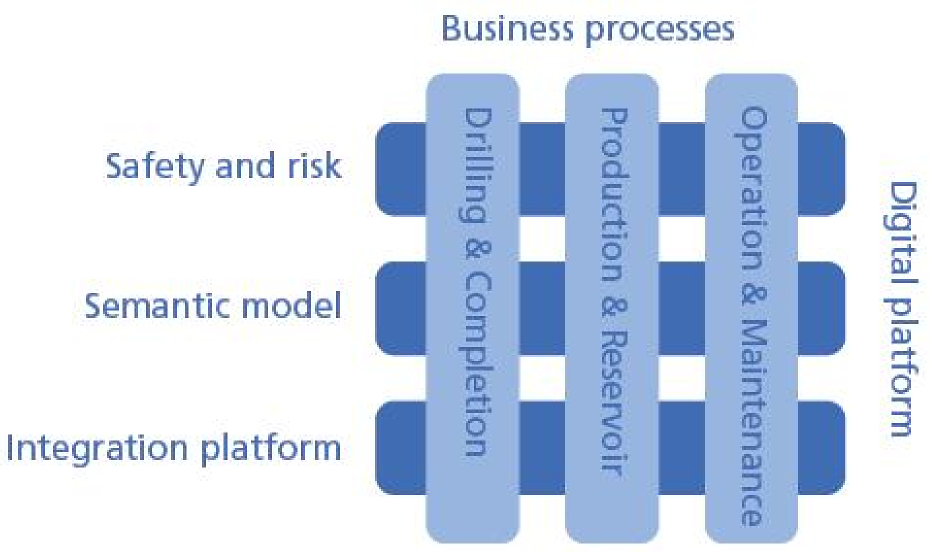

Figure 1.1: *This graph shows how technologies and business processes in IOHN interact.*

WITSML.

IOHN & AutoConRig

IO High North (IOHN) is a large joint industry project with the goal of using information technology to improve offshore operations in the arctic region. Norway and other countries are starting to develop oil and gas fields in such areas, and IOHN will provide an important contribution. As the IOHN wiki website[2] says it: ”The overall goal for the Integrated Operations in the High North (IOHN) project is to design, implement and demonstrate a reliable and robust architecture for Integrated Operations Generation 2 (IO G2). Existing open standards are used and extended when required and new standards are incubated to ensure interoperability, to facilitate integration and to transfer data. To make data-to-information-to-decisions work processes more efficient, information and knowledge models based on open standards are also developed and used.”

IOHN consists of several smaller activities that have specific projects attached to them. Figure 1.1 shows how the different activities in IOHN interact with each other. The activity that is concerned with ontology building is Activity 3. This activity, as well as activity 5 which concerns drilling, are the two that has the largest relevance to this thesis. From

[2]`https://www.posccaesar.org/wiki/IOHN`

participating slightly in the work behind these activities, I have gotten valuable input in the form of domain expert knowledge and feedback, in addition to some general understanding of the processes involved in such a large research program. This thesis hopes to contribute with input concerning ontology building in general and the drilling ontology in particular. One of the projects that I have had most involvement in is the activity 5 project Autoconrig.

Autoconrig is a research project aimed at automatizing as much of the drilling operation as possible. This is proposed being done primarily by implementing "smart agents". These smart agents each has a relatively simple task to monitor or control, e.g. sensor-monitoring or machinery-operation. Part of the problem is finding out exactly what must be included in the agents to be able to run the entire drilling operation. There is a lot of data and details which are not crucial, so identifying these are an important task, since having a simpler system means easier to implement and maintain. It must however still be able to perform every necessary task in drilling a well.

To have a common reference model which all the smart agents draw their knowledge from, an ontology has been proposed as the solution. By utilizing the complex structure expressible by OWL, the simple task of one smart agent is linked and related to other agents in an intuitive and, more importantly, correct way. Since the agents certainly will have to communicate, having each agent be aware of their structural and operational context is clearly desirable. For instance a motor agent should know what the motor is connected to and which operation involves it, such as the drawworks being used for lowering and lifting the string into the borehole. All this can be expressed in the ontology. It is desirable to examine to what extent ontologies can be utilized in a project such as Autoconrig.

AKSIO & CODIO

The AKSIO project[3] is a completed project that had the goal of making information retrieval from documents concerning drilling easier. The idea was to use an ontology to annotate documents with names from the ontology to reflect the actual content in the documents. This way, one could use the ontology to search for matches to the annotations as well as annotations that are related in some way through the ontology. This way of expanding the search would mean that more documents that might be relevant could be found as well. The project was meant mostly as a research prototype to see if this kind of technology was viable for the intended purpose. An

[3]Both AKSIO and CODIO are projects lead by Computas

implementation was created to test the concept, which worked fairly well. The help of domain experts in creating the ontology and access to actual documents from the drilling domain helped the success of this project.

The CODIO project is a research project where the goal is to create a system for decision-support in drilling operations. This sounds very much like the AutoConRig project, and there are certainly similarities. However, where AutoConRig has the ambition of automating large portions of the drilling process, CODIO only aims at giving decision support to the people performing the drilling. Central to their approach is a Bayesian network model of probability. The way this is supposed to work is by looking at sensor data and other input and feeding them into the network. The design of the network then decides what the output will be. The output we will get is mostly probabilities for an event to occur and a suggestion of which action that should be taken. An important part of this entire approach is that the decision reached should feedback and alter the probability model so that future similar cases are affected by decision taken prior to it.

In CODIO they propose to use an ontology mainly for limiting the size of the Bayesian network needed to be reasoned upon. The concepts that are common for the ontology and the network are based on the reasoning conducted at any time. From this it appears is that CODIO and AutoConRig might have slightly different uses for ontologies, as well as the fact the AutoConRig specifies no need for a Bayesian network.

- Since AutoConRig wants to automate everything and have agents control machinery, they want equipment and this machinery to be a part of the ontology. CODIO however is not (at this stage) interested in this.

- Both are interested in tagging or classifying sensor data, and other low level data. Having data clearly structured and related is clearly of interest and help to both, as the actual decision-making process relies heavily upon what is measured by the numerous sensors on many different types of equipment.

- At least AutoConRig, but most likely CODIO also, is interested in having events described by the ontology.

As in AutoConRig, CODIO is also dependent upon an ontology in the drilling domain. The work in this thesis will perhaps serve also here as valuable input to the ontology that is required by CODIO.

1.3 Data integration in the Oil&Gas domain

Data integration often deals with combining multiple sources of information into one output where all data is presented in the same format, such that it can easily be utilized in other systems. This is not an easy task to accomplish, and many systems for integrating data exist today which deal with these kinds of problems. None of them are perfect though, and much manual work must be done before they can function properly. ISO 15926 is a standardization effort that in part has problems such as these as one of its main uses. It will be briefly discussed later in the thesis. Other solutions will also be presented based on state-of-the-art semantic technology that is still an active research field.

Such solutions to data integration problems are relevant both for drilling and for production, though the two are a bit different in what needs they have. This thesis focuses mainly on drilling, but many things discussed are also relevant for a production environment. So as an overview thus far, this thesis will deal mostly with data(information/knowledge) representation and data integration in oil and gas drilling operations, and present attempts at solutions and use-cases relevant for drilling.

Now to take a step back and consider what exists to work with to try and reach closer to the goals and visions previously stated; that is the increased efficiency of hydrocarbon retrieval. I will begin by considering existing standards in the oil and gas industry and see where that takes us.

The focus of this thesis is on drilling, thus it is natural to begin by considering WITSML (Wellsite Information Transfer Standards Markup Language). This is a standard that was created with the purpose of transferring drilling data from wellsites to centers where the data can be stored and processed. WITSML also defines a way of querying the data stored in WITSML servers. The standard is on an XML format, and the structure is defined completely by a set of XML schema files. This subsequently means that WITSML data is stored in XML document files and the limitations of XML (such as the inherent tree structure, and lack of data identifiers other than through the structure itself) apply to WITSML. WITSML is therefore not able to express generic relationships between the data resources. Because of this there might be implicit relationships between data instances in WITSML that cannot be expressed due to limitations in XML. For the purpose of integrating WITSML data with other that data such relationships might be necessary.

In a related issue, the precision of the structure is also not good enough, as the standard opens up for interpretation in several cases. Several companies are in fact using the standard in slightly different ways today. This is clearly not a desirable situation, but as this is the current state of affairs it must be dealt with in a satisfactory manner. Implementing new standards is always a big challenge since most companies already have their own internal proprietary systems and must make them work together with the standard. In these cases, they may look for shortcuts and simplifications in the standard so that they minimize the effort required. This of course means that the implementation of the standard varies from company to company. If things are so bad that the various implementations are incompatible with each other, the standard almost seems meaningless, which is an unwanted situation. WITSML is however not completely as bad as just described, but it certainly has issues that need to be dealt with. That is however not the focus as such of this thesis. I will rather use the domain knowledge contained in WITSML and structure it in hopefully a better and richer.

As much as I have presented petroleum related topics thus far, the problems stated cannot be solved alone by a petroleum engineer or any other drilling domain expert. This is an informatics thesis and as such, it will focus on the technological aspects that pertain to the management of information. The solution for many of the problems in data integration and data representation involves creating an unambiguous vocabulary over the domain of interest.

- What we want to create is a vocabulary, which basically is a set of terms that are relevant to the domain of interest. This vocabulary should be structured through relationships in an appropriate way to reflect the domain of interest as closely as possible.
- 'Unambiguous' refers to the absence of multiple interpretations of the data structure. An unambiguous data structure has only one correct interpretation, and this interpretation should be made obvious so that misuse does not occur. In the case of WITSML as described earlier, it is not unambiguous, which is one of its flaws.

But to create such an unambiguous vocabulary is a difficult task that requires skills that most people in the oil and gas industry do not possess. They do however have important domain knowledge that is essential when creating these kinds of data models (which is the case for any domain). Domain experts are crucial when it comes to identifying important terms that describe the domain, but for structuring these terms a technology expert is needed. A joint effort of petroleum experts and informatics experts is clearly necessary to achieve the best results.

To be able to create this vocabulary representing drilling data better than WITSML does, a formalism is needed to be able to express the data structure. Such technology already exists and more is being developed and improved continuously. The World Wide Web Consortium (W3C) is a great resource for such technology and their Web Ontology Language (OWL) is a well suited formalism for just this kind of representation. Its most useful variant is based on description logics, which has an important position in this thesis. These ontologies that we can represent using OWL are logical vocabularies with a clearly defined semantics. They support reasoning which can extract implicit knowledge from the explicitly stated information in the ontology. This can yield new knowledge based on incomplete input only with the help of the logical structure of the ontology, since the explicit structure is more or less a template for how the information instances should be related to each other. Reasoning can thus in some cases deduce what kind of relationships between data exist that haven't been explicitly stated. This is useful both when creating the ontology and when using it.

Roughly described, an ontology consists of a TBox which is the intentional knowledge, and an ABox which is extensional knowledge. During creation, TBox reasoning is used to find flaws (inconsistencies) in the structure and also to find consequences of the statements entered. In actual use reasoning on the ABox, with the TBox providing the necessary relational structure, is most useful since this yields potential new information about data instances. It is worth noticing an analogy to relations databases here. TBox can be thought of similar to the relational schema, while the ABox can be thought as the tabular data. The difference however is that while in a relational database the schema is disregarded when doing queries on the data, in an ontology both the TBox and the ABox are important for querying. These properties make ontologies a very good formalism for representing complex data structures. However, contrary to regular databases where the schema is thrown away after creation, the TBox in an ontology is still an important part of the ontology when using it and especially when reasoning on it.

With this in mind, what I want to find out is how a W3C ontology based on knowledge mainly from WITSML, but also from other sources, can be constructed. This WITSML ontology will capture much of the drilling domain and be a general purpose ontology with "a little of everything" that can then be extended and specialized in several directions based on different uses for it. One of these uses could be some automated system for drilling, or other automated systems that rely heavily on robust computer systems. Also, as a vocabulary for the drilling domain such an ontology can become a standard for the entire domain. However, for data integration such an ontology is most likely not going to be efficient. A general purpose ontology trying to capture knowledge about the whole domain will likely be too large and slow to use for data integration that handles huge amounts of data and requires fast computation. Data integration with ontologies is dependent upon queries and query answering. For TBox satisfiability OWL2 is found to have a complexity of **NExpTime-hard**[4]. The topic of complexity classes with be further handled in section 3.3.1 on *DL-Lite*. Even though that is worst-case, it is relatively easy to use OWL constructs to create such an ontology. For data integration a much better worst-case is necessary. There are three so-called profiles defined as a part of OWL2. These are fragments of OWL2 with specific purposes intended. I will say more about these later in general, but the one that is most suited for data integration is OWL2 QL. This profile uses the description logic *DL-Lite*, which is a subset of OWL2. It excludes most of the constructs that make the complexity intractable. What remains is still a expressive enough formalism to be usable in data integration, and also other simple forms for modeling. And query answering in *DL-Lite* is shown to be in AC^0 [5], which is a very important property for handling large amount of data for integration. There exists an ontology framework called QuOnto which uses *DL-Lite* ontologies and is thus well suited for data integration. This framework is able to collect data directly from SQL databases through mappings to the ontology. So instead of using a regular OWL ontology for data integration, I will be creating a *DL-Lite* ontology also partially based on WITSML to be used in a data integration case.

Another important consideration to make is whether having a general purpose ontology is a sound approach. This ontology would have to be further extended with details later when the actual use of it is clearly defined. This is one scenario where a new extended ontology must relate to the existing ontology. Other such cases include an ontology that imports concepts from a different ontology, and splitting up an ontology into smaller parts. For my case specifically it might be interesting to see if two such ontologies can refer to each other, and in what way. Common for these cases is that there must be a theory of modularity which ensures that no problems arise when using several ontologies together. This is also a field that is actively researched, and I will spend some time considering the impact it has on the construction of my ontologies.

I have two aims with this work:

1. To create a general purpose vocabulary in OWL based on knowledge from WITSML. This vocabulary will capture much of the drilling domain in a somewhat superficial way, while more detailed specializations can be created as they are needed. The main strength of this ontology will be its unambiguity, which enables it to be used as a standard for domain knowledge representation. And also since OWL is based on description logics, the nice properties it thus inherits when it comes

[4]Much information regarding complexity of OWL at `http://www.cs.man.ac.uk/~ezolin/dl/`

[5][1] page 4

to computability makes it more than just a simple dictionary in that computers can "understand" its meaning. This means that e.g. computerized reasoning can lead to conclusions that haven't been explicitly stated but are nonetheless true.

2. Show how this vocabulary, or a *DL-Lite* stripped-down version of it, can be used for data integration in the drilling domain. In doing this I will explore some theoretical difficulties that arise and discuss possible solutions wherever I am in a position to do that.

However, towards reaching these goals there is also compliance with any existing ontologies in the domain to consider. As far as I am aware, there are no ontologies dedicated to the drilling domain as of now. There are however ontologies which intersect with the drilling domain. ISO 15926 is such an ontology. I mentioned ISO 15926 briefly further up already. It is a large repository for knowledge in the petroleum industry and other industries. It is mostly a taxonomy which defines classes in a textual way. Most of these definitions have not been explicitly stated in a logical way which would be natural to do in an OWL ontology. In addition to this, the modeling formalisms also differ from OWL so that translating from ISO 15926 to OWL is not a straight-forward operation. There are though methods in development that make this task easier. The reason such a translation could be desirable is that ISO 15926 is a large repository of domain knowledge, and it is after all an ISO standard. The creators of ISO 15926 are pushing to have the industry adopt it in their systems, a push which is slowly making progress.

My interest in ISO 15926 lies mainly in accessing whatever relevant drilling domain concepts it may contain, as well as considering the possibility to (partially) use the ISO 15926 part 2 as an upper ontology for my ontology work. Since ISO 15926 uses a different formalism from OWL as stated, this perhaps entails more difficulties than gains. Part 2 also contains some questionable modeling and some of it isn't used at all in ISO 15926 part 4 which is the reference library. One serious consideration that needs to be made is whether ISO 15926 Part 2 is the choice of upper ontology, or rather use a different one such as DOLCE[6] or BFO[7].

Another important topic that needs to be considered. When creating a single ontology with no ties to other ontologies, upper ontologies do not necessarily provide much help. Compliance with them could make the overall structure more understandable, but may not add much else. However when several ontologies or other knowledge representation systems are to communicate or or be integrated, upper ontologies help to make sure that the meaning of

[6]`http://www.loa-cnr.it/DOLCE.html`
[7]`http://www.ifomis.org/bfo`

classes and relations do not diverge in the different systems. This ensures that no misunderstandings or incompatibility over the most basic concepts occur. Upper ontologies basically provide templates for how an ontology should be modeled, and are as such very valuable tools.

Two of the most interesting upper ontologies now existing are DOLCE and BFO. They are both good examples of upper ontologies and arguably the best of what exists now. Most of the text concerning upper ontologies will be based on those two. DOLCE is bigger and more complex than BFO, but BFO also contains many important features an upper ontology should have.

Crucial in both ontologies, is the top level distinction between *endurants* and *perdurants*[8] (different names for them are *continuants* and *occurents* respectively). The difference between the two shows itself in the way they relate to time. Endurants can be said to be wholly present at any moment in time they exist, while perdurants consist of temporal parts such that they are only partially present at any moment in time. The property of endurants means that part-of relationships with endurants should have a time-index to be meaningful. The example "this keyboard is part of my computer"[9] is incomplete without saying when the keyboard is a part of the computer. However in "my youth is part of my life", which is a perdurant parthood, specifying time is not required.

Furthermore, endurants can be split into physical and non-physical endurants, and depending upon the level of detail in the upper ontology, even further distinction can be made. Similarly, perdurants can be divided into e.g. events, processes, phenomena, activities and states. They can have temporal or spatial parts. An example of this kind of parthood is "Proofreading is part of writing a thesis".

What we gain from using upper ontologies (and other standardized ontologies), is that clear and well-defined distinction is made between types of classes in the ontology. External context adds meaning to the ontology in a way that would be difficult to achieve without links to other ontologies. This makes it easier to understand the meaning of the ontology, even with little domain specific knowledge. Integration thus becomes easier to do. Also, certain modelling that is often difficult can be handled in a standardized way which makes the whole creation of the ontology a simpler task.

[8][2] page 10
[9]Example from [2] page 11

1.4 Structure of thesis

Chapter 2 — Oil drilling as a domain of interest

This chapter will introduce that most important sources of domain knowledge that I have used throughout the entire thesis. They are here considered one at a time.

Chapter 3 — Semantic technology

The chapter on semantic technology give a brief description of the current technologies. Then it goes more into detail on the relatively new *DL-Lite*, and the last portion tackles some of the challenges that the current technology has to deal with.

Chapter 4 — Creating the drilling ontology

In this chapter, the sources for domain knowledge are again considered, but this time as ontology construction sources in particular. I try to give methodologies that can be used to extract useful knowledge from these sources, and then create an actual drilling ontology based on this.

Chapter 5 — Application of the drilling ontology

The use cases described here are meant to rely on the ontology presented in the previous chapter. The first use case is somewhat an abstract description of a data integration application using an ontology framework called QuOnto. The second use case is a XML to RDF data conversion tool created in Java, which uses ontologies for improved quality.

Chapter 6 — Conclusion

The last chapter contains a summary of the thesis as well as thoughts on where further work could be applied.

Terms and Acronyms

A list of important terms and acronyms, and their meaning.

File locations

This lists all the external files, including links, created with relevance to this thesis.

Appendix A.1 — Statements

This appendix contains my findings and experiences in a specific method for extracting knowledge from domain experts.

Appendix A.2 — More on Mereology

Here I go into more detail on mereology than I do in the main part of the thesis.

Appendix A.3 — Normative vs Descriptive

This is a take on two different approaches to ontology design that may have implications at an abstract level.

Appendix A.4 — WITSML/XML to RDF/OWL conversion and problems concerning this

In this appendix I go into more of the thoughts behind the XML to RDF data converter presented in chapter 5.

Chapter 2

Oil drilling as a domain of interest

2.1 Standards and sources of knowledge

In this section I will present what I consider important standards and influences for the creation of the drilling ontology. Here I will simply present and explain them one by one, but in chapter 4 about the drilling ontology, the way of combining them as input to the creation of the drilling ontology will be discussed in detail.

2.1.1 WITSML

WITSML[1] (Wellsite Information Transfer Standards Markup Language) is an industry standard for transferring drilling data mainly from drilling installations (offshore) to data centers onshore, but it is also used in the exchange of data between partners onshore. It is maintained by Energistics, a consortium of many companies with interests in the drilling domain. They meet for discussions yearly to decide on the course WITSML should take and what should be included in or excluded from the standard. The fact that many of the most important members of the industry are represented in the consortium makes WITSML an important standard to consider, and a starting point for looking at data integration in the drilling domain. Since WITSML is meant to handle much of the data that in many cases is interesting to integrate, considering WITSML from an integration perspective as well as a more general knowledge representation perspective seems prudent. The data structures which WITSML is created to represent

[1] `http://www.energistics.org/witsml-standard`

should thus be a large part of what a drilling ontology should contain. I will handle WITSML's role in shaping the drilling ontologies more thoroughly in section 4.1.1.

WITSML is based on an older standard called WITS, to which they added the structure of a markup language to create WITSML as it is today. It is thus built on XML and the structure and contents are defined solely by a set of XML schema files (XSD files). This set consists of 20+ top level object schemas, from which the XML document files are created. Some of the most important of these top level schemas are[2]:

- log : Contains log data.
- mudLog : Contains log data about the mud in circulation.
- trajectory : Description of the trajectory or path that a wellbore follows.
- tubular : Information about which components a drillstring is made up of.
- well : Information about a well which in WITSML is defined as "a unique surface location from which wellbores are drilled into the Earth for the purpose of either (1) finding or producing underground resources; or (2) providing services related to the production of underground resources."
- wellbore : Information about a wellbore which in WITSML is defined as "a unique, oriented path from the bottom of a drilled borehole to the surface of the Earth. The path must not overlap or cross itself."
- wellLog : Contains log data about a well.

These object schemas include in a hierarchy a number of other schemas, in which schemas for datatypes form the foundation. The most simple of the datatypes are XSD types with a few value restrictions added to them. However a large part of the types are enumerated datatypes which list legal values for a number of properties. These can for instance be a list of all possible type of tubular components, or types of activities. In addition to these there are the quantities which refer to units of measure, which are important parts of representing measured or calculated data values. Transferring such data values is a large part of what WITSML is used for, mainly in the form of logs which are represented in many of the top level XML documents. An example of a portion of WITSML document structure may be expressed like this (here not in proper WITSML/XML syntax):

```
Wellbore
```

[2]which parts of WITSML that are important may vary from user to user

```
nameWell = 6507/7-A-42
name = A-42
...
commonData
    dTimCreation = 2001-04-30T08:15:00.000
    ...
```

Here “Wellbore” is a top level object, meaning that there are XML documents created containing at least one wellbore. Each of these wellbores then has several elements below it in normal XML fashion. For instance “commonData” is an imported schema and is as the name implies, common for many of the kinds of documents that can be created. The data value “dTimCreation” is described in WITSML as “When the data was created at the persistent data store.. . . ”.

Although WITSML is often stored in dedicated WITSML servers, the data may be stored in regular relational data bases instead of actually storing the XML documents as this is not very efficient. The WITSML standard also provides a querying language which is used to access WITSML data. This is simply the interface which must be standard, but actual implementations could vary greatly. As we will see, there are unfortunately more critical parts of WITSML where variation may occur as well.

For the purpose of precise knowledge representation as well as data integration, it is important to have a foundational model which is unambiguous in that the behavior/structure must be clearly defined in all cases. The same is true for a standard to function as a proper standard. In a large consortium all participants naturally have their wishes as to what should be included in a standard, so compromises are made to satisfy different interests. How this process evolved with WITSML is not clear (to me), but often these sort of situations can lead to a much more diffuse standard than is desirable. In the case of WITSML, several parts of the structure are loosely defined so that potentially conflicting documents can be created. Multiple interpretations and implementations exist which differ on some areas. For instance some use the standard in a way that says: a rig can have one well. Others use the standard in a way such that: a rig can have multiple wells. This might be a minor problem, but similar cases exist and they introduce difficulties when handling WITSML data from multiple sources. Another problem that could be worse to detect and handle, is one dealing with tubular components. Drill pipes in the particular type of tubular component which is used most of all. They are created with a standardized length, but part of the drill pipe is the joint which is supposed to be screwed into a different drill pipe. Whether or not this joint is included in the total length may vary from company to company. Since the calculation of the length of the drillstring is very important in drilling, this is something that must be made clear, but unfortunately WITSML provides no guidelines as to how this should be handled. Other cases dealing with interpretations of data may also exist. While in practice such problems might be rare, it questions WITSML as a standard.

As WITSML is based on XML, limitations inherent in XML expressivity and structure naturally apply to WITSML as well. The sort of complex relationships which might be desirable in a general domain knowledge representation, are not present in WITSML. There are few or no abstractions of knowledge relevant to the drilling domain in WITSML. However this is not a relevant part of a simple transfer format. WITSML deals with concrete data and entities only, and does not refer to any higher level of abstraction, which would help in describing the data in a more general manner. But for creating such a general representation, additional information about relationships between different kinds of data are needed. As an example of one difficulty that arises because of the lack of a clear abstraction on top, is how enumerations are used. While some enumerations are unproblematic, such as the listing of possible units for a unit of measure, others are not as straight-forward. The handling of tubular components is one of these. Even though it is correct that the long list of type of components denote different tubular components, nothing is said about how similar these must be and what the defining properties of a tubular component are. In fact, more or less the only thing tubular components have in common is that they are a part of a tubular. Other than that they may have very different properties. In WITSML the type itself is simply a property of tubular component so the question arises whether a more clear division among various types is needed. Although for its current purposes there may be no problems, knowledge representation based on WITSML must consider questions such as these. Many such relationships and abstractions lie implicit in the standard as it is today, but these are mostly in the heads of the domain experts that created the standard and cannot be pulled directly out of WITSML. Help from experts will be needed in doing this, which will be discussed further in section 4.1.1 and in section 4.1.6.

WITSML might be slightly more suited in its current form for data integration. The included query language in WITSML is usable for simple data extraction where we know exactly what data we are interested in and how it is structured. This suits integration where we are interested in consolidating limited sets of data. But since the query language relies on very explicit input, more complex integration dealing with incomplete data and uncertain structures will not be possible as it is. The query interface itself is defined by using XML files and simple pattern matching to fill in data in the XML elements provided. This limits the complexity of the queries that can be expressed. As the implementations may use a relational database for actual storage, using SQL for queries is surely a possibility.

This may often provide a workable solution, however the structure of the data is a limitation by itself as well. An ontology based data integration with use of relationships and semantics will require a completely new data model, which can of course be based on WITSML. But WITSML by itself will not suffice in most cases of complex data integration.

Despite its limitations, WITSML works well for its current purpose, but it is not good enough for use in representing domain knowledge on a general basis, nor for use in complex data integration. It is however a good source of domain knowledge and relevant data which should be included in a drilling ontology. As long as so much data is available in WITSML format, to be able to use these data in test cases for data integration has its advantages, so a model based on WITSML is beneficial. WITSML is also a well-known standard with many users, meaning that any WITSML-derived work will automatically have more influence and impact than most work done from scratch.

2.1.2 DDR - Daily Drilling Report

The daily drilling report is a standardized format for transferring data about daily drilling to the Norwegian oil authorities. By Norwegian law, all operators drilling in Norwegian areas have to hand in such a report every day to keep track of the drilling activity at the Norwegian continental shelf. The DDR standard itself is by large based on WITSML. It is defined as a single XML schema which refers to WITSML types for reference. Most of the types and also enumerations used in DDR are directly gathered from WITSML, but there are also elements in DDR which are not in WITSML, so DDR is not a proper subset of WITSML.

An example structure from DDR which shows the connection to WITSML:

```
<witsml:drillReport>
  <witsml:nameWell>34/10-A-32 C</witsml:nameWell>
  <witsml:nameWellbore>34/10-A-32 C</witsml:nameWellbore>
  <witsml:name>witsml:name</witsml:name>
  <witsml:dTimStart>2006-06-07T00:00:00.000</witsml:dTimStart>
  <witsml:dTimEnd>2006-06-07T23:00:00.000</witsml:dTimEnd>
  <witsml:versionKind>preliminary</witsml:versionKind>
  <witsml:createDate>2006-06-07T13:15:00.000</witsml:createDate>
  <witsml:wellAlias>
    <witsml:name>34/10-A-32 C</witsml:name>
    <witsml:namingSystem>NPD code</witsml:namingSystem>
  </witsml:wellAlias>
```

I will say more about DDR in section 5.1 where I describe a use case that uses the ontology framework QuOnto to do data integration on DDR data.

2.1.3 ISO 15926

ISO 15926 is a large ontology and information repository created by the POSC organization. Its main purpose is to be used as a reference library that in part has ontology structure. For the most part it contains information relevant to oil and gas, process and chemical industries. But it can also be used for other industries and businesses.

ISO 15926 is divided into several parts, some of which are part of the actual ontology and some of which are not. Part 2 is the upper ontology part that defines the topmost structure that all lower parts of the ontology must refer to. It is the smallest part of the ontology, with just a few hundred classes. Part 4 which is the reference library already consists of tens of thousands of classes, and is growing continuously. This reference library contains many classes relevant to the oil and gas domain, but a large part of it is not relevant at all. The detail level in part 4 stretches from general classes such as “pump” down to very specific pumps suitable for a particular job.

Part 7 of ISO 15926 introduces so-called called templates, which are meant to simplify the job of entering new data into the ontology. As ISO 15926 is not an OWL ontology, but rather a proprietary ontology language, it is more difficult for people without enough experience to use ISO 15926. Templates such as those in part 7 makes the job easier both of adding and gathering data, as well as interfacing ISO 15926 with OWL.

As the people working on ISO 15926 has recognized the value in being able to connect the ontology to OWL, more and more have been done in this direction. It is possible to extract data from the ontology and convert it to the OWL format automatically, and all the parts of ISO 15926-2 which are currently in use have been formulated in OWL. However because of the difference in foundation from the description logics in OWL, it is not a proper representation of the knowledge in part2. For this reason there is also being done work to create an ISO 15926 upper ontology based on OWL.

There are other parts as well, but they are not as relevant to this thesis.

2.1.4 Schlumberger Oilfield Glossary

The Schlumberger Oilfield Glossary[3] is an online repository of domain knowledge in the oil and gas domain. This includes drilling as well as production, and other subdomains. The focus will lie on the drilling part of

[3]`http://www.glossary.oilfield.slb.com/`

the repository. It is quite extensive with over 3000 entries in total, of which a substantial part is on drilling. Many of these entries describe various kinds of equipment, and how they are fitted together. But there is also information on processes and tasks performed on the drilling rig. Much of this, and the most interesting for this thesis, is what happens downhole while drilling. There are many entries which link to each other and describe the tools and equipment used while drilling.

As an example from the glossary, the entry for **logging while drilling (LWD)**:

“The measurement of formation properties during the excavation of the hole, or shortly thereafter, through the use of tools integrated into the bottomhole assembly. LWD, while sometimes risky and expensive, has the advantage of measuring properties of a formation before drilling fluids invade deeply. Further, many wellbores prove to be difficult or even impossible to measure with conventional wireline tools, especially highly deviated wells. In these situations, the LWD measurement ensures that some measurement of the subsurface is captured in the event that wireline operations are not possible. ”

We will in chapter 4 see how useful knowledge can be extracted from texts like this.

2.1.5 AKSIO

AKSIO has already been discussed in the introduction, so this section will simply reiterate some of the important points made there.

AKSIO was a collaborative project where the goal was to heighten the quality of data/documents returned in searches and queries regarding drilling/petroleum operations. For this purpose an ontology was created, and the idea was that data/documents should be tagged with concepts from the ontology, and then through the relations in the ontology concepts related to the query-word would also be found. This project’s purpose was not a finished product, but it was meant as a research effort and a prototype to build upon later. The value in AKSIO lies mostly in the fact that the ontology was created in collaboration with domain experts, so we will assume that the information is good. The concept names defined are likely to be the ones actually used in the domain of drilling, and this is important to capture. The ontology is mostly just a concept-hierarchy with very few roles, but the ones created provide some relational information.

2.1.6 Domain experts

Domain experts are an important source of information in creating all sorts of knowledge representation systems. There will always be unclarities when trying to obtain knowledge from just reading documentation. Domain experts are a valuable source of information that otherwise can be difficult to acquire. Direct querying and questioning of these experts can give understanding of the most difficult parts of the domain. They know their domain better than anyone and are often the same people who will be using the system/ontology, meaning that their input will directly influence the systems they themselves will be using.

In the drilling domain, domain experts can give help to give the whole picture of the operation, and explain largely in which order processes happen, and what is dependant upon what. This is knowledge that often is poorly represented in documentation, and having it explained by an experts and being able to ask relevant questions is quite valuable. Having gained this insight, the ontology developer should dig into as much of the other sources as possible and get to know the details. After acquiring a certain level of understanding and also finding out where the problems and difficulties are, the domain experts prove a very important source for solving those problems.

2.2 Specific problem and solutions

While the gathering of the domain knowledge into a single document by itself has value, applying this collection of knowledge to solving specific problems makes the effort all the more worthwhile. One such specific problem, that also was discussed at length in the introduction, is data extraction and integration. While the knowledge itself does not enable data integration, it does provide the foundation for either creating tools for data integration, or for using existing tools. I will look at both of these ways in which data integration can be achieved.

Common for both of the solutions is that they utilize ontologies as a way of representing domain knowledge and connect the actual data to these ontologies. Since the task of creating all the software necessary for data integration is rather large as a single part of a master thesis, I will mostly describe the way it can be achieved, but also provide some software implementations that are useful as parts of a full integration. The actual ontologies used are closely linked to the drilling ontology that is a central part of this thesis. Chapter 4 will deal with the drilling ontology in detail, and chapter 5 will present how this relates to the use cases described here.

2.2.1 WITSML and DDR

The focus of this part of the thesis is on the conversion and integration of WITSML and DDR data. As these solutions both rely heavily upon ontologies, the details on these solutions and how they utilize ontologies will be presented in chapter 5, after the ontologies and the technical aspects of them have been presented in chapters 3 and 4.

Proprietary solution

An important part of data integration is the conversion of data into a single format that is easy to work with when doing the actual integration. This proprietary solution focuses on this aspect. It provides a generic way of converting data from any XML format, thus including WITSML, to RDF and OWL. This is done with the aim that the integration will be conducted with OWL ontologies as an integral part and the data in RDF connected to these ontologies. This is where the link to the drilling ontology will become apparent. Here I also provide an implementation in Java of the converter.

QuOnto solution

QuOnto is an ontology representation and reasoning framework well suited for data integration using ontologies in *DL-Lite*. In theory it can collect data from any kind of data storage and integrate these, but there are not so many actual implementations yet. The data is then connected to the ontology using user-defined mappings. These are besides the ontology the most important part of the process. The software that is using this framework then simply has to query the ontology using a version of SPARQL extended with SQL-like statements, and the data will be extracted on the correct format and be associated with the correct ontology class. This can be done for multiple data sources each with a unique mapping to the ontology. This way data represented in various ways can be properly integrated.

Chapter 3

Semantic technology

This chapter deals with the theoretical foundations used to express ontologies, mostly concerning OWL. I will give an overview of the main technologies relevant and then go into a little more detail. The second half of this chapter will present emerging technologies based on OWL as well as issues related to OWL that have significance for creating a drilling ontology, but also other ontologies.

3.1 Overview over established technologies

The technologies that make the foundation of what is becoming mainstream semantic technology are first and foremost RDF (Resource Description Framework), OWL (Web Ontology Language) and SPARQL (SPARQL Protocol and RDF Query Language). These three make up the most important formalisms for working with semantic content, and their position in the semantic web stack can be seen in figure 3.1. RDF is most known in its XML serialized form, but there is no necessary link between the two. RDF is conceptually a general directed graph structure, while XML is strictly defined by its syntax which is in a tree-form. This RDF graph is built out of triples with a Subject-Predicate-Object structure. RDF is as such a very powerful representational formalism.

As powerful as RDF is, in itself it does not specify more concrete uses. It does however provide the formal constructs to do this such that vocabularies with RDF structure can be created. One of the most popular RDF vocabularies is FOAF (friend of a friend) for describing people and their relations. This vocabulary is as most RDF vocabularies simply a syntactic extension upon RDF, and thus uses the semantic foundation of RDF without adding anything. There are also formal vocabularies on RDF which extend both

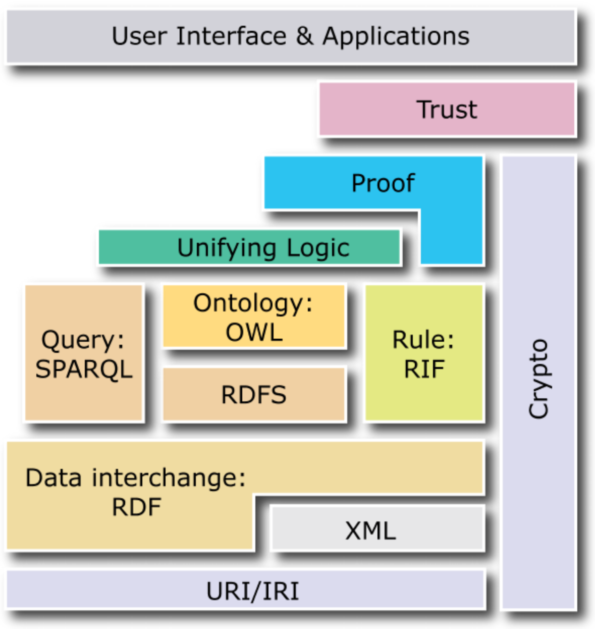

Figure 3.1: *This illustration shows one take on the semantic web stack.*

syntax and semantics. RDFS and OWL are the most prominent of these.

RDFS (Resource Description Framework Schema) does not appear much in this thesis, but it is worth mentioning to complete the picture. As the name implies it was created to give RDF a meta-level in the form of a schema. This is different from e.g. XML schema as RDFS also has a formal semantics defined. This semantics provides subclass relations and other constructs for creating a class/concept taxonomy. There are two main ways in which the semantics for a taxonomy can be defined; intensional and extensional. Simply put, in the extensional semantics classes are defined by the set of its individual members. What this means is that in an extensional semantics, two classes with exactly the same individual members are equivalent classes. In an intensional semantics this is however not the case. Even with exactly the same members, two classes cannot be inferred to be equal. In the case of the formal RDF vocabularies, RDFS is intensional while OWL is extensional.

OWL[1], the Web Ontology Language, was created as a formal vocabulary over RDF for modelling full-fledged ontologies. While the difference from RDFS has been established they both are similar in that they facilitate taxonomies. As such the subclass relations (and some others) mean the same. What differs is the interpretation of classes. Besides that, OWL is

[1]The focus will be on the OWL variants based on description logics. OWL-FULL is largely ignored

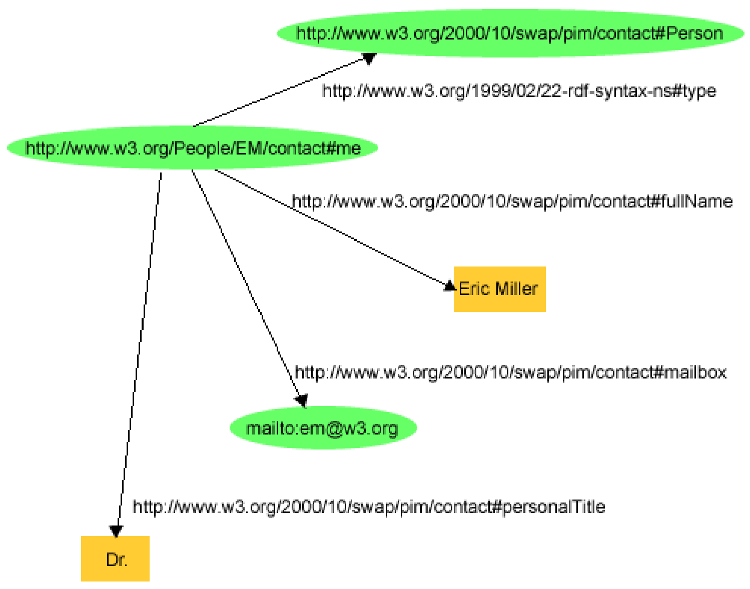
http://www.w3.org/2000/10/swap/pim/contact#Person
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/People/EM/contact#me
http://www.w3.org/2000/10/swap/pim/contact#fullName
Eric Miller
http://www.w3.org/2000/10/swap/pim/contact#mailbox
mailto:em@w3.org
http://www.w3.org/2000/10/swap/pim/contact#personalTitle
Dr.

Figure 3.2: *This graph is an example of how RDF can be visualized.*

a much larger formalism than RDFS, and can express way more complex structures. This is thanks to the foundation of OWL which lies in description logics[3]. Because of limitation to the first version of OWL, a second version of OWL has reached the status of recommendation at W3C. This OWL2 adds a number of important constructs, such as qualified number restrictions, and a richer set of relational constructs.

3.2 Established technologies in further detail

3.2.1 RDF

RDF (Resource Description Framework) is a W3C recommendation intended for describing data, or any kind of computerized representation of resources in general. All RDF statements are triples on the form Subject-Predicate-Object, e.g.

```
Lars livesIn Norway
```

Here Lars is the subject, livesIn is the predicate and Norway is the object. Similar kinds of statements can be used to create a full interconnected structure. In this manner RDF can be viewed as a graph, where the subjects and objects are nodes, and the predicates are edges. An example graph from W3C can be seen in figure 3.2.

An important property of RDF is that it provides for identification of resources based on URIs (Uniform Resource Identifier). These URIs generalize the URLs used for identifying web pages. A significant difference and limitation of URIs is that while URLs in a proper network uniquely points to a single web page, URIs comes with no such guarantee. This is because URIs have a much broader area of usage and is not tied to any specific location. While the URL "http://www.larsdomain.no/Lars" would point to a single web document in that specific location, the URI "http://www.larsdomain.no/Lars" have no such universal single meaning. It is simply a name that can anyone can use to describe any kind of resource, regardless of what the text string implies. To alleviate this, it is generally agreed that people use domains they control when creating URIs for describing resources. This way I can assure a conceived universal meaning for my URI "http://www.larsdomain.no/Lars" if I control the domain "larsdomain.no". Any further mention of unique URIs or identifiers will refer to this perceived uniqueness established through agreement of use.

With this established, we can see that the triple above lacks domains to be properly unique. Fortunately I have introduced the domain "larsdomain.no" that I will use to provide uniqueness of my resources.

```
http://www.larsdomain.no/Lars  http://www.larsdomain.no/livesIn
  http://www.larsdomain.no/Norway
```

What I have done here is to make sure that all three parts of the statement are unique, by identifying them with a (fictive) internet domain. Even though I undoubtedly want to ensure that "Lars" is unique, this is not actually the case with the two other. Both the predicate "livesIn" and the object "Norway" should already have good existing RDF definitions. It is much better then to use those existing resources instead of creating my own "Norway". This way I most likely expand my knowledge greatly, since a resource like "Norway" should be included in many triples. Thus an even better triple might look something like this :

```
http://www.larsdomain.no/Lars
  http://www.personconcepts.com/livesIn
    http://www.norway.no/Norway
```

Now I have altered the predicate and the object to reflect what could be existing RDF resources. This is a great way of reusing other work and also connecting your own RDF graph to a larger whole.

When creating RDF graphs with common domain names it is useful to introduce namespaces as a way to save space and make the triples more readable. A namespace defines and abbreviates a common prefix for the items in a vocabulary and usually corresponds to the URI up to, but not including, the local name. We can for instance define the namespace "myns" to mean “http://www.larsdomain.no/”. This way the first resource from above can be written as “myns:Lars”. This is common when using RDF and its derivatives such as OWL.

RDF also provides a set of built-in resources with a well-defined semantics. An example of these is rdf:type, which can be used as a predicate to express a object-memberOf-class kind of relationship. A complete overview of the RDF semantics can be found at `http://www.w3.org/TR/rdf-mt/`.

RDF itself is an abstract resource representation framework, which can utilize different serializations (file formats). The most common is **XML**, but others such as **n3**[2] and **Turtle**[3] are also used. The same naturally applies to OWL as well.

3.2.2 OWL

OWL is the Web Ontology Language, a W3C recommendation with the purpose of providing a standardized ontology language. OWL initially had three variant: OWL-LITE, OWL-DL and OWL-FULL[4]. The most useful and interesting one is by far OWL-DL. It is based upon a description logic with the designation SHOIN(D)[4]. Currently OWL version 2 with the description logic SROIQ(D)[5] is becoming the new standard for representing OWL ontologies. In addition to the complete OWL2 language, three sublanguages of OWL2 called profiles are presented as ontology languages aimed at more specific uses. These three are OWL2 RL, OWL2 QL[5] and OWL2 RL. They are described further in section 3.4.5.

As was explained above, OWL is built upon RDF. OWL extends RDF as a formal vocabulary which has a formally defined semantics, which is the strength of OWL. Much more complex structures and relationships can be expressed with a semantic foundation in OWL than with simply RDF. Per definition everything in OWL is also proper RDF, but the simpler semantics of RDF does not interpret the triples to the extent that OWL does. The underlying description logics of OWL provides semantics that enables OWL to be as expressive as it is. In logics we talk about axioms as the statements that describe the relationships between classes and relations. Examples of such axioms can be:

$$classA \sqsubseteq classB$$

$$classA \sqsubseteq \exists relationR.classB$$

[2]`http://www.w3.org/DesignIssues/Notation3.html`

[3]`http://www.w3.org/TeamSubmission/turtle/`

[4]OWL-LITE is a less expressive fragment of OWL-DL and OWL-FULL is a much more expressive language that is undecidable. OWL-LITE and OWL-FULL will not be discussed any further in this thesis.

[5]particularly interesting since it uses *DL-Lite* as logic foundation

The first axiom says that classA is a subclass of classB, while the second axiom says that classA is the subclass of all things that have a relationR relation to classB. This does not look like anything RDF triples can express outright. To do that OWL provides for the syntactic link that connects description logics and RDF. Generally a subsumption axiom is a more complex structure than an RDF triple, and as such must be expressed using several triples. The two axioms from above can be express as triples such as this, grouped by the axiom they express:

```
classA       rdf:type            owl:Class
classB       rdf:type            owl:Class
classA       owl:subClass        classB


classA       rdf:type            owl:Class
classB       rdf:type            owl:Class
relationR    rdf:type            owl:ObjectProperty
classA       owl:subClassOf      anonymous1
anonymous1   rdf:type            owl:Class
anonymous1   owl:Restriction     anonymous2
anonymous2   rdf:type            owl:Restriction
anonymous2   owl:onProperty      relationR
anonymous2   owl:someValuesFrom  classB
```

As we can see from the listing above, the number of triples can be quite large for even a relatively simple axiom. Fortunately ontology building tools do this job for us, so effort can go into actual creation of the ontology.

Since description logics build on a long tradition of logic theories and the ability to reason efficiently upon these, OWL follows in the same fashion. As a decidable fragment of first order logic[6], this means that inconsistencies in the ontology can be discovered automatically, and interesting consequences that were either intended or unintended can be derived from ABox and TBox reasoning.

Decidability is an important aspect when it comes to reasoning about ontologies. OWL-DL has the nice property that it is decidable, meaning that all consequences can be derived in finite time. This could however be non-deterministic exponential time, so in worst case scenarios things might not look as good after all. For this reason, many research groups are working on smaller fragments of DL such that they retain as much expressivity as possible while bringing reasoning down to polynomial time and below. The OWL2 profiles relate to such fragments, and *DL-Lite* (not to be confused with OWL-LITE) is the fragment that is used in the profile OWL2 QL. It

[6]Description Logic Handbook[6] section 1.7.2

will be discussed further in section 3.3.1.

3.2.3 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is the query language W3C recommends for querying RDF graphs. It is a simple variable substitution based query language, where triples are given and variables in those triples are substituted with all possibilities found in the RDF graph. It is also possible to restrict the results by constricting the values returned in similar ways to SQL. A simple SPARQL example query:

```
SELECT ?name ?age ?
WHERE ?person foaf:name ?name . ?person foaf:age ?age
```

The capitalized words are reserved keywords, the words with a preceding ? are variables, while the rest of the words such as foaf:name should be occurring in the RDF graph. "foaf" is here an existing RDF vocabulary "friend of a friend" that models social relations.

SPARQL is a powerful tool for querying RDF data, but it gets a little more complicated to use with OWL. As we have seen in the previous section on OWL, many triples are often needed for expressing even simple axioms. This means that at least when writing queries by hand, they quickly become large and difficult to handle. If the computer automatically generates queries based on simpler input, this should not be a problem however.

There is more to SPARQL than the simple example query from above. Constructs for e.g. ordering and filtering also exist. The W3C specification for SPARQL with more details on this can be found at `http://www.w3.org/TR/rdf-sparql-query/`.

3.3 Emerging technologies

3.3.1 *DL-Lite*

When the purpose of our ontology is data integration and other cases that handle large amounts of data, all of OWL/OWL2 might prove way too expressive compared to what we actually need. That is, the expressiveness also means that reasoning and query answering potentially become really slow. OWL has NExpTime and OWL2 has 2NexpTime complexity in consistency and satisfiability reasoning with regards to the TBox size[7, 8], and query answering is not even completely known. This would be very bad if we are dealing with huge amounts of data, which often will be the case in integration for oil and gas applications.

DL-Lite[1, 7] refers to a whole family of description logics, with the common property that they all are "lighter" than the traditional DLs such as SHOIN and SROIQ. It was developed just for the purpose of being able to guarantee fast reasoning and query answering, while capturing some of the most popular modeling formalisms, such as Entity-Relationship model and UML class diagrams. In this family of *DL-Lite* logics, I will focus on some of the ones that are the least expressive, and thus are contained in the better end of complexity classes. This of course because the OWL2 profile "QL" is based on one of these logics, furthermore the same is true for the ontology framework QuOnto.

We will begin by describing the most expressive *DL-Lite* and restrict it until we reach the ones we are after.

DL-Lite$_{bool}^{\mathcal{RN}}$

This version of *DL-Lite* is the most expressive and its complexity in various cases is too high for use in data integration. It has an ExpTime combined complexity[7], and coNP complexity in data complexity, where we have instance checking and query answering[8]. I will not go into the details of each complexity class relevant here, but simply establish their order of complexity[9] of tractability and refer to [1] section 3.3 for more information.

$$AC^0 \subseteq LogSpace \subseteq NLogSpace \subseteq P \subseteq NP \subseteq ExpTime$$

The logic *DL-Lite*$_{bool}^{\mathcal{RN}}$ has concepts C and roles R defined as follows:

$$R ::= P_i | P_i^-,$$

$$B ::= \bot | A_i | \geq qR,$$
$$C ::= B | \neg C | C_1 \sqcap C_2,$$

where B is a basic concept, A is a concept name, P is a role name, and q is a positive integer.

A *DL-Lite*$_{bool}^{\mathcal{RN}}$ TBox $\mathcal{T}$ is then a finite set of concept and role inclusions of the form:

$$C_1 \sqsubseteq C_2,$$

$$R_1 \sqsubseteq R_2$$

[7][1] section 3.2. Combined complexity is the measure where the entire knowledge base K is regarded as an input

[8][1] section 3.2. Data complexity is the measure where the TBox is regarded to be fixed, while the ABox is counted as input. Conversely TBox complexity is the measure where the ABox is regarded to be fixed, while the TBox is counted as input.

[9]a small fraction of the complexity classes available

and an ABox $\mathcal{A}$ is a finite set of assertions of the form:

$$A_k(a_i),$$

$$\neg A_k(a_i),$$

$$P_k(a_i, a_j),$$

$$\neg P_k(a_i, a_j)$$

Together these two form a *DL-Lite*$^{\mathcal{RN}}_{bool}$ knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. Further explanation and the semantic interpretation can be found in [1] section 2.1.

To start out with this logic and gain the fragments we are after we have to restrict it along three axes: (i) the Boolean operators *bool* on concepts, (ii) the number restrictions ($\mathcal{N}$) and (iii) the role inclusions ($\mathcal{R}$).

Restrictions on the inclusion axioms

If we restrict the concept inclusions of a *DL-Lite*$^{\mathcal{RN}}_{bool}$ TBox to:

$$B_1 \sqsubseteq B_2,$$

$$B_1 \sqsubseteq \neg B_2,$$

$$\neg B_1 \sqsubseteq B_2,$$

where B_i are basic concepts, our TBox will be called a *Krom TBox.*

If we restrict the concept inclusions of a *DL-Lite*$^{\mathcal{RN}}_{bool}$ TBox to:

$$\sqcap_k B_k \sqsubseteq B$$

our TBox will be called a *Horn TBox.*

Finally, if we restrict the concept inclusions of a *DL-Lite*$^{\mathcal{RN}}_{bool}$ TBox to:

$$B_1 \sqsubseteq B_2,$$

$$B_1 \sqsubseteq \neg B_2,$$

our TBox will be called a *core TBox.* As $B_1 \sqsubseteq \neg B_2$ is equivalent to $B_1 \sqcap B_2 \sqsubseteq \bot$, core TBoxes can be regarded as sitting in the intersection of Krom and Horn TBoxes.

The focus will from now on be on the fragments of *DL-Lite*$^{\mathcal{RN}}_{core}$.

Further restriction

We will now restrict further by limiting number restrictions and role inclusions[10]:

- The fragment of $DL\text{-}Lite_{core}^{\mathcal{RN}}$ without number restrictions $\geq qR$, for $q \geq 2$, but with role inclusions will be denoted by $DL\text{-}Lite_{core}^{\mathcal{R}}$. Note that this means we still have existential concepts $\exists R$.
- The fragment of $DL\text{-}Lite_{core}^{\mathcal{RN}}$ without role inclusions, but with functionality constraints and existential concepts $\exists R$ denoted $DL\text{-}Lite_{core}^{\mathcal{F}}$.

What is interesting with these fragments that have just been described is that $DL\text{-}Lite_{core}^{\mathcal{R}}$ is the exact logic that is the basis for the OWL 2 QL profile, and the most expressive variant that can be used in QuOnto is the combination of $DL\text{-}Lite_{core}^{\mathcal{R}}$ and $DL\text{-}Lite_{core}^{\mathcal{F}}$, which is called $DL\text{-}Lite_{\mathcal{A}}$.

For all these three fragments the combined complexity of the satisfiability problem is $\leq$ NLogSpace, and the data complexity for query answering is AC^0. This particular property has great implications for how *DL-Lite* relates to SQL. The common relational database system SQL also has a query answering complexity of AC^0. This means that *DL-Lite* can use a relational database management system (RDBMS) to answer queries. Large amounts of data that is interesting to integrate is stored in RDBMSs, so this connection is quite valuable to have.

The people behind *DL-Lite* have created a whole framework named QuOnto for developing and implementing ontologies in *DL-Lite*, and querying against data from many SQL databases. This framework also facilitates data integration. They even defined a query language based on both SQL and SPARQL called SparSQL that tries to combine the best from both query languages while still retaining the AC^0 complexity class. Using this SparSQL query language, it is possible to query SQL data directly through mapping to the ontology, while getting the reasoning capabilities of **DL-Lite**. This means that a much richer set of results can be obtained than with just normal SQL queries.

DL-Lite vs OWL2

DL-Lite is as stated a fragment of OWL2. This means that not all ontologies in OWL2 can be expressed in *DL-Lite* without removing axioms. As a general OWL2 ontology is more expressive than a general *DL-Lite* ontology, linked to this expressiveness is also complexity. Because of this, an OWL2

[10]Note the there are other fragments with their specific restrictions that have not been mentioned here.

ontology will need much more resources for computing queries and reasoning than a *DL-Lite* ontology. For an ontology handling large amounts of data, it is clearly desirable to keep the complexity down so that each operation on the data takes as little time as possible. *DL-Lite* is a suitable fragment for such cases where the ontology is not so complex, but the amounts of data are large. As previously stated, this is perfect for use cases on data integration.

DL-Lite is created to be a simple subset of OWL2 that retains certain properties while having much lower computational complexity than all of OWL2. OWL2 has a complexity with regard to data size (ABox) of at least coNP-Hard[9], and a combined TBox and ABox complexity of ExpTime-hard or possibly even worse. It is the combination of constructs available in OWL2 that yields this rather horrendous complexity. Most of these constructs have typically large branching when doing inference on them. Disjunction on the left hand side, and nominals in any form are examples of constructs that in any case blow up the complexity exponentially. *DL-Lite* then naturally cannot include constructs such as these, but also other constructs have rather bad impacts on the complexity.

The following OWL constructs are the ones that are available in OWL 2 QL[11]:

- subclass axioms (SubClassOf)
- class expression equivalence (EquivalentClasses)
- class expression disjointness (DisjointClasses)
- inverse object properties (InverseObjectProperties)
- property inclusion (SubObjectPropertyOf not involving property chains and SubDataPropertyOf)
- property equivalence (EquivalentObjectProperties and EquivalentDataProperties)
- property domain (ObjectPropertyDomain and DataPropertyDomain)
- property range (ObjectPropertyRange and DataPropertyRange)
- disjoint properties (DisjointObjectProperties and DisjointDataProperties)
- symmetric properties (SymmetricObjectProperty)
- reflexive properties (ReflexiveObjectProperty)
- irreflexive properties (IrreflexiveObjectProperty)

[11]List from W3C `http://www.w3.org/TR/owl2-profiles/#OWL_2_QL`

- asymmetric properties (AsymmetricObjectProperty)
- assertions other than individual equality assertions and negative property assertions (DifferentIndividuals, ClassAssertion, ObjectPropertyAssertion, and DataPropertyAssertion)

And these are the ones that are in OWL2 but not in OWL 2 QL: (list from W3C)

- existential quantification to a class expression or a data range (ObjectSomeValuesFrom and DataSomeValuesFrom) in the subclass position
- self-restriction (ObjectHasSelf)
- existential quantification to an individual or a literal (ObjectHasValue, DataHasValue)
- enumeration of individuals and literals (ObjectOneOf, DataOneOf)
- universal quantification to a class expression or a data range (ObjectAllValuesFrom, DataAllValuesFrom)
- cardinality restrictions (ObjectMaxCardinality, ObjectMinCardinality, ObjectExactCardinality, DataMaxCardinality, DataMinCardinality, DataExactCardinality)
- disjunction (ObjectUnionOf, DisjointUnion, and DataUnionOf)
- property inclusions (SubObjectPropertyOf) involving property chains
- functional and inverse-functional properties (FunctionalObjectProperty, InverseFunctionalObjectProperty, and FunctionalDataProperty)
- transitive properties (TransitiveObjectProperty)
- keys (HasKey)
- individual equality assertions and negative property assertions

Compared to all of OWL2, this is clearly a limitation when it comes to expressiveness. Some constructs such as qualified number restrictions and disjunction would often be desirable to use. But there is a natural correlation between expressiveness and complexity. High expressiveness means high complexity and low complexity means low expressiveness (however note that low expressiveness does not necessarily imply low complexity, and that high complexity does not necessarily imply high expressiveness). So a trade-off must be made in *DL-Lite* if we are to be able to use it as intended, handling huge amounts of data (ABox instances). *DL-Lite* needs to have LogSpace complexity with regards to data to be able to use easily against relational

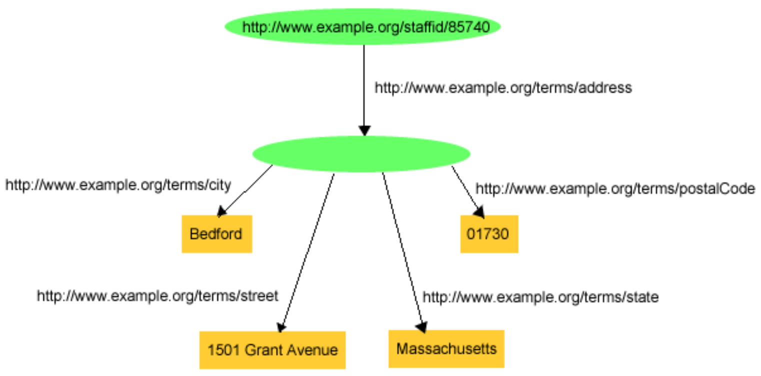
http://www.example.org/staffid/85740
http://www.example.org/terms/address
http://www.example.org/terms/city
Bedford
http://www.example.org/terms/postalCode
01730
http://www.example.org/terms/street
1501 Grant Avenue
http://www.example.org/terms/state
Massachusetts

Figure 3.3: *This graph show how n-ary relations can be handled in RDF and OWL.*

databases. Several variants of *DL-Lite* retain this property, with *DL-Lite*_A probably being the most interesting one.

3.4 OWL Challenges and limitations

While OWL is an expressive formalism, it has its limitations and there exist fields that are currently researched. I will briefly present a few of these here. Some that I stumbled upon while creating my ontologies will be explained more in detail in later chapters.

3.4.1 n-ary Relations

OWL only supports binary relations natively, and in many cases this can be a limitation. If we for instance want to create a structure for addresses and connect this to whoever lives at the address, then an n-ary relation would be a natural way to do this. We would then want a person in the relation as well as the city, state, street, and so on.

This is however not possible because OWL, and the description logic underneath it, only supports binary relations. For this problem reification is one solution. This means that we create a class out of the relation and then create instances of this class when using it in the A-Box. W3C has made an illustration of just this, which is shown in figure **??**.

In this example of a reification, we have an address relation from a person instance to an anonymous instance of some class that is the reified n-ary relation. From that instance then is relations to each of the individual components of the address: city, street, state and postalCode.

This approach is most likely the best solution there is to the n-ary relation problem, and also a good one when it is agreed upon and handled correctly. One can for instance use some sort of macro that behind the scenes translates the intended n-ary relation into a suitable OWL reification. This would however mean there must be some kind of middleware between the ontology and the application using it so that the meaning of the n-ary relation is properly maintained.

As the limitation of binary relations are an integral part of description logics, there seems to be no easy way to fix this at the core level. Solution such as using reification might prove the only possibility.

3.4.2 Mereology; partOf relations

Mereology[12] is the theory of parthood relations. It is an old philosophical discipline, and has thus a long history behind it. Many considerations has been made in a strict philosophical manner in this field, but from a computer science and engineering point of view, the results are fewer. Now as ontologies are gaining ground, these theories are finding concrete problems to be utilized in. This means that much work is being done that deals with computer-specific aspects. Efficiency, expressiveness and computability are factors that have to be considered in this respect. I will try to shed some light on this.

Properly representing such a part of relational hierarchy is something that can be potentially a large task. It can either be done very simple without much consideration on specialized use, or it can be made with many relations that are only to be used with certain classes. This approach would require much work, but might be worth it in a very expressive ontology. Below I present a few alternatives for creating a part of hierarchy, using the drilling domain to illustrate the possibilities and differences.

The four alternatives presented are:

1. large relational hierarchy with basically a distinct partOf relation for each use
2. a single (or few) partOf relation where the difference are made apparent through the classes
3. smart relational hierarchy based on theories in mereology
4. a single partOf relation where no distinction of use is made

[12]`http://plato.stanford.edu/entries/mereology/`

Alternative 1

Many subrelations of a "partOf" relation (structuralPartOf, functional-PartOf, ...) that through the relations give additional meaning to the partOf relation. Instead of using a single partOf relation for everything that is part of something else, several subrelations of partOf is added. This can be beneficial in that it will be easy to distinguish between different kind of partOf relations, and grouping them together.

The approach in this alternative has a high level of granularity with regards to partOf relations and class membership. This can best be illustrated through an example:

We have a class BottomholeAssembly, which is subclass of the following classes:

```
  TubularSection
  hasBHAComponent some BHAComponent
  hasDrillBit exactly 1 DrillBit
  hasDrillCollar some DrillCollar
```

To put it more verbosely, a BottomHoleAssembly is subclass of the conjunction of: everything that is a TubularSection, everything that has a BHAComponent, everything that has exactly 1 DrillBit, and everything that has a DrillCollar.

The relation hierarchy of interest here is:

```
hasPart
  hasPhysicalPart
    hasStructuralPart
      hasTubularComponent
        hasBHAComponent
          hasDrillBit
          hasDrillCollar
```

and the class hierarchy of interest here is:

```
TubularComponent
  BHAComponent
    DrillBit
    DrillCollar
```

meaning that each class has its own unique relation. While there are advantages to this approach, as described above, there are also problems. If we now create instances of each class in the BottomholeAssembly restriction, and define the instances of the same classes to be different from each other

[13], we see that our BottomholeAssembly instance must have the following:

- at least one harBHAComponent-relation to a BHAComponent instance,
- exactly one hasDrillBit-relation to a DrillBit instance,
- at least one hasDrillCollar-relation to a DrillCollar instance.

The problem with this approach is that if we make two DrillBit instances, and connect one through the hasDrillBit relation and the other through the hasBHAComponent relation, we will not have any inconsistency even though we would want (possibly expect) that to happen, as we have stated that we only want a relation to one DrillBit instance. The problem is of course that we have explicitly stated that we want one hasDrillBit relation to a DrillBit, and that does not exclude the possibility of several hasBHAComponent relations to other DrillBits.

This will however always be the case when using subrelations, so that defining the axioms using hasStructuralPart still makes it possible to add many instances using the superrelation hasPhysicalPart (and hasPart). It may indeed seem like using subrelations is risky business when it comes to reasoning, if the relations aren't used exactly as intended. Of course, since hasDrillBit is a subrelation of hasStructuralPart there are no problems using hasDrillBit even if the axiom is using hasStructuralPart.

One might wonder if it even is necessary to have such a high level of granularity in subrelations. As the axiom filler classes themselves are single classes, having single relations that are only used for those classes might essentially be to express the same thing twice. This might point in the direction that a single partOf relation without a hierarchy is good enough.

To use another example from a different domain. Consider food represented by a class "Food", and two relations "like" and "love". We make the assumption that "love" is a subrelation of "like", since it sounds reasonable that, everything that we love we also like. We may now set out to express a number of arbitrary axioms about a "Person" class where we express "Person"'s feelings toward "Food". Assume we have an axiom stating that "Person" can only love one Food, and possible other axioms. An ABox assertion stating that "Peter loves bananas" then will surely mean that we cannot also have an assertion "Peter loves apples". However, it is quite possible to say that "Peter likes apples". There is nothing odd about this at all, but it shows in a simpler way the difficulties in the previous example.

[13]The absence of the unique name assumption in DL makes it necessary to explicitly state that two instances actually aren't the same thing.

Alternative 2

The second alternative is to put some restriction on the class that is the whole to give an implication of what kind of part-whole relationship is relevant for the class. Then use an elementary partOf relation from the parts to the whole. Any meaning that implies structural part or functional part and such should then appear through the restrictions on the whole. To illustrate using an example:

```
    x partOf CirculationSystem
```

Since CirculationSystem is in this case thought to be a class that expresses not one single physical entity but is an abstraction where its parts share common properties, it thus has no structural parts. Any parts must then be of other kinds. To express that it has e.g. functional parts we might make it subclass of a FunctionalSystem, and then separate this from actual physical objects such that the use of partOf in this case is clearly different from something like "Engine partOf Car".

Alternative 3

Do something like what is described this Cognitive Science Journal article[10]. Here is described a more general view that is meant as a top-level partOf-hierarchy. They describe a separation into 6 different kinds of partOf / Whole relationships with examples:

Component / Integral Object	- handle / cup
Member / Collection	- tree / forest
Portion / Mass	- slice / pie
Stuff / Object	- steel / bike
Feature / Activity	- paying / shopping
Place / Area	- oasis / desert

This means that there will be several different partOf relationships. This may make it more correct with regards to reality, but will complicate things a great deal for the people using the ontology. Which relation to use may not always be obvious, and there might be cases when none really fits. This could be cleverly masked though, so that it will appear simple to the user which relation to choose in any case.

However as this is a sort of partitioning of partOf membership that is suited for top-level ontologies, it should not be something that every creator of ontologies must deal with. A much more likely scenario is that when an ontology designer connects an ontology to a top-level ontology, the lower-level partOf relations should be linked to the relevant top-level partOf relation. This speaks in the favor of multiple partOf relations, since if just one partOf relation is used there would arise complications when connecting to a top-level ontology with partOf relations such as the ones described here.

Other similar descriptions of mereology exist, such as the Stanford encyclopedia of philosophy `http://plato.stanford.edu/entries/mereology/`

Alternative 4

The last alternative is not to make any distinction between different kinds of partOf relations at all. This would be a very simple approach with a single partOf relation. Its strength is obviously the simplicity and ease of having fewer relations to worry about. Whenever a part-whole relationship is needed, this single relation would provide that. In ontologies that describe very limited domains with similar kinds of classes, this kind of approach may suffice. Especially in cases where the ontology is to be linked to a top-level ontology as described in the previous section, the simplicity of the ontology could either be a positive or a negative thing. A small limited domain ontology where most part-whole relationships are of similar type would be easy to align with a top-level ontology using a more complex partOf hierarchy. A large domain ontology that uses a single partOf relation for many kinds of part-whole relationships would however require many changes to align it with a top-level ontology.

These considerations should be able to serve as a guideline to when a single partOf relation is enough.

Which one to use when

The question of which of these approaches (or even different ones) are best to use in an ontology, is not easy to answer. From what I can gather at this time, the purpose and the level of general granularity of the ontology being constructed dictates which of these alternatives is best to use. There do exist some guidelines for how this should be conducted, such as the W3C article on this that contains some good pointers, `http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/`. But still this problem requires further inquiry to create a methodology for choosing the best variant in each case.

In the drilling ontology, which will be described in the next chapter, I have opted to a combination of the first and second alternatives. This both since I have been unsure of which one to choose, but also to test reasoning capabilities and get a feel for the differences.

3.4.3 Multiple path problem

The problem with multiple paths is one that is well-known among researchers that know OWL, but not so much among people who are little by little trying to use OWL in commercial applications. In many cases their ontologies might of such a simple nature it won't be a problem anyway. But the problem exists and appears quickly in more complex ontologies. That being said, it is easy enough to make tiny ontology where this problem arises.

What I here mean by path is a set of TBox axioms linking classes together in some way through relations. An example of this could be:

$$Person \sqsubseteq \exists worksFor.Company$$

$$Company \sqsubseteq \exists owns.Equipment$$

With these two axioms there is a conceptual path from Person to Company to Equipment. This by itself poses no problems. Now add a third axiom:

$$Person \sqsubseteq \exists worksOn.Equipment$$

This axiom creates a different conceptual path from Person to Equipment. Judging from the way both these paths connect Person with Equipment, it would seem fair that if something were to happen with the equipment the company owns, then this would mean that something happens with the equipment the person works on. If we now add an axiom that says that "A poor company has only broken equipment":

$$Company \sqcap PoorCompany \sqsubseteq \forall owns.BrokenEquipment$$

we would expect that the equipment the person is working on is also broken equipment, as expressed in the axiom:

$$Person \sqcap \exists worksFor.(Company \sqcap PoorCompany) \sqsubseteq \exists worksOn.BrokenEquipment$$

This is however not the case. The reason for this lies in the tree model property of OWL, which says that if an OWL ontology has a model, then it has a model with a tree-like relational structure as well [11]. This means that even though there exist a model for the ontology where the person works on broken equipment, there also exist a model where the person works on good equipment. And since reasoning is only able to derive what is true for all models, there is no way that the last axiom can be derived. What this implies is that even though we can describe arbitrary relational structures in OWL, we are only able to reason on a "tree-like subset" of the ontology. A graphical representation of the example is shown in figure 3.4.

There is no easy solution to this problem, but [11] proposes an extension to OWL called structured objects that deals with the problem in a nice way.

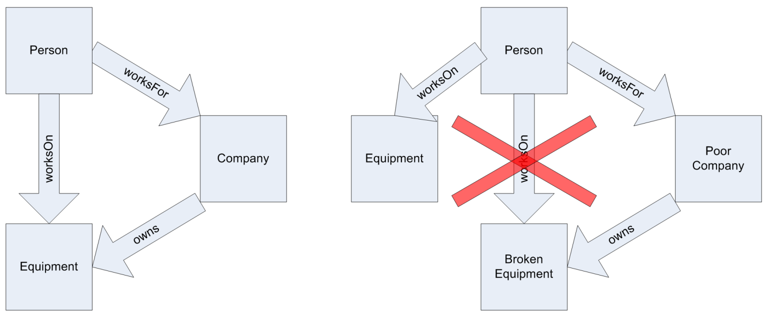

Figure 3.4: *This graph illustrates the example of multiple paths. The figure on the left shows the state of the three first axioms. The figure on the left shows the state we get when the new axiom is added.*

It basically proposes to create arbitrary graph structures within the TBox of the ontology in a way such that they are separated from the rest of the ontology. They can then have SWRL-like rules work on them to fix the problem illustrated here without sacrificing decidability.

3.4.4 Consequences that cannot be expressed directly

There are certain consequences through reasoning that cannot be properly expressed in OWL as it is currently. This limitation has close ties to the reason for the multiple path problem previously described. Below I will describe two such limitations expressed in a traditional way similar to first order rules. The lack of variables in OWL is again what makes this problem appear. We thus cannot capture ABox-consequences such as:

```
Person(x) and Woman(y) and hasSibling(x,y) -> hasSister(x,y)
```

Or such as:

```
Tubular(x) and TubularComponent(y) and hasPart(x,y) ->
 hasTubularComponent(x,y)
```

In the first example here, it is natural for us humans to immediately understand that a sibling that is a woman must be a sister. The second example is equivalent to the first, only it uses names from the drilling ontology. It is desirable to be able to express this in ontologies as well. But this is not possible to do directly without utilizing tricks which make things more complicated that necessary.

What we want to be able to do is to axiomatize in the TBox the example from above, such that when a person has a sibling that is a woman, the axiom will tell us that the woman is the sister of the person.

This can be solved by the following:

1. Composition
   hasSister = hasSibling o womanID$^{\Delta}$
   woman$^{\Delta}$ is here the diagonal of the class Woman, which is the same as the identity function of the class. The significance of this is further described below. This is the best solution currently available if we want to restrict ourselves to OWL only. It makes things complicated though, and reasoning on compositions is not possible in *DL-Lite*.

2. Use a rule language such as SWRL for such consequences instead of adding anything to OWL. This has its advantages and disadvantages. It saves us from adding unnecessary complexity to the ontology, but it makes us rely on a formalism outside of OWL that in many cases is not decidable.

3. Adding construct to OWL that reflects the composition above.

I will focus on the first possible solution to the problem and outline it a little further. OWL2 has added among other things the possibility to use compositions of relations, which means that this kind of solution is in principle possible. Logically what this means is if we have two binary relations $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ then the composition $R \circ S$ is defined as

$$R \circ S = \{(x, z) \in X \times Z : (x, y) \in R \wedge (y, z) \in S\}$$

In the example from above R is hasSibling and S is womanID. In the case of womanID it is a diagonal relation, meaning that the domain and the range of the relation are equal (womanID $\subseteq Y \times Y$), then we extract the diagonal [14] from this cross product of the set shared by the domain and the range. This sort of relation is not by default supported by OWL, so a proof is needed to make sure it can be used without any problems.

Diagonal of a set

Problem description: To define the diagonal $A^{\Delta} = \{(a, a) : a \in A\}$ of a set A.

Axioms We introduce a primitive relation dA and axiomatize it as follows:

[14]the identity such that the set of the diagonal is equal to $\{(y, y) : y \in Y\}$

$$\exists dA.\top \sqsubseteq A \qquad \text{(Domain of } dA \text{ is upper-bounded by A)} \tag{3.1}$$

$$A \sqsubseteq \exists dA.\top \qquad \text{(Domain of } dA \text{ is lower-bounded by A)} \tag{3.2}$$

$$\top \sqsubseteq \leq 1 dA.\top \qquad \text{Functionality of } dA \tag{3.3}$$

$$\exists dA.\top \sqsubseteq \exists dA.Self \qquad \text{Reflexivity} \tag{3.4}$$

Proof of correctness Let A be a concept name. We show that $(a,a) \in (A^{\mathcal{I}})^{\Delta} = dA^{\mathcal{I}}$ for all interpretations $\mathcal{I}$ of the axioms.

LHS $\subseteq$ RHS: Suppose $(a,a) \in LHS$. Then $a \in A^{\mathcal{I}}$, whence $a \in (\exists dA.\top)^{\mathcal{I}}$ by axiom 2. Hence $a \in (\exists dA.Self)^{\mathcal{I}}$ by axiom 4. It follows by the definition of $\exists R.Self$ (where R is any relation), that $(a,a) \in dA^{\mathcal{I}}$, which is what we wished to show.

RHS $\subseteq$ LHS: Suppose $(a,b) \in dA^{\mathcal{I}}$. It suffices to show that $a \in A^{\mathcal{I}}$ and that $a = b$. It follows from the assumption and the semantics of existential restrictions that $a \in (\exists dA.\top)^{\mathcal{I}}$. Hence, we have $a \in A^{\mathcal{I}}$, by axiom 1. It remains to show that $a = b$. Since $a \in (\exists dA.\top)^{\mathcal{I}}$ we have that $a \in (\exists dA.Self)^{\mathcal{I}}$, by axiom 4. Hence $(a,a) \in dA^{\mathcal{I}}$. Now, $(a,b) \in dA^{\mathcal{I}}$ too, whence, since dA is functional by axiom 3, it follows that $a = b$ as desired.

$\square$

Once this proof is settled, we need to make sure that this composition in OWL actually yields the consequences we are after. By the definition of composition the hasSister will now be the composition of hasSibling and womanID, but it is also interesting to see that when actually used in an ontology, reasoning gives us the results we want. What we want to show is that when a Person that has a sibling which is a Man, this sibling is actually inferred to be a brother, and likewise with Woman and sister. To show this, we establish the following class hierarchy:

```
Person
    Woman
    Man
```

then we establish these relations:

```
dMan
dWoman
hasSibling
hasBrother
```

```
    hasSister
```

where dMan and dWoman are the diagonals of Man and Woman. The relaions hasBrother and hasSister must have defined the crucial compositions that are what make this work. For hasBrother this composition (also called property chain) is

```
    hasSibling  o  dMan  ->  hasBrother
```

and for hasSister it is:

```
    hasSibling  o  dWoman  ->  hasSister
```

What remains now is the proper axiomatization of the diagonals. In the OWL files `http://heim.ifi.uio.no/larsove/ontology/siblingsister2.owl` and `http://heim.ifi.uio.no/larsove/ontology/siblingsister.owl` are two attempts at this. In the first file I tried to axiomatize the diagonal completely in the RBox (all the axioms restrict the dMan and dWoman relations directly) by using the Functional and Reflexive RBox constructs. This did not work properly however, as the reasoner inferred that dMan and dWomen were equivalent relations. The reason for this is that Reflexivity and Functionality are interpreted as properties on all individuals in the domain, and not just the individuals of Man and Woman.

To deal with this problem I made the second version where the diagonal relations are axiomatized in their respective class instead. This version functions flawlessly and a Person having a sibling which is a Man is inferred to have a brother; likewise with Woman and sister.

Aside from the extra axioms that increase the size of the ontology, there is a slight problem. In the relation hierarchy it would be nice if we could have hasBrother and hasSister as sub-relations of hasSibling. But this is unfortunately not possible. Since the compositions in hasBrother and hasSister include the hasSibling relation, we cannot at the same time have the relations be sub-relations of hasSibling.

A similar problem where composition can be utilized

Also, we have a related case where compositions can be used as a solution. We again have an ABox-consequence that cannot be captured:

```
Person(x) and hasFather(x,y) and hasBrother(y,z) -> hasUncle(x,z)
```

To express this we can create the composition:

```
hasFather(x,y)  o  hasBrother(y,z)  ->  hasUncle(x,z)
```

This will work nicely (`http://heim.ifi.uio.no/larsove/ontology/uncle.owl`), but there are problems concerning this. We would want the relation hasUncle to be irreflexive and asymmetric to avoid cases where a person is uncle to himself, and **person x** is uncle to **person z** while at the same time **person z** is uncle to **person x**. Unfortunately the OWL2 RBox constructs for irreflexivity and asymmetricity cannot be used when the **hasUncle** relation is defined by a composition.

3.4.5 Scalability and sublanguages

One thing that needs to be considered when creating an ontology, is how scalable it needs to be. For this we must obviously know something about what its intended use is. An ontology that will handle huge amounts of ABox data will require other properties than an ontology that is mostly going to be used solely as a TBox terminology.

Before OWL2, there were not much in the way of standardizing simpler fragments of OWL, but still a lot of research and development were done to create less complex ontology languages for more specific purposes. Most of these focus on reducing the expressivity of OWL to be able to handle large amounts of data more efficiently. When using the constructs OWL provide, it is fairly simple to create an ontology that scales horribly both with regards to the size of the TBox and the size of the ABox. For example, when computing concept satisfiability a complexity class of ExpTime and even worse can be obtained simply by the use of disjunctions[15].

To cope with these problems, W3C adds with the standardization of OWL2 also three "profiles" (ref http://www.w3.org/TR/owl2-profiles/), which are such fragments of OWL2 aimed at particular uses. These three are together with their main properties:

OWL 2 EL

Quote from `http://www.w3.org/TR/owl2-profiles/`

> OWL 2 EL is particularly useful in applications employing ontologies that contain very large numbers of properties and/or classes. This profile captures the expressive power used by many such ontologies and is a subset of OWL 2 for which the basic reasoning problems can be performed in time that is polynomial with respect to the size of the ontology [EL++] (see Section 5 for more information on computational complexity). Dedicated

[15] [12] section 5

> reasoning algorithms for this profile are available and have been demonstrated to be implementable in a highly scalable way. The EL acronym reflects the profile's basis in the EL family of description logics [EL++], logics that provide only Existential quantification.

OWL 2 QL

Quote from `http://www.w3.org/TR/owl2-profiles/`

> OWL 2 QL is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL 2 QL, conjunctive query answering can be implemented using conventional relational database systems. Using a suitable reasoning technique, sound and complete conjunctive query answering can be performed in LOGSPACE with respect to the size of the data (assertions). As in OWL 2 EL, polynomial time algorithms can be used to implement the ontology consistency and class expression subsumption reasoning problems. The expressive power of the profile is necessarily quite limited, although it does include most of the main features of conceptual models such as UML class diagrams and ER diagrams. The QL acronym reflects the fact that query answering in this profile can be implemented by rewriting queries into a standard relational Query Language.

OWL 2 QL the OWL2 profile that is based on *DL-Lite*, so this is quite relevant for the thesis.

OWL 2 RL

Quote from `http://www.w3.org/TR/owl2-profiles/`

> OWL 2 RL is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate OWL 2 applications that can trade the full expressivity of the language for efficiency, as well as RDF(S) applications that need some added expressivity. OWL 2 RL reasoning systems can be implemented using rule-based reasoning engines. The ontology consistency, class expression satisfiability, class expression subsumption, instance checking, and conjunctive query answering problems can be solved in time that is polynomial with respect to the size of the ontology. The

RL acronym reflects the fact that reasoning in this profile can be implemented using a standard Rule Language.

3.4.6 Modularity

As ontologies grow larger and larger, it might not be desirable to keep everything in one monolithic ontology, but rather to split it up into several smaller modules. This could either be a static configuration, or perhaps even dynamic modules that are extracted based on which part of the ontology we are interested in at a given time. When we create an ontology from scratch, it might also be prudent to start modularizing immediately to avoid extensive work later. In many cases there will also be existing ontologies we would want to import into our own and use parts of. For all these cases a good theory must lie behind that ensures that there will be no problems with this modularization. [13] establishes a theory behind modularization and presents a relatively simple approximation that extract modules which are close to the optimal solution; the module that consists of exactly what we need and nothing else. This article bases much of its results on the concepts established and described in [14]. As the details of this theory are somewhat extensive, I will not tackle them in this thesis.

I will however approach modularity from a higher perspective and look at how it might be relevant for the construction of a drilling ontology. A few examples of modularization use that might be appropriate in different cases:

- split up existing ontology into static parts
- split up existing ontology dynamically on request
- create ontologies from scratch that together describe a particular domain
- importing ontologies other people have created

As will be seen in the next chapter, what I have eventually chosen to do with the Drilling Ontology is to split it up to accomodate various uses while retaining a common core set of axioms.

Chapter 4

Creating the drilling ontology

In this chapter I will present how the sources of domain knowledge influenced the creation of the drilling ontology. Then I will describe in part the process of building the ontology. This section involves both the concretization of drilling domain knowledge, as well as some considerations in ontology engineering. Finally I will present parts of the drilling ontology itself.

4.1 Source contributions

In chapter 2 the most important sources for domain knowledge were presented one at a time. Now the focus will be on combining these sources and how to extract relevant knowledge to create a drilling ontology. I will first go through each of the sources and describe how they influenced the creation of the ontology. Some sources has affected specific parts of the ontology, while others have been more about the overall picture. This will be somewhat elaborated later in this chapter. Then I will present how the knowledge gained from each of the sources affected each other and lead to the drilling ontology. It is worth noticing that the ontology is by no means a finished product yet, and other considerations especially by domain experts may lead to different drilling ontologies in the future.

4.1.1 WITSML

WITSML (wellsite information transfer standard markup language) is an important standard when it comes to transferring drilling data between various facilities onshore and offshore. As such it naturally contains much low-level information such as data from measurements and how to properly represent and identify these. This provides a good foundation to building the parts of the ontology that deals specifically with low-level data[1]. This is clearly the sort of data that often need to be integrated. Enabling the ontology to deal with this kind of data is therefore paramount to its success in data integration.

However, in a more general-purpose drilling ontology describing only low-level data by itself will not suffice. There is in any domain high-level knowledge[2] as well. And properly connecting the high-level concepts with low-level data is important for the entirety of the drilling ontology. I therefore also want to check if there is any such high-level knowledge in WITSML; meaning I want not only to extract the data values from measurements, but also how and by what machinery these were measured, and how they relate to other parts of the ontology. For this purpose the XML schemas where the standard is defined is what needs to be closer examined.

There are 24 schemas which are defined as the top level, meaning that the actual document files are created in adherence to these 24 schemas. The rest of the schemas are included by other ones and participates in creating a hierarchy of complex XML datatypes which has elements that in turn have other datatypes and so on until the lower levels that consist of simple datatypes. This hierarchy of schema files the defines the many datatypes in use is a valuable resource for an ontology that is to be able to handle WITSML data. The simplest part to start with is the definitions of these datatypes starting from the lower levels. For actual measurement data the schema files define relatively simple types such as string and numbers at the bottom and elaborate upon them creating complex datatypes that have both a value and a denominator. This datatype is then used in a particular place in a schema file[3].

An example of this among many is **obj_cementJob** which has an element **mdHole** that has type **measuredDepthCoord**. This type in turn is a complex type consisting of a value content with type **abstractMeasure** and denominator which is the unit of measure **MeasuredDepthUom**. This unit of measure is an enumerated data type with the possible value **m,**

[1]Low-level data means most information that can be expressed easily using common numerical and textual datatypes and which are largely atomic in that they cannot be subdivided. It mostly refers to numerical measurements/calculations and simple strings.

[2]High-level knowledge is in contrast to low-level data information about relations between concepts that can either be physical or abstract entities. Not to be confused with upper ontology!

[3]An important consideration to make here is that even though everything in XSD are defined as datatypes, we want to make a distinction between types that describe measurement data and types that describe the documents which contains these data. Unfortunately this distinction is non-existent in WITSML

ft, ftUS. The type **abstractMeasure** is created as a common type for all types that have some unit of measure, and the value is defined to be **abstractDouble** which in turn is defined as the XSD double where the empty value is disallowed.

As is apparent from this example, the type hierarchy in WITSML is complex and very many similar cases like this exist. Apart from the datatypes of the measured values themselves, enumerated datatypes also are an important part. Not only in the definition of units of measure as seen in the example, but also many other kinds such as types of material, types of tubular components and type of risk subcategory. All of these will have to be represented in the ontology for it to handle all of WITSML properly.

High-level

When it comes to high-level knowledge WITSML is not nearly as rich as when it comes to low-level data. Since WITSML is built upon XML it per definition describes a format for a specific document structure with content. As such it does not distinguish between low-level data and high-level descriptions. Since everything in XML schema are datatypes there is no conceptual difference between any low and high level of knowledge. But for an ontology this is a much more interesting question. What we then need to do is try to extract knowledge about what the documents describe implicitly. There are two distinct cases of higher-level knowledge I have been able to extract; one related to physical structures and components, and one related to a process<->object distinction.

Physical structures and components

There are several references to physical structures and components in WITSML. This is the kind of information that will help put the measurement data in a broader context. For instance the manner of how drillstrings/tubulars are composed is extensively described. This is done through large sets of enumerations of different types of components, which are in turn used in complex datatypes to create a representation of a partOf relationship. To illustrate what such a partOf hierarchy can look like:

```
Tubular1
   TubularComponent1.1
      Jar1.1.1
      Connection1.1.2
   TubularComponent1.2
      Connection1.2.1
```

This just shows how each entity is part of a larger whole. All of them of course many other attributes as well. The various type of tubular components are listed in an enumeration, and each tubular component element has to specify what type it is. This differs from the parts that a tubular component in turn can have. In the example above **Jar** and **Connection** are already "atomic" entities in that they have no further specialization in the form of a type. A few differences in representation like this exist in WITSML, and will have to be handled properly when creating the ontology. Also, there unfortunately is no information in WITSML that indicates what kind of parts a certain kind of tubular component has. Meaning that the person or computer that creates a WITSML document receives little feedback on what is allowed and not.

Process-Object distinction

Some of the 24 schemas describe objects at a certain time x, while other describe processes lasting from time x to time y. Thus there is a distinction here that should be made, but this is not clearly stated in any way in the WITSML standard[4]. From observing in which fashion each document uses elements with the type **timeStamp**, I have come up with the following grouping of documents:

```
Object
    fluidsReport
    log
    message
    mudLog
    opsReport
    realtime
    rig
    surveyProgram
    trajectoryStation
    wbGeometry
    well
    wellbore
    wellLog

Process
    bhaRun
    cementJob
    convCore
    risk
```

[4] as far as I have been able to find out

```
    sidewallCore
    trajectory

Neither
    dtsInstalledSystem
    dtsMeasurement
    formationMarker
    target
    tubular
```

Note that the documents that ended up in "Neither" often also have some associated time, but this does not mean they necessarily describe processes or objects. Many of them are simply aggregations of data values that relates to some other document which in turn have a proper reference point in time. The same is true for many of the documents put in "Object", however they themselves have a fixed time reference so they are at least initially put among objects. All the documents grouped under "Process" have defined clearly a start time and end time.

The grouping done above only reflects how the documents deal with time, which in itself is a difficult thing to handle in ontologies, but there are other aspects to what constitutes an object and a process as well. The section above concerning physical structures will have relevant to what constitutes a proper object in an ontological sense. In the case of documents such as **log**, **message** and **mudLog** they do not describe any objects which exist outside of the documents themselves. They are simply collections of measurement data gathered at a certain time. In contrast, things like tubular and well are clearly physical objects, yet the documents do not provide a time value for them. When considering this, it is clearly necessary to create a more detailed grouping of the document.

```
PhysicalObject
    rig
    trajectory*
    tubular
    wbGeometry*
    well
    wellbore

AggrevativeObject
    fluidsReport
    log
    message
    mudLog
    opsReport
    realtime
    surveryProgram
    trajectoryStation
    wbGeometry*
    wellLog

Process
    bhaRun
    cementJob
    convCore
    risk
    sidewallCore
    trajectory*

Neither
    dtsInstalledSystem
    dtsMeasurement
    formationMarker
    target

    * is explained below
```

Even though the grouping above expresses what the documents describe in a better way, that does not mean the ontology can be created on the basis of this alone. Some of the documents can be interpreted in ways that are not specified clearly in WITSML, but can nonetheless be correct ways of viewing them. For instance the **wbGeometry** document describes physical casings that are placed inside wellbores to protect them from collapse, but from the criteria concerning the use of time and data that are examined here they fit just as well among aggregative objects. Another such case is **trajectory**, which from the WITSML documents is described as having measurements starting and stopping at certain points in time. A bit of insight into the domain would however indicate that a trajectory is not a process but an object/entity describing in this case the path which drilling follow. Ambiguities like this cannot be solved simply by examining the documents, but is dependant upon WITSML and domain experts.

Continuing on the topic of distinguishing objects from processes, I will present an example from WITSML that illustrates both the difference between the two, and also some interconnectiveness between documents. Many of the document types must refer to which well and which wellbore they belong to, these wells and wellbores being defined in some other document. But other than this, documents of a certain schema are mostly independent from the rest. If we now have a **well**, a **wellbore**, a **tubular** and a **bhaRun**, the schema files defines how these must refer to each other.

- The **well** is on the top in that it has no references to other documents.
- The **wellbore** must refer to a **well**
- The **tubular** must refer to both a **well** and a **wellbore**
- The **bhaRun** must refer to one of each of the above.

This is interesting because it shows a relationship between physical objects and a process. Such relationships are valuable for an ontology that is going to describe the domain on both a low and a high level. Unfortunately there are not many other similar relational structures in WITSML.

Connecting the low and high level

So in what way does the measurement data and other low-level information relate to the considerations made concerning what the documents describe? In the example above we can see that there is at least some references between documents. But there is not nearly enough to create an expressive ontology. This means that linking low-level data vertically to the entities above it will be fairly easy, but connecting data from different documents that may be related is going to be more difficult. There are often references to well and wellbore, and in the case above from bhaRun to tubular. But that is basically all there is, and how the data from the various documents relate is not handled at all. This small amount of interdependence between documents is one of the weaknesses in WITSML, which is largely felt when querying databases of WITSML data. It implies that information about connections between data from different documents will have to be gathered elsewhere.

Summing up

To sum up, what WITSML provides is an extensive set of low-level data descriptions. This is done through hierarchy of datatypes to describe all the various kinds of measurements, as well as enumerations on types of components, states, denotations and so on. It then gives these types some context by linking them to element values in the many documents types that WITSML defines. The strength of WITSML lies clearly in the low-level data. There are links between these data and higher-level abstractions as well, but WITSML does not distinguish between any low and high level of representation, since in the XSD formalism used to define WITSML everything is defined as either simple or complex datatypes. Also between the entities of high-level information there are implied very few relations. There is for instance nothing that restricts what data elements a tubular component must have with regards to the type of the component.

WITSML offers much when it comes to data representation, which is an important part of integration, but other sources will have to provide the bulk of the higher-level relationships.

4.1.2 ISO 15926

The POSC Caesar life-cycle ontology standardized in ISO 15926 is being more and more actively used in the gas and oil business. This makes it interesting to see what it might contain concerning drilling.

ISO 15926 is a huge structure with thousands of classes, and is as such difficult to navigate through. Finding relevant knowledge about drilling have proven difficult, either because it is not there or because it is represented in non-obvious ways. One of the things that it does contain, is much information from WITSML. This will be more closely examined in the next section.

As for any relevant information in ISO 15926 not from WITSML, there seems to be many concepts that relate to oil drilling. The problem when it comes to ontologies, is that the many concepts that are interesting are not related to each other. They are just placed into the general ISO 15926 hierarchy, e.g. saying that "well casing" is subclass of "lining" and has classification "mechanical equipment class". We would like to have some information regarding how "well casing" relates to other classes, but no such information is provided. When it comes to domain knowledge ISO 15926 has something to offer when it comes to the subclass hierarchy, but not much else as of yet.

Another aspect where ISO 15926 has more to offer is as an higher or upper ontology. Connecting the concepts in the drilling ontology to the ISO 15926 part2 as a higher/upper ontology is beneficial if it is to be compatible with all the knowledge outside of the drilling domain that ISO 15926 contains. Also, as application based on ISO 15926 are created, compliance with part2 assures that the drilling ontology is in some way compatible with these applications. However the formalism used to express the ISO 15926 part2 is different from the description logics that OWL is based on. It will therefore not be easy to achieve compliance. The drilling ontology does not require ISO 15926 to function, so it is in any case something which can be added at a later point.

The new version of part 2 that is being written in OWL is a different matter however. The formalism is of course compatible, so what remains to see is whether or not the upper ontology modelling can be used with the drilling ontology. Unfortunately this ISO 15926 OWL version did come into existence until quite late in the work on this thesis, so I have not been able to investigate whether it can be used or not.

4.1.3 ISO 15926 and WITSML

ISO 15926 might rather prove useful at the very low level, beginning with datatypes and data from various sources such as sensors and other measurements. As a part of the IIP-project WITSML data was shuffled into ISO 15926. All datatypes, things that are defined as XML schema simple or complex types, have been added to ISO 15926. There has however, not been done much work trying to properly integrate the WITSML types with the existing ISO 15926 datatypes. Some work has been done, such as redeeming WITSML low-level types with the existing EXPRESS[5] based ISO 15926 types.

Unfortunately when adding WITSML types to ISO 15926, hierarchical information has been lost. It appears that only some of the low-level datatype information is "properly" included. From what I can see, the enumerated datatypes are added so that all the enumerated values are included. Also, all the units of measures from WITSML are included. In both these cases though, the references to extensions are lost (subtypes/supertypes). For instance, the WITSML datatype "massConcentrationMeasure", which is the datatype actually being referenced for use in an XML document is not included at all. Rather "massConcentrationUom", a unit of measure, is what is added to ISO 15926. It is correctly classified as a unit of measure and has been added to the ISO 159265 hierarchy. The problem is that there is no reference to how it fits in the WITSML hierarchy. The WITSML schema states that "massConcentrationMeasure" has a unit of measure "witsml:massConcentrationUom" and has supertype "witsml:abstractMeasure" which has supertype "witsml:abstractDouble" which has supertype "xsd:double". All this information is lost. This means that even if the low-level xsd and witsml datatypes are properly aligned with ISO 15926, there will be no relation to the more complex types of WITSML.

Now what does this mean for the creation of an ontology based on WITSML and ISO 15926? This means that using the WITSML parts of ISO 15926 directly is probably not a good idea. Relations are crucial to an ontology, and with so few relationships from WITSML added to ISO 15926, much potentially useful information is lost. When not even the explicit relational information from WITSML is added to ISO 15926, looking for the implicit

[5]http://en.wikipedia.org/wiki/EXPRESS_(data_modeling_language)

relations will most likely be even harder. The WITSML data added to ISO 15926 might however be useful for an ontology based on WITSML to interface against ISO 15926. At least there will exist a starting point for further integration.

4.1.4 Schlumberger Oilfield Glossary

This is an online repository, freely available, that contains much information about Oilfields operations and equipment. Within this repository there is a large section concerning drilling `http://www.glossary.oilfield.slb.com/search.cfm?Discipline=Drilling` that contains much useful information that can be utilized when creating an ontology. Each entry in the glossary describes the term and also links to documents describing similar terms and related terms. For instance **Rig** links to **mud**, **derrick**, **mast**, **topdrive** and others. These links can be used to get an idea of how a term relates to other terms, which is essential when creating an ontology. It is however limited in the way that we seldom get any additional information about what sort of relation we have. From the text one can perhaps in certain cases assume that there is a structural “part-whole” relation, and in other cases some sort of “uses/utilizes” relation. Also we get no bounds and restrictions on the relations. Without an domain expert to explain further just what kind of relationship we are dealing with it is difficult to get high quality knowledge.

But the glossary still is a valuable source of gaining intuition and understanding necessary for building a good ontology. And with the cooperation of a domain expert ontology building can be conducted at a faster pace with resources like this handily available.

4.1.5 AKSIO

AKSIO with its 235 classes contains much useful information that the other sources presented do not deal with. Much of this information is about events and states. But there is also information about equipment, operations, and other relevant concepts. It is however mostly a taxonomy and while it does include 17 relations that are used to a varying extent, there is not much information about relationships. But in return the relationships it does contain are quite useful to build upon and use as a basis. Especially relating events to states is one of the strengths of AKSIO. An example of such a relationship is:

```
PackOff  causes  LostCirculation or StuckPipe
```

where **PackOff** is an event and **LostCirculation** and **StuckPipe** are states. In a similar fashion there are relationships between equipment and operations:

```
Casing  implementsBarrier  Drilling or Intervention or Production
```

Where **Casing** is equipment and **Drilling**, **Intervention** and **Production** are operations. Contrary to the previous example however, this time the meaning of the relation is not obvious. While **causes** is easy to understand and pretty much unambiguous, **implementsBarrier**, which is not explained in any way in AKSIO, is more cryptic. This again shows the need for domain experts to guide and advice whenever difficulties arise.

I have also done a IOHN deliverable on the conversion of the AKSIO ontology to OWL-DL, that is located in `http://heim.ifi.uio.no/larsove/master/IOHN/IOHN_deliv_aksio_merged.pdf`.

4.1.6 Domain Experts

Domain experts differ from the other kinds of sources of knowledge, in the obvious way that they are not documents and repositories, but actual people. This naturally make them a different source to relate to. But it also mean they are potentially the most valuable source, since the communication in this case is going both ways. As opposed to written documents where the ontology engineer can only read and try to understand, when a domain expert is available anything that is unclear can be resolved simply by asking the right questions. Besides helping with refining and correcting the modelling done on the ontology, they also can provide much new information according to the needs of the ontology engineer.

There are various ways of extracting good knowledge from domain experts. An obvious way is to work closely with the expert when building the ontology and this way get continuous input on important modelling decisions. But often, as is the case for me in this thesis, the access to experts is limited. In this case other ways of efficiently getting good knowledge need to be attempted. With limited time available for meetings and such, the ontology engineer needs to be well prepared to ask the right questions that are difficult to figure out. For this reason it is prudent that the ontology engineer has gotten some insight into the domain through other sources even before meeting with the domain expert for the first time.

Another way of extracting information that is not in real-time is by asking very simple but precise questions that the expert can answer at his/her own leisure. I have described such a method in further detail in the appendix A.1 with an example of use included.

4.2 Overall ontology structure

When working to combine sources into an ontology, it is helpful to have a goal and purpose of the knowledge gathered in mind. With a clear goal and perhaps even a rough path there, we can more naturally discard unwanted information and focus on what we want from all the sources.

In the case of this thesis the overall goal is to create a drilling ontology and our sources are the ones mentioned above. Simply having the goal of creating an ontology is still a difficult and open problem. A general drilling ontology, or a general ontology in any domain, that can be used for anything concerning drilling seems quite difficult, as we can never anticipate all the uses people might have for the ontology. This means that having a few concrete uses for the ontology will make it easier to combine the sources in a meaningful manner and construct the ontology.

As have already been presented earlier in this thesis, arguably the most important use for the drilling ontology is data integration and conversion. This means that when designing the ontology this particular purpose should be considered throughout the entire process. But at least some degree of generic knowledge representation of the drilling domain is also interesting to be able to capture. Both to get an extensive ontological structure of the domain and to accommodate for future uses of the ontology. As has been discussed in the previous chapter, OWL comes in several variants with varying degree of expressivity and complexity. Just as data integration is largely dependent upon a less complex and quite efficient ontology, other uses may need a more comprehensive or altogether different way of formalising parts of the domain.

In light of these considerations, I have after several iterations of trial and error in building a drilling ontology arrived at the conclusion that a modular design based on a common core with appropriate extensions is best. The modules that the drilling ontology will consist of is in its current iteration are:

- A **Core module** that largely consists of a taxonomy of the most important classes for representing the drilling domain. Exactly what classes should belong here is something that may change, and smaller modules may be separated out of the core. In addition to having a taxonomy, the core should have the most general relations that are universal for the domain. It is important that the axioms of the core are no more expressive that *DL-Lite* can handle, or else the *DL-Lite* module that extends the core would not actually be in *DL-Lite*. Chapter 3 describes exactly what is allowed in *DL-Lite*.
- A ***DL-Lite* module** that has the purpose of being a data integration

ontology. Other than the fact that QuOnto uses *DL-Lite* ontologies, it is an efficient OWL variant that can be used in tools similar to QuOnto. This module will mostly add axioms using the existential quantifier to relate classes to each other through relations.

- A **Universal quantifiers module** that is not inhibited by the *DL-Lite* restrictions. This module is free to use any OWL2 constructs available. As the name of the module implies, it adds axioms using universal quantifiers to further enrichen the domain ontology. Since this module does not use any existential quantifiers, it is meant to be possible to create an additional ontology which includes both the *DL-Lite* module and this module for an even richer ontology.

Although this approach seems like a good direction to move in, the best level of modularity is perhaps not yet reached. The split into modules described here partly reflects the purposes presented in the thesis. But especially the core could yet be split into smaller parts. It might be prudent to create an own module for low-level data representation, and keep the core at generally a higher level. This could be tackled in later derivative/similar work.

I also have done a IOHN deliverable on this ontology structure more aimed specifically at WITSML. It is to be found at `http://heim.ifi.uio.no/larsove/master/IOHN/IOHN_deliv_witsml_superstructure_merged.pdf`.

4.3 Combining the sources

In this section I will first regard the overall combining of sources needed for creating the modules in the ontology, and then say a few things about each of the modules and how they differ.

Finding the sources and describing them one by one is one thing, combining them is another. Different sources often have different ways of representing the knowledge we are after, in such a way that the same information may reside in two sources and yet we are not immediately able to tell that they are the same. Also pieces of information that are to related to each other must be identified properly. So a considerable part of the work has to go into normalizing the sources; finding out when two different sources are describing the same thing, and finding out when they complement each other. In some cases two sources may approach the domain of interest in completely different ways, potentially making this job more difficult.

Common for most of the sources described above is that they provide large amounts of information, but they don't say much explicitly about relations between the various instances of information. This means that in part the combining can be done by taking concepts from the different sources and creating a taxonomy simply based on the meaning of the concepts themselves. In the cases where concepts relate to other concepts from other sources this relational context must also be carefully considered when combining. Also the fact that there may be similar information represented in different ways among the sources should be expected.

As WITSML is the only source providing large amounts of low-level information, it will be the most important source for a considerable portion of the ontology. It does not provide much in the higher levels however, so there the Schlumberger repository and AKSIO will have much more to offer. ISO 15926 will have some significance both at the highest and at the lowest level. The sources will roughly contribute in the following manner:

- **WITSML :** low-level data/formalism, some physical equipment and facilities, document data
- **ISO 15926 :** low-level formalism, upper ontology contributions
- **Schlumberger** : higher level concepts with relations, generic knowledge and understanding
- **AKSIO :** event, states, some equipment, processes and various other
- **Domain experts :** refining all the above, relationships, classifications, specializations

Combining low-level information

At the low level WITSML and partially ISO 15926 are providing the information we have to work with. In previous sections both WITSML and ISO 15926 have been presented. WITSML low level data types are based on XSD types, while ISO 15926 low level data types are based on Express. Also in RDF and OWL, datatypes based on XSD types are possible to use[6]. Because of this I have decided to disregard the ISO 15926 datatype representation in this iteration of the drilling ontology. As a consequence, the datatype model of the drilling ontology is built solely upon WITSML. This reverbes well with the case of data integration, as much of the data to be integrated is most likely stored in WITSML.

Since WITSML contains quite much low-level information, the ontology presented in this thesis is not including all that information. I have basically extracted some parts of WITSML and showed how they can fit into the ontological structure. Perhaps the most important of these are the *units of measure*. They give units to every kind of data value that are expressed in WITSML. For the purpose of data integration, such information is

[6]`http://www.w3.org/TR/swbp-xsch-datatypes/`

paramount to keep intact. Examples of such units of measure in WITSML are **meter, foot, ampere, meterPerSecond**.

Combining high level information

At the high level, WITSML is of less significance than on the low level. It does provide some key concepts though that are central to the drilling domain. Some of these are **bhaRun, Tubular, Well, Wellbore**. From AKSIO there are various concepts that should fit together with these WITSML concepts nicely. When it comes to states and events there is no common concepts between WITSML and AKSIO so there should be no clash occurring. Also with the equipment descriptions in AKSIO, the number of concepts is much higher than what WITSML provides, and in the few cases where there are common concepts, there is not enough relational context given for there to be a problem. Examples of this are:

- **Casing** from AKSIO and **wbGeometry** from WITSML. As was discussed briefly in the section about WITSML, **wbGeometry** is the WITSML way of representing the casing of wellbores, so it is safe to say that the two casing concepts are talking about the same thing. Although the WITSML casing provides much low level data as context, AKSIO has simply a relation that points to what oil-related operation it belongs to (in this case Drilling, Intervention, Production). While this on the large scale is important to know, there is no clash with the WITSML representation.
- **Drillstring** from AKSIO and **Tubular** from WITSML. In this case it is actually less clear whether or not the two concepts mean exactly the same thing. Some things seems to indicate they are the same, while others indicate that Drillstring is a specialization/subclass of Tubular. I have gotten the same signals from domain experts as well, so it would seem that this is somewhat ambiguous. In the drilling ontology I have decided that Drillstring is a subclass of Tubular, and don't see any apparent problems with this.

One important thing to notice is that common for these examples is that the naming of the concepts are different in AKSIO and WITSML even though they are talking about the same or closely related things. This goes to show that it is impossible to get anywhere in combining these sources properly without some degree of domain knowledge.

For this purpose firstly the Schlumberger Oilfield Glossary is a good resource. It provides the ontology designer short and often good descriptions of various concepts that are relevant. This way some context is made known to the designer of the ontology, making it much easier to handle cases such as those above. In addition the glossary entries also introduce new concepts to the designer that may prove important to include in the ontology.

This alone is often not enough though, and domain experts then are the ultimate source of knowledge. They both clarify difficulties that the designer has no chance of doing by himself, and can provide a context for the concepts that are not expressed anywhere else, as much of this is implicit in the other sources. An example of this is the **Tubular-Drillstring** case explained above, where eventually with the help of domain experts I eventually decided that **Drillstring** is a subclass of **Tubular**. Apart from clarifications, domain experts certainly also can provide many new concepts as well as relations between those already existing.

Combining the low and high level

WITSML is the source of domain knowledge that has had the greatest impact of all the sources I have utilized. It provides the main foundation for processing of actual data in the ontology. With a focus on data integration this is quite important. As has also been made apparent there aren't much higher level knowledge in WITSML. It does however provide some of the necessary connections between the low level and the high level. As was introduced in the section about WITSML, there are certain elements of physical structures such as **rig, tubular, tubularComponent, well, wellbore**, and processes such as **bhaRun, cementJob, risk**. These few classes are important and central in that they obviously have a strong connection to the low level of WITSML. As the low level of the ontology is decided to be more or less identical to the low level of WITSML, this means that the rest of the ontology is dependant upon this link between high and low that the WITSML concepts provides. This also reverbes nicely with the fact that these WITSML concepts are among the most important in the drilling domain.

So for the parts of the ontology's high level that are not derived from WITSML; for them to be able to access low level data, they are dependant upon the establishment of relationships either directly with the low level, or through connections with the WITSML concepts. Which one of these two approaches is best varies from case to case. But as most of the data in WITSML is describing the higher level concepts, it seems natural to most often connect other concepts to these WITSML concepts rather that directly to the low level data. This is again a task that would benefit greatly from domain expert guidance, as such connections between concepts are most likely not explicitly stated in any of the other sources.

4.4 Standardized Ontologies

Part of the vision behind semantic technology and the semantic web is the idea of reusability of ontologies. An important part in realizing this is to establish standardized and upper ontologies as common references. This section will go into more detail on the “whys” and “hows” of this topic.

4.4.1 Upper ontologies

As was introduced in chapter 1, having an upper ontology above the domain ontology is sometimes necessary and often helpful. In the case of the drilling ontology, it is strictly not necessary for representing the domain of interest by itself. But to be able to connect to other ontologies through the common ground that upper ontologies provide is a great strength. And it is indeed helpful to have the formal specifications of an upper ontology to at least partially guide the construction of the ontology.

With the drilling ontology I have decided to use parts of BFO and DOLCE as my upper ontology. The reason for not using the entirety of an upper ontology is that they are quite large so it would be a too big task to try to create a drilling ontology based on an entire upper ontology. In addition, these upper ontologies are created mostly by people who do not really consider reasoning and computability very closely, so reasoning with these upper ontologies is bound to be in the range of difficult to impossible. Nonetheless, some important parts have been included in the drilling ontology.

Objects and Processes

This is a distinction that has been hinted at earlier in this chapter. In the introduction I presented them in the context of upper ontologies and DOLCE as *endurants* and *perdurants*. They are defined as such:

- *endurants* : These are entities that do not have temporal parts, but they exist in time. This means that when an *endurant* exists, the entirety of it exists at any time. The entity is not spread out over time. And yet they are dependant upon time, as they (at least physical entities) begin to exist and cease to exist at some time. This property means that ideally any relationship with such entities must have a time factor to make sure that the entity actually exists at the time we want the relationship to be valid. DOLCE describes *endurants* as “participating” in *perdurants*, such as for instance a person participates in a discussion. *Perdurants* do not have this property.

- *perdurants* : These are entities that do have temporal parts. This means that they are not present in their entirety at any moment in time, but are rather pieced together by parts occurring at different points in time. These parts are not necessarily explicitly stated in the ontology because such a level of detail may not be interesting, but this does not alter the fact that *perdurants* are not related to time in the way *endurants* are. They can perhaps be thought of as a chain of events, where an event is on the time axis the smallest *endurant* entity possible.

Everything that can be described as objects are designated as *endurants*, while processes are *perdurants*. To illustrate with some examples from the drilling domain:

- **bhaRun** is a process and thus a *perdurant* because it describes an operation on an oilrig that lasts from point x to point y in time. In further detail a **bhaRun** is defined as an operation lasting from when a drilling is inserted into the wellbore until it is extracted.
- **wellbore** is an object and thus an *endurant* because its existence is not spread out over time. Even though a wellbore is increasing in size over time because of drilling performed, this can be seen as the size property of the wellbore increasing. The wellbore will not be identical to how it was at a previous time, but in the manner DOLCE describes *endurants* by participation, we have already stated that wellbores participate in bhaRun operations.

Unfortunately OWL does not handle time in any built-in fashion so if this is needed it must either be modelled manually, or a time ontology must be imported/referred to. For this reason I have decided not to tackle the time issue in any complete fashion, but the ontology simply stores the time low-level data from WITSML as regular datatype values when they occur. I do however anticipate that it will not be too difficult to add complete time-modelling at a later point, as this would in some sense be like taking snapshots of the ontology at different points in time and reasoning over the set of snapshots[7].

4.4.2 Ontological context

As has been explained in section 3.2.1, URIs do not in fact by themselves refer uniquely to a single entity, contrary to what URLs do and contrary to what many people seem to think. This actually carries great relevance to the use of upper ontologies and other standardized ontologies. If it were

[7]This is my current understanding of how time can be handled. It might turn out quite different!

so that an URI would uniquely identify a single entity in the real world, then there would be no doubt among people what each and every resource of an ontology means. This is however not the case, since even though we might give classes and relations describing names, this is alone not always enough to properly define what we are trying to describe. For the purpose of reasoning these names of course are irrelevant, as the computer that does the computations has no way of understanding or even being ”aware” of any link between ontology resources and real-world entities. The reasoner does it job solely based on the ontologies that humans gives the computer as input.

For understanding the meaning of the classes and relations for us humans the situation is different. It is important to realize that ontologies only describe subsets of reality at a certain level of abstraction. Resources in the ontology can only refer and relate to real-world entities through human interpretation. This interpretation is heavily dependent upon the surrounding context. When building an ontology we are of course interested in keeping it as unambiguous as possible[8]. This means not just unambiguous in logical interpretation that is crucial to reasoners, but also unambiguous in human interpretation. The difficulties here lies in the wide variety of knowledge each person might possess. Human knowledge is not as clearly defined as computerized knowledge in ontologies, so there can be an infinite number of ways to interpret a single class name without any context. People have different experiences and have their very own personal context connected to concepts that make such simple context-less representation ambiguous.

Of course in an ontology, no class is completely without context, as the domain ontology itself provides context that greatly limits the possibilities of interpretation. But this would often mean that large portions of the ontology will have to be examined and understood to achieve the intended interpretation. So instead of leaving everything up to the domain ontology, having standardized ontologies that provide well-known context is a very helpful tool.

Aside from upper ontologies which are meant to provide high level and abstract descriptions and context, other standardized ontologies also exist that provide context closer to the actual domain ontology.

Examples of such standards that are actually in use are:

- Friend of a Friend (FOAF), a social relationship vocabulary, at `http://www.foaf-project.org/`
- Dublin Core, a more general metadata vocabulary, at `http://dublincore.org/`

[8]exceptions to this perhaps exist?

- SKOS, a knowledge organization vocabulary, at `http://www.w3.org/2004/02/skos/`

4.5 Ontology engineering methodology

This section will describe some methodology that can be applied to creating ontologies for specific domains. At first when I started on this thesis I had no knowledge on how to proceed with creating an ontology, so what this section describes has been acquired over a long period of time. This fact is also partially reflected in how the drilling ontology turned out, which will be further discussed in section 4.6.

4.5.1 Proposed methods

An ontology consists of a set of axioms, and the process of building an ontology is that of creating axioms that describe our domain of interest, often with a specific purpose in mind. One such specific purpose is data-integration, which is highly relevant for this thesis.

Unstructured textual knowledge

The most straightforward and easiest way to get this done is to translate textual definitions into axioms by simple methods such as having nouns become classes and verbs become roles. Since things that are nouns often are a good match for classes, and verb a good match for roles, this appears to be a good approach. An example of this could be the statement

- A car has four wheels

We can see that this sentence has the structure car (noun) has (verb) four wheels (noun). This we would then translate into the axiom

$$Car \sqsubseteq =4\ hasWheel.Wheel$$

Other similar cases can also be devised to extract axioms from more complex sentence structures.

This simple way of transforming definitions to axioms depends on that the text used as ontology source consists of precise statements such as the one above. With natural language in normal pieces of text there are bound to be many sentences that are not at all relevant as sources for axioms. Sifting through whole texts and extracting just the relevant sentences is in itself potentially a huge job. But with a good method that automates the conversion from text to axioms the effort might just be worth it.

A problem that quickly can arise from using such methods, is that the vocabulary in use gets really large really fast. If the large number of different words in use in normal language is directly included in the axioms, we might end up with an ontology full of terms that should be related to each other, but are not because of synonyms[9], homonyms[10] and polysemes[11]. In many cases this may be fixed trivially by replacing all occurrences of one term with the other that means the same. But often it will not be as simple. The domain being described has much to do with what meaning lies in every word. This domain context must be considered for the axioms to turn out correctly. This brings us again to domain experts.

The job domain experts can do in the creation of ontologies as already been handled in section 4.1.6. I also describe how asking them direct questions in the form of statements in the appendix A.1. But domain experts also have an important role if methods as described in this section are to be utilized. They are absolutely necessary when there is doubt about the exact meaning of a term in the domain of interest. Such knowledge must be provided to work out problems of synonyms, homonyms, and polysemes as described in the previous paragraph.

Structured knowledge

Besides textual knowledge, there are sources for creating ontologies that are already on a structured format. Such formats can be everything from XML to SQL. Common for any structured format is that they, contrary to unstructured textual knowledge, are easy to parse. The key difference is that normal text is in natural languages that is a result of generations of evolutionary development. This means that only descriptive rules can be applied, and there are many exceptions to these rules that make parsing difficult. And there is of course the problem of synonyms, homonyms and polysemes described above. Structured knowledge however is normatively defined, and as such there is (ideally) a single way to understand the structure. Regularity and lack of exceptions also makes it easy for computers to work with knowledge on such formats. A text in appendix A.3 goes into detail on the difference between descriptive and normative ontology building.

I will illustrate by using a piece of WITSML data.

```
<tubular>
```

[9]Different spelled words with same or similar meaning.
[10]Words that share spelling and pronunciation but have different meaning.
[11]Words with the same spelling and related meanings.

```
    <tubularComponent>
      <typeTubularComp>DrillPipe</typeTubularComp>
    </tubularComponent>
  </tubular>
```

In this piece of XML it is obvious that **tubular** and **tubularComponent** are somehow related. And as the latter is below the former in the structure, it is a fair assessment that in this case **tubularComponent** is a part of **tubular**. And furthermore **tubularComponent** is of the type **DrillPipe**. Such formalised knowledge is much easier to extract automatically by a computer than unstructured textual knowledge.

There are however difficulties here as well. Although less of a problem than with textual knowledge, there are bound to be information that is not relevant for the ontology we are building. The choice whether or not a particular piece of information should be axiomatized must be made by someone with domain knowledge. Here again the domain experts are crucial parts of the process. They are also necessary for filling out missing pieces of the puzzle. In the WITSML example above there was no actual explicitly stated information that said anything about what kind of relationship there was between **tubular** and **tubularComponent**. Since I myself possess some domain knowledge I know that there should be a partOf relationship between the two, but generally domain experts must provide this kind of knowledge.

Another important part of ontology building is creating explicit descriptions with axioms based on implicit domain knowledge. This means extracting information that is not expressed directly either in structured or unstructured format. An expert in any particular domain has presumptions about the meaning of terms and knowledge that is regarded as obvious. For an ontology engineer this knowledge is difficult to extract alone. In the case of WITSML which is defined by a set of XML schemas, finding the implicit axioms is hard since there are no nouns or verbs to use as hooks. Most of the modelling conventions and methods used are in the heads of the people who created the schemas and the domain experts who provided input. In these cases we cannot do much without access to the same people or other domain experts with similar knowledge.

The recently published book on semantic technology, "Foundations of Semantic Web Technologies" [15], describes in chapter 8 methodologies for constructing ontologies. It is much more extensive than my short description here, but many of the approaches, such as the extraction of textual information and the use of domain experts, seem similar. Although late in the process, I have tweaked this section slightly as a result of the contents of that book.

4.6 The Drilling Ontology

4.6.1 Early design experiences

The most influential sources of domain knowledge have been presented earlier in this chapter. Among them WITSML has had the biggest impact on the building of the ontology. Especially early on in the process WITSML was used to obtain a core of important concepts to expand on. The main focus was on the processes involved in drilling and on the equipment used to drill wellbores.

Initially I had little knowledge on how to build an ontology and did not have the proposed methods above clearly in my mind nor on paper, so I simply started out by trying to arrange concepts extracted from WITSML and Schlumberger in a reasonable taxonomy[12] and then add whatever relations I might extract from the domain knowledge context. Even with this seemingly simplistic approach, it was useful to have an idea of what I was trying to achieve building the ontology.

I did not have a real goal for building the ontology at first, which turned out quite problematic. The initial thought was to capture the entire drilling domain in an ontology which could then be used for any purpose in this domain. The problem with this is that the lack of direction in the ontology engineering makes every design decision excruciatingly difficult. Since for such an ontology every possible use ought to be considered, the process of building the ontology becomes really slow and tedious. And there is really no way to anticipate every possible use for an ontology.

In hindsight I can now see that I should have prepared much better a methodology from the start before starting on creating the actual drilling ontology. It turned out to be a much more difficult process than expected. What I tried to do was to start out with very little domain knowledge (and also relatively little ontology knowledge) and create an ontology from scratch. This was a much harder task than anticipated. Much time went into reading and comprehending sources of domain knowledge. Trying to implement this into the ontology without having a specific method at hand proved to be very difficult!

Lessons learned

As I gained understanding of both the domain and of ontology engineering, I achieved a more systematic approach to building the ontology. It seems

[12]subclass hierarchy

clear to me now that when creating an ontology, at least one of the following overall strategies should be applied:

1. There must be a specific purpose in mind for the ontology when creating it. This allows the creators to focus on what is necessary to express in the ontology. The purpose will thus serve as a reference to check the ontology against to see if it fulfills its purpose. This process will work even better if a sort of weighting function or fitness function is specified with regards to the purpose of the ontology. If one can regularly check to see if the ontology is able to express/solve what is intended, the creation of the ontology should work much more smoothly. An example of such a specific purpose is data integration, which is a key topic in this thesis. For data integration speed and the ability to handle large amounts of data are important, so that is something that should guide the construction of the ontology.

2. There must be a common method of creating ontologies that describe specific domains. Such a method should be devised as a general way of creating ontology that describes a specific domain. Many kinds of criteria are possible to include in such a method. But common for them should be that they give concrete ways of adding domain knowledge in the form of axioms to the ontology. This can be expressed e.g. as a sentence of “Noun verb noun” that should be added to the ontology in a specific way. But other possibilities surely exist. This could be investigated further to create a whole toolbox of such methods.

3. A simpler variant of the previous could alternatively be devised. This means a simple method consisting of a few design principles. These should describe what a class might be used for and what a property might be used for and so forth. This method should definitely be combined with a specific purpose so that the ontology construction can proceed smoothly.

If such rigorous principles of design aren’t employed, the ontology construction can become much more of a tedious and difficult process than is necessary. This I have experienced early on during the process of designing the drilling ontology. It should also be possible to use the strategies in combination as well for even more strict design principles, e.g. combining specific purpose with general methods. However whether or not this could cause unforeseen problems will have to be investigated properly[13].

[13] not done in this thesis

Combine with specific methods

When such strategies and principles are properly established, they should be combined with the methods introduced in section 4.5. Having principles and overall goals as a guideline when starting the actual construction will be very helpful in deciding which axioms are necessary and which ones are irrelevant.

4.6.2 My application of engineering methods

For the drilling ontology, the principle that turned out to be the most important was the one concerning specific purpose. Even though I lacked such a clear purpose for the ontology early on, eventually data integration became the use case for the ontology to focus on. This gave a sense of direction in the ontology building.

Using unstructured textual definitions as described above is partially what I have done by using the definitions from the Schlumberger Oilfield Glossary described in section 4.1.4. An example of this is the definition of **casing collar**:

- The threaded collar used to connect two joints of casing. ...

From this we can create the axiom:

$$CasingCollar \sqsubseteq = 2\ connects.CasingJoint$$

Likewise for structured definitions, such as WITSML, this was explained in detail above in the section on structured knowledge.

The value of domain experts

I had some access to domain experts who gave valuable input both in the form of explanations on how processes work in the domain, and more direct input to the ontology. At one point I tested a way of extracting domain knowledge from experts by presenting them a set of statements that should either be confirmed or rebutted. This was introduced in section 4.1.6 about domain experts. The details of this experiment is in the appendix section A.1. What I got out of this was that while my understanding of many parts of the domain got clearer by a simple answer, there were also some of the statements that did not have such a simple solution. The comments I got on those could even lead to more confusion, which had to be inquired further upon to reach a satisfactory answer.

Just as helpful was sitting down with domain experts and getting firsthand help when actually building the ontology. Even though this time was limited compared to the time spent alone working on the ontology, much of the ontology was created just in such close collaboration with domain experts. This just goes to show that proper domain understanding is difficult for the ontology engineer to acquire, and that such knowledge is paramount to efficient and correct ontology construction.

Modularization of the ontology

As said, the ontology was at first created without any specific purpose in mind. This also meant that even though modularity was considered a topic of interest, there was no plans how to use efficiently. This gradually changed however as specific uses for the ontology appeared. The specifics of the modularization I decided upon has already been introduced in section 4.2. Now I will say more about what each of the three modules contain.

The process of splitting up and modularizing the ontology happened rather late, so the distribution of axioms is somewhat skewed in favor of the core module. This is however natural for in many cases where the extending module only refines certain parts of the core vocabulary. As it appears at this time, the entire taxonomy is contained in the core module, meaning that none of the other two modules adds any new classes. What they do add however are a few axioms to test the limits of expressivity. The core is of course contained in *DL-Lite* also, but the specific *DL-Lite* module adds some axioms that aren't needed in the core. These axioms are intended for use in data integration, and shows how similar axioms can be added later on. The same is true for the module with universal quantifiers. It shows how such axioms may be added. This module is however not suited for data integration at all, and has as of now no concrete applications.

I will here simply present a limited version of the taxonomy, as well as the relation hierarchy. For details, the files themselves should be viewed in an ontology tool/editor such as Protege[14]. The ontology module files are located in `http://heim.ifi.uio.no/larsove/ontology/`.

Taxonomy

This shows the class hierarchy to a certain level of granularity.

```
Thing
    Endurant
```

[14]http://protege.stanford.edu/

```
        Object
            NonPhysical
                GeometryType
                JobResponsibility
                JobTitle
                KnowledgeResource
                Quantity
                Service
                Survey
                SurveyStation
                Trajectory
                TrajectoryStation
UnitOfMeasure
                WellDatum
            Physical
                Annulus
                BoreHole
                CirculationSystem
                Company      - could fit better under NonPhysical
                CoreSample
                Cuttings
                Facility
                Field
                Formation
                GeographicArea
                Hydraulics
                Lease
                Location
                Oilfield
                Rig
                RigEquipment
                Slot
                Substance
                Tubular
                TubularComponent
                TubularSection
                Well
                Wellbore
                WellboreGeometry
                Wellpath
    Perdurant
        Event
            BlowOut
            CasingCollapse
            CollisionWithOtherWell
            DamagedBit
            DroppedObjectFromSurface
            Kick
            PackOff
            StuckEquipment
        Process
            RigActivity
            Run
        State
```

Relation hierarchy

```
topObjectProperty
    hasUnitOfMeasure
    belongsToField
    belongsToRig
    belongsToWell
    followedByActivity
    hasActivity
    hasContractor
    hasDevelopmentStatus
    hasDrillString
    hasDrillingActivity
    hasFacility
    hasField
    hasGeometryType
    hasJobResponsibility
    hasJobTitle
    hasLocation
    hasOffsetWell
    hasOperator
    hasPart
        hasBit
        hasBitNozzle
        hasTubularComponent
        hasTubularSection
            hasBHA
    hasProductionType
    hasSlot
    hasSurvey
    hasSurveyStation
    hasTrajectory
    hasTrajectoryStation
    hasWellDatum
```

```
hasWellPath
hasWellbore
hasWellboreGeometry
involvesRotatingEquipment
isPartOfCirculationSystem
isPerformedByServiceCompany
liesInGeographicArea
    liesInArea
    liesInBlock
    liesInCountry
    liesInField
performsActivity
worksOnFacility
worksOnWellbore
```

Chapter 5

Application of the drilling ontology

I will in this chapter present two cases in which the drilling ontology can be used for data conversion and integration. The first case describes an ontology framework called **QuOnto** that makes use of the OWL2 QL profile's *DL-Lite* ontology language for efficient data integration. I will not go into detail of QuOnto[1], but rather give a general description of what it does and how it works.

In the second case I will aim at creating a piece of prototype software that converts data from XML (therein WITSML) to RDF. There is no actual data integration done, but the data conversion facilitates easier data integration by presenting the data in RDF.

5.1 QuOnto

QuOnto is a Java ontology representation and reasoning tool[2] that is well suited as a framework for integrating data using ontologies, which is explained in detail in [37]. Specifically through the use of *DL-Lite* ontologies that were introduced in section 3.3.1. The reason for using *DL-Lite* instead of OWL2 has to do with efficiency. As has been established also in chapter 3, OWL2 does not handle large amount of data efficiently. *DL-Lite* however has much better performance when it comes to handling data. This fact is an important part of what makes QuOnto with *DL-Lite* viable for data integration.

[1]There is a concurrent thesis done by another student that deals specifically with QuOnto

[2]`http://www.dis.uniroma1.it/~quonto/`

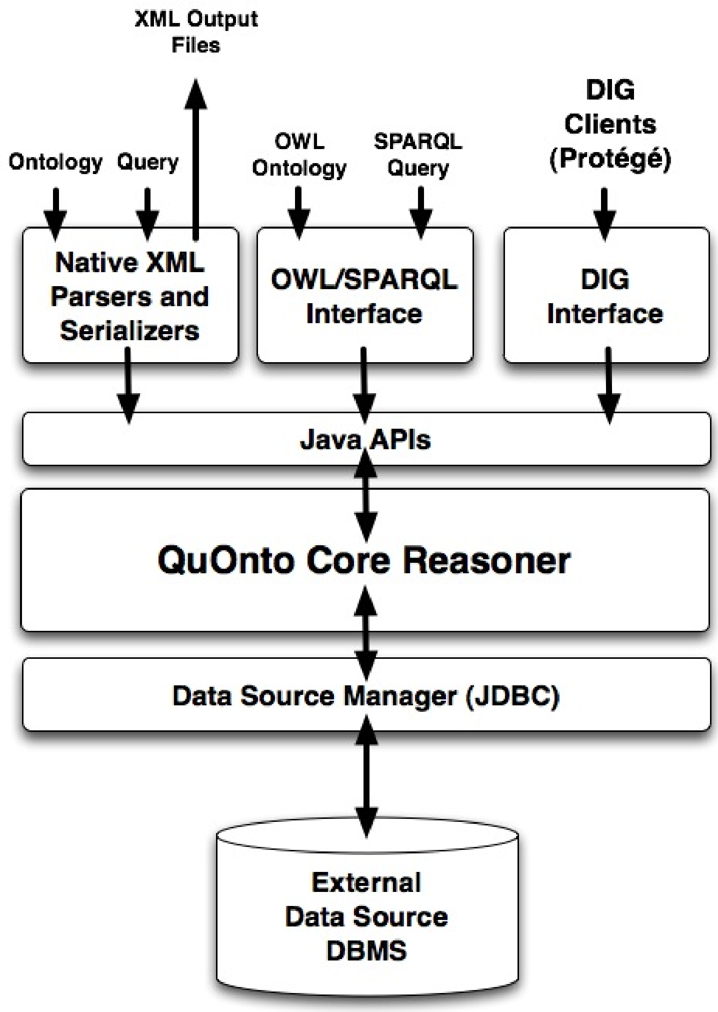

Figure 5.1: *This graph illustrates the architecture of QuOnto.*

Another important aspect of QuOnto, is that it is able to gather data from any kind of source through the use of mappings. These mappings connect e.g. SQL databases with QuOnto and the ontology it uses. Queries can then be performed on the combination of the ontology and the SQL data. This powerful real-time mapping conversion makes it fairly simple to accomplish data integration through the use of QuOnto. The overall architecture of QuOnto is illustrated in figure 5.1.

The particular use case that the drilling ontology might prove useful in, is the integration of DDR (Daily Drilling Report) data. DDR, which was introduced in section 2.1.2, is based upon and has a lot in common with WITSML. Therefore it should be relatively easy to use the drilling ontology together with DDR data since the ontology is also heavily influenced by WITSML. The *DL-Lite* restriction set by QuOnto limits the expressivity of the ontologies that can be utilized, but as that is taken into consideration when constructing the drilling ontology, that particular aspect should not pose any problems.

5.2 Data from WITSML/XML to RDF

An important aspect of being able to use WITSML data in an ontology is of course the ability to convert WITSML data to a format directly usable by OWL. RDF which OWL is built upon is the natural choice for such a semantic data format. This both enables ABox reasoning to be performed on the converted data and it facilitates for data integration by using a common ontology with multiple data sources. The task is then set to create a prototype where some WITSML test data is to be converted to RDF triples.

Here in this section I will describe first the thoughts I had on how to proceed with converting WITSML data to RDF, and the go on to present the implementation.

In addition appendix A.4 contains more background information and thoughts on the conversion of XML data.

5.2.1 Two approaches to XML $\longrightarrow$ RDF conversion

As a general overview I will present two overall approaches to XML-RDF data conversion that includes programming and some testing. One is a bottom-up approach where only the XML data is considered by itself. The other is more of a hybrid between top-down and bottom-up where there must be provided a template in the form of an ontology that guides the conversion. I will first give a list of steps involved in each of the two approaches using an example with WITSML tubular data, and then explain more in detail. Even though this describes conversion from XML, I believe these methods apply to any source of data.

The following two lists present each of the approaches to converting data from XML to RDF. Each step in the list has in parentheses a suggestion for tools or software that can aid in fulfilling the task. The entries that have a "+" prefix are the ones that are absolutely necessary, while the rest are mainly for improving quality and testing.

Approach 1 - hybrid top-down

1. + Create ontology from tubular-data OR expand the drilling ontology (Protege)
2. + Convert data (XSLT or Java/Jena)
3. Improve ontology based on converted data (Protege)
4. Queries on data and ontology (SPARQL, DL Query)
5. Connect tubular ontology to drilling ontology (Protege)
6. Improve ontologies (Protege)
7. Test Queries (SPARQL, DL Query)

This first approach involves creating an ontology as a template before converting data to RDF. This ontology can either be created from scratch in cases where there is no existing semantic context for the XML data to be converted; or in other cases, like here with the drilling ontology, there exists an ontology that is to use the data after it has been converted. In this case, the template ontology can either be a small part of the whole existing ontology, or it can be created as an extension that shares key resources with the whole. It is thought that the templates for WITSML to RDF conversion should be based on the drilling ontology (which is quite natural as the drilling ontology itself is based largely on WITSML).

After the template ontology has been created, the data conversion can be performed. The details of this will be explained in the next section that present the implementation, but an important part of the conversion is a set of tags in the template ontology that link XML elements with ontology resources. These tags are what enable the conversion to proceed automatically and hopefully without errors.

The remaining steps from the list above are optional. They help with improving quality of the conversion and discovering errors, but extra manual work has to be done. There may be cases where results from the data conversion show us that the template ontology should be altered. The whole ontology, in our case the drilling ontology, may also need updating due to feedback from the conversion. In any case, test queries on the converted RDF data should be conducted to ensure the results are as expected.

Approach 2 - bottom-up

1. + Convert tubular-data to RDF-triplets (XSLT, Java/Jena)
2. Improve triples (Manual editing, some RDF editor)
3. Queries on RDF-data (SPARQL, DL Query)
4. Connect RDF-data to drilling ontology (Java/Jena, Protege)
5. Improve (Protege)
6. Test Queries (SPARQL, DL Query)

The second approach looks solely at the XML data. Compared to the first approach, the meta-information in the XML documents is now more important. Since we will not have any reference for conversion in the form of a template, the RDF is created by observing the XML-structure, element/attribute names and the values themselves. This means that no additional knowledge or context, other than what the XML document contains, is added.

All of the remaining steps are similar to in the previous approach meant for testing and improving quality. After the RDF-triples has been created, we may want to ensure that the automated conversion worked well. This can for instance be done by giving feedback to the conversion software about changes we want to the structure we desire. The conversion will then have to be carried out once more in an iterative process. This is just a suggestion, and other methods may work. Such processes is easier in this approach since the conversion itself is quite simple.

The fact that this approach does not provide any semantic context means that the converted RDF data must be connected more or less manually to ontologies for it to be used in larger settings[3]. The difference between this and the first approach is that the data may be of a very different structure than what the ontology is prepared for. By using a small ontology in between in the first approach, we ensure that the converted data will be compatible or close-to-compatible with the drilling ontology. This approach will in general require much more work to have the RDF-triples connect to the whole drilling ontology.

[3]such context can of course be added in the conversion process, but then simplicity is lost and we get an approach more similar to the first one

The approaches compared

Which of these two approached is best probably varies. I however suspect that having some semantic context in the form of an ontology is of great help when using the RDF data further. If the bottom-up approach is used, there must eventually be added some context that give the converted data more meaning. Without such context, the whole process of converting data to the semantic RDF format seems rather pointless.

The implementation presented in the next section facilitates both approaches so they can be compared on a case-by-case basis if necessary.

5.3 Implementation of XML $\longrightarrow$ RDF converter

This XML to RDF converter is a simple java application that aims at doing semi-automated conversion from XML data to RDF triples. It is not fully automated because it ideally needs a template ontology that provides a guideline on how the conversion is to be done. It is however also possible to map the XML data directly into RDF, but in this case no surrounding structure and meaning will be introduced. My work was done in Java with Jena handling the RDF-specific parts, but any language can potentially be used.

The conversion process will now be explained step-by-step with references to the configuration file whenever it is relevant (the format of config files is that of Java Property files, which is more or less the same as Windows .ini files). While the XML parsing only has one possible way of execution, the RDF conversion can be done in two ways. One is by simply converting the XML into RDF without any extra context being introduced, and the other is converting the XML using a template ontology that serves as a guide for the program. How this template must be created will be explained.

As opposed to XML, in RDF there is a stronger ID requirement for all resources, since the structure is a general graph and not a tree. In a tree structure the location in the structure by itself restricts the relations possible, while in a general graph any resource can relate to any other resource. This poses a potential problem when converting data from XML to RDF. In this converter there is not yet a good solution to this. What is done is that a incrementing number is added to the tail of each XML-tag name so that pseudo-uniqueness in naming is guaranteed (combined with namespaces it should be fairly safe). It is then up to the related data to distinguish one occurrence of an XML-tag from another. In WITSML this can partially be done thanks to various ID attributes to many of the tags.

Whether or not these IDs can be used to create a good enough RDF name is something that needs further inquiry.

5.3.1 XML Parsing

When parsing the XML documents, the program searches for and handles 3 different cases. This can be exemplified by the following piece of WITSML:

```
<tubular>
  <tubularComponent>
    <typeTubularComp>DrillPipe</typeTubularComp>
    <id uom="m">0.1087120026350020</id>
    <typeMaterial>Steel</typeMaterial>
    <vendor>199</vendor>
  </tubularComponent>
</tubular>
```

Case 1

XML elements that have no values, but only other elements as children. The element can also have attributes. The following from the example above is handled by case 1:

```
<tubular>
  <tubularComponent>
```

Case 2

XML elements that have values, but no attributes. This often can be a realization of XML simple datatypes. The following from the example above is handled by case 2:

```
    <typeTubularComp>DrillPipe</typeTubularComp>
    <typeMaterial>Steel</typeMaterial>
    <vendor>199</vendor>
```

Note that the parent element, which in this case is <tubularComponent>, also is is important in the way this will be converted.

Case 3

XML elements that have values and attributes. This often can be a realization of XML complex datatypes. The following from the example above is handled by case 3:

```
<id uom="m">0.1087120026350020</id>
```

Also here the parent element is important in the conversion.

5.3.2 RDF conversion

As stated before, a version with and one without template ontologies is possible to use. In both of them I will explain how they handle input from the XML parser split into each of the 3 cases introduced there.

- *config reference :* the variable ”templateOntology” must be ”true” to use template ontologies.
- *config reference :* the variable ”namespace” defines the namespace of all the RDF resources created during the conversion.
- *config reference :* the variable ”outputFile” must contain the path and name of the output file.
- *config reference :* the variable ”outputFormat” must contain the format for outputting the data to file. Possible values are RDF/XML , RDF/XML-ABBREV , N-TRIPLE , TURTLE, N3
- *config reference :* the variable ”verbose” should be either true or false and decides whether or not to give a verbose conversion output, mostly for debugging.

5.3.3 RDF conversion without template ontology

This conversion is as simple as it gets. It uses the XML element names to create suitable RDF resources, which are connected according to the structure in the XML documents.

Case 1

The element and their child elements are connected by creating triples such as:

```
Element has ChildElement1..n
```

Where ”has” is a property created using the same namespace as the data, while the attributes are added such as:

```
Element attrName1..n attrValue1..n
```

From the WITSML example above:

```
tubular1 has tubularComponent2
```

Case 2

The parent element is connected to the element's value using triples such as:

```
parentElement element elementValue
```

where "element" is acting as a property. From our WITSML example above:

```
tubularComponent2 vendor 199
```

Case 3

The parent element is connected to an anonymous RDF node, which is then connected to both the value and the attributes.

```
parentElement element anon1..n
anon1..n attrName1..n attrValue1..n
anon1..n hasValue elementValue
```

From our WITSML above:

```
tubularComponent2 id anon1
anon1 uom "m"
anon1 hasValue 0.1087120026350020
```

5.3.4 RDF conversion with template ontology

What this conversion process does is to simply do exactly the same thing to the XML data as has been done by the template ontology. Every XML element should have a corresponding resource in the ontology so that the program knows exactly how the data should be converted. Any missing XML element reference in the ontology results in warning messages when converting.

The conversion is done by querying the ontology using SPARQL and extracting the necessary information relevant to the XML data currently being converted. Two pieces of information is crucial in this process. One is the template instance of whatever class is relevant, the other is the annotation property marker that connects the RDF resource to a specific XML tag. Again an example from our WITSML:

The annotation property used as a marker is called ”witsml_element”. In the ontology there are two classes ”Tubular” and ”TubularComponent”. We have in the ontology the triples,

```
Tubular witsml_element "tubular"
TubularComponent witsml_element "tubularComponent"
```

meaning that the two ontology classes are both linked to their respective XML tag counterpart. Also the two classes have template instances ”template_tubular” and ”template_tubularComponent” respectively.It is paramount that the naming of the templates is done such as: template_<XML-tag> Note that only classes in the ontology that refers to XML elements which has other elements as children need to have template instances. Any connection between the template instances is also important to show. From the WITSML example there is a triple:

```
template_tubular hasPart template_tubularComponent
```

This shows that tubulars has tubular components as parts; so if any tubulars appear in a WITSML document with tubularComponent child elements, they will be given the same kind of relationship.

- *config reference :* the variable ”templateOntologyX”, where X is a number from 1...n must contain the location (either local file or URL) of the files containing the template ontology(ies).
- *config reference :* the variable ”templateInstanceNamespace” must contain the namespace of the template instances used in the conversion.
- *config reference :* the variable ”ontologyXMLmarker” must contain the name of the annotation property that shows what XML tag the ontology resource is linked to.
- *config reference :* the variable ”ontologyXMLmarkerNamespace” must contain the namespace of the marker above.
- *config reference :* the variable ”outputOntology” should either be blank or contain a unique URI. In the case of containing a URI, it means that the output file will be not just an RDF file, but an OWL ontology (although the only triple added is ontology-URI rdf:type owl:Ontology).
- *config reference :* the variable ”outputOntologyText” provides a comment text for the output. Only applicable if ”outputOntology” is not blank.
- *config reference :* the variable ”importOntology” must be either true or false, and makes the output file import the template ontologies for

added context. Only applicable if ”outputOntology” is not blank.

Case 1

This case dictates a need for marker annotation properties on ontology classes that corresponds to XML-tags and markers on the properties used for the XML attributes. Template instances are needed for the classes, and connections between the instances and also from instances to attribute values are necessary. Illustrating using the WITSML example:

```
<tubular>
  <tubularComponent>
```

As described above, these two should in the template ontology have corresponding instances ”template_tubular” and ”template_tubularComponent”. They are also connected using a ”partOf” object property. From this we get the new RDF triple:

```
tubular1 hasPart tubularComponent2
```

using the incrementing numbers for unique naming as described in x.y.z

Case 2

This case dictates a need for marker annotation properties on ontology properties that corresponds to XML-tags. Illustrating using the WITSML example:

```
<typeTubularComp>DrillPipe</typeTubularComp>
<typeMaterial>Steel</typeMaterial>
<vendor>199</vendor>
```

By using the marker annotation property ”witsml_element”, we find that the corresponding ontology resources for the elements are ”hasTypeTubularComponent”, ”hasMaterialType” and ”hasVendor”. ”hasMaterialType” is an object property, which in this case means that it signifies an enumerated datatype. The other two are datatype properties and there is no restrictions on the value range. From this we get the new RDF triples:

```
tubularComponent2 hasTypeTubularComponent "DrillPipe"
tubularComponent2 hasMaterialType Steel
tubularComponent2 hasVendor 199
```

Case 3

This case dictates a need for marker annotation properties on the classes corresponding to the XML-tags. These classes will have anonymous instances generated, which in turn connect to the data in the value and attributes from the XML. The XML attribute names also needs marker annotation properties. Illustrating using the WITSML example:

```
<id uom="m">0.1087120026350020</id>
```

In the ontology, we find that the corresponding ontology resources for ”id” and ”uom” are ”InnerDiameter” and ”hasUOM”. ”hasUOM” is here an object property which signifies an enumerated datatype, but that is not necessary. The new RDF triples are:

```
tubularComponent2 hasInnerDiameter anon2
anon2 hasUOM "m"
anon2 hasValueFloat 0.1087120026350020
```

5.3.5 Possible improvements and changes

The software prototype created has not been extensively tested so there might still be bugs in the code. Likewise the conversion process have some limitations that could become an obstacle to certain XML files, but this will be possible to improve on. Template ontologies now have to be created by hand, but it should be possible to semi-automatically create this ontology if the XML data is accompanied by XSD schema files (which is just what WITSML provides).

There are also many things that can be done to improve user friendliness, such as a more interactive user interface, better documentation.

Chapter 6

Conclusion

6.1 Summary

The main focus of this thesis has been on semantic technologies and ontologies in general, and their use in the oil and gas drilling domain in particular. I have attempted to create a prototype for a drilling ontology that captures much of the core concepts in the domain. This ontology was then used partially in the conversion of XML to semantic RDF data, which is an important step in accomplishing data integration. I have also explained how this ontology can be used in the ontology framework QuOnto for actual data integration. This is however not attempted by me, as another thesis parallel to mine is dealing with the actual use of QuOnto.

To be able to construct this ontology, an important task was to gather and present sources for domain knowledge that might be useful in the creation of a drilling ontology. These sources are:

- WITSML
- DDR - Daily Drilling Report
- ISO 15926
- Schlumberger Oilfield Glossary
- AKSIO
- Domain Experts

Out of these, WITSML and the Schlumberger Oilfield Glossary were two of the most contributing sources, especially early on in the process. In addition to these two, domain experts were the most enlightening source of knowledge. Such experts helped to add and refine knowledge to the ontology that would have been impossible for myself from just examining textual sources.

While building the actual drilling ontology, I also presented various aspects of the technologies that form the foundation that makes ontologies possible. These are:

- RDF
- SPARQL
- OWL2 / OWL-DL
- *DL-Lite*, which is formalism of the OWL2 profile OWL2 QL

In addition to presenting the most relevant technologies, difficulties and limitations inherent in these were examined. The most important of these were:

- The use of partOf relations
- Consequences that cannot be expressed directly in OWL
- Modularity

The sources for domain knowledge are not in their original format suited directly for use in ontologies. Therefore the relevant information must be extracted from them depending on how the knowledge is represented. A large portion of the thesis deals with this problem, and presents some specifics of extracting relevant knowledge from all of the sources presented earlier, including domain experts.

When the domain knowledge has been compacted and made better suited for ontology construction, methods and principles for actual ontology engineering should also be established. This was also done while keeping focus on the drilling domain.

With both the domain and technological contexts established, the actual drilling ontology was then presented. This drilling ontology is a prototype that contains core concepts in the domain, as well as specifics into some areas to illustrate how the continued construction can be conducted. Through a number of iterations the ontology ended up as a set of three modules, where the parts are:

- A **Core module** that largely consists of a taxonomy of the most important classes for representing the drilling domain.
- A ***DL-Lite* module** that has the purpose of being a data integration ontology.

- A **Universal quantifiers module** that is not inhibited by the *DL-Lite* restrictions. This module is free to use any OWL2 constructs available.

Although the two extending modules can contain a large number of axioms, the prototype ontologies currently has only a few axioms. As most of the axioms do not influence the complexity in any way that puts it outside of *DL-Lite*, the **Core module** is by far the largest module. Whether or not it should be smaller and more axioms should be separated into another module is something that will have to be considered more closely in the future.

The final part of the thesis introduced two specific uses for the ontology. These uses are closely linked to data integration. The first case is that of the ontology representation and reasoning framework QuOnto. It is able to use *DL-Lite* ontologies specifically for the purpose of data integration. As such the drilling ontology with the *DL-Lite* module was created with QuOnto utilization in mind. However, as there is another concurrent thesis working specifically on QuOnto use, I simply presented QuOnto at an abstract level.

The second use for the ontology was in a data conversion tool that I created using Java and Jena[1]. This tool handles converting XML data to semantic RDF data. It can do so either without any connection to an ontology, but also with an ontology as a template for the conversion. This latter case is the most interesting one, as it gives the converted data valuable context that will surely be useful in data integration. As such this tool, while not conducting any data integration by itself, facilitates for it.

6.2 Future work

There are a number of areas this thesis touches upon where more work can be done. I will present some of the most important ones that I have identified.

Improve and extend the drilling ontology

The drilling ontology as it currently appears is a prototype for what such an ontology could look like. There are certainly parts of it that can be improved, both with regards to the domain knowledge representation, but also in the technological aspect when it comes to modelling and engineering.

[1]`http://jena.sourceforge.net/`

Further modularization of the drilling ontology

The three modules that the drilling ontology currently consists of are not as developed as they could have been. The core module is rather extensive, and parts of it could perhaps be extracted in separate modules; one such candidate that I have identified is the part with units of measure. Meanwhile the two other modules can be expanded with more axioms to better fulfill their potential. This should however be done in correlation to actual use cases.

Do more work on investigating ontology engineering principles

As much innovative work is being done in general when it comes to ontologies, the engineering of them is also a field where progress is constantly made. Methodologies and principles are bound to appear that makes the ontology construction process easier than it is today. Work to help improve on this is paramount to a future commercial success of ontology based systems.

More theoretical work on OWL2 and beyond

This is also an area where much research is being conducted. The topics presented in this thesis are but a few of the challenges that exist. And as I did not give definitive answers to them, much work can be done to improve the quality of ontology languages.

Extend the use cases and do a full-scale data integration test

The use cases presented in this thesis were not large projects that properly show the capabilities of ontology based data integration, but rather small tests that establish that there is potential inherent in ontologies. The next step should then be to do a larger scale test where data integration is conducted. QuOnto is probably the best candidate for a system to be implemented with relative ease, as this is already working framework while my Java solution is still a simple converter.

Terms and Acronyms

AKSIO Active Knowledge Management for Integrated Operations. Project for semantic search through documents

Class OWL term meaning the same as Concept

Concept DL term for unary predicate

DL Description Logic , family of logic languages that OWL and its variants are based on

DL-Lite Subset of DL that focuses on efficient data handling and querying

IO Integrated Operations

IOHN Integrated Operations in the High North

OLF Oljeindustriens Landsforening (The Norwegian Oil Industry Association)

OWL Web ontology language , W3C recommendation for ontology language

Property In OWL this is the same as Relation in DL

RDF Resource Description Framework , W3C recommendation for semantic described data/resources

RDFS RDF Schema

Relation DL term for binary predicate

Role OWL term meaning the same as Relation

SPARQL SPARQL Protocol and RDF Query Language

WITSML Wellsite Information Transfer Standards Markup Language

File locations

ZIP-file with all of the files below at `http://heim.ifi.uio.no/larsove/master/masterALL.zip`

The pdf file of this document at `http://heim.ifi.uio.no/larsove/master/master.pdf`

The core module of the drilling ontology at `http://heim.ifi.uio.no/larsove/ontology/Drilling_ontology_core.owl`

The DL-Lite module at `http://heim.ifi.uio.no/larsove/ontology/Drilling_ontology_DL-Lite.owl`

The universal quantifier module at `http://heim.ifi.uio.no/larsove/ontology/Drilling_ontology_UniQuant.owl`

The AKSIO ontology converted to OWL-DL at `http://heim.ifi.uio.no/larsove/ontology/aksio_converted-2009-01-16.owl`

The first test ontology for compositions that works fine at `http://heim.ifi.uio.no/larsove/ontology/siblingsister.owl`

The second test ontology for compositions that does not work properly at `http://heim.ifi.uio.no/larsove/ontology/siblingsister2.owl`

The third test ontology for compositions at `http://heim.ifi.uio.no/larsove/ontology/uncle.owl`

A collection of the files needed for XML to RDF conversion at `http://heim.ifi.uio.no/larsove/master/XMLtoRDF.zip`

IOHN AKSIO conversion deliverable at `http://heim.ifi.uio.no/larsove/master/IOHN/IOHN_deliv_aksio_merged.pdf`

IOHN Drilling ontology deliverable at `http://heim.ifi.uio.no/larsove/master/IOHN/IOHN_deliv_witsml_superstructure_merged.pdf`

IOHN XML to RDF converter deliverable at `http://heim.ifi.uio.no/larsove/master/IOHN/IOHN_deliv_xml2rdf_converter_merged.pdf`

Bibliography

[1] A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev, "The dl-lite family and relations," *J. Artif. Int. Res.*, vol. 36, no. 1, pp. 1–69, 2009.

[2] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, and L. Schneider, "The wonderweb library of foundational ontologies preliminary report," in *WonderWeb Deliverable D17*, 2003.

[3] T. Liebig, "Reasoning with owl – system support and insights –," Tech. Rep. TR-2006-04, Ulm University, Ulm, Germany, September 2006.

[4] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen, "From SHIQ and RDF to OWL: The making of a web ontology language," *Journal of Web Semantics*, vol. 1, no. 1, pp. 7–26, 2003.

[5] I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible sroiq," in *In KR*, pp. 57–67, AAAI Press, 2006.

[6] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds., *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.

[7] D. Calvanese, M. Lenzerini, R. Rosati, and G. Vetere, "Dl-lite: Practical reasoning for rich dls," in *In Proc. DL 2004, volume 104 of CEUR Workshop Proceedings*, 2004.

[8] I. Horrocks and U. Sattler, "A tableaux decision procedure for shoiq," in *In Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI*, pp. 448–453, Morgan, 2005.

[9] D. Calvanese, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati, "Linking data to ontologies the description logic dl-litea," in *In Proc. of OWLED 2006*, 2006.

[10] M. E. Winston, R. Chaffin, and D. Herrmann, "A taxonomy of part-whole relations," *Cognitive Science*, vol. 11, no. 4, pp. 417–444, 1987.

[11] B. Motik, B. Cuenca Grau, and U. Sattler, "Structured objects in owl:

representation and reasoning," in *WWW '08: Proceeding of the 17th international conference on World Wide Web*, (New York, NY, USA), pp. 555–564, ACM, 2008.

[12] I. R. Horrocks, "Using an expressive description logic: Fact or fiction?," in *In Proc. of KR-98*, pp. 636–647, Morgan Kaufmann, 1998.

[13] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler, "Just the right amount: extracting modules from ontologies," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*, (New York, NY, USA), pp. 717–726, ACM, 2007.

[14] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler, "A logical framework for modularity of ontologies," in *In Proc. IJCAI-2007*, pp. 298–304, AAAI, 2007.

[15] M. K. Pascal Hitzler and S. Rudolph, "Foundations of semantic web technologies," 2010.

[16] B. Motik, "On the properties of metamodeling in owl," in *In 4th Int. Semantic Web Conf. (ISWC 2005*, pp. 548–562, 2005.

[17] M. Lewis, D. Cameron, S. Xie, and I. B. Arpinar, "Es3n: A semantic approach to data management in sensor networks."

[18] M. Imai, Y. Hirota, S. Satake, H. Kawashima, and H. K. Yokhama, "Semantic connection between everyday objects and a sensor network."

[19] L. Bermudez, J. Graybeal, and R. Arko, "A marine platforms ontology: Experiences and lessons."

[20] D. P. Pothipruk, Q. A. O. Owls, and R. O. T. Semantic, "An optimization for query answering on alc database."

[21] J. Bao and V. Honavar, "Adapt owl as a modular ontology language," in *OWLED*, 2006.

[22] B. Smith, W. Ceusters, B. Klagges, J. Kohler, A. Kumar, J. Lomax, C. Mungall, F. Neuhaus, A. Rector, and C. Rosse, "Relations in biomedical ontologies," *Genome Biology*, vol. 6, no. 5, p. R46, 2005.

[23] I. Horrocks, "Practical reasoning for very expressive description logics," in *Journal of the Interest Group in Pure and Applied Logics 8*, pp. 293–323, 2000.

[24] I. Horrocks, O. Kutz, and U. Sattler, "The irresistible sriq," in *In Proc. of OWL: Experiences and Directions*, 2005.

[25] I. Horrocks and U. Sattler, "Ontology reasoning in the shoq(d) description logic," in *In Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001*, pp. 199–204, Morgan Kaufmann, 2001.

[26] B. N. Grosof, “Description logic programs: Combining logic programs with description logic,” pp. 48–57, ACM, 2003.

[27] E. Grädel and Q. Yuri, “Why are modal logics so robustly decidable?.”

[28] B. Motik, R. Shearer, and I. Horrocks, “Optimized reasoning in description logics using hypertableaux,” in *Proc. of the 21st Int. Conf. on Automated Deduction (CADE-21)*, vol. 4603 of *Lecture Notes in Artificial Intelligence*, pp. 67–83, Springer, 2007.

[29] S. T. Polyak and A. Tate, “A common process ontology for process-centred organisations. knowledge based systems,” in *A Coalition Force Scenario ‘Binni — Gateway to the Golden Bowl of Africa’”, Proceedings of the International Workshop on Knowledge-Based Planning for Coalition Forces*, pp. 115–125, 2000.

[30] H. Herre, B. Heller, P. Burek, R. Hoehndorf, F. Loebe, and H. Michalek, “General formal ontology (gfo): A foundational ontology integrating objects and processes. part i: Basic principles (version 1.0),” Onto-Med Report 8, Research Group Ontologies in Medicine (Onto-Med), University of Leipzig, 2006.

[31] H. S. Pinto, J. P. Martins, and J. A. P. Martins, “Revising and extending the units of measure ”subontology”.”

[32] W. C. A, B. S. B, and J. F. A, “Ontology and medical terminology: Why description logics are not enough.”

[33] J. Lehmann and J. Breuker, “On defining ontologies and typologies of objects and of processes for causal reasoning,” in *Proceedings of IEEE Standard Upper Ontology, pages 31 – 36, Menlo Park*, 2001.

[34] P. Bedi and S. Marwaha, “Versioning owl ontologies using temporal tags,” 2007.

[35] F. Baader, I. Horrocks, and U. Sattler, “Description Logics,” in *Handbook of Knowledge Representation* (F. van Harmelen, V. Lifschitz, and B. Porter, eds.), Elsevier, 2007.

[36] C. Corona, E. D. Pasquale, A. Poggi, M. Ruzzi, and F. Savo, “When owl met dl-lite....”

[37] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, and M. Ruzzi, “Data integration through dl-litea ontologies,” in *Revised Selected Papers of the 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008)* (K.-D. Schewe and B. Thalheim, eds.), vol. 4925 of *Lecture Notes in Computer Science*, pp. 26–47, Springer, 2008.

[38] R. Kontchakov, F. Wolter, and M. Zakharyaschev, “Modularity in dl-lite.”

[39] R. Kontchakov, L. Pulina, U. Sattler, T. Schneider, P. Selmer, F. Wolter, and M. Zakharyaschev, “Minimal module extraction from dl-lite ontologies using qbf solvers,” in *IJCAI’09: Proceedings of the 21st international jont conference on Artifical intelligence*, (San Francisco, CA, USA), pp. 836–841, Morgan Kaufmann Publishers Inc., 2009.

[40] R. Kontchakov and M. Zakharyaschev, “Dl - lite and role inclusions,” in *ASWC ’08: Proceedings of the 3rd Asian Semantic Web Conference on The Semantic Web*, (Berlin, Heidelberg), pp. 16–30, Springer-Verlag, 2008.

[41] A. Borgida, “On importing knowledge from ontologies.,” pp. 91–112, 2009.

[42] C. Parent and S. Spaccapietra, “An overview of modularity,” pp. 5–23, 2009.

[43] J. S. Aitken, B. L. Webber, and J. B. L. Bard, “Part-of relations in anatomy ontologies: A proposal for rdfs and owl formalisations,” in *Proc. PSB*, pp. 166–177, 2004.

[44] A. Borgida and R. J. Brachman, “Conceptual modeling with description logics,” pp. 349–372, 2003.

[45] C. M. Keet, “Part-whole relations in object-role models. 2nd international workshop on objectrole modelling (orm 2006,” in *In: OTM Workshops*, pp. 2–3, Springer-Verlag.

[46] R. Stevens and M. Stevens, “A family history knowledge base using owl 2,” in *OWLED*, 2008.

[47] B. Motik, B. C. Grau, I. Horrocks, and U. Sattler, “Representing structured objects using description graphs,” in *KR*, pp. 296–306, 2008.

[48] B. Motik, B. C. Grau, and U. Sattler, “The representation of structured objects in dls using description graphs*.”

Appendix

Appendix A

Bonus Material

A.1 Statements

The problem of acquiring domain knowledge about the system he is creating always exists for any kind of system designer. This is no less true for ontology design. If the designer himself is to learn everything he knows about the domain to be able to create a good enough system, he will have to study long and a lot more than he probably has time for. This is clearly not a realistic scenario. The domain experts have in many cases years of valuable experience which is not easily acquired by someone new to the field. On the other hand, having the domain experts learning how to create the system themselves is equally difficult, and will require them to study lots as well. This should all be well-known facts to people designing systems of any kind. When a systems is to be made, certainly much time will be spent in meetings and sessions between the system designers and the domain experts. This often takes a lot of time, as acquiring sufficient amounts of knowledge to be able to make a satisfiable system is not an easy task. Many iterations of prototyping, which the experts then look at and discard are bound to occur.

While this is often a necessary process, I believe it can be made more efficient. When time is sparse for some of the parties involved, arranging actual meetings or workshops may be difficult and time-consuming, thus there aren't nearly enough of them to achieve the progress we would want. I believe this is the case in many projects, where the system designers are fully committed, while the domain experts have their normal schedule and have to squeeze in the correspondence and meetings dealing with the project whenever they can. What can be fruitful in these scenarios is having the designers read some material upfront and learn things about the domain before even meeting any of the experts. This I see as a natural way of approaching a new domain, and is surely done in real projects. Reading up on the domain, they will most likely fail to grasp properly it at first, and have questions about a great many things. Indeed, early on they might not even know what they don't know, an undesirable state of confusion. Meeting the domain experts and engaging in dialogue is most certainly fruitful for getting a clearer picture of the domain. But as mentioned, chances for this might be sparse.

What I believe is a good substitute is a form of questioning. This might sound obvious, but in my experience often the way questions are asked is too complicated and unstructured. Even though normal questions through email or phone conversations takes less time than having actual meetings, it appears inefficient as the questions might be unclear and the domain expert will perhaps not understand the question properly, or will explain the answer in a way that is difficult to understand for the engineer. Compared to actual meetings where a higher degree of conversation back and forth can occur, the benefit will certainly be a lot lower as well, so the cost-efficiency might turn out even lower than spending the time for actual meetings!

So what I propose is to structure the questioning a lot better. Not in content but in form and complexity. And instead of actual questions, make elementary statements. These can be statements that the designer believes either to be true or false, but that belief should not be conveyed to the experts when presenting the statements to avoid influencing the answers. Having many simple elementary statements which the designed believe either to be true or false, the domain experts can then quickly read each statement and declare them true or false. Also the option of additional commenting on the statement should be available. This way I believe the domain experts can run through a list of lots of statements fairly quick, giving the designer valuable input to work on. This can be repeated, and through each iteration the designer should be able to state more and more relevant things getting to the core difficulties of the domain, and this way getting good knowledge. Of course, this alone is probably not enough, and should be combined with regular meetings and other activities (like reading documentation, tutorials, books, ...). But whenever designing a system, opinion and input from the domain experts with years of experience should in most cases be the best way of quickly understanding the domain.

One important prerequisite for this method to work, is that the designer has a concrete goal when making the statements. He should be aiming for somewhere: a particular design, model, function, or something else. If the designer is only interested in general knowledge about the domain as a whole, only relying on these statements may not give the big picture that he aims for. There will always be things and relations the designer haven't thought about, which can best be explained by the experts. This is also the reason why the most efficient way is to start by acquiring overall knowledge and the big picture, and then combine statements with other correspondence to narrow down on the problem at hand (such as creating an ontology).

When it comes to creating ontologies, they are somewhat special since good design of them will require a much better understanding of the domain than with many other applications. But it still is a concrete goal, where statements can be useful. For instance for determining class-names, subClass relations and other relations and their naming. An important part of ontology design is determining what level of abstraction we want, and also where to divide the TBox from the Abox (higher abstraction means most likely a smaller TBox with instances at a "high" conceptual level. Compare having Car instances to having instances of a VW Golf). This decision however should be carefully reached in discussion with the domain experts and the users of the systems being created. Statements may be useful here also though!

It boils down to reduction in complexity compared to normal questioning. If the statements all are short propositions, then they should be easy and quick to comprehend and give an answer to. Especially since the answer is either true or false. However since the answer is so simple, actually extracting useful information might require a lot more statements than you would need with other forms of questioning. And there will be more work for the designer to combine the answers to the statements and consider their relevance. But as the designer is assumed to have much more time available than the domain expert, the amount of work should be balanced out with regards to time. I will present some methodology aimed at getting as much and as correct knowledge out of the statements and their answers as possible.

A.1.1 Methodology

As to what kind of statements should be stated, there are certain methods that might be followed:

- Redundancy! Make rephrased statements more than once, especially the ones you feel are important. Alter details if necessary, but be aware of the changes made to statements. Also when they are rephrased the meaning may change.

- Make a statement that you think is true, and then later make an opposite statement (perhaps rephrased) trying to create contradictions. If the experts answers true or false on both, you know that something is fishy and should be more carefully inquired, either with more statements or as a good question for the next actual meeting.

- Try to have a "path/graph/tree of statements" such that many of the statements are connected in a way. This will happen naturally if you start with complex statements/questions and break them into smaller parts. Thinking this way in a e.g. hierarchy and having a model of how to fit together the answers may be a good idea. So, starting with general statements, break them into parts down to “atomic” statements. This of course requires much work, perhaps way too much. But it could also be a general method applicable to many different domains, so adaption could turn out easy once developed.

Having the domain experts agreeing to such an approach, such that they answer properly each statement is required of course, and persuasion will not be addressed here.

A.1.2 Test case

I have had the opportunity to test this theory on a domain expert from NOV (National Oilwell Varco) and got some interesting results.

```
Statement:  casing is a process
True/False: True
Comment:    Can also be a steel pipe inserted into a well bore
to seal off fluids and keep the hole from caving in.

Statement:  a run is a process
True/False: True
Comment:    Can also refer to a specific run with a drill
 string - a run would then be given a run number.

Statement:  cementing is a process
True/False: True
Comment:

Statement:  an operation is a process
True/False: True
Comment:

Statement:  a Run is any operation that puts a string down a
 borehole
True/False: True
Comment:

Statement:  a Run lasts from when the string is inserted until
 it is extracted
True/False: True
```

```
Comment:

Statement:  a WITSML-bhaRun is the same as a Run
True/False: False
Comment:    A WITSML-bhaRun is a type of run.

Statement:  all Runs must use a string
True/False: False
Comment:    Can also be a Coiled Tubing Run. (Not a string, but
 rather a Wire)

Statement:  the tip of the string is always a bottomhole
assembly (BHA)
True/False: False
Comment:    I.e. When running a Casing Run - There is no BHA.

Statement:  string is the same as drillstring
True/False: False
Comment:    I.e. Casing String. A drillstring is a complete
 string incl. BHA.

Statement:  wellbore is the same as borehole
True/False: True
Comment:

Statement:  a Run does not always use a string down the borehole
True/False: True
Comment:    Ref, Coiled tubing.

Statement:  a drillstring is a tubular
True/False: True
Comment:

Statement:  a rig only drills one borehole at a time;
True/False:
Comment:    Depends how rig is defined. If Rig is defined as the
 Derreck (the drilling tower) then this is true. If Rig is
 defined as a drilling facility, then this would be false,
 since drilling rig can have multiple Derricks.

Statement:  WITSML-wbGeometry describes one casing
True/False: False
Comment:    Describes a well (annular) section given a certain
 geometry/diameter.
```

Statement: only Runs uses drillstrings
True/False: True
Comment:

Statement: a Run has a trajectory and a target
True/False: False
Comment: A run follows a certain trajectory, or makes one in case of drilling run.

Statement: a borehole has a trajectory and a target
True/False: True
Comment:

Statement: casing is not a Run
True/False: False
Comment: Casing run exists.

Statement: casing is not tubular
True/False: False
Comment:

Statement: cementing is a Run
True/False: True
Comment:

Statement: cementing is only done right after casing
True/False: False
Comment: Can also be done to strengthen well sections.

Statement: trajectory is a wellpath
True/False: True
Comment:

Statement: a drillstring always uses a bottomhole assembly
True/False: True
Comment:

Statement: all operations that insert something into the borehole are Runs
True/False: True
Comment:

Statement: there are Runs that aren’t WITSML-bhaRun

```
True/False: True
Comment:

Statement:  all stages of casing are done in the same way
True/False: True
Comment:    Fist casing is not necessarily cemented in place.

Statement:  cementing is always done right after casing
True/False: True
Comment:

Statement:  a WITSML-tubular is only made out of
 WITSML-tubularComponents
True/False: True
Comment:

Statement:  WITSML data is not a continuous stream of live data
True/False: True
Comment:    Some of the documents (i.e realtime data) is sent
 frequently, and some (I.e. well) are sent infrequently.

Statement:  WITSML data are sent as XML documents and stored on
 a server
True/False: False
Comment:    WITSML data are sent as XML documents, but they are
 not always stored as documents, rather the information
 contained in them is extracted and inserted into other
 systems / databases.

Statement:  WITSML documents are sent from drilling-rigs
 sporadically, and not necessarily often.
True/False: True
Comment:

Statement:  some type of WITSML documents are sent more often
than others (which ones?)
True/False: True
Comment:    RealTime, Log, WellLog are sent often.

Statement:  an actual XML document file is described by one of
the schemas with the obj_ prefix
True/False: True
Comment:
```

The feedback contain much good information, but the answers were not as simple as I expected. Many of the answers were not just true or false, but had further comments. I expected that some of the statements would get comments in addition to a simple true or false, but as many as I got. This is however still useful since the amount of insight into domain increases, and the bases for ontology design decisions improve.

A.2 More on Mereology

One important distinction to make design-time is whether the ontology is meant only for use when actually drilling, or if it is to be more of a general life-cycle ontology. In the case of just actual drilling, things can be much more explicit and detailed. This is because everything that is true when drilling can be directly expressed. In the second case a more general approach must be made. Many things that are true when drilling aren't necessarily true at other times, so these things cannot be expressed (at least in the core ontology). To illustrate using an example:

Having the class DrillBit and the class BottomholeAssembly, we know that there is a connection between those two. While drilling, a bottomhole assembly must always have a part that is a drillbit, and a drillbit is always a part of a bottomhole assembly. These two statements can then be explicitly made as restrictions in the ontology if we are only concerned with drill-time. However, when not drilling e.g. the equipment in storage, things are different. It will still be true that a bottomhole assembly always has a part that is drillbit, but it will not be true that a drillbit must be part of a bottomhole assembly. The distinction here is that a drillbit is a defining part of a bottomhole assembly; a bottomhole assembly is not whole without a drillbit. The same is not true about drillbits. They are full-worthy drillbits without being part of any bottomholeassembly. A meronomy[1] runs one way only, from the parts up towards the whole. Meaning that the parts define the whole, while the whole says less about its parts. Drillbits are in this context atomic parts (leaf nodes in a meronomy) and will thus always be a full-worthy drillbit in any conceivable use for the ontology. Bottomhole assembly however is a "complex object" (inner nodes in a meronomy) defined by its parts, one of them being drillbit. We cannot say anything beyond this for sure if we make no assumptions as to how the ontology will be used. It is up to the scenarios where the ontology is used to dictate any additional restraints made on the description of the part-whole relationship of drillbit and bottomhole assembly, as well as other similar cases.

[1]Mereology: the theory of part-whole relationships. Meronomy: a set of concepts classified in a part-whole structure.

One must also take care when using inverse roles for mereology. “hasPart” and “partOf” should be made inverse, but they may certainly have different uses, again dependent upon context. A particular bottomhole assembly may only be part of one drillstring at a time, but if we set out to model what happens during time-elapse, that one bottomhole assembly may become part of different drillstrings. This particular problem may however be remedied by having each bottomhole assembly be a whole for the sole purpose of one drillstring. And reuse of its parts would then create a different whole for another drillstring. This does not eliminate the problem however, as atomic parts such as “Drillbit” still retain the potential problem. However, as discussed above. the context may dictate how many restrictions we may want to put on a class. In a specific context it may very well be prudent to say that a drillbit be part of only one bottomhole assembly, and likewise a bottomhole assembly be part of only one drillstring.

A.3 Normative vs Descriptive

In any kind of knowledge representational(KR) system[2], there are introduced biases and assumptions about the world represented. Traditionally most computer based KR systems have been strictly defined by rules[3] and the data entered must fit these rules perfectly. Within these rules lies the biases and assumptions. Whenever the creators defined the system in question, they carried along with them their own experiences, wishes and agendas for the system, as well as the same for the actual users of the system to the best of their ability (herein lies an obvious challenge in communicating the users’ requirements properly, but that is a different discussion).

What essentially is done in such systems, is that a closed defined system of representation is created. Whether this be a relational database at the bottom with middleware and software on top, or any other incarnations, this is mostly true today. Altering the representation of certain data or knowledge can range from fairly easy on small systems to difficult or nearly impossible within larger more complex systems. Problems linked to altering representation in existing systems should not arise where the domain is more or less static or where the purpose of the system does not change. However many systems, especially nowadays with the advent of so many

[2]The word “system” is in this text given the meaning of “computer based collection of software that works together in achieving a given purpose/purposes”. In this context this purpose is most often knowledge representation, but should be clearly stated when it differs.

[3]The word “rule” here can be any kind of schema or statement that in collections/sets clearly define what is expressible and not by the system. Not to be confused with logic based rules, nor general ways of axiomatizing.

web-communities and socially based websites. Places such as facebook, myspace, flickr, youtube and many others are generating huge amounts of data that is not static by nature but changes as the demands of users change and websites struggle to keep up with these. Whereas clearly defined small company databases can remain unaltered over long periods of time, huge databases that interact across organizations are much more sensitive to change and are incredibly expensive to alter.

The core of the problem lies in the assumption implicit in the rules that defines a normative system. In such systems, the extent of expressivity is defined solely by the rules. If something is not expressible with regards to the rules, then that something is by definition not part of the system. If we still wanted to facilitate adding this particular something, then we would either have to alter the rules and thereby endangering all of the data already in the system; or we would have to create some ad hoc exception to the rules and handle that appropriately throughout the software. This is the old-fashioned and computer-friendly approach to representational systems.

In contrast to this we have what we can call a descriptive system. This is fundamentally different in that it does not create a system defined solely by some ruleset, but rather tries to describe a certain domain of interest that already exists (whether it be in the “real” world or in “cyberspace”). The advantage of this is that we aren’t bound by any self-imposed restrictions to our domain, but are free to expand and improve our representation to a much greater extent (as far as our representational formalism allows). This has certain drawbacks however. In our search for regularity in information and ease of representation, which must be conducted to some extent in a formalized computer system, the nature of such descriptive approaches entails the existence of exceptions.

A resource can be a class, an instance or even a property in different contexts. This makes it so hard to create a general descriptive ontology of a certain domain, even if that domain is really simple. A normative ontology would however not have these problems, since it by nature is meant to dictate exactly how things are to be interpreted.

But another problem with descriptive ontologies lies just in this potential misalignment with reality. The logics behind the ontologies are there for the purpose of giving unambiguous meaning to resources, but in reality things are often ambiguous. For example, homonyms are words that are spelled the same way but have different meanings. They cannot be expressed as simply themselves, as this would mean that the different meanings of the words are part of the same ontology resources. In a small ontology for a specific purpose, we are able to avoid this problem to a fair extent, since homonyms have nothing to do with each other and are thus parts of unrelated domains. However the more general the purpose of the ontology gets, the more ambiguities from the real world appear. This in by nature inherent in everything human, and cannot be avoided when dealing with trying to describe the real world.

An obvious way of dealing with this is simply to specify the exact purpose of the ontology, without any built-in support for other uses. It will certainly be possible to use the ontology for other purposes, but many assumptions with regards to how resources are handled may not fit. In this way of specifying a purpose what is really done is to make the ontology more normative, in that it dictates clear meaning.

A descriptive ontology in ambiguous domains will not be able to express as much as a normative ontology. Many axioms which clearly defines specific meaning will potentially exclude uses for the ontology that are interesting. This can be solved partially by modularizing the ontology, where there is a core set of axioms that is purely descriptive and excludes no purposes for use. Each specific purpose must in this case create a specific ontology which can import the core ontology. At least some level of reuseability will be accomplished by doing so, but instead of having a single domain ontology, there will be many that can possibly create confusion.

It is important to note that this distinction between descriptive and normative ontologies has to do with how the ontology designer(s) approach the problem of creating an ontology. It is not something that necessarily yields different ontologies. If the domain of interest is unambiguous as it is, the resulting ontology may turn out the same regardless of whether the descriptive or the normative approach is used. The two extreme cases can be described as such:

- **100% descriptive -** The ontology is designed with any conceivable purpose in mind. This extremity can result in an ontology that to be able to accommodate every conceivable purpose says almost nothing, and is thus useless.

- **100% normative -** The ontology is designed with one single purpose in mind. This extremity is not as fateful as the descriptive one. The rigidity obtained here assures full compliance with the use the ontology is designed for, but other purposes may also be able to use the ontology without much trouble.

A.4 WITSML/XML to RDF/OWL conversion and problems concerning this

This section goes more into the difficulties and problems that arose when creating the converter for XML data to RDF/OWL. The text is not as polished as section 5.2, but contains more detail in certain areas. Some of the text overlaps with other parts of the thesis, which is natural as many topics for the converter are important for the whole thesis.

A.4.1 Structured datatypes in RDF and OWL

This section contains pros and cons on different approaches for representing low level drilling data in RDF / OWL; mainly concerning predicate = property (binary only) or predicate = class. In the first case, we have a very simple representation which should be fast to process and easy to convert to and from. In the second case we have a more complex representation which potentially could prove slower and more difficult to use.

To illustrate using an example:

```
<Person id="Peter">
  <Height unit="cm">180</Height>
</Person>
```

Now we show the RDF conversion using the two approaches. A simple representation, only one RDF triple per tag is needed:

```
    Peter rdf:type Person .
    Peter hasHeight_cm 180 .
```

Pros

- easy to convert
- easy to process

Cons

- might lose semantic content
- cannot have n-ary relations
- implicit structure of properties

```
    Peter rdf:type Person .
    Peter hasHeight anon1 .
    anon1 hasValue 180 .
    anon1 hasUnit cm .
    (anon1 rdf:type Height .)
```

A complex representation, several RDF triples needed:

Pros

- no implicit structure
- retains semantic content

Cons

- more difficult to convert automatically
- SPARQL queries are more difficult to formulate

Note that when using the simple approach, there is little doubt what the name of the property should be. In our example, "hasHeight_cm" or similar variations are natural. When using the complex approach, this is not as easy. Since there is no longer a single property from subject to object (from Peter to 180), many possibilities arise. The conversion showed above is just one suggestion for a naming scheme. There are considerations to be made concerning how much we want to express, and if we get problematic redundancy.

The instance "anon1", which can be made of type "Height", represents the complex property. Saying "Peter hasHeight anon1" is then similar to saying that Peter has a height that is a Height. This sounds like redundant information. Perhaps it is enough to say that "Peter has anon1" or even "Peter _ anon1", making the property anonymous. These are questions to which I have no immediate answer. As for the properties from "anon1", they seem more sure. "hasValue" and "hasUnit" are reasonable names for those properties, but these too are open to discussion.

Another problem that needs addressing is the different scenarios of use for the results concerning structured datatypes above. There are two extremes and a middle ground to consider. The two extremes are using simple representation for all properties, and the other using complex representation for all properties, while the middle ground is somewhere between (thus most likely more difficult to implement and use).

The complex representation for all demands that all XML elements are to be transformed into RDF nodes (OWL instances), while element values are RDF literals (OWL concrete domains connected through datatype properties). The RDF predicates (properties) in this case would either be the same for all relationships (say, a single "has" property), or something more complex e.g. each relationship has its own RDF predicate (property). This approach makes it easy to simulate n-ary relations as described above. It does however also mean that relations that can be expressed simply with a single triple becomes unnecessary complex. Creating restrictions in the OWL ontology on top may also become more cumbersome, but this must be more carefully investigated.

The simple representation for all ensures few triples and easy conversion from XML. This also means that the pros and cons from above will be true for the structured datatypes. Whether or not simple representation is enough must be evaluated on a case-by-case basis.

The middle ground assures that structured datatypes are properly expressed, while simple datatypes are expressed with a single triple like “subject datatype-property value”. This approach makes it necessary to treat the two variants differently. We will add to our Person Peter:

```
<Person id="Peter">
  <Height unit="cm">180</Height>
  <Occupation>Journalist</Occupation>
</Person>
```

This will then yield these RDF triples:

```
    Peter rdf:type Person .
    Peter hasHeight anon1 .
    anon1 hasValue 180 .
    anon1 hasUnit cm .
    (anon1 rdf:type Height .)
    Peter hasOccupation "journalist" .
```

The potential problem here is that through “hasOccupation” we get the value “journalist” directly, while through “hasHeight” we get an anonymous instance, which itself has a property to both the value “180” and to the unit “cm”. These are two different ways to get information about Peter, which may not always be obvious. If this middle ground method is to be used, it must be made clear which properties use a simple representation and which properties use a complex one. Maybe this can be helped through naming conventions and such as well. It must in any case be more carefully investigated.