UNIVERSITY OF OSLO
Department of Informatics

SemTask — Semantic Task Support in Integrated Operations

Master thesis

Aleksander Blomskøld
[aleksabl@ifi.uio.no]

Fredrik Klingenberg
[fredrkl@ifi.uio.no]

August 1, 2008

Abstract

StatoilHydro and other oil companies are introducing Integrated Operations (IO) in the management of their oil and gas assets. According to OLF, IO has an estimated value potential of 300 billion NOK on the Norwegian Continental Shelf. IO includes increased use of real-time data, more collaboration in multidisciplinary teams, and increased automation. With the increased use of real-time data, operators may experience information overload. This thesis focuses on the "Daily Production Optimization" (DPO) work process at StatoilHydro. A structured workflow application, called SemTask, is presented. Its main purpose is to help operators overcome the information overload problem by suggesting which data sources to use in different situations. It supports operators in executing the DPO work process, ensuring that the right tasks are done at the right time. SemTask uses ontologies in OWL to describe workflows, data sources, and workflow execution states. The workflow ontology is based on the OMG-standard Business Process Modeling Notation (BPMN). The ontologies form an extendable, flexible model. We propose using rules in Jena, a Semantic Web framework, to implement a workflow execution engine. A prototype that shows that this leads to an elegant solution was created. The SemTask model, and how it deals with the DPO scenario as well as possible further extensions, is discussed. SemTask offers an extendable solution for data source suggestions and active help to operators during execution of workflows in Integrated Operations.

Acknowledgements

This master thesis is the result of a collaboration project between the Department of Informatics, at the University of Oslo, and StatoilHydro. Many people have helped us to prepare this thesis. First we would like to thank Trond Lilleng (Integrated Operations coordinator at StatoilHydro) who was the initiator of this project, Torhild Rio (Hydro IS Partner) who gave us an initial introduction to APOS, and Espen Halvorsen, our main contact person in StatoilHydro. Secondly, we thank Jon Henrik Forssell, Magne Valen-Sendstad, and Pål Rylandsholm from Det Norske Veritas (DNV), especially for the ISO 15926-workshop, and the PRODML-presentation held by Jan Ingvar Riveland (Tietoenator). We would also like to thank Frédéric Verhelst and Bård Henning Tvedt from Epsis for the two-day seminar that gave an excellent introduction to oil reservoir and oil production. Kjetil Torvanger (Technical Manager in Cronus Automation West) also deserves our acknowledgement for the introduction to the Process Data Portal. Thanks to the people at the IRC-channels ##logic, #swig, and #jena on Freenode for always giving quick and helpful answers to our questions. We appreciate and thank Olaf Owe (professor, UiO/IfI) for the time he spent answering our questions about Pi calculus and Maude. Fredrik would like to thank Computas for the summer job in 2007 that gave him a good insight in ontologies and workflow systems for the oil and gas industry. From Computas we would especially like to thank David Norheim for answering questions about OWL and Steinar Carlsen for the introduction to BPMN and access to the BPMN 1.1 specification. We would also like to thank Rolf Schwitter at Macquarie University, Sydney for introducing us to the exciting field of Semantic Web. Last we thank our advisors Arild Waaler (professor, UiO/IfI), especially for helping us with issues regarding the ontologies and establishing the contact with Epsis and DNV, and Roar Fjellheim (Computas and professor II, UiO/IfI) for creating the problem description, establishing the contact with Hydro, proofreading, and giving us general feedback during the work with this thesis.

Contents

Figures

Listings

Chapter 1

Introduction

> The best way to be boring is to leave nothing out
>
> Voltaire

1.1 Problem statement — Semantic task support

1.1.1 Original problem statement

The candidates shall examine the tasks and information sources for operators in Hydro's Operations Center of Oseberg East at Sandsli (Bergen). A task ontology shall be established (in OWL), which can be used to describe the tasks / work processes and how these are related to different situations and information sources. Moreover, the candidate shall show how a reasoning system based on the ontology can help the operators to select the right source of information in various working conditions (normal conditions and critical situations).

1.1.2 Changes

The original problem statement has been slightly changed during the work with this thesis. As we did not get as much time with the operators at Oseberg East as we needed, we never learned enough about their different working conditions, especially in critical situations. As a consequence, we only looked at how a workflow system based on ontologies could help the operators in normal working conditions.

1.2 Motivation

The oil and gas industry is highly technical. Large sums are invested in research and development, and cutting edge technology is used. Currently there is an important focus on Integrated Operations (IO). IO is a new way of working with increasing focus on multidisciplinary teams and use of real-time data. The goal is to make better decisions, which result in lower costs and increased incomes. One problem with increased use of real-time data is the possibility of information overload. It has been said that operators use about 40% of their working time searching for the information they need. The problem will not be smaller in the future when even more real-time data is available for the operators.

When experienced operators are leaving, new ones need to be trained. One way of reducing the problem of losing intellectual capital, is by having good *workflow systems*. They help inexperienced employees to do the work the experienced employees might just know by heart.

1.3 SemTask

SemTask is an abbreviation for Semantic Task support. This thesis presents the SemTask application. It is a workflow system based on *ontologies*, meant to help the operators in Hydro's Operations Center of Oseberg East getting suggestions on data sources to look at while working on a particular task. It can also serve as a general workflow system helping the employees doing the right tasks at the right time. We have ideas on how SemTask may work, and have proposed an internal model based on ontologies.

1.4 Organization of thesis

Chapter 2 — Integrated Operations and StatoilHydro

To understand the problem statement one needs to understand sufficiently enough of the oil and gas domain. This chapter gives an introduction to oil and gas production. The oil industry is currently undergoing some huge technological and organizational changes, which go by the name "Integrated Operations", and these are discussed. The chapter also gives information about StatoilHydro and the key IT applications the company uses to support workflow and information storage and retrieval. At last, the chapter presents some of the most important industry standards in the field of oil and gas production.

This chapter is a summarization of the information collected from meetings and seminars at StatoilHydro and Det Norske Veritas DNV in addition to available IO material.

Chapter 3 — Workflow

A major focus of Integrated Operations is to improve the work processes. To be able to carry out these changes in an organization, workflow systems are of great help. This chapter is a summarization of workflow literature with: what workflow systems are, why they are important, and examples of workflow languages and systems that exist today. SoluDyne, Hydro's workflow system, and the Business Process Modeling Notation (BPMN) are presented.

This chapter outlines the weaknesses of SoluDyne and explains why BPMN is a better alternative as a workflow notation. An example of how SoluDyne diagrams can be mapped into BPMN is also presented.

Chapter 4 — Introducing SemTask

This chapter presents the most important work process for this thesis: Daily Production Optimization (DPO). It is a work process that the operators at Hydro's operation center of Oseberg East go through every day.

We outline the weaknesses and optimization potentials of the current way DPO is done. A scenario exemplifies how these weaknesses can be addressed by using the *SemTask* application. Then a comprehensive mapping from DPO in SoluDyne into BPMN shows how much knowledge can be transferred from documentation into workflow diagrams. This mapping is the first step in making the SemTask scenario possible. The chapter also provides an overview on how the rest of SemTask is to be implemented.

Chapter 5 — Ontology

Ontologies will be discussed in this chapter, as they are a key part of the SemTask architecture. An introduction to the Web Ontology Language (OWL) along with some of the other *Semantic Web* technologies is presented. The chapter outlines different ways of representing the workflow model in SemTask, and explains why ontologies are the best choice.

Chapter 6 — SemTask ontologies

In this chapter the four ontologies we created, which are part of the SemTask architecture, are presented: BPMN, SemTask, Data, and Execution. The chapter starts with some general information on methodology and high level design decisions regarding the ontologies. Then some design decisions around the BPMN ontology are explained.

As BPMN currently does not have a standardized serialization format, we propose using our ontology for this. The ontology is also compared to the sBPMN ontology by the SUPER project, which also brings semantics into BPMN.

Finally we have created a highly modular and extendable architecture used for workflow execution using ontologies. This architecture is thoroughly explained.

Chapter 7 — Execution model

Until now a model which can represent workflows, with state along with data sources, has been discussed. This chapter explains how an execution model based upon *rules* could work. We start with a general introduction to rules and rule languages. In SemTask, the rules are separated from the model. This makes it easy to extend the functionality by adding rules without altering the model. We have created examples of rules that partly implement the workflow execution engine. Thoughts about how the rest of the different workflow diagram elements could be implemented, to complete the workflow execution engine, are presented.

Chapter 8 — Conclusion and further work

The last chapter contains conclusions and offers some thoughts about further work and possible extensions of SemTask.

Acronyms

A list of acronyms used in this thesis is included on page 100. For a list of oil-specific terms we refer to The Norwegian Petroleum Directorate "Terms and definitions"-page at `http://www.npd.no/English/Om+OD/Nyttig/Olje-ABC/Ordliste/olje_ordliste.htm`.

Appendix A — The SemTask OWL Ontologies

The appendix gives the full source of our four ontologies. The source is presented in the OWL Abstract Syntax format.

Appendix B — The RDF Documents

In this appendix two RDF documents are presented in full source. First, the DPO workflow RDF document is presented (based upon the BPMN and the Semantic Task Support ontologies). Then an RDF document of some data sources is presented (based upon the Data ontology). The RDF documents are written in the N3 format.

Appendix C — Jena Rules

The last appendix presents two rules which show some basics of how a workflow may be executed with the help of the Jena Semantic Web Framework.

Chapter 2

Integrated Operations and StatoilHydro

This chapter gives a brief introduction to oil and gas production on the Norwegian Continental Shelf (NCS). Some of the general challenges of the oil and gas industry are outlined, and an introduction to Integrated Operations is presented. In addition, it is focused on ex-Hydro at Sandsli and how the Hydro operators are coping with their daily tasks. StatoilHydro's workflow systems, IT applications, and data sources with regards to production optimization are also presented.

2.1 Introduction to offshore oil and gas production

The Norwegian oil adventure started in the middle of the 1960s, when the Ekofisk field was discovered and developed. Since then more than 60 oil and gas fields have been successfully developed and recovered. Oil and gas production on the NCS is a highly technical task involving several engineering disciplines.

The life cycle of an offshore oil and gas project starts with searching for hydrocarbons[1]. Data is among others gathered with airplanes (measuring magnetic fields) and with seismic survey boats. The data is transformed into visual models and analyzed by geologists. If there is a high enough probability that sufficient amounts of hydrocarbons exist, test drillings start. This is done with drilling rigs which drill exploration wells. These wells are either wildcat wells (used to check whether hydrocarbons exist) or appraisal wells (used to determine the size of a petroleum deposit). Rock and sand

[1]Hydrocarbons is a term used for oil, gas, condensate, and Natural Gas Liquids (NGL).

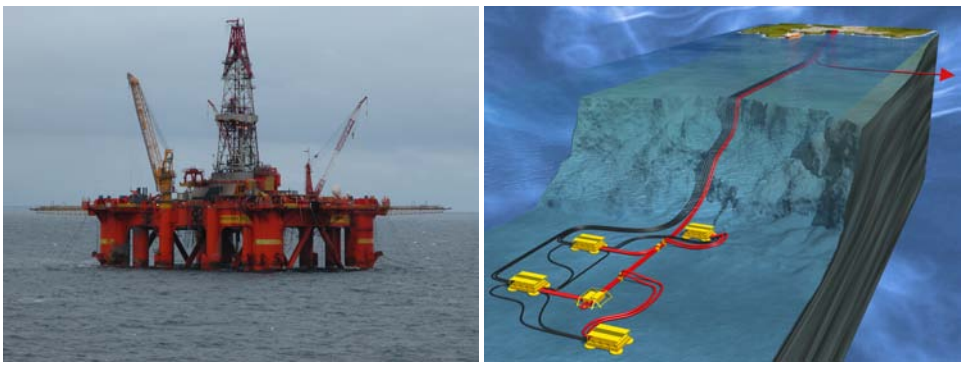


Figure 2.1: *A picture of a normal oil platform with a topside production plant, and a graphical representation of an underwater installation that bring fluids to an onshore production plant.*

samples are brought onshore to be interpreted by different geologists and other oil experts. If the experts find the discovery commercially viable the recovery (or production) phase can start.

Several types of production platforms exist: those that are completely subsurface with the processing plant onshore (for instance Ormen Lange), the more usual ones with an oil rig with a topside processing plant, and floating boat-like processing plants. Figure 2.1 shows how these two types of platform look like.

After the exploration phase has finished, the exploration wells are plugged and new production wells are drilled. There are three types of wells: *recovery wells*, where oil and gas are flowing to the processing plant, *injection wells*, which are used for regulating the pressure of the reservoir, and *observation wells*, which are used for observation and testing purposes. Some of the wells may be a combination of these three types. The wells are drilled with the help of real-time drilling data (including seismic data) and with the presence of multidisciplinary teams including geologists, physicists, oil and drilling experts. In order to produce maximum amounts of hydrocarbons from a field, it is crucial that the wells are drilled in order to cover as much of the field as possible. For an offshore field, tens of wells are typically drilled.

The production phase starts when the production platforms are placed offshore, and the first production wells are drilled. This phase typically lasts for a couple of decades. It involves monitoring, maintenance, regular production optimization, and various types of manual day-to-day work.

2.1.1 Production optimization

One part of the production phase is to perform production optimization. Production optimization is to monitor several measurements from sensors and adjust some parameters that lead to enhanced production. Examples of measurements include temperatures, wellhead pressure rates, flow rates, and well/pipe/tank/process capacities. Short-term parameters that may be adjusted are mainly choke values (regulations of valves) and adjustment of pressure with the injection wells. Examples of long-term actions that can be done to increase production are: drilling new wells, pump acids in the reservoir to increase the flow, and increasing topside processing capacities by replacing or improving equipment.

There are several things that make production optimization difficult. One problem is the large amount of data available. As more information may actually lead to worse decisions [Schroder et al. 1967], *information overload* is a big problem. There may be tens of thousands of sensors connected to a field/platform sending data several times a minute. Finding the important information is difficult. To cope with this problem, the industry is trying to come up with new ways to visualize the data to make searching for information easier. Computer programs watching for measurements that are outside predetermined threshold values and abnormal trends (like a well pressure that is falling unnaturally fast) may also help to minimize the information overload problem.

Another problem is *uncertainty*. Each measurement is associated with an amount of uncertainty. There are not enough sensors to cover all the necessary information. For instance, it is very difficult to get enough information to decide the ratio of water/gas/oil in a production well. This ratio is important to know when dealing with production optimization, as one always wants to minimize the amount of water to pump up and increase the amount of oil. What typically is known is the well's volume flow in total and the ratio after the well has mixed its fluids with other wells in the field. With a manual process it is possible to find the ratio in a single well, but one cannot check this continuously. Hence, the data used for making decisions might be out-of-date.

A third problem that makes production optimization difficult is that a local optimization may affect the global properties of the reservoir. For instance, if one wants to decrease the choke in one production well and increase the choke on a second because there is a larger ratio of oil in the first one, a third well may be affected because of the changed pressure. Complex reservoir simulations need therefore to be run at regular intervals, to help the operators making the right decisions.

Yet another issue about production optimization is that there is often a trade off between higher productivity and lower safety levels. Increasing the well pressure may yield a higher production rate, but it will probably also increase the risk for equipment damage, etc.

2.2 Integrated Operations

The oil and gas industry sees a large economic potential in increased use of IT. Integrated Operations (IO) is an initiative meant to change the way people work, enabled by IT tools. IO is a broad term used for increased collaboration across multiple locations and disciplines, increased automation, more streamlined work processes, and increased use of real-time data. Implementing IO involves organizational changes and new information and communication technologies. It promises to increase efficiency, reduce costs, and enhance hydrocarbon recovery. Moderate calculations by The Norwegian Oil Industry Association (OLF) state that the value potential of implementing IO on the Norwegian continental shelf is at least 300 billion NOK [OLF 2007].

Throughout the industry, different definitions and opinions about IO exist. Examples of similar terms are: "Smart Operations", "eOperations", and "Smart Field" to name a few. As StatoilHydro, OLF, and The Norwegian Oil and Energy Ministry are using "Integrated Operations"; this term will be used here. Some in the industry focus mostly on increased collaboration between vendors and producers, some on increased automation of work processes, while others focus on better use of real-time data. IO is defined by the Norwegian Government to be: "Use of information technology to change the work processes to achieve better decisions, steer by remote control equipment and processes and to move functions and personnel to land" [OLF 2006]. Information technology has been used for several decades by the oil companies. So even if the IO definition includes the key elements: "Use of information technology", "change in the work processes", and "the relocation of functions" these are not exclusive to the definition of IO. What IO does to optimize the industry is to use IT as a strategic tool for increasing the effectiveness of work tasks and make the decision processes in oil and gas production on a large scale easier. IO has also introduced a comprehensive redistribution of work tasks between units placed on land and at sea.

2.2.1 Benefits

There are several reasons for introducing IO. The economic motives include:

- Increased exploitation [Ministry of Petroleum and Energy 2007a]

- Accelerated production 5%-10% [OLF 2005]
- Reduced operating costs 20%-30% [OLF 2005]
- Higher safety levels
- Extended field life (See figure 2.2 on the following page)
- Enhanced oil and gas recovery (EOR) 3%-4% [OLF 2005]

2.2.2 The IO development

IO enables increased interaction and collaboration between offshore platforms and onshore support centers. Concentrating information and decision making onshore make it possible to let experts work on different offshore fields during the day. Real-time data from sensors are transferred via high capacity fiber links from offshore facilities to operational centers on land. Here multidisciplinary teams collaborate to optimize operations and rapidly solve problems, often using video conference equipment.

The traditional practices have been: self-sustainable fields, specialized onshore units, and periodic onshore support. There is limited work process interaction across disciplines, between onshore and offshore, and across companies. Fragmented problem solving and solutions created in isolation are some of the unfortunate consequences of today's work processes.

"Integrated Work Processes will most likely be implemented in two stages, i.e. first by Generation 1 (G1), then by Generation 2 (G2) processes" [OLF 2005]. G1 of IO is improving and, to some extent, solving the problem of fragmented problem solving. This is done by using integrated onshore and offshore processes and centers, and also providing continuous onshore support. The disciplines work together in making plans and executing them. Some of the platforms are running pilots, while others are already into the implementation phase. Well planning was, in the traditional way before G1, done in a more or less sequential way. Implementation of G1 leads to a more integrated process where several people with different expertise work together. However, it is a potential improvement related to real-time integration of acquired data and updating of geological and reservoir models. In production optimization several improvements are brought into life as a result of implementing G1 work processes. One example is that onshore functions are more available for the offshore operators.

G1 work processes are implemented in different ways depending on the company in question. However, several common features are present:

- Onshore centers staffed by professionals

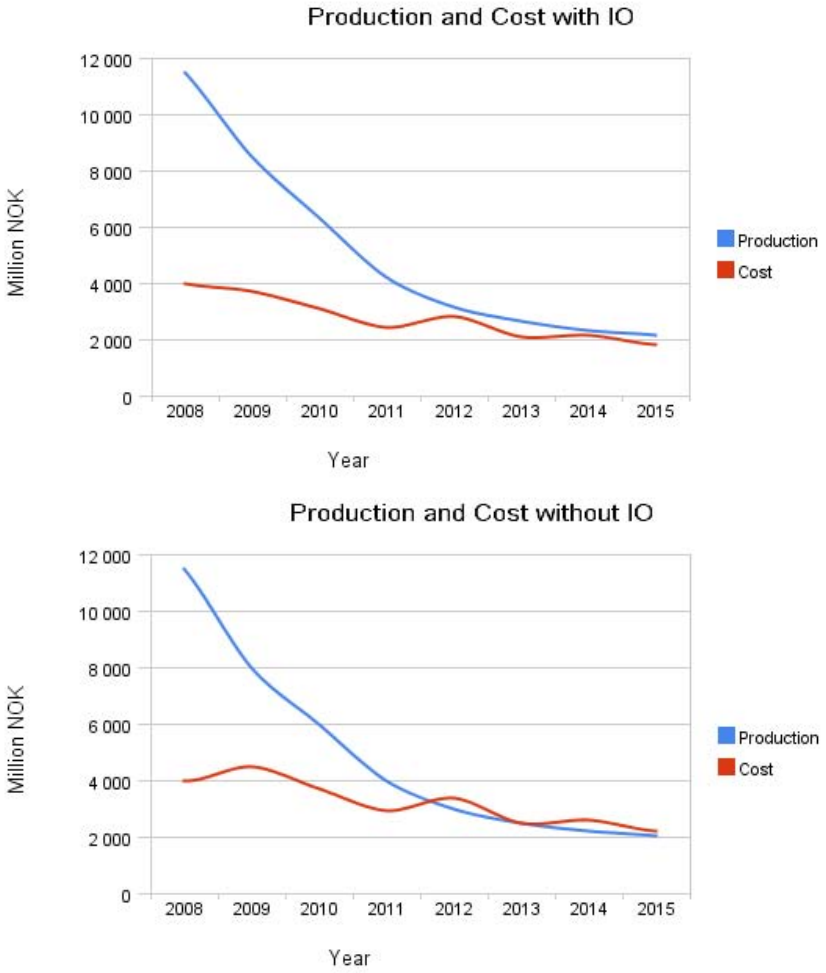


Figure 2.2: *This is an example of how cost and production at an oil platform may develop with and without IO. When the cost is reduced with 15% and the production is increased with a small percentage, the lifetime of the platform may be increased substantially. The numbers used for making these graphs are based upon [OLF 2006]. The Brage platform is a concrete example of a field that has increased its lifetime, probably for at least 10 years. [Holst and Nystad 2007]*

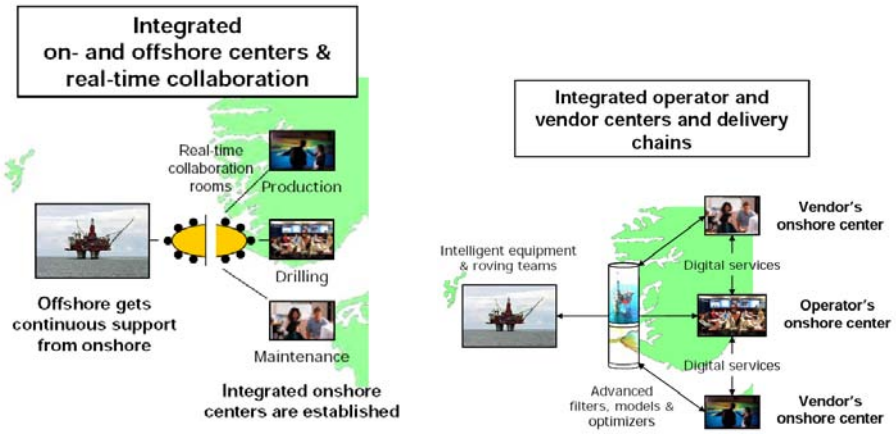


Figure 2.3: *Differences between G1 and G2 work processes*

- 24/7 onshore support for areas like drilling
- "What-if" analysis performed onshore
- Direct involvement of production and maintenance vendors

In well completion, the phase where a well is prepared for production of oil and gas, virtual reality models of completion are being used. In addition, real-time monitoring of the wells with advanced equipment is utilized.

Trends and benefits of G1 will be enhanced further in G2 IO. Operators offshore in Norway have already started prototyping and piloting G2 processes. Digital services will be introduced, and vendors will be included in a different manner. "The vendors will participate actively in the operation of fields through delivery of digital drilling, well, production, and maintenance services" [OLF 2005]. The field operators will be operational 24/7. Daily production will be automated and run safely. Complex IT applications will be used to a large extent in providing useful 3D models, statistical and analytical decisions guidance to name a few areas. More intelligent filtering, automation of processes and decisions will prevent information overload for the operators.

The next and final step of IO would be to automate more and more of the work processes, which depends heavily on IT applications.

2.2.3 Implementation requirements

Several requirements must be met to make G2 processes a reality. Development and implementation of G2 processes will naturally be more

demanding than G1 processes as they are more complex. Digital infrastructure is necessary to support connections between vendors and operators. An information security framework for reliable and secure connection with third party access will be required. Data standards are necessary to ensure that data is correct and understood correctly across disciplines and companies, and they need to be developed and implemented. Here Semantic Web technology is a promising approach [Teijgeler 2007]. "New technologies and solutions for broad scale instrumentation of installations, real-time gathering of information and real-time optimization of operations are required" [OLF 2005]. Organizational changes will be a necessary precondition to be able to fully take advantage of G2 processes. Several of these issues have been addressed by OLF projects.

2.2.4 Slow adopters

Unfortunately, the oil industry is a conservative business. Although there are strong forces working for modernizing the business (such as OLF), there are also people who are more satisfied with the ways things currently work. It has been said that many of the changes with IO we are seeing happening now, happened more than a decade ago in the automotive industry [Øystein Fossen 2007]. There may be several reasons for the somewhat slow transition to IO. One is that offshore oil production companies earn a lot of money, even though they are lagging in their use of technology. Another problem with introducing new technology in an oil company is that they are rather short-handed on human resources. Every work-hour spent on testing and evaluating new technology is an hour lost spending on short-term production optimization.

David Ottesen, CEO in Ziebel (a multinational company working with smart wells), said this about the situation: "The technology is here, yes. It's the determination we are lacking. The contracts are designed such that suppliers earn more when things take more time and more people are used. That is the exact opposite of what we should be striving towards." [Rasen 2008]

2.3 StatoilHydro

StatoilHydro is one of the largest offshore oil companies in the world. It operates in about 40 countries, and is worth more than 500 billion NOK.[2]. The company has most of its activities in Norway, and is the largest operator

[2]StatoilHydro was worth about 540 billion NOK at 31. December 2007 [StatoilHydro 2008].

on the NCS (about 80% of the total production) [Ministry of Petroleum and Energy 2007b].

2.3.1 Statoil and Hydro merger

During the work with this thesis the two companies Hydro and Statoil merged. The companies are now harmonizing their different processes, policies, and IT applications. In areas with differences in IT applications between the two companies we will mainly focus on Hydro's application unless otherwise specified. For instance for documenting work processes Statoil used Business Process Management (BPM) along with a software called Docmap, and Hydro used APOS ("ArbeidsProsess Orientert Styringssystem" or "Work Process Oriented Administration System") and a software called SoluDyne[3]. As a basis for this thesis we will use ex-Hydro's system, APOS.

2.3.2 Hydro at Sandsli — Oseberg East

StatoilHydro has a department at Sandsli, Bergen that was previously owned by Hydro. At this department several offshore oil platforms are operated, including the Oseberg East.

Operator rooms

Each platform has its own dedicated operator room onshore. In the operator room dedicated for Oseberg East, video conferencing equipment is installed. This helps to share experts between different platforms, and let more people work on land. The video conference equipment has possibilities for showing live video and shared IT applications on large screens. It is used several times a day to have meetings with offshore and onshore personnel. Some of the meetings are recurring, such as the daily morning meeting (going through plans for the day) and the daily production optimization meetings.

A 3D display is used for getting a visual representation of the reservoir served by the platform. Wells can be seen along with geological data.

2.3.3 Data

StatoilHydro uses many IT applications for oil and gas development, divided into the following categories:

[3] http://www.soludyne.com/

- Well design
- Flow assistance
- Well surveillance
- Production optimization
- Reporting

As previously mentioned, the oil industry is continuously increasing their use of data, especially real-time data. Each oil field has thousands of sensors regularly sending measurements in fixed intervals. Currently most of the fields operated by the operating center at StatoilHydro in Sandsli have data coming in at an interval of 4 seconds. This makes the load of data very high. The data is currently stored in a database called IP21 from AspenTech[4]. To help the operators visualize and make use of the data a portal layered on top of IP21, named Process Data Portal, is being utilized.

Process Data Portal

Traditionally, different technologies are chosen for different oil and gas platforms. This leads to different interfaces and operating environments. Operators may therefore potentially waste time looking for the required piece of data. Also, if an operator is working with different platforms, she might not be familiar with the interfaces, which may be cumbersome. Because of this, the Process Data Portal (PDP) was created. Its purpose is to streamline access to real-time data from different sensors, historical data, alarms and events, and finally reports. In addition, it is an integration layer for other applications. PDP is originally from former Hydro.

With PDP the topology has changed considerably from what can be seen at figure 2.4 on the next page to more structured as can be seen in figure 2.5.

We will not go into detail on the new topology. However, a quick overview explains which direction IO is going. Two IP21 databases which are mirrors of one another are present; one located onshore, the other located offshore.

PDP has an abstraction layer between the applications and the IP21 database. It handles data requests from applications so clients do not have to deal with the native IP21 format. There is also a web server with different applications. Both onshore and offshore users access the applications in the same way. Data moves both to and from the central IP21 database. Data going back to the database are typically the results of heavy computations so they only need to be done once.

[4] `http://www.aspentech.com/`

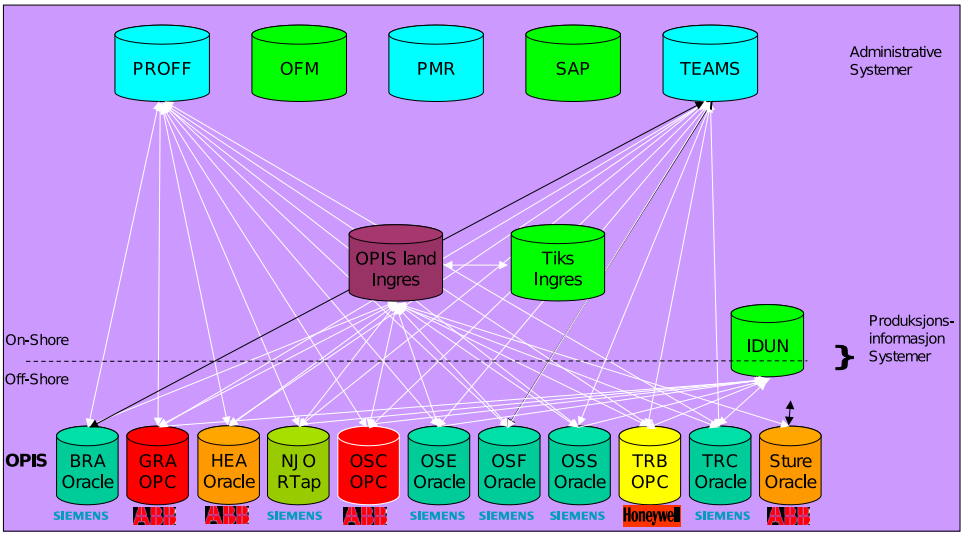


Figure 2.4: *Topology before PDP*

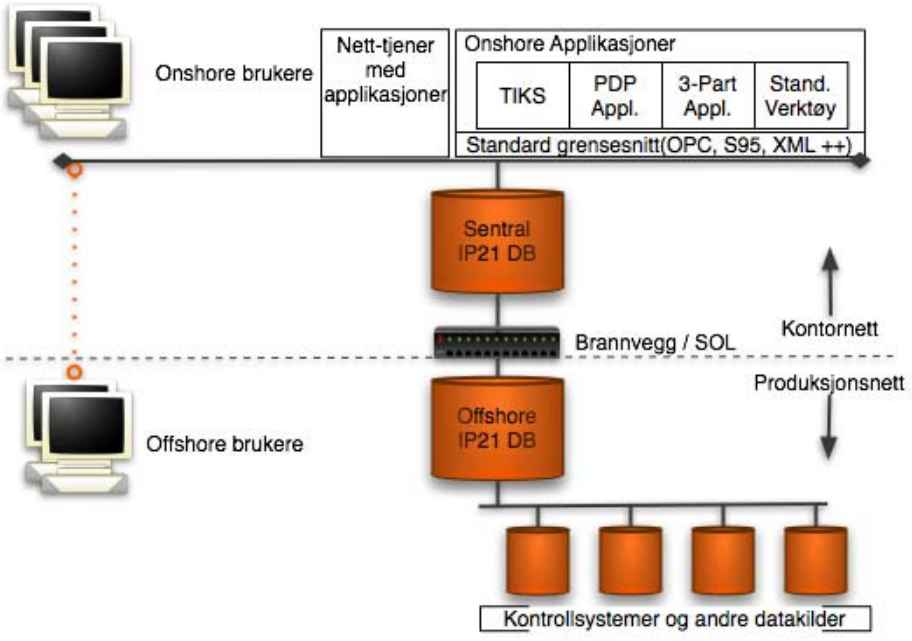


Figure 2.5: *Topology with PDP*

Figure 2.6: *SharePoint interface for the daily morning meetings*

PDP is in addition a package of different standard applications like "production overview" and "subsea well outline". Consequently, the operator does not have to use much time searching for data. A fixed template has been made out of Web Parts which is integrated in SharePoint[5]. The Web Parts are dynamic in the sense that one can click on them to bring up more detailed information or other numbers. See figure 2.6 for a screenshot.

Another example of PDP would be subsea well overview with alarms (see figure 2.7 on the following page). Here the operator can easily see the actual production compared to the planned/estimated one for the wells. Different color codes, red for "alarm", green for "normal", and yellow for "need attention", make it easy to focus on the subsea wells that need watched closely. One has the option to look at the state of the system in intervals of 1hr, 6hr, 24hr, and 72hr to see trends and to get a good general view of the wells at hand.

2.4 Industry standards

As IO is about collaboration between different entities, much work is being done to make petroleum specific standards.

[5]SharePoint is an intranet portal solution for Microsoft — see `http://en.wikipedia.org/wiki/SharePoint` for more information.

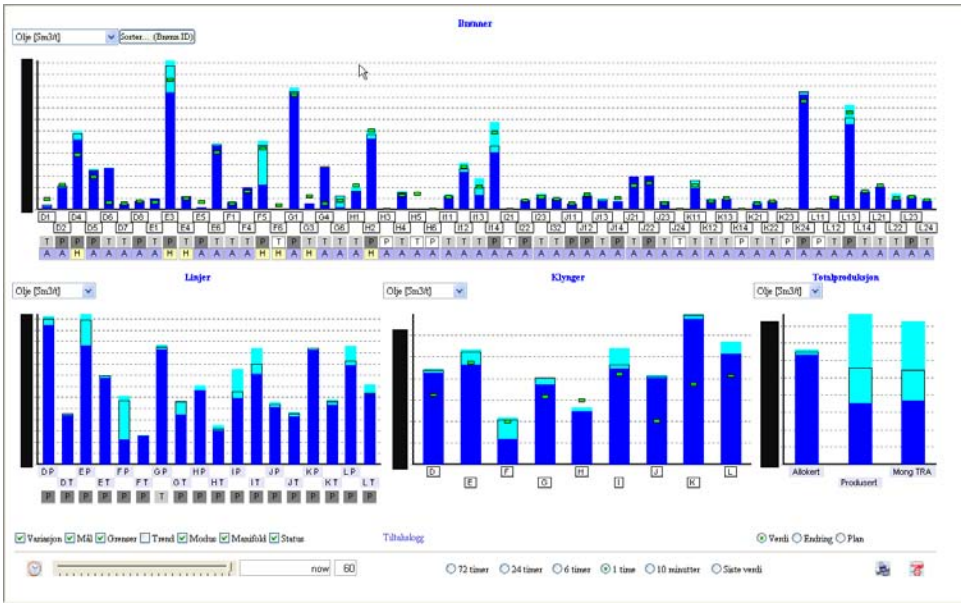


Figure 2.7: *A screenshot showing the ActiveX control for subsea well monitoring*

2.4.1 ISO 15926

ISO 15926 ("Industrial automation systems and integration — Integration of life-cycle data for process plants including oil and gas production facilities") is a large standard for data integration. It has its roots from a European Union project started in 1991, and it is not currently finished. It consists of seven parts, where the first five have been standardized, and the remaining two have just been submitted to ISO for review.

The standard is a large ontology that covers definitions of meta objects (thing, class, individual), generic engineering concepts (such as physical objects, connections, activities, etc.), and text book concepts typically found in processing plants in the petroleum industries (pump, reducer, heat, etc.) [David Leal 2005]. It will also cover more user-defined classes such as company commodity classes and catalogues from suppliers and manufactures. The ontology is quite large — part 4 of the standard, which is called the reference library, covers about 100.000 classes important to process plants and the petroleum industry. The standard covers everything from specific types of pumps and "A standard gas hour" to "Class of class of physical object".

A main use of ISO15926 is to ease data integration. Today much time is being spent converting data to different data systems from different entities

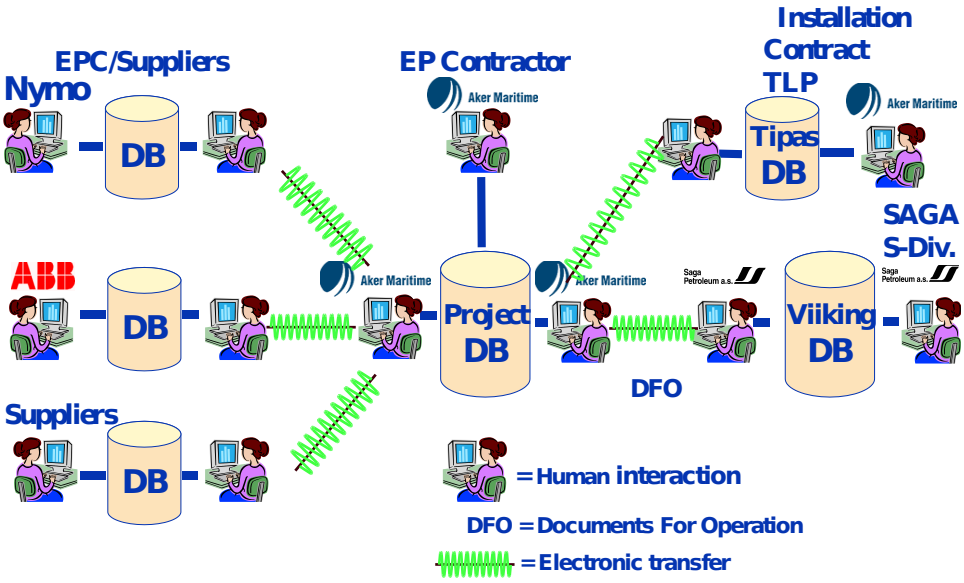


Figure 2.8: *An example from a real-world project of how much human interaction is required because different entities use different data formats. The figure is taken from [Valen-Sendstad 2008].*

in the value chain. In figure 2.8 one can see an example of how much human interaction is done in a real-world project.

For more information about ISO 15926 see the Wikipedia-page[6] or the standards themselves [ISO 2004; 2003; 2007].

2.4.2 WITSML

WITSML [Energistics 2008] (Wellsite Information Transfer Standard Markup Language) is a standard for data access and integration in the domain of well drilling. It is being developed by POSC/Energistics[7] and is much used in the industry.

Technically, the standard is a set of XML schemas (XSDs) with documented descriptions of terms. The standard also covers an API for accessing the data, which are based upon web services (SOAP and WSDL).

For more information about WITSML see the WITSML-Wikipedia page[8]

[6] `http://en.wikipedia.org/wiki/ISO_15926`

[7] POSC (Petrotechnical Open Standards Consortium) — see `http://posc.org`. In 2006 POSC was rebranded to Energistics — see `http://www.energistics.org`.

[8] See `http://en.wikipedia.org/wiki/Wellsite_information_transfer_standard_markup_language`

or the WITSML homepage [9]

2.4.3 PRODML

"[PRODML is] a shared solution for information exchange allowing Upstream Oil & Gas Companies more easily execute those software applications used to optimize production" [Morneau 2007]. PRODML (Production Markup Language) is a standard for data integration between different applications dealing with production optimization and production reporting [Energistics 2006]. It was created by a working group with participants from 13 upstream oil and service companies in the oil and gas domain [Shipley et al. 2008]. It was later handed over to POSC/Energistics for maintenance. The standard is based upon two Norwegian standards that were used for production reports sent to government agencies and to partners: Partner Production Daily Reporting and Monthly Production Reporting Data Transfer Standard.

More specifically the data that PRODML can represent is mostly about what flows where, how much and when. An example of a statement represented in PRODML is: "X Sm^3 of condensate flowed between pipe connection A and pipe connection B between timestamp X and timestamp Y". The aim is that all sensors should produce PRODML-data. The data should then be processed by production optimization applications or sent directly to partners or government agencies for reporting.

PRODML is built up much in the same way as WITSML. There are XML schemas that describe how to represent some terms found inn ISO 15926. The standard covers a data access API in the same way as WITSML.

For more information about PRODML, see the PRODML homepage[10].

[9] http://www.witsml.org/.
[10] http://www.prodml.org/

Chapter 3

Workflow

> The object of all work is production or accomplishment and to either of these ends there must be forethought, system, planning, intelligence, and honest purpose, as well as perspiration.
>
> Thomas Edison

The previous chapter presented an overview of the oil industry in general, and StatoilHydro in particular. Additionally an introduction to Integrated Operations along with there challenges and opportunities were made. As a part of those challenges, changed work processes were proposed to handle the issue with fragmented problem solving process and to enable support of tasks in multidisciplinary teams. To be able to change the work processes, they should first be documented. This documentation is often done in a *Workflow System*.

This chapter takes a look at the area of workflow and workflow systems. *SoluDyne*, the workflow system StatoilHydro uses today is presented, as well as *Business Process Modeling Notation* (BPMN), an OMG workflow notation standard.

3.1 Definition

According to the Dictionary of Computer Science, Engineering, and Technology, workflow is defined as:

"Workflow: multiple tasks/steps/activities, of which there are two types: simple, representing indivisible activities, and compound, representing those

which can be decomposed into sub activities. An entire workflow can be regarded as a large compound task. The workflow defines the relationships among the activities in a project, from start to finish. Activities are related by different types of trigger relation. Activities may be triggered by external events or by other activities. Also the movement of documents around an organization for purposes including sign-off, evaluation, performing activities in a process and co-writing." [Laplante 2000]

A more informal definition and more according to the use of StatoilHydro would include the control and flow of tasks. The flow is formally described and is repeatable and reliable. The workflow can be described in a static way where participants, data and processes are linked together. Some workflow descriptions may be executable, while others may only serve as a manual or documentation for humans. It helps the people working together to have a common understanding of the overall goal and how they can get there. By formally describing the workflow one can also better document and see improvements. These improvements can then be included in the better and newer workflows. In that way not only the work of the participants in the workflow, but also the workflow itself is always the current best practice. This means that the workflow describes the best way of doing something. An example could be the workflow of buying a movie ticket. Here the participants are the customer and the ticket seller. Resources are money and ticket. The order of events is not incidental, for instance: the seller must first make sure that she has more tickets to sell before selling to the buyer. For a more comprehensive description of workflow systems see [Georgakopoulos et al. 1995].

There is also the question of what level of abstraction the workflow should describe. Should one describe the inner workings of every aspect of the workflow components? To continue the "selling a movie ticket" example, should one remove the workflow part that is included in the process of checking whether there are any tickets left? In other words should one model the database transaction, data transfer protocols, etc.? This clearly depends on the domain. From a cinema's standpoint the database transactions concerned in the "ticket selling" is irrelevant.

The workflow can be described using illustrative symbols representing a given functionality or process. The Business Process Modeling Notation has for example the symbols shown in figure 3.3 on page 30. Also here the level of abstraction comes into play. The language might have elements describing low level operations or higher level elements.

3.2 Different workflow systems

In general there exist three kinds of workflow applications. The first two support long-lived workflows that are controlled by humans, and tend to focus on human interaction: sending documents from one person to another, nice user interface, etc. The term "human-centric" is often used for such workflows.

The human-centric workflow systems may either be passive or active. Passive workflow systems are just a type of documentation systems. SoluDyne, discussed in section 3.6 on page 25, is such a workflow system. Users can read and write documents and workflow diagrams. Active workflow systems have an execution engine. The engine interprets events (e.g., timer going of, documents submitted to server), and acts on these according to the predefined business processes. It also facilitates the flow of information, tasks, and events.

The other type of workflow applications has little or no concept of human roles. These workflows are typically written in BPEL or similar language. They focus on automating interactions among software components (often Web Services), and are sometimes called "engineering workflows".

3.3 Why use workflow systems?

Introducing workflow into a business or company leads to several benefits: more formal understanding of the key processes and discovery of improvements. One can argue that by having a formal notation and diagram of a work process it is more likely to be improved upon due to the fact that improvements can more easily be implemented and documented. Not just in the making of the workflow diagrams, but also later there is a possibility to discover improvements. There are side effects when improving the diagrams such as increased safety. Errors are less likely to occur when work processes are formal and documented. They are directly connected to the safety of onshore and offshore personnel. Equipment that once had a function can become redundant as a result of work process improvements or other more cost efficient solutions is discovered. Efficiency could increase and ultimately save money and time.

Knowledge and expertise are often centered on a few individuals. It would be more beneficial to have it in the company itself. In that way the company is not as dependent on the individuals, but keeps the knowledge independent of the working staff. When key individuals leave the company, replacing them becomes easier. Training inexperienced crew operators is easier when

manuals of working tasks are available. Computer applications that guide the users through the workflow may be especially beneficial for inexperienced personnel. Another aspect of separating knowledge from experts in the company by using workflow systems, is that process optimization is easier. Optimization is difficult when there is great emphasis on human knowledge and judgment. In strict workflow systems optimization does not depend so much on the skill and knowledge of the operator, but is instead on the workflow itself. As a result of optimization the productivity could also be improved.

By having a formal notation and diagram of the work processes, they are more likely to be improved. This is because improvements can be more easily implemented and documented in diagrams. One may discover improvements, not just in creating workflow diagrams, but also later in their lifecycle. Finally by introducing workflow diagrams the work processes go from ad-hoc human oriented control to well documented, repeatable, and predictable.

With active workflow systems the tasks will be executed in the same way given that the data input is the same. As a result one gets no deviations from the initial optimized workflow due to human interactions as could be the case with passive human centric workflow.

3.4 Limitations

Even though workflow systems clearly have several advantages, they also have weaknesses. There are areas and uses where other modeling languages are used with better results. One such language is $\Pi - calculus$, which is a theory of mobile processes. The word mobility refers to the movement of links in a space of linked processes. An example could be the changing connections between telephones and the network of base stations as they are carried around[1]. A regular workflow language/notation has problems in accomplishing this.

The execution of a single workflow is easily done, but problems could occur in an execution of several tightly connected parallel workflows: two workflows from two different vendors trying to simultaneously buy the last product from a service provider for example.

Workflow systems often rely on one database and one workflow engine. This architecture could easily become a bottleneck in a large enterprise.

[1]For more information about the Pi-calculus see [Sangiorgi and Walker 2003, Milner 1999].

A limitation due to different vendors and their proprietary workflow notations is the lack of collaboration between different workflow systems. Workflow systems incorporate in their design very concrete and exclusive interpretation of the world which makes it difficult to federate different systems [Alonso and Schek 1996, Jae-Yoon Jung and Hoontae Kim and Suk-Ho Kang 2006]. To overcome this, efforts are underway to make a standard workflow/process notation (section 3.7 on page 29 presents BPMN, which is such a standard). In our domain of interest, Integrated Operations, where different vendors, oil and gas companies and different experts are to collaborate, such a standard is of high relevance.

3.5 Business Process Management

When talking about workflow, Business Process Management (BPM) and business processes are often mentioned. A business process is: "a collection of related, structured activities that produce a service or product that meet the needs of a client" [Wikipedia 2008a]. One can argue that these processes are of great value and that a company would like them to be as efficient as possible. Business processes are implemented as *information processes* or *material processes*. Information processes relate to automated tasks and partially automated tasks (i.e., tasks performed by programs and tasks performed by humans interacting with computers, respectively) [Georgakopoulos et al. 1995]. Material processes refer to human tasks rooted in the physical world (i.e., moving, storing, transforming, and assembling physical objects). In this thesis we will focus on business processes related to information processes.

BPM is the method for managing business processes. Workflows fit into the BPM by covering the first two parts of the BPM lifecycle: Design and Modeling (see figure 3.1 on the next page).

3.6 SoluDyne

For documenting best practices, workflows, and so on, the pre-Hydro part of StatoilHydro uses a solution from the Norwegian company SoluDyne. The product, also named SoluDyne, has several modules where the workflow module is one of them. The workflow notation is a proprietary one, created by the company. We were not able to get any documentation from SoluDyne despite our emails. However, some documentation was found in [Engum and Wathne 2004] and the diagram elements where more or less self-explanatory. The notation consists of "role lanes" keeping track of who the responsible

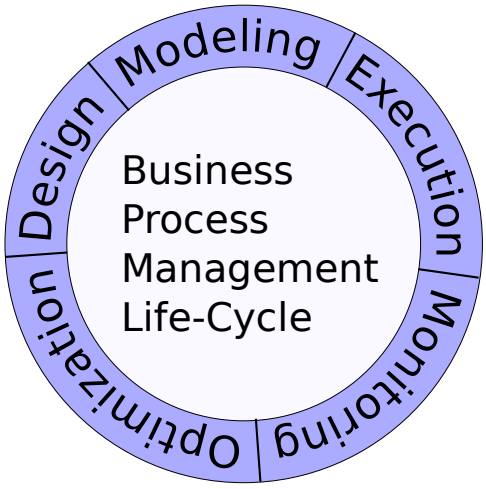


Figure 3.1: *The BPM Life-Cycle*

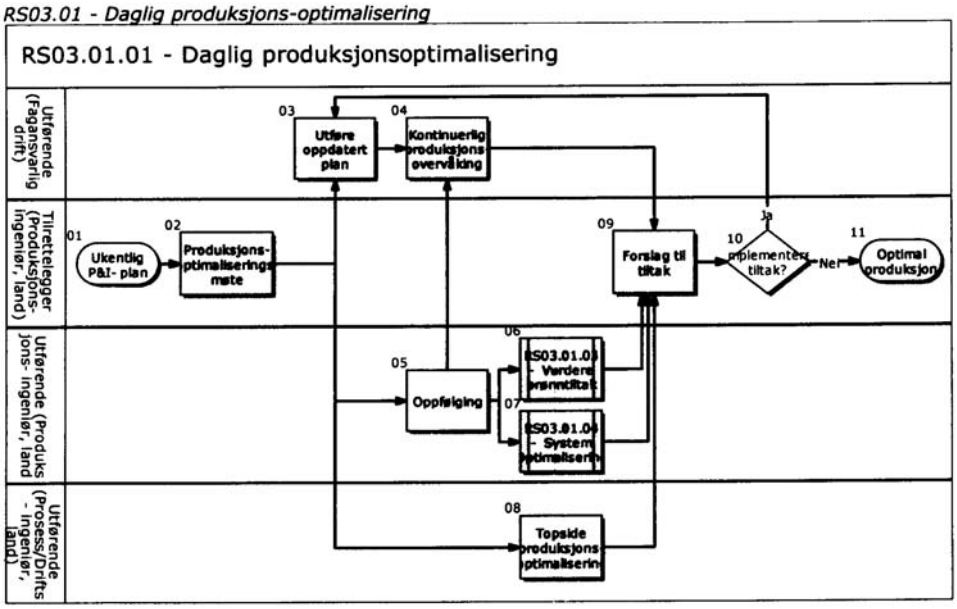


Figure 3.2: *Example of a workflow diagram in the SoluDyne system used by the operators at StatoilHydro, Sandsli*

person or group of the activity is. Simple atomic and compound activities are represented by squares.

A simple example of a SoluDyne workflow diagram is shown in figure 3.2 on the preceding page. The diagram is in Norwegian, but when talking about the tasks and roles we use their English translations. The numbers in the diagram and text help to see which task is discussed. The diagram is also translated into BPMN in figure 3.4 on page 32 using the English translation.

The four participants are:

1. Executing (Technical manager, operation)
2. Organizer (production engineer, onshore)
3. Executing (Production engineer, onshore)
4. Executing (Process/Operation engineer, onshore)

The arrow from an activity can split into several others such as “02 - Production Optimization Meeting” splitting into: “03 - Execution of updated plan”, “05 - Production follow up”, and “08 - Topside production optimization”. We assume there is an implicit fork present. Two arrows can join into one like it does before “04 Need for process”. Neither here is the exact semantics defined, but we assume it means a regular XOR-join operation (i.e. wait until both have finished) where two separate flows merge into one.

By clicking on a process one can move into a lower level of abstraction and see the workflow diagram representing the selected process. At the leaf nodes of the workflows one may find further documents and descriptions explaining the step. These details are typically written in a Microsoft Word document or similar. Improvements in a work process could be discovered as it is being used or as a result of an analysis. The workflow description should be continuously updated to reflect such improvements. In this way the diagrams and the documents describe the current best practice of an operation. The level of abstraction of the workflows is relatively high. This is due to the fact that they apply to all pre-Hydro oil and gas platforms belonging to StatoilHydro. Details belonging to individual fields are up to the specific platform at hand to determine.

3.6.1 Problems

There are several problems and areas of possible improvements regarding the workflow solution with SoluDyne. The notation is not very rich in its possibilities, and it has been mentioned that some of the details from the work processes were lost when expressing it in that notation.

The workflow diagrams are of such high level of abstraction that it is in reality up to the main operator and her level of experience to control the tasks in operator room and not up to the strict rules or methodology. The workflow documents are mainly a guideline more than a strict definition of how things should be done. After attending several well meetings, we observed how some of the much used workflow diagrams are used in practice. The high-level workflow diagram was shown on a shared screen by offshore and onshore. This acted as an agenda for the meeting. However, the diagrams were mostly used as a checklist making sure that the operators had covered all the main important parts of a process.

3.6.2 Alternatives to SoluDyne

The probably best known workflow language is the Business Process Execution Language (BPEL). It is an XML-based language for describing a business process in which most of the tasks represent interaction between the process and external Web Services [OMG 2005]. As BPEL is used for orchestrating services more than documentation of human workflow it is not seen as a real alternative in our context. There is a relation between the graphical notation BPMN and BPEL. BPMN can be translated into BPEL, which is specified in the specification. However, there is not a one-to-one translation and there exist elements in BPMN that cannot be mapped into BPEL, and vice versa [Ouyang et al. 2006, Recker and Mendling: 2006].

Another workflow language is the Yet Another Workflow Language (YAWL[2]), which came out as a result of a project with researchers at Eindhoven University of Technology and Queensland University of Technology as participants. It has a clear and formal semantics [YAWL Foundation 2007]. YAWL has not got the number of users and is not a standard, and for that reason not considered in this thesis.

BPMN is a much more known process modeling notation than SoluDyne. It is a public standard, as opposed to SoluDyne which is a proprietary solution used only by a few companies. The community and resources available to support BPMN is considerable. There are several other alternatives that could be used, but as BPMN is arising as the most important standard for modeling business processes we are using this in our master thesis.

[2] `http://www.yawlfoundation.org/`

3.7 BPMN

BPMN [Stephen A. White 2004] is a standardized graphical modeling notation used for drawing workflow diagrams of business processes. It was developed by the Business Process Management Initiative (BPMI[3]), but is now being maintained by the Object Management Group (OMG[4]) after their merge in 2005 [Wikipedia 2007]. As of July 2008, the current adopted version is 1.0 and a new version, 2.0, is currently under construction [OMG 2008].

The designers of BPMN had two goals for their notation. First they wanted it to be simple to create diagrams. Secondly they wanted it to be able to handle the complexity inherent in business processes. The solution was to create a small set of notation symbols grouped into different sets. This makes it easy for the modeler to create a basic diagram which captures most of the intended meaning of the business process itself. The small set of symbols can be extended to support the requirements for complexity without dramatically changing the look and feel of the diagram. An example is the gateway object. A regular gateway is depicted as a diamond with nothing inside. One can further expand the semantics by adding a star inside making it a "complex decision".

We will briefly explain the basic elements making up most BPMN diagrams[5]. An example will be depicted as well. Figure 3.3 on the following page depicts the basic elements. Examples of diagrams in BPMN are found in chapter 4 (figure 4.3 on page 43 and figure 4.4 on page 44).

The basic elements are divided into:

- Flow Objects
- Connecting Objects
- Swimlanes
- Artifacts

3.7.1 Flow objects

These are objects that appear in the flow of the workflow diagram. There are three different kinds of flow objects:

[3] `http://www.bpmi.org/`

[4] Object Management Group is also the group behind the ubiquitous UML-standard. `http://www.omg.org/`.

[5] There are several other entities making up BPMN. For a full overview, see: `http://www.bpmn.org/`.

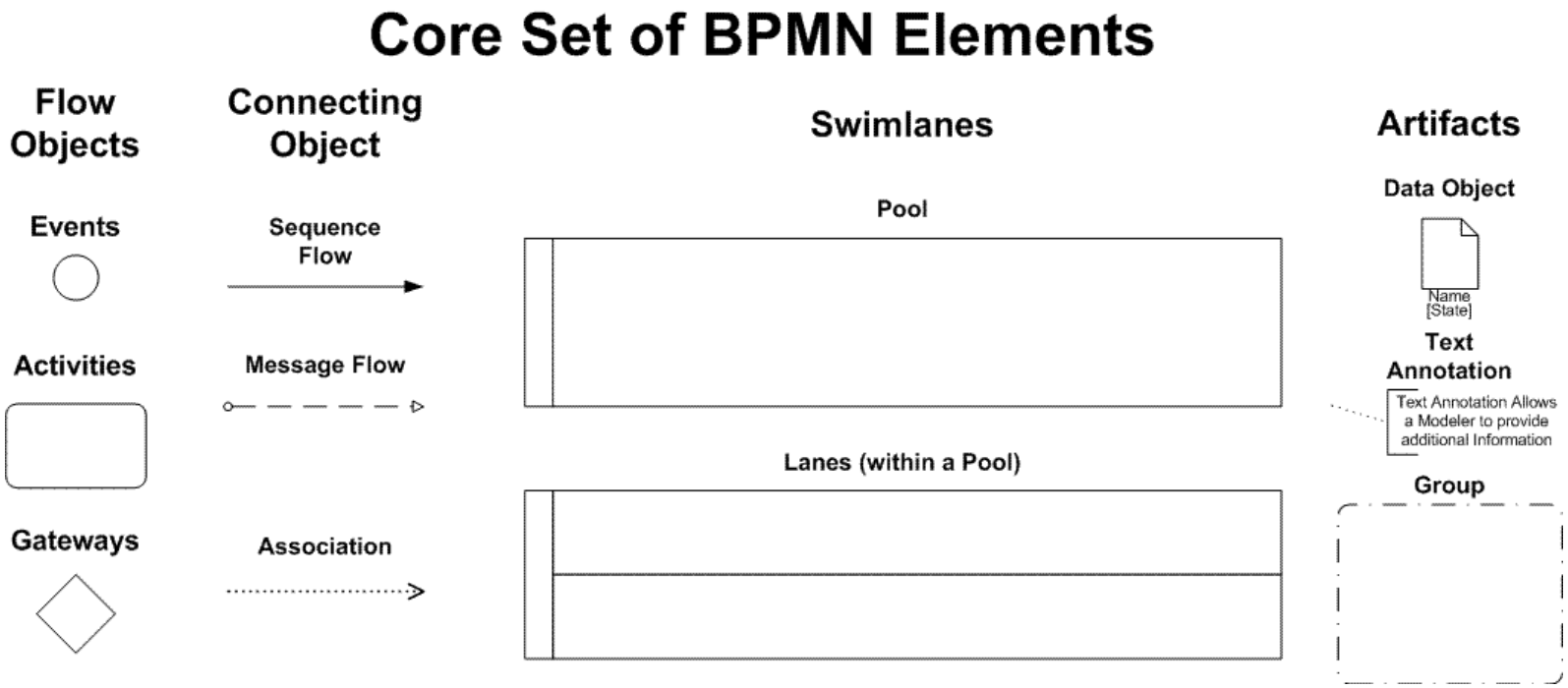


Figure 3.3: *The basic elements making up BPMN*

Events

An event is something that may happen during the course of a business process. Events affect the flow of the business, and usually have a trigger and a result. They can start, interrupt or end the flow. There are several different event-types, like message event, timer event, etc.

Activities

This is something that is to be done (e.g., "find reservation"). We differentiate between compound and atomic activities where the atomic ones are called tasks. Compound activities can be broken down into a finer level of detail. If the activities have more details, and can be expanded, they are called sub-processes.

Gateways

Gateways are depicted as diamond shaped figures. In BPMN they are divided into two types, forks and joins, which again can be divided into several different sub types. Forks split the flow into two or more directions, while joins take two or more flows and merge them together into one.

3.7.2 Connecting objects

The connecting objects are represented by arrows in a diagram. They are used to represent paths of messages, sequences, and associations.

3.7.3 Swimlanes

There are two types of swimlanes. The "main" one is called a *Pool* and represents participants in business-to-business processes. The other is called a *Lane* which most often represents different roles. All elements must belong to a pool or a lane.

3.7.4 Artifacts

Artifacts are other types of graphical elements. They are divided into three: Data Objects, Text Annotation, and Group. These are present in the notation to be able to show information beyond the basic flow-chart structure of the process.

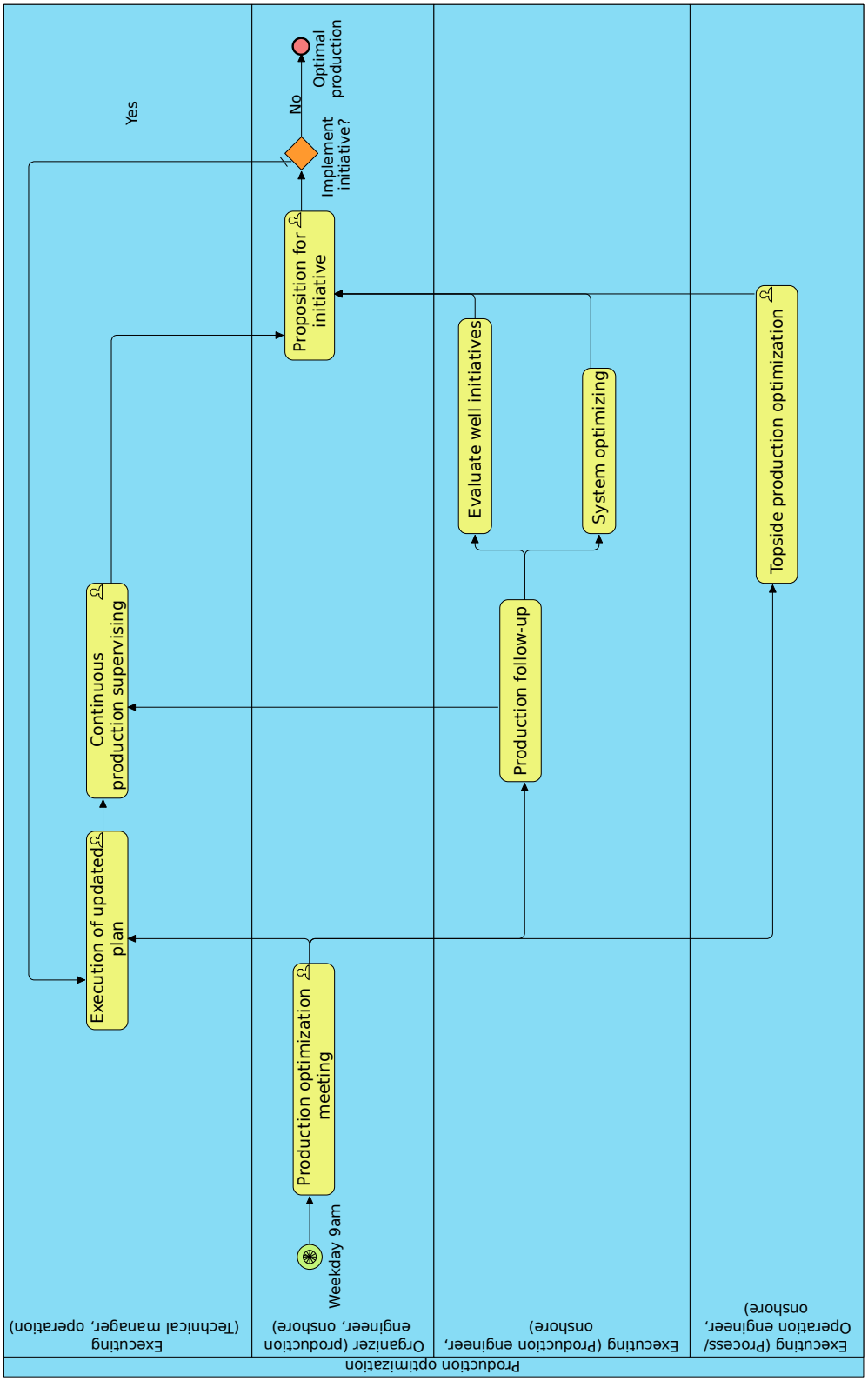


Figure 3.4: *The same diagram as found in figure 3.2, only represented in BPMN instead of SoluDyne*

3.7.5 Benefits over SoluDyne

BPMN has a broader notation than SoluDyne, with several types of events depicting what triggers the event. SoluDyne has only a blank start event type. BPMN makes it possible to show flow of data using artifacts with associations, something which is not possible in SoluDyne. BPMN, in contrast of SoluDyne, makes it possible to show different types of forks and joins, gateways, default paths, exceptions, if a task is user-based or application-based, etc. The extra power of BPMN enables the architects who draw the diagram to express their diagrams closer to the reality, compared to the diagrams expressed in for instance SoluDyne.

Chapter 4

Introducing SemTask

The previous chapters gave an introduction to Integrated Operations and workflow. This chapter will bring these two aspects together, and explain how they form the SemTask application. SemTask is an abbreviation for Semantic Task support. It is an application meant to help the operators make better decisions, faster. SemTask will be explored with its functionality at a high level. It gives an indication of what role SemTask may play in IO as well as how it relates to other applications available at StatoilHydro. Reasons why SemTask is needed and what it solves are examined.

Workflow is a central and fundamental part of SemTask as they form the foundation of work process execution. The ambiguous and weak semantics in the SoluDyne diagrams is not adequate, so BPMN is used as a replacement. The new diagrams have additional semantics because of the higher expressiveness of BPMN. Additional information found in the APOS documentation is also added into the diagrams. The conversion and the diagrams themselves will be explained.

A scenario where SemTask is used is introduced, based on Hydro's work process "Daily Production Optimization" (DPO). DPO is a central part of exemplifying how SemTask works, and will be explained in detail. Let us start by looking at the common work process DPO, and outline some of the problems with today's work method.

4.1 Daily Production Optimization (DPO)

DPO is a work processes the operators at StatoilHydro at Sandsli execute every weekday, designed to get maximum production out of an oil field. The process starts each day with a meeting where production increasing initiatives are discussed.

Different activities of this work process will be shown in detail. This is important for the understanding of the scenario, and for several other parts of this thesis.

The process involves several people and resources. The SoluDyne version is depicted in figure 3.2 on page 26. The workflow consists of four role-lanes, or participants. From top to bottom:

- Executing (Technical manager operation)
- Organizer (Production engineer, onshore)
- Executing (Production engineer, onshore)
- Executing (Process/operation engineer, onshore)

The organizer owns the daily morning meeting (also called production optimization meeting), and is responsible for accepting or rejecting optimization suggestions.

Following is the descriptions of each of the tasks/sub processes in the work process DPO, as described in APOS. The interactions and connections between the given tasks and sub processes will then become more evident.

4.1.1 Description

01 — Weekly P&I Plan The Production and Injection plan is taken as a starting point for the meeting.

02 — Production optimization meeting The first activity in the workflow is the production optimization meeting. It has both onshore and offshore participants. The result of the meeting is a list of activities, consisting of production increasing initiatives that the technical manager for operations uses as input in the activity "Execution of updated plan".

All participants of the meeting should create an overview of different types of data. The Production engineer should in advance create an overview of:

- Well capacities
- Process capacities
- Well status
- Variances from the production and injection plan as well as the results from the previous day
- Need for change of steering parameters

The technical team should be prepared for questions about:

- Categorization of losses in relation to the budget
- Capacity constraints in the plant
- Planned activities

A representative from the operation group should be prepared to give input about:

- Capacity constraints
- Process technical issues that may affect the production

03 — Execute updated plan The plan for the day should be updated with the action list from the daily morning meeting. It must also be updated when new actions are found. If it is not possible to follow the plan, the technical manager should discuss this with the organizer (production engineer).

04 — Continuous production supervision The production is supervised by the Executing (Technical manager operation). When new initiatives are found these are sent to the Organizer (Production engineer, onshore). Trends and analysis for the wells are some of the parameters that are supervised.

05 — Production follow-up Production follow-up is a sub-process consisting of several activities. The production engineer is dedicated to production follow-up, and she should be located in the operation room at least 4 hours daily. The action list from the production optimization meeting should be checked to see the effects of the planned changes. The production engineer should continuously evaluate the wells and processes to look for additional optimization potentials. If an optimization potential is found, it should be tested with computer simulations before it is implemented.

08 — Topside production optimization One onshore production engineer should be dedicated to finding production increasing potential regarding the topside processing plant.

09 — Proposition for initiative The organizer (the onshore production engineer) should consider any action that may increase the production. If there is a need for collaboration between onshore and offshore, the organizer

arranges this meeting to discuss this. If the proposition is accepted, it is sent to the Executive (Technical manager) who adds it to the plan for the day.

4.1.2 DPO — Optimization potentials

SoluDyne is a documentation application, and does its intended meaning in a good way. The operators use it to look up documentation about the work process when needed. However, some of the information about processes was lost while creating the documentation as a result of SoluDyne's lack of expressiveness.

There is no automation of work processes. Operators can easily forget tasks in a complex workflow. Ultimately, the workflow system should be executed. In that way no tasks are missed or done in an incorrect order.

It is claimed that engineers use up to 40% of their working time to look for the right data to use. The number may actually grow in the future as the availability of real-time data increases. The workflow application should assist the operators in getting the right data at the right time.

The current work processes have no associated flow of data. Data sources that a person uses while performing a task may be useful for the users of the subsequent tasks. Decisions are most likely based upon them.

Some of the problems with today's workflow system have been outlined. As the different parts of SemTask are explained more benefits and details are presented. But let us take it one step further looking at a scenario where SemTask is used to solve some of these optimization potentials. The scenario is based on the DPO process.

4.2 Scenario

4.2.1 Participants

- Frida — Executing (Technical manager operation, offshore)
- Per — Organizer (Production engineer, onshore)
- Pernille — Executing (Production engineer, onshore)
- Didrik — Executing (Process/operation engineer, onshore)

1. SemTask notifies Per that it is time for the process "Daily Production Optimization". Per asks SemTask to notify the other participants

(Frida, Pernille, and Didrik) that they should connect to start this workflow. They all notify SemTask at their presence and head for the meeting room. As Per is the meeting owner he gets an overview of who have joined the meeting, and whether anyone is currently missing. SemTask knows the connections between tasks and data and can therefore infer that the weekly Production and Injection plan (P&I) should be available to study. SemTask brings the P&I plan up on the screen.

2. As this is a meeting between participants onshore and offshore, SemTask suggests opening a video conference to connect to Frida who is offshore. After the video conferencing system has connected the teams, SemTask automatically shows some important key information like well and process capacities, well statuses, and differences between yesterday's actual and planned production. The key information is gathered in a preconfigured workspace in PDP. Per suspects a negative trend in oil production in well A6. He clicks on this well to get more information, and uses PDP to look at the historical data in different intervals. His suspicion turns out to be true, and upon investigating the pressure level for the well in question he sees a quite dramatic decrease. He asks Frida about her opinion as she can see the same as Per on her screen. SemTask will automatically present information about well A6 like type, parts and manufactures, and start-up date, on the screen as Per looks at the production data for that specific well. Per and the others decide to contact an expert for her opinion as the pressure decrease is abnormal, since this may suggest that something is dramatically wrong, or that it is just a sensor malfunction. Per clicks on the pressure sensor for well A6 to get information about manufacturer and contact person. The expert who is listed as contact person is currently not available and SemTask automatically suggests another person registered as a contact for the well pressure sensor. It turns out that there is a known problem with this type of sensor, and an action to survey the well and continue the normal production is made. The surveillance initiative action is logged in PDP and Per instructs SemTask to continue the process.

3. As the meeting ends, everyone goes to their own working place. Now it is time for Frida, Pernille, and Didrik to work in parallel. Frida gets the activity "Execute updated plan" activated as well as "Continuous production supervising". SemTask shows her the action list from the production optimization meeting along with a planning program that lets her put different activities up at different times. SemTask also shows production supervision information available in PDP. She does not see any immediate optimization potentials at the moment, but is prepared to give them, with the help of SemTask, to Per at any time.

Frida sees the newly made "surveillance" action, and follows it up.

4. Didrik gets the activity "Topside production optimization" activated on his screen. SemTask brings relevant information to this activity, which is an overview of the oil and gas capacities, pollution rates in the waste water, alarm, and event logs. He notices that the pollution rate is increasing more than it normally should. As he is unsure of the cause so he writes in SemTask that he is concerned about the pollution rates in the waste water from the processing plant, and that he needs to have a meeting with some of the engineers offshore. SemTask sends his reports to Per.

5. Per receives Didrik's report, and gets the screen for "Proposition for initiative" shown on his display. As the report from Didrik covers a concern about the processing plant, Per sees the same data of the processing plant as Didrik did when he asked for a meeting. SemTask automatically presents a list of offshore engineers who might have experience with the problem and the equipment mentioned in the report. Per is also able to see the schedule of the different engineers and the meeting room, which enables him to find a time for the meeting between Didrik and the offshore engineer.

6. At this point Mona, which is currently working with a domain expert regarding a malfunction in one of the subsea well, needs help from Per. She knows that Per is busy working on the "Daily Production Optimization" process. For that reason, she decides to bring up SemTask to start the monitoring function of that particular process. SemTask shows the tasks that Per is currently working on. Mona knows how important these tasks are, and decides not to interrupt Per. Since the SemTask application is already running, she checks out some statistics of the tasks being done in the DPO-process. She notices that only a small percentage of the initiatives from the "Topside production optimization" task is actually implemented, and decides to follow it up later.

7. While Frida and Didrik are working on their tasks, SemTask shows the task "Follow-up" on Pernille's screen. SemTask presents the action-list from the daily morning meeting, key information from each of the wells, etc. Each of the activities in the action list has a responsible person associated with it. If a planned activity seems to have a negative effect on the production, it will be easy for Pernille to contact that person through SemTask. Pernille is aware of the fact that this oil field has a known problem with network optimization, so she checks whether a network model exists. If no model exists, she asks SemTask to start the network optimization task. This leads SemTask to start the associated applications, which are GAP and MaxPro. Pernille

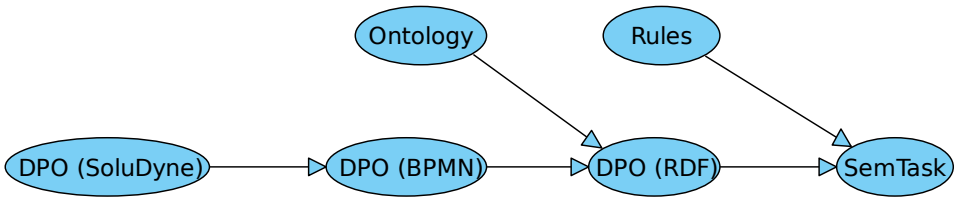


Figure 4.1: *Overview of the approach in making SemTask*

creates the model while SemTask has already brought up relevant process capacities to aid Pernille. SemTask is aware of the "System optimization process", so it foresees the need for data to test the model. It checks whether production or test data is available, and gives the choice to Pernille of which to use for testing the network model. Pernille chooses production data and SemTask asks Pernille if the model is OK. Pernille answers SemTask by checking if the model passes some requirements. SemTask brings her to the activity "Execution of optimization" and brings up the "checklist for system optimization". She notifies SemTask as she is satisfied with the model, and the subprocess is finished.

8. If Frida, Pernille, or Didrik has an action proposal, it will automatically be sent to Per through SemTask. Per will get the action proposal along with who made the proposal, what data was used, attached documents, etc. If he accepts the proposal, it is sent to Frida who puts it in the plan of the day.

4.3 SemTask implementation overview

There are several approaches to building an application that fulfills the scenario. Technological and architectural decisions become evident as the different parts of SemTask are discussed (e.g., why use ontologies instead of XML-Schema and why the ontologies are structured in the following way).

The workflow diagrams are now in SoluDyne, and need to be converted into BPMN. Once they are in BPMN they are to be converted into RDF instances using a set of ontologies. Rules are used to implement the execution engine of SemTask so that these workflow diagrams can be executed. Figure 4.1 illustrates how the different parts of SemTask are related to each other.

A more technical diagram is presented in figure 4.2 on page 42. How the different ontologies relate to each other will become evident in chapter 6 on page 63. Chapter 7 deals with the integration between the reasoner, rules and ontologies using Java and a Semantic Web framework called Jena.

Graphical User Interface is not treated in this thesis, but the general idea, and ideas for further work are presented in chapter 8 on page 95.

4.3.1 Our focus on SemTask

SemTask is a complicated workflow and decision support application. It consists of several parts, including a graphical user interface, a client/server architecture, logic to start up the right applications with the right parameters, etc. We will not focus on any of those components in SemTask. What we will focus on is the functionality of showing the right data at the right time. This will require a further formalizing of the workflow diagrams, along with a formalizing of necessary input data and data sources. In section 8.2 on page 97 several possible extensions of the SemTask application are examined.

4.4 SoluDyne to BPMN

As described in section 3.6 on page 25, the Hydro part of StatoilHydro uses SoluDyne for describing and documenting their work processes. The SoluDyne system is both used for drawing, storing, and displaying the diagrams, and for connecting documentation (like short texts, documents, and spreadsheets) to the diagrams. As the SoluDyne workflow diagram notation lacks documentation (like what is the exact meaning of the different diagram shapes), and only provides the simplest opportunities for drawing workflow diagrams, we have chosen to translate the original SoluDyne diagram to BPMN.

The BPMN version of DPO sets out to capture more of the underlying semantics and details, found in the APOS documentation, that were originally lost as a result of modeling the workflow in SoluDyne. Figure 3.2 on page 26 shows the original diagram in the SoluDyne system and our translation with some additions are found on figure 4.3 on page 43.

In order to create the enhanced BPMN-version, we studied the DPO-workflow and the supporting documentation from APOS [Hydro 2008]. In addition, we have emailed and had meetings with Espen Halvorsen at StatoilHydro and Kjetil Torvanger from Cronus Engineering. This was done to be able to understand not only the diagram, but also the process that it intends to capture. The conversion from SoluDyne diagrams to BPMN diagrams is not straight forward so we will briefly explain it. The conversion described here is for the specific process DPO, and the conversion shows signs of the fact that additional information is added to the resulting diagram.

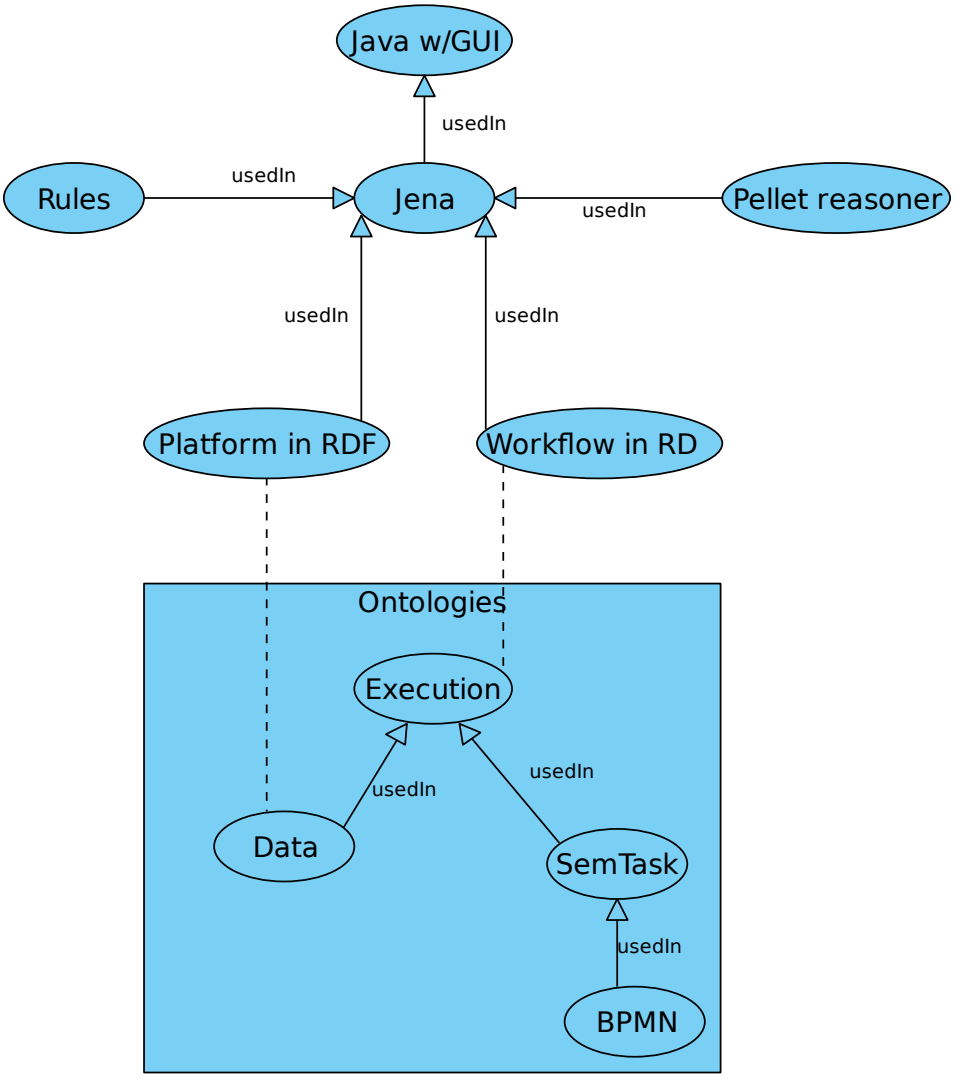


Figure 4.2: *The architecture of the SemTask application*

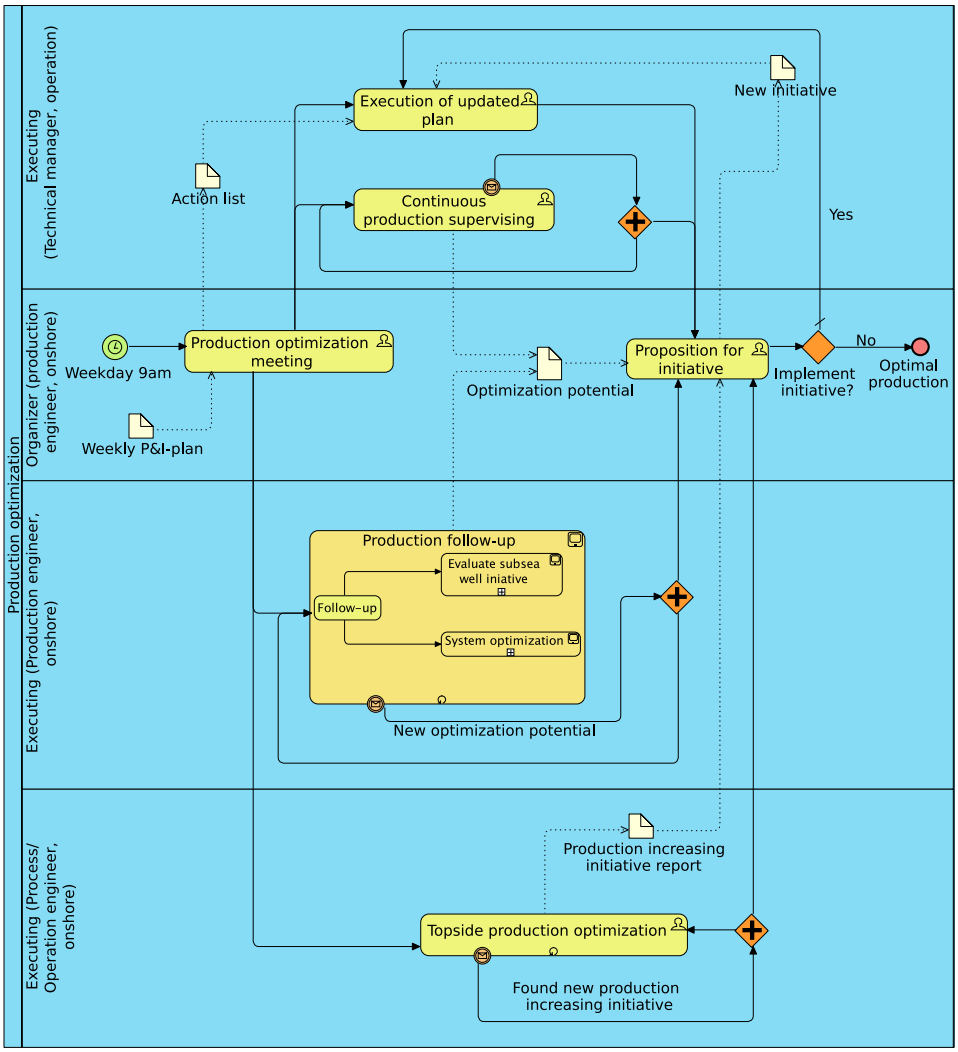
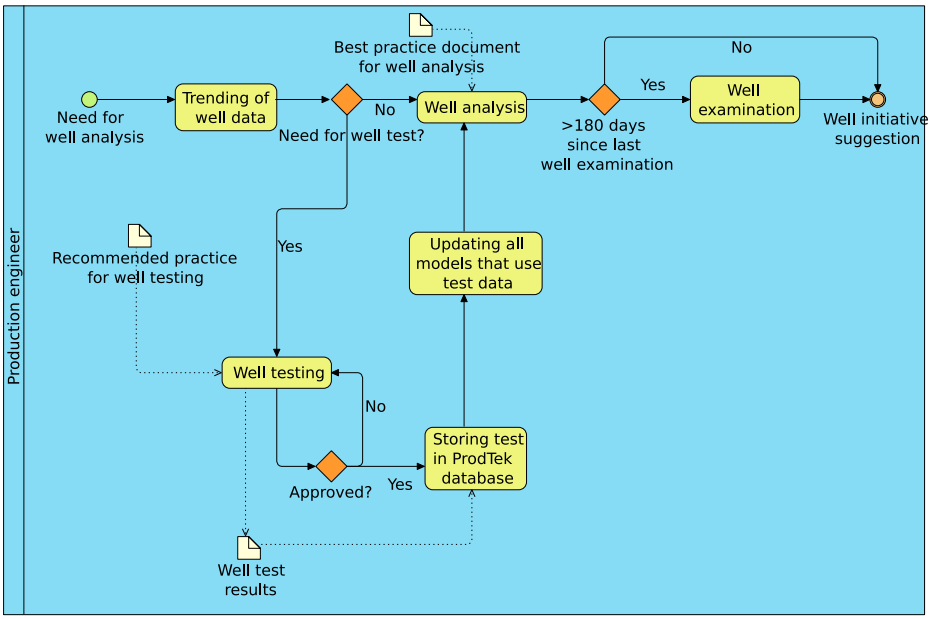
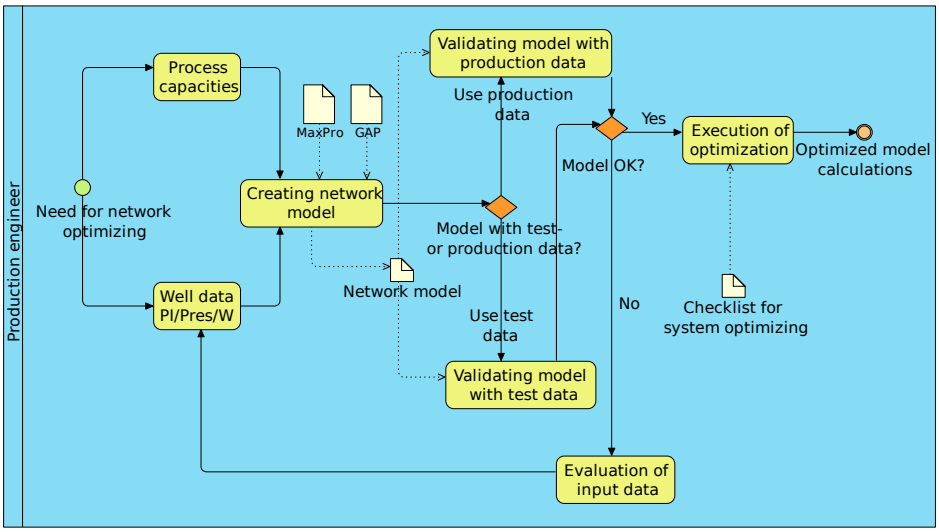


Figure 4.3: *Daily Production Optimization in BPMN*



(a) Evaluate Subsea Well Initiative



(b) System Optimization

Figure 4.4: *The sub-processes of Production follow-up*

In the SoluDyne version we find the activity “05-Follow up” that triggers both “System Optimization” and “Evaluate Subsea Well Initiative”. These are performed in a sequential manner, as depicted, before handling the flow over to “Proposition for initiative”. Following the documentation for the process “Follow up” and our research, which can be read in section 4.1.1, we chose to model it as a sub-process. The reason is that the two processes: “System optimization” and “Evaluate subsea well initiative” are parts of the ongoing “Follow up” process. It is a looping process and each time the “Executing (Production engineer, onshore)” finds an optimization potential, an event kicks off. The event results in sending a data document to the “Proposition for initiative” process. Associations to the data objects show what activity is producing the data, and where it is being used as input. The control is given both to the “Proposition for initiative” process and back to the “Follow up” process.

In much the same way as the “Follow up”-activity the “Topside production optimization” has undergone some changes to the new diagram. It is looped, and has an intermediate event connected to it in the similar way as the “Follow-Up” process. We also assume that once a production increasing initiative is found, the process does not stop, but sends a document to the responsible person for the entire “Daily production optimization”, and continue to look for further initiatives. In this case that person is “Organizer (Production engineer, onshore)”.

In the original SoluDyne diagram “03-Execute updated plan” and “04-Continuous production supervising” these tasks are performed in sequence. However, we interpret its meaning to be that they are performed in parallel. One should always do “Continuous Production Supervising” and it is not the case that one should first start supervising after the executed plan task is finished. Also in much the same way as the “Follow up” and the “Topside production optimization” activities, the “Continuous Production Supervising” activity includes an intermediate event. This event ultimately sends a data document “Optimization Potential” to the “Organizer (Production engineer, onshore)”, and continues with the production supervising. For the “Execute updated plan” we have extended the semantics by showing the data used as input both from “Production optimization meeting” and “Proposition for Initiative”.

SoluDyne lacks the feature of showing data flow in workflow diagrams. The ability to do this is a feature of BPMN, so we have tried to use it where it seems feasible. Both input of data, output of data, and data flow are shown. For instance can we see that the Production optimization meeting uses the Weekly Production and Injection-plan as input, and that it produces an Action list that the activity Execution of updated plan uses.

Chapter 5

Ontology

> We are getting into semantics again. If we use words, there is a very grave danger they will be misinterpreted.
>
> H. R. Haldeman

The previous chapter gave an introduction to SemTask by going through a scenario. This chapter will present how data sources and workflow diagrams may be represented to be able to fulfill the requirements of the scenario. An introduction to ontologies will be presented along with the main alternatives. The conclusion is that using ontologies to represent data and workflow lead to several advantages. These advantages will be outlined and explained. Why ontologies seems to be the best option as a representation foundation for the SemTask application will also be discussed.

5.1 An introduction

5.1.1 Definition

The term ontology can have different meanings. It originates from a subfield of philosophy concerning the study of the nature of existence, the branch of metaphysics concerned with identifying the kinds of things that actually exist and how to describe them [Antoniou and van Harmelen 2008]. The meaning in computer science is a bit different. The most used definition is probably the following: "An ontology is an explicit and formal specification of a conceptualization" [Gruber 1995]. A more informal, and perhaps more explanatory, definition would be: an ontology is a hierarchy of concepts,

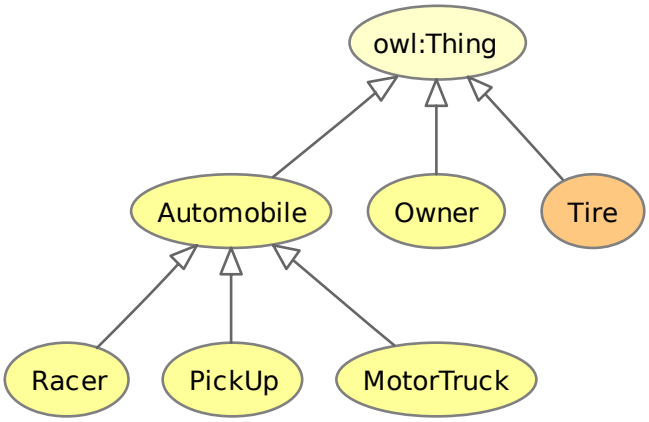


Figure 5.1: *An example of an ontology. The figure is made using Protégé and visualized using the OWLViz plug-in. See* `http://www.co-ode.org/downloads/owlviz/`

which can be seen as classes, and relations between these classes.

5.1.2 Example

A simple ontology about Automobiles is presented in figure 5.1. Here we have *Thing* as the super class with three subclasses: *Automobile*, *Owner*, and *Tire*. The class *Automobile* is further divided into *Racer*, *PickUp*, and *MotorTruck*. Every class is disjoint from its siblings (i.e., an instance of *Automobile* cannot also be an instance of *Tire*). Once this simple concept/class hierarchy is in place one can define the properties. One often differentiates between object and datatype properties. Properties are relations going from an object to another object for object properties and to datatype (such as natural numbers or strings) for datatype properties. Properties have a domain and a range. The domain consists of the elements that the relation "goes from',' and the range consists of the elements that the relation "goes to". We could then define an object property called *owns* with the domain: *Owner* and range: *Automobile*. An example of a datatype property could be *numberOfDoors* having *Automobile* as domain, and the natural numbers as range.

There are several other constructs available in an ontology language than the ones mentioned in the example. The available constructs define the expressiveness of the ontology language at hand[1].

[1]For a good introduction to making ontologies see [Horridge et al. 2004].

5.1.3 Classes

An ontology consists of classes and properties. Depending on the ontology language you can state different things about the classes. Typically axioms include:

- Disjointness of classes: If class A and B are disjoint, no instance can be a member of both A and B.
- Inheritance/subsumption: If class A inherits class B (B subsumes A) then all instances of A are automatically also instances of B.
- Complex definitions: A class can be defined to contain exactly all instances that have a property of a specific type or list all its members, for instance.
- Complex inheritance: A class can be defined to include all instances that have a property of a specific type, for instance.

In an ontology, there are two kinds of classes: defined classes and normal classes. A defined class has one or more *sufficient* conditions, meaning that if an instance has met all these conditions, it is automatically a member of that class. An example is value partitions where a class is defined exactly to be the union of its disjoint subclasses. Another example is one that specifies exactly which members (instances) it has. These kinds of defined classes are often called enumerated classes or enumerated types.

5.1.4 Properties

Properties connect instances. It is possible to specify several facts about the properties in an ontology (depending on the expressiveness of the ontology language), including:

- Domain and range: If D is the domain and R is the range of a property P then for all x and y: if P(x,y) then x is a member of D and y is a member of R
- The inverse of a property: If P and S are inverse properties then for all x and y: P(x,y) if and only if S(y,x).
- Reflexivity: If P is a reflexive property and C is in the domain of P then for all x in C we have P(x,x).
- Property inheritance: If P and S are properties and S inherits P then: for all x and y, if S(x,y) then P(x,y).
- Symmetry: If P is symmetric then for all x and y, if P(x,y) then P(y,x).

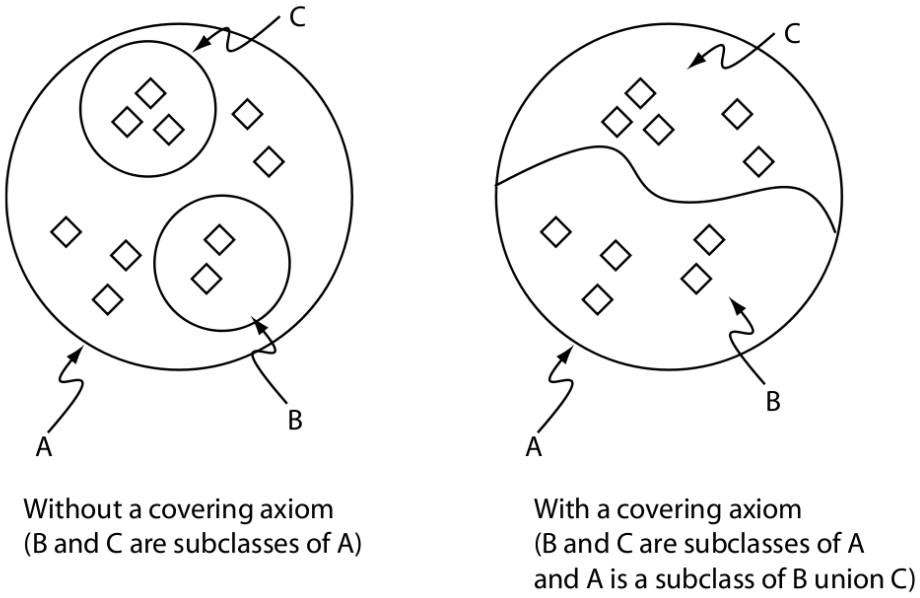


Figure 5.2: *The use of a covering axiom to get a value partition class. The figure is taken from [Horridge et al. 2004]*

- Transitivity: If P is transitive then for all x, y, and z, if P(x,y) and P(y,z) then P(x,z).
- Functionality: If P is functional then for all x, y, and z: if P(x,y) and P(x,z) then y=z.
- Inverse-functionality: If P is inverse-functional then for all x, y, and z: if P(x,y) and P(z,y) then x=z.

5.1.5 Language requirements

There are several requirements for ontology languages. The main ones are: well-defined syntax, efficient reasoning support, a formal semantics, sufficient expressive power, and convenience of expression [Antoniou and van Harmelen 2008]. Well-defined syntax is clear from the fact that otherwise a computer would not be able to parse the ontology description. A formal semantics means that the meaning of a word should not be a result of human interpretation or a computer program implementation. Algebra in Mathematics is an example of a language with a well-defined semantics.

5.1.6 TBox and ABox

When discussing knowledge bases[2] a distinction is often made between the terminology knowledge and the asserted knowledge. These parts are called the TBox and ABox respectively. There are no logical distinctions between the TBox and the ABox, but as we shall see, reasoning algorithms often work on either one of them. Having a term for these different parts of the knowledge base is therefore convenient. An analogy is classes and objects in object oriented programming. The classes can be compared to the TBox, and the objects which belong to a class can be compared to instances in the ABox.

TBox

The TBox is a set of *schema axioms* that define the classes and the relations/roles with their respective properties. The concept and role constructors used and allowed are constrained by the ontology language in question. We have, in the most general case, inclusion ($A \sqsubseteq B$) and equivalence axioms ($A \equiv B$) where A and B can be class or relation/property. More expressive languages have additional axioms, such as those mentioned in section 5.1.3 and 5.1.4

Following the example about Automobiles a schema axiom could be: $Racer \sqsubseteq Automobile$ which means that *Automobile* subsumes *Racer*. Another example of a schema axiom could be: $Mother \equiv Woman \sqcap \exists hasChild.Person$ which says that the class *Mother* is defined as all objects belonging to the *Woman* class and which has one or more properties of the kind *hasChild* with the class *Person* as range.

There is much more to a TBox than what is stated here, and the curious reader is referred to [Baader et al. 2007].

ABox

The ABox on the other hand is a set of *data axioms* that define a specific state of affair of an application domain in terms of concepts and roles [Baader et al. 2007]. In the ABox we define individuals by giving them names; let us, call three individuals: a, b, and c. We then make two kinds of assertions about them called concepts assertions and role assertions like C(a) and R(a, b), where C is a concept and R is a role.

[2]Knowledge base is defined as a set of sentences, expressed in a knowledge representation language (e.g., ontology language), each representing some assertion about the world [Russell and Norvig 2003].

In the example of Automobiles example data axioms could be that BMW-Z8 is an instance of the class *Racer* and Ford-Ranger is an instance of the class *PickUp*. Further, we can say drivesFaster(BMW-Z8, Ford-Ranger), which is an example of a role assertion.

In practical implementations, there may be other constructors for ABox axioms available, e.g. differentFrom, which states that the individuals are different from each other.

5.1.7 Expressiveness vs. decidability

There is a tradeoff between expressiveness and decidability when dealing with ontology languages. We want the ontology language to be as expressive as possible making us able to express what we want. However, the language in question should be decidable meaning that there are reasoning algorithms that are sound and complete and will stop in a finite time. There is ongoing research to extend ontology languages with new constructs making us able to express more without losing decidability.

Although one would like the ontology language to be decidable one can work with undecidable languages as well. The ontology could for example be used as a taxonomy where we do not need support for reasoning. An example in the domain of oil and gas production is the Active Knowledge Management in the petroleum industry — AKSIO project [Norheim and Fjellheim 2006]. Here an ontology defined in an undecidable ontology language is used for expanding a text based search by looking at synonyms and relative terms of the initial search term.

5.1.8 Reasoning/inference tasks

Concerning the TBox, several inference tasks are relevant. Satisfiability of a concept/class is the reasoning task to see whether there is a model[3] in which the concept is non-empty; i.e., find out if the concept can have members. Subsumption is to reason about whether one concept is always a generalization of another, that is if a membership of one class always results in the membership of another class. Equivalence of classes is the inference task of checking whether two classes always must have the same members.

Concerning the ABox, the reasoning task is instance checking, i.e., checking which classes an individual belongs to.

[3]Model is here referred to as the model theory in Description Logic. See [Baader et al. 2007] for a thorough introduction.

It is shown that the reasoning tasks: checking satisfiability of a class, if two classes are equivalent, and if two classes are disjoint, can all be done using only subsumption. If the negation of concepts is allowed in the language, the reasoning tasks can be reduced to unsatisfiability. As a result of this, a reasoner typically only implements either subsumption or unsatisfiability, depending on the knowledge representation language. For a more comprehensive introduction to reasoners, reasoning tasks, and the algorithms behind see [Donini et al. 1996].

In the development of large ontologies, like in the ISO 15926 standard mentioned in section 2.4.1 where there are over 100.000 classes, automatically reasoning support is necessary since reasoning "by-hand" would take too long time. It is surprisingly easy to create unsatisfiable concepts for instance. When running subsumption checking one will see all classes, intended or unintended, that a given class is a sub class of. This information can be used to improve the ontology.

Let us look at one example of reasoning that uses the example in section 5.1.2 on page 47. We define *Automobile* and *Owner* to be disjoint from *Tire*, as well as defining an object property called *hasTireBrand* with the domain of *Automobile* and range *Tire*. We state that *both Automobile* and *Owner* have exactly one instance of the *hasTireBrand* as a necessary condition for all the elements of the class. It is tempting to conclude that the reasoner would infer that the class *Owner* is inconsistent since the domain of the *hasTireBrand* is only *Automobile*. However, the reasoner would instead infer that the class *Owner* is subclass of *Automobile*. If the two classes were disjoint the reasoner would detect an inconsistency in the class *Owner* meaning that it cannot have any members.

5.2 Short Introduction to Semantic Web

Many of the technologies we use in this master thesis are part of the Semantic Web stack. It can be illuminating to shortly present the idea behind Semantic Web, and how the technologies relate to one another.

The web we know of today consists mostly of content suitable for human consumption [Antoniou and van Harmelen 2008]. By introducing the Semantic Web, or Web 3.0, researchers are trying to change this, i.e., making the content understandable for computers as well. There are several consequences to this such as better web searches because the computer knows the meaning, i.e., the semantics, not just the syntax of the web content. Different web content from different sources could also be more smartly merged before presented to the end user since the meaning of the content is machine accessible. These are examples of what has been called

"the Semantic Web vision" that states that the web should be a web of data that is machine accessible, thus making the examples given possible. The Semantic Web initiative came from Tim Berners-Lee (the "father of the web"). He explains the Semantic Web vision with agents, machine accessible data, and reasoning of data in the famous Scientific American article: The Semantic Web [Berners-Lee et al. 2001].

Although mainstream adoption of the Semantic Web is still some years away it has been a considerable progress in the development of, and use of standards, languages, technologies, and applications [Cardoso 2007]. Because of this, the Semantic Web has reached a level of maturity so that big companies can apply the technology. For example, OWL is used for data integration by companies like Audi, Boeing, and Hewlett Packard [Antoniou and van Harmelen 2008]. Using ontologies for data integration is further explained in section 5.4.2 on page 61. Companies like Vodafone, Google, and Oracle are all working for a smarter web by developing Semantic Web applications [Cardoso 2007].

The technologies that are a part of the Semantic Web are layered in a stack. The stack by W3C, naming all the key technologies and showing how they depend on each other, is found in figure 5.3 on page 56. This stack has been changed several times and will probably change again. However, the main interesting technologies in the development of SemTask are all W3C recommendations (what other organizations would have called a standard): XML, RDF, RDFS, OWL, and SPARQL; and are likely to be in the Semantic Web stack in the future as well. Looking at the Semantic Web stack one can roughly divide it into elements concerning syntax, data, ontology and logic. URI and XML define the syntax, RDF defines the data layer, OWL and RDFS are used in the ontology layer and rules are used in the logic layer. SPARQL is the W3C recommendation for query language over RDF. Concerning SemTask, the Ontology technology of the Semantic Web Stack is of main interest. However, let us quickly go through the technologies constituting the foundation of the Semantic Web stack: XML, RDF, RDFS, OWL, and SPARQL.

5.2.1 XML

eXtensible Markup Language (XML) [Maler et al. 2006] is a W3C recommendation for a syntactic format used for representing tree-structures of information. XML allows representing machine-accessible information and allows the user to define markup for their documents using tags.

XML is a widely used standard in all types of businesses. The petroleum industry has also chosen XML as the syntactic standard, with PRODML

and WITSML as examples.

5.2.2 RDF

RDF stands for Resource Description Format and is a W3C recommendation [Carroll and Klyne 2004]. It is a simple data model used to describe resources and the relations between them. The core concepts of RDF are: resources and literals. A resource can be though of as an object and is given an identity by a unique URI (e.g., URL, ISBN, or telephone number). Literals are atomic values (e.g., strings, integers, and booleans).

Statements are triples consisting of a subject, predicate, object. The subject can either be a resource or literal. Predicates are a special type of resource denoting the relation between different resources and is identified using URL. The object can either be a literal or another resource. As XML defines trees RDF takes this one step further and gives the possibility of representing directed graphs.

RDF/XML is a W3C recommendation [Beckett 2004] and defines a syntax for serializing RDF graphs. There are other languages for serializing RDF like Turtle [David Beckett 2007], and Notation3 (N3) [Tim Berners-Lee 2006]. The N3 notation is a compact and readable alternative to RDF/XML, and is extended to allow greater expressiveness.

Since the RDF documents in the appendix are written in N3 format a quick introduction is beneficial. Statements are separated using ".". There are several shortcuts in N3. Looking at listing 5.1 on the following page one sees that the empty prefix is defined as "#" meaning the current document. When writing multiple statements about the same subject there are two shortcuts: ";" introduces another property and "," introduces another object with the same predicate and subject. The rdf:type property is abbreviated in N3 to just "a".

5.2.3 RDFS

Resource Description Framework Schema (RDFS) is a W3C recommendation [Brickley and Guha 2004]. It is an extensible knowledge representation language and is used to structure RDF resources and provide a vocabulary/ontology for RDF. RDFS forms the basic ontology language in the Semantic Web stack. It is a primitive ontology language [Antoniou and van Harmelen 2008] with the ability to define classes, properties with their domain and range and hierarchy of these classes and properties.

Listing 5.1: *Example of N3 notation*

```
@prefix : <#> .
@prefix bpmn: <http://www.ifi.uio.no/fredaleks/semtask/
    bpmn#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix protege: <http://protege.stanford.edu/plugins/owl/
    protege#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix semtask: <http://www.ifi.uio.no/fredaleks/semtask/
    semtask#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix xsp: <http://www.owl-ontologies.com/2005/08/07/xsp
    .owl#> .

:actionList a semtask:FormDocument;
  bpmn:hasConnectingFrom
     :actionListToExecutionOfUpdatedPlan;
  bpmn:hasConnectingTo
     :productonOptimzationMeetingToActionList;
  bpmn:hasName "Action List"^^xsd:string .
```

5.2.4 OWL

OWL [van Harmelen and McGuinness 2004] is a richer ontology language than RDFS. It is built upon RDFS and has in addition the ability to describe relations between classes (e.g. disjointness), characteristics on properties (e.g. symmetry), and enumerated classes, to name a few. OWL will be explained in further detail in section 5.3 on page 57.

5.2.5 SPARQL

SPARQL [Prud'hommeaux and Seaborne 2008] is a recursive acronym that stands for SPARQL Protocol and RDF Query Language. It is a query language used to query RDF triples. It has been a W3C working draft since October 2005 and became a W3C recommendation in 2008. It is supported by most of the Semantic Web tools, such as Pellet, Protégé, and Jena. The syntax is based upon SQL.

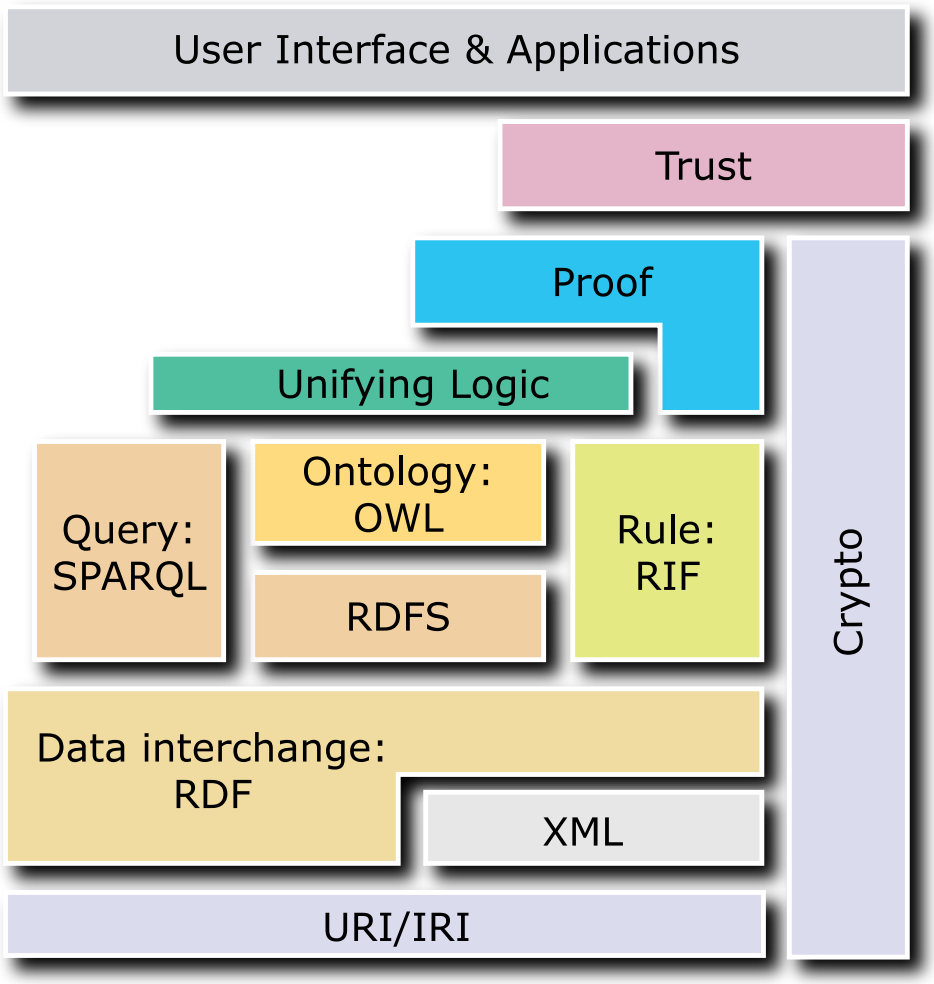


Figure 5.3: *The Semantic Web Layer Cake — from* `http://www.w3.org/2007/03/layerCake.png`

5.3 Web Ontology Language

OWL (Web Ontology Language) is the W3C recommendation for the ontology language used in the Semantic Web [van Harmelen and McGuinness 2004]. It came out of the need to have a much richer expressiveness than RDFS that primarily is restricted to class and property hierarchy with the ability to define domain and range for properties. OWL supports most of the axioms mentioned in section 5.1.3 and section 5.1.4.

OWL is divided into three sublanguages, each having different restrictions on what can be expressed.

5.3.1 OWL Full

OWL-Full does not add any restrictions on the OWL primitives. Thus being the most expressive language of OWL. However, the drawback of the expressiveness of OWL-Full is that the language is undecidable. That implies that it is not possible to have efficient and complete reasoning support.

5.3.2 OWL-DL

OWL-DL is a sublanguage of OWL-Full that restricts how the constructors of OWL and RDF can be used to make the language decidable. The advantage is that we in many practical cases get efficient reasoning support [Antoniou and van Harmelen 2008], the disadvantage is less expressiveness.

5.3.3 OWL Lite

OWL Lite is a sublanguage of OWL-DL and adds further restrictions on OWL-DL. This leads to less expressiveness, but somewhat better computational properties. The language can also be easier to understand for beginners.

5.3.4 OWL representation

There are several standards for serializing an OWL document. One is to use OWL XML, which is criticized for being bloaty. The ontologies made in this thesis are represented in the OWL Abstract Syntax. See [Peter F. Patel-Schneider and Horrocks 2004] for the specification.

5.4 Representation of SemTask

Some of the most evident alternatives to using ontologies for representing the workflow model and the data model in SemTask are:

- graphical workflow language (BPMN, UML, YAWL ...)
- workflow language for execution (BPEL, YAWL, ...)
- Data models (e.g., databases or XML-schemas)
- specialized oil-specific standard (WITSML, PRODML, ...)

5.4.1 Alternatives

Graphical workflow language

Using a graphical workflow notation is a possibility for representing the workflow diagrams. UML (Unified Modeling Language), which is probably best known for class, object and use case diagrams, and BPMN are some of the alternatives. Neither of these graphical languages are suitable for representing dataflow and workflow in one single diagram.

BPMN has a rich set of possibilities when it comes to modeling workflows. There are three shortcomings of using BPMN as-is: First, BPMN is a notation with currently no standard way of serializing a diagram. Secondly, it is not possible to model different types of data and data sources in BPMN. Thirdly, the behavioral semantics in BPMN is not clearly defined in the specification [Wong and Gibbons 2008].

UML is slightly less powerful than BPMN when it comes to representing workflows [Eloranta et al. 2006], and has all the same shortcomings as BPMN.

Ontologies have their foundation in set theory and description logic. Using ontologies for representing workflow diagrams would define a strict semantic interpretation of the diagrams. UML and BPMN have their foundation in a specification that is not sufficient for a machine to do reasoning about.

Workflow language for execution

Several workflow languages for execution exist, where BPEL is probably the most well-known. There are execution engines able to execute BPEL processes. However, they are not as well supported for human tasks as computer based tasks. BPEL is better adapted for the collaboration of different web services than connecting resources and processes.

Data models

Data models (e.g., databases or XML-schemas) define the structure and integrity of their data sets. This structure is often only intended for one enterprise, or application, and not intended to be shared by other applications. Ontologies should be as generic and task-independent as possible [Spyns et al. 2002]. G2 processes of IO include the integration of different vendors into the processes. The integration of their applications and data would be easier using ontologies.

Another big advantage of using ontologies instead of data models is the ability to do reasoning (see section 5.4.2 on the following page for example).

Specialized oil-specific standard

Data can be represented in a specialized oil-specific standard, like PRODML. The problem with most of these standards is that they are quite narrow, so they do not capture all the data we would need. Although ISO 15926 contains many oil-specific data, it is missing some core relations we would like to have (like that temperature is relevant to pressure).

We are not aware of any oil-specific standards for modeling workflows.

Data representation

UML class diagrams can represent data as class diagrams, which could be implemented using databases or Java. However, representing data in an ontology makes it more accessible for reasoning. Scalability of the systems based on these ontologies will be better since all parties use the same data representation. Alternatively they could map their internal data representation to the one provided by the ontology, instead of mapping between all the different internal representations used by the different parties they collaborate with.

WSML

WSML (Web Service Modeling Language) [de Bruijn et al. 2005b] is an ontology language meant for representing semantic descriptions of web services. It is part of the WSMO (Web Service Modeling Ontology) [de Bruijn et al. 2005a] project. The language comes in five flavors each with different expressiveness:

- *WSML Core* which is the least expressive one, based upon the intersection between Description Logic and Horn Logic.
- *WSML DL* which is based upon Description Logic. It is somewhat compatible with OWL (it has in general a bit lesser expressiveness than OWL, but better data type support) [Keller et al. 2006], and conversion tools exist[4].
- *WSML Flight* which is based upon Horn Logic. It has a higher expressiveness than WSML Core, and a different expressiveness than WSML DL.
- *WSML Rules* which is an undecidable extension to WSML Flight.
- *WSML Full* which unifies WSML Rules and WSML DL.

WSML is meant to be used for workflow between computer applications. It is not meant to describe human based workflow systems and diagrams.

5.4.2 Using OWL for Semantic Task Support

As mentioned earlier, there are several good workflow languages and tools to represent workflow, both as a graphical notation and as an execution language. With that in mind, why should one use ontologies to represent workflow and data?

Workflow diagram reasoning

One of the benefits of using ontologies and OWL for workflow diagram representation is, as mentioned, their foundation in set theory and description logic. This enables for workflow diagram reasoning. Reasoning can be done either to do semantic checks (e.g., check whether all tasks are connected to a lane), or to infer new knowledge (e.g., fork gateways that have several conditional sequence flows to be decision gateways).

Merging of workflow and data

None of the alternative solutions to OWL treats processes and data in the same model. Doing so will yield a couple of interesting possibilities. For instance, the execution of an activity may in it self yield new understanding of data (e.g., an alarm may not be critical if an operator has just finished checking the problem). The influence may also go the other way: data may

[4]The WSML2OWL translator — `http://tools.sti-innsbruck.at/wsml/wsml2owl-translator/v0.1/`.

lead to the execution of activities. One such example could be a sensor showing levels above the predefined threshold that would automatically trigger a task to address the problem.

Common data and workflow representation

G2 of IO includes a more tightly integration of vendors and other third party companies. A standard for data integration is therefore crucial to be able to collaborate. Using Semantic Web technology has been suggested for the Norwegian petroleum industry [Gulla et al. 2006]. There are several ongoing ontology projects including OLF's dataintegration project[5] and the ISO 15926 project. However, SemTask takes this even further by also making the workflow itself common for the different parties. Now different tools, formats, and workflow products are used by the different oil and gas participants. A common workflow and data, represented as ontologies, through the entire collaboration chain between the different parties will take IO one step closer to G2. Not only is there a common vocabulary, but also a logically consistent meaning of the terms in the vocabulary.

Data source classification

Data sources should be classified in a taxonomy, and should in addition have relations between themselves. This will make SemTask able to suggest data sources in different situation. For instance, if one does not have access to some piece of information one might be interested in the information located higher up in the taxonomy. One may not have access to a specific production plan for a specific day. SemTask may then infer that a more general production plan for the week might be interesting. Similarly, if the production plan is available and an operator asks for it, SemTask may then suggest related data that might be interesting, like different trend analyzes. Ontologies are a very good solution for this.

Data integration

One of the major benefits of using ontologies instead of an ad-hoc data model for SemTask is the ease of data integration.

OWL is also used to make ontologies for data integration. It makes it possible to map information from one source to information from another source, even if their syntactical representations are not equal.

[5] `http://www.olf.no/io/dataintegrasjon/` for more information about the dataintegration project.

Why OWL?

OWL is chosen as ontology language for several reasons. It is by far the most used ontology language [Cardoso 2007]. RDFS is not expressive enough. Its lack of expressiveness for several use cases identified by the Web Ontology Group of W3C[6] ultimately resulted in using DAML+OIL [Horrocks 2002] as the starting point for creating OWL. As OWL is the most used ontology language, and also a W3C recommendation this leads to the development of tools, research and a good community in the context of ontology engineering.

In addition, the Norwegian oil industry has decided to rely on the Semantic Web technology as a platform for future Integrated Operations [Strasunskas and Tomassen 2007].

[6] http://www.w3.org/2001/sw/WebOnt/

Chapter 6

SemTask ontologies

So far we have presented ontologies in general. How ontologies fit in the Semantic Web, different languages used for representing ontologies, and reasons why OWL is the ontology language of our choice have been presented. This chapter follows this up by examining the development of each of the ontologies in SemTask, as well as development issues such as methodology, reuse, and other design decisions. We will outline what role each ontology has in SemTask, and how they relate to each other. This is important as they form the foundation for executing the DPO-process.

6.1 Topology

The topology of the ontologies in SemTask results from the fact that we wanted to clearly separate four different aspects:

- BPMN specification — The foundation for representing BPMN diagrams.
- SemTask — An extension of the BPMN specification, needed to represent APOS-diagrams with additional semantics.
- Data — Describing data sources, and how different sources of data relate to one another.
- Execution — Used for representing state in a diagram.

This architecture is made with StatoilHydro in mind. However, it is possible for other companies to use it. They can start by using the BPMN and Execution module. This also makes it easier for the integration of workflow between different companies.

An illustration of the topology is shown in figure 6.1.

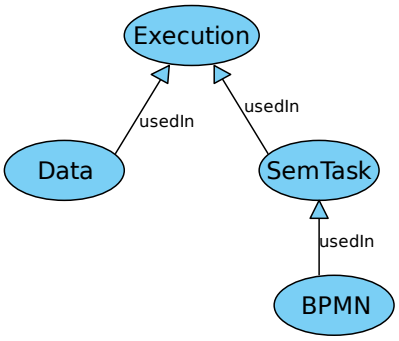


Figure 6.1: *The topology of the ontologies in SemTask*

We wanted a good process description language as a foundation, so we made an ontology based on the BPMN specification. Using this ontology, it is possible to describe specific business diagrams in RDF. However, to be able to describe the data sources in DPO, some extensions to the BPMN ontology were necessary. These extensions are included in the *SemTask* ontology.

The SemTask ontology includes the BPMN ontology making it possible to model the APOS diagrams (for instance DPO) with specific documents like "Weekly P&I-plan" (see figure 4.3 on page 43), specifying documents to be shown for example. With both of these ontologies, it is possible to represent the workflow diagrams in RDF.

The *Data* ontology describes different data sources and how they relate to one another. An instance of this ontology represents data sources found on a specific oil and gas platform. The ontology is used with the *Execution* ontology to execute the diagrams and pass data from one task to another.

The Execution ontology extends the SemTask ontology opening for the possibility to pass *Decisions* and *Tasks* with the use of data in the diagram.

6.2 Development

6.2.1 Tools

We used Protégé[1] for editing the ontologies and Pellet[2] as reasoner since they are free (Open Source), of high quality, and fit our needs.

[1]See [Noy et al. 2001] or `http://protege.stanford.edu/` for more information about Protégé.

[2]See [Sirin et al. 2007] or the Pellet homepage: `http://pellet.owldl.com/` for more information.

6.2.2 Methodology

When making the ontologies we followed mainly the development methodology described in [Noy and McGuinness 2001]. We used an iterative approach and mainly followed the phases: determine scope, consider reuse, define taxonomy, define properties, define facets, define instances, and check consistency.

6.2.3 Upper ontologies

An upper ontology is a general framework of concepts [Russell and Norvig 2003]. It consists of high level concepts such as "Places", "Measurements", "Physical Objects", etc. Other ontologies may use these upper ontologies by letting the concepts defined in the ontology extending concepts in the upper ontology. One could have used an upper ontology, such as The Standard Upper Ontology[3] or Cyc[4], in our ontologies. However, bringing in upper ontologies at this stage would make the SemTask prototype unnecessary complex. Different upper ontologies used by the different parties in IO could lead to different semantics for the same terms. Although some upper ontologies are used more than other there is currently no de-facto standard [Wikipedia 2008b].

6.2.4 Reuse

There are mainly two different views on building ontologies based on reuse: "by assembling, extending, specializing and adapting, other ontologies which are parts of the resulting ontology" or "by merging different ontologies on the same or similar subject into a single one that unifies all of them" [Pinto and Martins 2000]. We tried mainly to find ontologies in the area of workflow or something similar to BPMN to reuse in one of the two ways described.

There are several sites one could use to search for ontologies, including: Swoogle [5], Schemaweb[6], the DAML.org ontologies site[7], and normal search engines such as Google. Concerning the BPMN ontology the only ontology we found was the sBPMN ontology. sBPMN uses WSML instead of OWL, which is one of the reasons for not using it. A comparison of our ontology and sBPMN is presented in section 6.4.6.

[3] http://suo.ieee.org/
[4] http://www.opencyc.org/
[5] http://swoogle.umbc.edu/
[6] http://www.schemaweb.info/
[7] http://www.daml.org/ontologies/

One aspect of making ontologies is reusing existing ontologies, another is making the resulting ontology easy to reuse. Related to the latter is ontology modularization, which is the mechanism to extract only a subset, an ontology module, of the original ontology [Pinto and Martins 2000]. Talking about composing and decomposing ontologies it is argued that: "ontologies cannot be decomposed into semantically distinct components, we cannot predict the scope of a (local) change, and how to reuse parts of ontologies or safely compose them are open problems" [Sattler et al. 2006a]. We have therefore not made any effort to make the ontologies easily accessible for reuse.

The BPMN ontology is relatively small in size and is a general formalization of BPMN. Others may use it for their ontology integration, and it can be argued that when a small set of modular, highly reusable ontologies are available, large ontologies for specific purposes can more easily be assembled. Ontology modularization is a complex area, and different approaches are used to accomplish ease of ontology integration.[8]

6.2.5 Other design decisions

There are many design decisions to make while creating an ontology. One of the simplest is naming conventions. We have chosen to use singular for class names as it is most used in practice [Noy and McGuinness 2001]. We use UpperCamelCase for classes and lowerCamelCase for properties. Individuals (instances) in the ontologies are named using UpperCamelCase, while instances in the RDF-documents are named using lowerCamelCase.

In each of the ontologies we decided to have one super-class that all the other classes extended, one super-property for all the object properties and one super-property for all the datatype properties. We did so to organize the properties and classes in the different ontologies.

6.3 Ontology statistics

Listed below are some statistics showing how many classes, properties, and individuals each ontology have. It gives an indication of their sizes compared to each other.

	BPMN	**Data**	**SemTask**	**Execution**
Classes	77	15	18	11
Data Properties	4	7	6	4
Object Properties	13	16	6	17
Individuals	8	0	6	0

[8]For more information about ontology modularization see [Sattler et al. 2006b].

6.4 BPMN Ontology

As seen in the SemTask scenario (chapter 4), we need a formalization of the workflow for the SemTask application to work. We will achieve this formalization by creating an ontology describing workflows. The ontology is based upon the Business Process Modeling Notation (BPMN), which was discussed in section 3.7, and has been written in OWL-DL.

Representing BPMN in OWL has already been suggested [Endert et al. 2007], and some preconditions have been set on how the ontology should be [Hepp and Roman 2007]. Most of the literature on using semantic technology in terms of workflow stems from Semantic Business Process Management (SBPM) [Hepp et al. 2005] and Semantic Web Services (SWS) [Cabral et al. 2004] which is more about orchestrating Web Services and less about human interaction than what we need for SemTask.

All the elements of BPMN have been formalized. Only a part of the ontology is used in modeling the DPO workflow. But by modeling the entire specification, we have a good foundation for extending the SemTask application with more BPMN elements (e.g., exceptions, messages, and loops). BPMN is also a well-known industry standard, and can be used by all the different parties in future G2 of IO. Specific extensions of BPMN can be layered on top in the *SemTask* ontology as described in section 6.5.

The ontology as a whole can be found in appendix A.1 on page 113. An example where this ontology (and the next Semantic Task Support Ontology) is in use can be found in appendix B.1 on page 132 where an RDF-document encoded in N3-format of the DPO-workflow is found.

6.4.1 Overall strategy

Scope

Our goal for the ontology is to be able to create an RDF document representing a BPMN diagram. There are several details in the BPMN specification that could be modeled; however, the gain for this project would be small. There is also a rule of thumb to only model at most one extra level of detail [Noy and McGuinness 2001] of what the ontology is used for.

Data model vs. reasoning model

One must choose if the ontology should be used primarily as a data model or as a reasoning model. We tried to accomplish both. This lead to the decision

to use OWL-DL, which is decidable but still pretty expressive resulting in the fact that we get an ontology for the BPMN specification that can be reasoned upon.

6.4.2 Classes

The first step was to go through the BPMN Specification [OMG 2006], find concepts and make them into OWL-classes. The top class in our hierarchy is *BPMNElement*. We made four subclasses of this class, representing the four groups all BPMN-elements may be placed within: (some direct subclasses in parenthesis)

- *Artifact* (*Annotation*, *DataObject*, *Group*)
- *Connecting* (*Association*, *MessageFlow*, *SequenceFlow*)
- *Flow* (*Activity*, *Event*, *Gateway*)
- *Grouping* (*Lane*, *Pool*)

All these classes are disjoint. The subclasses of *Artifact* and *Grouping* are pretty much self-explanatory. *Flow* and *Connecting* however have some design decisions worth mentioning.

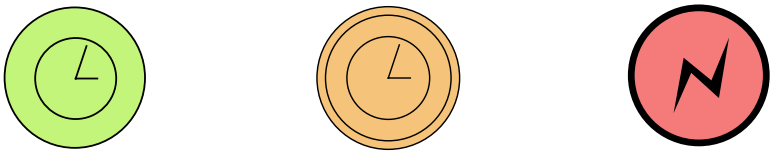
Flow

The flow class represents the flow objects in BPMN. It is a defined class, and all its members must be in a lane. All its subclasses are disjoint from one another. These are the subclasses of Flow:

- Event
- Activity
- Gateway

There are two ways the BPMN-events may be divided into different groups. Either they may be divided into different groups according to what the BPMN-specification calls "main types" (Start, Intermediate or End) or they may be divided according to what type of event they are (e.g., cancel-event or timer-event). An example of three types of events is found in figure 6.2 on the following page. To model these events, one may use one of three approaches:

1. Just use inheritance (needs multiple inheritance)
2. Use fields of an enumerated type



(a) TimerStartEvent (b) TimerIntermediateEvent (c) ErrorEndEvent

Figure 6.2: *Three kinds of BPMN-events*

3. A mixture of the two former (inheritance from one class and one field of an enumerated type)

The second approach has been used in both wsper.org's meta-model of BPMN [Jean-Jacques Dubray 2007] and in the meta-model found in OMG's upcoming BPMN 2.0 specification [OMG 2007]. This approach has the advantage that the result will have fewer classes than with the other approaches. It also has the advantage over the first approach that it may directly be mapped to a language which does not support multiple inheritance (e.g. Java). However, our choice was the first approach — inheritance. The reasons for why we chose multiple inheritance were twofold: First of all it feels intuitive and second, OWL-DL supports it. A convention in ontology making is to have all subclasses to be a "is-a" relation to its super class. If it feels unnatural to say subclass B "is-a" A (which is the super class) one might want to reconsider the hierarchy. The modeling of BPMN events using multiple inheritance supports this convention. Also, the number of classes did not grow to an unacceptable level. As base-classes for the events we made all the main types, as well as all the event types. The concrete events are inherited from two of these groups. For instance, ErrorEndEvent inherits from the class EndEvent and the class ErrorEvent. Not all combinations are valid. For instance, the BPMN-specification does not allow an event that has main-type end, and event type timer, no TimerEndEvent is therefore found in our ontology. This type of restriction would have required more work to model if we had not chosen a multiple inheritance approach. An intercept of the OWL-code describing events can be found in listing 6.1 on the next page.

A BPMN-activity can be either a Task or a Sub-Process, depending on if it is compound or not. An activity can be one of: "normal", "looped", "multiple instance", or "compensation". A sub process can be all these, in addition to type "ad-hoc". Each activity, independent of being task or sub-process, has a type that can be: "Service", "Receive", "Send", "User", "Script", "Manual", "Reference", or "None" describing how it should be executed. Here, we have the same modeling alternatives as with the events. Modeling this with just inheritance would require about 100 classes. We chose therefore to model

Listing 6.1: *Some of the OWL-code regarding the Events*

```
#Event Main types:
Class(a:StartEvent partial
 a:Event)
Class(a:IntermediateEvent partial
 a:Event)
Class(a:EndEvent partial
 a:Event)

#Some event types:
Class(a:ErrorEvent partial
 a:Event)
Class(a:TimerEvent partial
 a:Event)
#(...)

#Some events:
Class(a:ErrorEndEvent partial
   a:ErrorEvent
   a:EndEvent)
Class(a:TimerStartEvent partial
   a:TimerEvent
   a:StartEvent)
Class(a:TimerIntermediateEvent partial
   a:TimerEvent
   a:IntermediateEvent)
#(...)
```

Listing 6.2: *Some of the OWL-code regarding the Activities*

```
#Definition of the value partition: ActivityTaskType
Class(a:ActivityTaskType complete
 oneOf(a:None a:User a:Manual a:Receive a:Script a:Service
      a:Send a:Reference))
ObjectProperty(a:hasActivityTaskType Functional
 domain(unionOf(owl:Thing a:Activity))
 range(a:ActivityTaskType))

#An example of an activity
Class(a:CompensationActivity partial
 a:Activity)

Class(a:SubProcess partial
 restriction(a:hasStartEvent cardinality(1))
 a:Activity)

Class(a:CompensationSubProcess partial
 a:SubProcess
 a:CompensationActivity)
```

Listing 6.3: *Example RDF-N3 code of a ServiceCompensationSubProcess*

```
:exampleCompSubProcService  a bpmn:CompensationSubProcess;
 bpmn:hasActivityTaskType bpmn:Service;
# (more properties here)
 .
```

the execution-type as an enumerated field. In listing 6.2 one can see some of the OWL code making up the activities, and in listing 6.3 one can see how a compound service sub-process looks like in RDF.

Gateways are pretty straightforward to model. The *Gateway* class has two subclasses named *MergeGateway* and *ParallellForkOrJoinGateway*. The MergeGateway has *OrMergeGateway* and *XorMergeGateway* as its subclasses. The ParallellForkOrJoinGateway has two defined subclasses, a *ParallellForkGateway*, which is defined as a ParallellForkOrJoinGateway with more than one SequenceFlow coming out of it, and a *ParallellJoinGateway*, which is defined as a ParallellForkOrJoinGateway with more than one *SequenceFlow* coming in to.

Listing 6.4: *An example of a sequence flow*

```
:implementInitiativeToOptimalProduction      a bpmn:
    ConditionalSequenceFlow;
 bpmn:connectedFrom :decisionImplementInitiative;
 bpmn:connectedTo :optimalProduction;
 bpmn:hasCondition "No"^^xsd:string .
```

Connecting

Connecting is the group of different sequence flows (Normal, Default or Conditional), message flow and association (directed or undirected). The first intuitive approach to model this is to use OWL-properties instead of having them as classes. However, if we want to attach attributes to these sequence flows — such as name, id, conditions, etc., we would have to use OWL-Full since OWL-DL does not support adding properties to properties. Modeling these as classes would fix this problem, but resulting in the ontology being more complicated. The RDF documents based on the ontology would also be more complicated. However, as a result of using OWL-DL, these drawbacks had to be included in the ontology. In listing 6.4 one can see the RDF code for the sequence flow between "Implement initiative?" and "Optimal production" from the BPMN-diagram found in figure 4.3 on page 43.

All the connecting classes: *SequenceFlow*, *MessageFlow*, and *Association* are sub-classes of *Connecting*. The sub-classes of *SequenceFlow* are: *NormalSequenceFlow*, *DefaultSequenceFlow*, and *ConditionalSequenceFlow*.

6.4.3 Properties

The datatype properties in our ontology are straight forward and consist of:

- hasCondition: used for setting conditions on properties
- hasMessage: used for specifying messages on message flows.
- hasName: used for setting the name of the elements, like "Follow-up" or "New initiative".

They are all functional and have all String as range.

The object properties and their inverse:

- hasActivityTaskType: used to specify the activity task type

- hasStartEvent: used to specify the start event that begins a sub-process
- hasConnecting ⇔ connecting. With sub-properties:
  - hasConnectingTo ⇔ connectingTo
  - hasConnectingFrom ⇔ connectingFrom
- isInLane ⇔ hasObject: specifies which lane a BPMN-element is in
- isInInPool ⇔ hasLane: specifies which pool a lane is in

bpmnStartEvent is both functional and inverse-functional (a sub-process can only have one start-event which is the beginning of that sub-process, and that start-event can only be a part of one sub-process). isInLane, isInPool, connectedTo, hasActivityTaskType are all functional.

6.4.4 Deviations from the BPMN specification

We have made the restriction that a subprocess must have one, and only one, start event that starts the entire process within. This is accomplished by adding the property *hasStartEvent* having *SubProcess* as domain and *StartEvent* as range. The property is added as a necessary condition on the class *SubProcess* with the cardinality of 1.

6.4.5 BPMN as language

BPMN is a notation, and not a language. As a result of this, no standard storage format exists at the moment. There are however work going on to create such a format, but there are weaknesses in all of them. XPDL 2.0, OMG's BPDM, and Intalio's XML are three proposals. To create a portable format for the BPMN diagrams a unique mapping from the diagram to the storage format is necessary. This storage format could for example be XML. It has been argued that the format should reflect the terminology and semantics of the diagram, and that neither of XPDL 2.0, OMG's BPDM nor Intalio's XML does so [Silver 2007].

By making an ontology from BPMN we have made it into a vocabulary with strong semantics. The storage format could then be in any of the RDF syntax formats such as:

- N3
- Turtle
- RDF/XML

Our format supports most of BPMN's semantics. What it currently does not support is the graphically placements (i.e. x and y coordinates) and sizes of the elements. This is something that should be addressed before it can be used as a standard for serializing BPMN diagrams.

By specifying the standard in an ontology instead of an XML Schema (such as XSD), one can express the standard on a higher level of abstraction. [Klein et al. 2001] Using an ontology to specify a storage format and for instance RDF/XML to serializing the diagrams have several advantages:

- Unambiguous syntax
- Tool support (for instance automatic generation of Java classes out of the ontology)
- Semantic checks (for instance that a stop-event cannot have any sequence flows from it. These types of checks can easily be done with already built libraries/reasoning engines.)

6.4.6 Comparison to the sBPMN ontology

We found another BPMN ontology called sBPMN [Abramowicz et al. 2007]. This ontology was created as a part of the SUPER[9] research project. The ontology is a part of a larger stack of ontologies that aims at bringing semantics to Business Process Management (BPM). As mentioned earlier, Semantic BPM deals with orchestrating web services.

The ontology language used for sBPMN is WSML FLIGHT, which is not compatible with OWL. However, it gives sense to look at some of the similarities and differences between our BPMN ontology and the sBPMN ontology. Many of the same design decisions were made with the two ontologies. For instance, the Events were modeled as multiple inheritance, SequenceFlows as classes, and both ontologies had about the same class hierarchy. However, while our approach was to create an ontology which models the BPMN-specification as close as possible, the sBPMN is an ontology with some practical concepts not found in the BPMN-specification added. For instance, we find concepts such as Participant, Transaction, and WebService, which are not found in the BPMN-specification.

[9]SUPER is an abbreviation for Semantic Utilised for Process Management within and between Enterprises — `http://www.ip-super.org`.

6.5 Semantic task support ontology

The purpose of the *SemTask* ontology is to extend the BPMN ontology so that one can model the DPO-process with specialized data elements, and also put extra classes which are not part of the BPMN specification. With only the BPMN ontology one cannot represent the specific data elements like "Weekly P&I plan" in other ways than just stating that it is a *DataObject*.

Since the ontology is small one might ask why it has not directly been included in the *BPMN* ontology. The main reason for this is that we wanted to clearly separate the BPMN specification from the other parts of SemTask. Therefore, possible future extensions of the BPMN ontology should be placed in this ontology.

The idea of this extension is based on the fact that in the DPO there are activities that use a fixed set of data. If we look at the BPMN diagrams (see figure 4.3 on page 43) we find three types of data usage — either a type of document (e.g., "Weekly P&I plan"), a computer application (e.g., "MaxPro") or generated data ("Optimization Potential").

The OWL-ontology can be found in appendix A.2 on page 122. An example where this ontology is in use can be found in appendix B.1 on page 132 where an RDF-document of the DPO-workflow is found.

6.5.1 Classes

The ontology has classes which extend the *DataObject* class from the BPMN ontology with specific documents. The documents defined in the ontology are: *URLBasedDocument*, *ApplicationDocument*, and *FormDocument*. These classes are all disjoint from each other. The ontology also has a class related to the FormDocument: *Field*.

A URLBasedDocument represents a document that may be accessed with a given URL. Examples of these are normal web pages (either on intranet or on the internet), or PDP workspaces. Local files can also be represented with a URL (e.g., `file://`). An ApplicationDocument represents a description of how to execute a program. It typically has a URL pointing to where the executable program is, along with a list of program parameters that should be passed to the program.

A FormDocument represents a form to be filled out. An example of use in DPO is the "Optimization Potential"-data sent from the activities "Production follow-up" and "Continuous production supervising" to "Proposition for initiative". The idea is that "Optimization Potential" could be modeled as document form. It consists of fields like "Name of

Listing 6.5: *A FormDocument in RDF*

```
:optimizationPotential a semtask:FormDocument;
  bpmn:hasConnectingFrom :>
      optmizationPotentialToPropositionForInitiative;
  bpmn:hasConnectingTo :>
      continuousProductionSupervisingToOptmizationPotential >
      ,
  :productonFollowUpToOptmizationPotential;
  bpmn:hasName "Optimization Potential"^^xsd:string;
  semtask:hasField :estimatedRisk,
  :estimatedValue,
  :longDescription,
  :shortDescription .

:shortDescription a semtask:Field;
  semtask:hasName "Short description of the optimization >
      potential"^^xsd:string;
  semtask:isInForm :optimizationPotential .
:estimatedValue a semtask:Field;
  semtask:hasName "Estimated Value in NOK"^^xsd:string;
  semtask:isInForm :optimizationPotential .
#(more fields...)
```

Listing 6.6: *The definition of DecisionGateway*

```
Class(a:DecisionGateway complete
 intersectionOf(b:Gateway restriction(b:hasConnectingFrom >
    allValuesFrom(b:ConditionalSequenceFlow)) restriction(>
    b:hasConnectingFrom minCardinality(2))))
```

proposal", "Estimated value", etc. An intercept of the RDF-code forming this FormDocument is found in listing 6.5.

The ontology is also meant to have general extensions of the BPMN specification (e.g., the *DecisionGateway* class). It defines gateways where a user needs to do a decision (e.g. "Implement initiative?"), and has least two conditional sequence flow going out from it. The definition can be found in listing 6.6.

6.5.2 Properties

The datatype properties are: *programParameters*, *URL*, *hasName*, and *isBeginningEvent*. The two former properties have already been explained. *hasName* is used to set the name of a field. *isBeginningEvent* is a property

that is set on the BPMN start event if it is the start event of the entire diagram. It is beneficial to have such a property when one looks at the execution of a diagram.

The only object property in the ontology is *hasField*. It has FormDocument as domain and Field as property.

6.6 Data ontology

This section presents the ontology used for representing the data sources that are needed for the SemTask scenario to work. The purpose of the ontology is to provide the users of SemTask options for further evaluating decisions and tasks by looking at relevant data sources to the ones the users have looked at. The previously mentioned scenario contains examples of suggestions of data sources.

The ontology can be found in appendix A.3 on page 125, and an example RDF-document using the ontology is found in appendix B.2 on page 151.

The data source ontology is only meant to cover a subset required for SemTask. The scope is limited to deal with the work process DPO, and the functionality mentioned in the SemTask scenario. As the ontology is made without help from domain experts, only very simple mappings are made. Ultimately, a more comprehensive ontology should be created. This ontology might reuse some ideas from PRODML, or some concepts from the ambitious ISO 15926 Ontology, and it should be created with much more interaction with domain experts.

The Data ontology is inherited in the Execution ontology, and used by properties there. How the Data ontology in the Execution ontology works will be explained in section 6.7.

The Data ontology is an ontology concerning data sources, and is not an ontology of data itself. This makes an instance of the ontology able to express different types of data sources (e.g., sensors), where they are placed, how they are related to one another, and how one could access data from these data sources (i.e., a PDP-ID). If one would like to refer to a concrete measurement, one could refer to the data source and the exact time of the measurement.

There are several reasons for not using the production data directly in SemTask. As previously described in section 2.3.3 on page 14, StatoilHydro's PDP application has optimized the flow and processing of data by having one centralized access point for it, along with several standardized applications to present the information. When decisions in a workflow are connected to

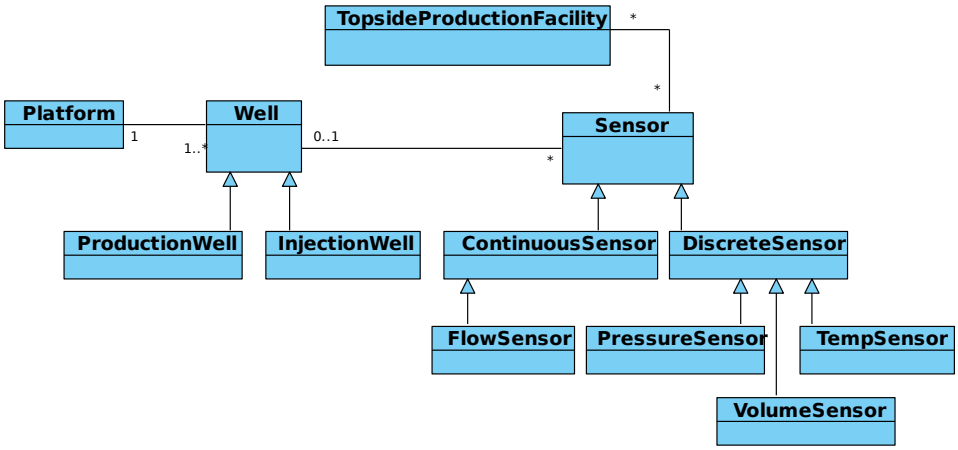


Figure 6.3: *A UML class diagram representing the data model*

the sources of data being used at different times, one could easily use the existing tools to look at this data. This is beneficial since one does not need to create these tools again, and since operators at StatoilHydro are used to a set of tools, and do not want to use time to learn new ones.

We decided to model the concepts with their properties and relations as a data model represented in UML first. By doing this it is easier to get a sense of how the data sources are related to one another, compared to what it is describing it with text. It turns out that the Data ontology is very simple, and can be almost directly mapped from the UML representation to an ontology.

6.6.1 Data model

Before creating an ontology for the data needed to represent the SemTask scenario, we will present a more simplified data model as a UML Class Diagram. The diagram, which is found in figure 6.3, is a basic description of the data sources needed, and how they relate to one another.

We have two types of wells: injection wells and production wells. We have sensors that either may be on a well or on a topside production facility. These topside production facilities may be things like a separator, a water cleaner, tanks, and so on. Both well and sensor classes are constructed in a hierarchy using multiple inheritance.

6.6.2 The Data Ontology in OWL

We will not go into too much detail on how the conversion from the UML Class Diagram to an OWL-DL is done. The main reason is that the conversion is pretty straight forward, like the class *Well* and the subclasses *ProductionWell* and *InjectionWell* converts to exactly that in the ontology with the two subclasses disjoint. This can also be read by looking at the OWL code. This is valid for most of the data model.

Classes

The classes are more or less taken directly from the UML-diagram.

Properties

There are two datatype properties in the ontology. Both are functional:

- hasDescription: used to attach a short description of a given data source
- hasPDPId: used to set the PDP-id of the data element

The object properties are:

- artifactBelongsTo $\Leftrightarrow$ hasArtifact: used to set which platform an artifact belongs to
- isLocatedOn $\Leftrightarrow$ hasSensor: used to set where a sensor is located, i.e., a well, an artifact or just a platform
- wellBelongingTo $\Leftrightarrow$ hasWell: sets which platform a well is connected to
- relevantData : set relevant data. Sub-properties:
  - relevantArtifact: used to connect relevant artifacts together.
  - relevantSensor: used to connect relevant sensors together.
  - relevantWell: used to connect relevant wells together. If two wells are very close to each other for instance, they may be relevant to one another.

The first three of the object properties are functional. The last four properties are symmetric and transitive.

6.6.3 Instances of the ontology

An instance (an RDF document using the ontology) of the Data ontology will typically be a description of an oil and gas platform. This instance will be reused in the execution of all work processes related to that platform. The instance will have information of which wells, topside artifacts, and sensors available on that platform.

6.6.4 Extensions

As mentioned, this ontology is made without interaction with domain experts. This makes it less interesting than it could have been. Further extensions of the ontology could be made by having more classes, especially defined ones. What also can take this ontology further is the introduction of logical rules. Rules will be discussed in section 7.2. With rules one may state terminological knowledge such as "If a sensor x is of type TempSensor and is located on Well y, and one has another sensor z of type PressureSensor also located on y, then: x isRelevantTo z." These kinds of statements would make the ontology more useful as one would not need to state all the isRelevantTo-statements explicitly for each sensor, for instance.

6.7 Execution ontology

With the two ontologies: BPMN and SemTask, we can represent a workflow diagram that represents the BPMN-extended SoluDyne diagrams previously seen in figure 4.3 on page 43. In other words, with the ontologies it is possible to represent a static picture of a workflow like the DPO. But some of the functionality of SemTask relies on the notion of state, and the dynamic behavior of a process. This is something which is not found in the BPMN specification [Wohed et al. 2006], and therefore not in our BPMN-ontology either. The Execution ontology is introduced for making SemTask capable of representing state in an execution of a workflow.

This section gives an overview of the ontology. As the ontology forms the foundation of workflow diagram execution, it can be illuminating to explain the overall idea of execution first. This idea evolved during our work, and will be explained. How it evolved will, as it clarifies some of the design decisions of the execution ontology itself, be described. Rules are used in the execution engine of the SemTask application, and will be thoroughly explained in chapter 7.

6.7.1 The overall idea

Maybe the most intuitive approach of representing workflow diagram states would be to have a status attribute on each of the elements, in the workflow diagram, denoting the state they are in. Elements meaning: tasks, subtasks, gateways, and events. The status attribute would typically be an enumeration of:

- Ready: It is ready to be started.
- Ongoing: Someone is currently working with this task.
- Waiting: It is not possible to start the task at the moment, and its neither finished nor ongoing.
- Finished: The task is finished.

The activities would then change their status during the execution of a workflow. One example would be the "Production optimization meeting" task. It would initially have the status of *Ready* when the workflow is initialized, changed to *Ongoing* as the meeting starts, and to *Finished* when the meeting is finished. Then subsequent tasks will automatically be started by changing their status from *Waiting* to *Ready*.

Data sources would only be connected to the activities that use or produce them. An example is the "Topside production optimization" task, which produces "Production increasing initiative report" or "Production optimization meeting", which uses "Weekly P&I plan" as seen in the DPO-process (denoted in figure 4.3 on page 43).

This idea had several weaknesses. One is where two different tasks, connected to the same subsequent task, finishes. The subsequent task would then only get the state Ready. The information that it was two tasks responsible for this task getting Ready is lost, hence also the information that this task needs to be done twice. A possible solution would be to let each task have a property *counter* that reflects how many times the task should be run. However, you will not be able to see why it should be run. After having worked on the initial idea we noticed a more important problem. Data is sent between all the different tasks, not only between those which have dataflow modeled in the BPMN-diagrams such as "Topside production optimization" and "Proposition for initiative". The execution model needed to reflect this.

This lead to a new architecture. We discovered that one could divide the data being sent into decisions and tasks. In the BPMN diagram created for DPO we only show flow of data for the data documents, this convention should be followed as showing dataflow between all tasks/subtasks would

complicate instead of clarifying the diagrams. Also these data documents, such as "Production increasing initiative report" change their meaning from denoting the data being produced and used by activities, to being schemas or forms for the activities to include each time data is to be sent to a subsequent activity.

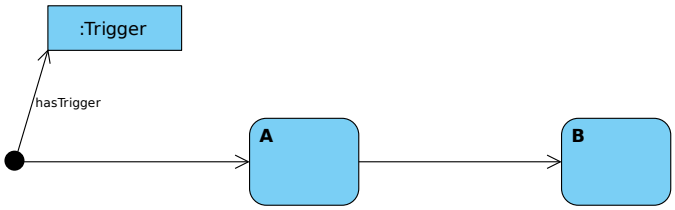
Flow elements (like Activities) in the BPMN specifications have tasks connected to themselves where the tasks in turn have relevant data sources. Out of the tasks, decisions are made. The tasks have also a relation *hasReason* pointing to the trigger or decision that was responsible for its creation. The effectiveness of the workflow can be increased as the data needed for a task is immediately available.

When SemTask shall suggest data sources for an operator it does so by looking at the current Task. It checks the reason for why the Task was made (typically an earlier made Decision), and sees what data were used to make that Decision. That data is suggested, along with relevant data (set by the relevantData property). Let us look at an execution example shown in figure 6.7.2, which uses tasks, triggers, and decisions.

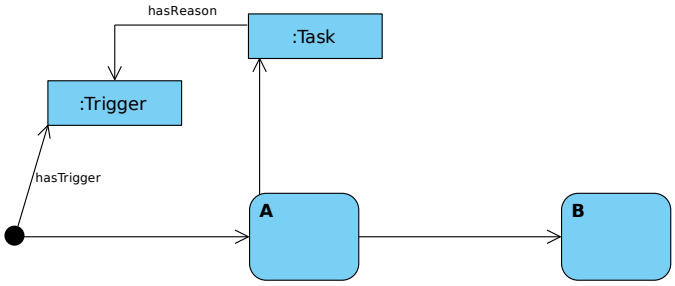
6.7.2 Example execution

First an event is triggered as a result of a timer going of. The triggering is represented by an instance of the Trigger class, connected to the activity with a hasTrigger-property. An activity A is then assigned a task, because of the Trigger. The task shows the reason for its creation with the *hasReason* property. When a person is done with the Task, a decision is made. This decision may have usesData, hasDocument, and hasDescription properties attached to it. This decision is a result of the task, and this is indicated by the *isResultOf* property. The decision is recorded on the activity A before it passes on to the subsequent activity B as a new Task. The decision made from activity A to add a task to activity B is reflected in the fact that the task assigned to B has a relation *hasReason* from the task to the decision.

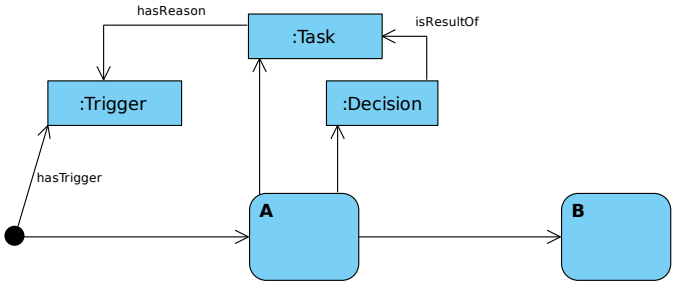
The tasks and decisions are never deleted throughout the execution. This enables backtracking, so one can see the reason for why a task has been assigned to an activity, along with the reason for that reason again (recursively). It is then possible to go back as many steps as one would like. The overall monitoring of the workflow execution can get a full history of the tasks and decisions made.



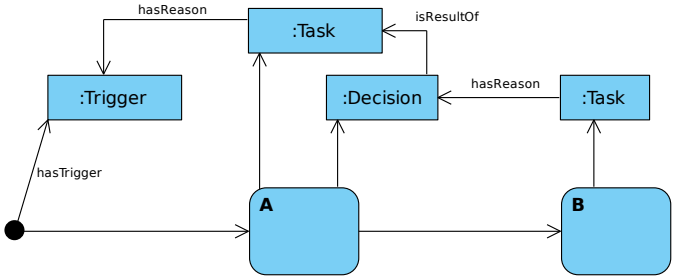
(a) Step 1. The circle at the left side represents an event. The two rectangles, A and B, represent tasks. The arrow between the event and A, and the arrow between A and B represents a sequence flow. We can see that the event has been triggered.



(b) Step 2. The execution engine attached a new task to A, since the event was triggered, and there is a sequence flow between the event and A.



(c) Step 3. After the owner of activity A is finished with the task, she creates a decision.



(d) Step 4. The execution engine attached a new task to B, since A had a decision, and there is a sequence flow between A and B.

Figure 6.4: *An illustration of four execution steps*

6.7.3 Classes

There are three classes regarding the general execution in the Execution ontology: *Decision*, *Task*, and *Trigger*. The general idea is that zero or more Tasks are assigned to each activity. Decisions are made from Tasks; these Decisions may lead to new Tasks, again passed to subsequent activities. Triggers are connected to events, and lead to creation of new tasks. Besides these classes there are two classes regarding the forms mentioned in the SemTask-ontology: *FilledForm* and *FilledField*.

Decisions typically contain a textual description along with some attached documents. The data used for making that decision should also be attached to an instance of Decision.

As Decisions and Tasks are made, their relevant data are attached using the belonging properties. The Decisions and Tasks are not deleted throughout the execution of the diagrams. One can therefore always backtrack and see the relevant data that was used for a Task or Decision.

FilledForm and FilledField are used to represent forms that are filled out with information.

6.7.4 Properties

The properties and their inverse are:

- triggerBelongsTo ⇔ hasTrigger: connects a Trigger to an Event
- taskBelongsTo ⇔ hasTask: connects a Task to an Activity
- decisionBelongsTo ⇔ hasDecision: connects a Decision to a Task
- hasFilledField ⇔ belongsToFilledForm: sets which FilledForm a FilledField belongs to
- hasFilledForm ⇔ usedInDecision: used to connect a Decision to a FilledForm
- isOfFormType: used to denote which kind of Form the FilledForm is of
- hasDescription: used to have a textual description of Decisions and Tasks
- isResultOf ⇔ resultsIn: used to connect the Decisions to Tasks
- usesData: used to denote what data a Task or Decision is using

- hasReason: used to denote the reason why a Task was created (either a Trigger or a Decision)
- hasDocument: used to add documents to decisions

Chapter 7

Execution model

The previous chapter presented the ontologies in SemTask, and how they relate to each other. This chapter will use these ontologies together with rules to form the executional part of SemTask.

The chapter starts with limiting the scope of the execution engine we are going to create. Different alternatives for creating the workflow execution engine are outlined, along with an explanation of why we chose to use rules. An introduction to rules is presented, with a list of current Semantic Web implementations, and how rule engines work by doing triple pattern matching. Then we go through one of the rules we made for creating Tasks out of Triggers. The chapter ends by explaining how this execution engine may fulfill some parts of the SemTask scenario.

7.1 Introduction

Building an enterprise workflow execution engine is a difficult task since workflows can be highly complex with synchronization between tasks, exceptions, distributed workflows, and communication between tasks/workflows, etc. However, the functionality of the execution part of SemTask, described in this chapter, is limited to a few areas: automatic suggestions of relevant data, handling of tasks, and monitoring.

One of the key features of SemTask is the suggestions of relevant data sources. We use some simple techniques to infer which data sources might be relevant for a given task. First we use the data sources that are relevant (denoted by a relevantData-property) for the data that the operator is currently examining. If the operator is for instance currently looking at the well temperature in well A1, and that sensor has a relevantData-property pointing to the well-head pressure in well A1, that data source will be

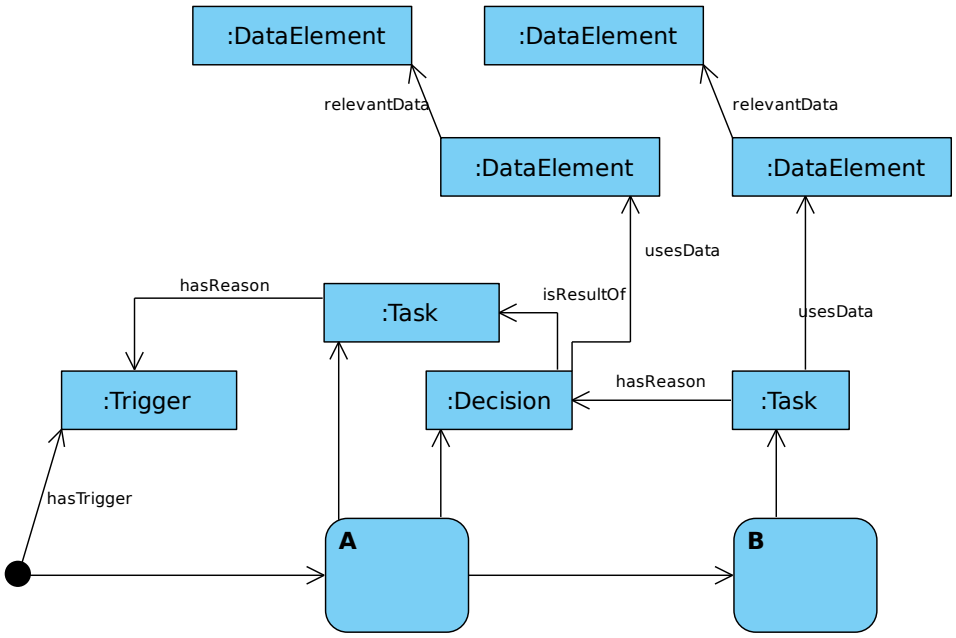


Figure 7.1: *An example of relevant data. If a person is working with Task B, all the DataElement-objects seen here will be suggested as relevant data.*

suggested. Second we see on the hasReason-property of the current Task. If it points to a Decision which again has some usesData-properties, these data sources would also be suggested.

An illustration of relevant data can be seen in figure 7.1. If a person works with the task connected to B, then the data source he has been looking at is relevant, and similar data elements to that (found by the relevantData property) may also be relevant. The Task connected to B was created because of a previous Decision connected to A. That Decision was made based on a data source. This data source as well as similar data sources may also be relevant.

The SemTask architecture is based on a centralized server-client architecture. The clients are the participants in a workflow execution (e.g., Per and Didrik from the scenario), and the server is responsible for the execution itself. The focus in this chapter will be on the execution engine on the server-side. The implementation communication between the client/server, security, exceptions, Graphical User Interface (GUI), etc., belong to further extensions of SemTask.

7.1.1 Execution alternatives

Building an engine to support the different aspects of workflow diagram execution can have several different solutions. Let us briefly look at the two approaches we considered.

Java approach

Maybe the most obvious way is to use an imperative language such as Java with a corresponding framework to deal with RDF. Benefits include great tool support, excellent community, and mature technology. The downside is that with imperative languages it is a clear separation between specification and implementation. The implementation of a complex execution engine is doomed to introduce errors and deviations from the specification. As it is a language mismatch between RDF/OWL and Java, one would need many lines of code for doing simple things such as accessing a property of a given instance. This makes the code base large, and therefore hard to maintain.

Rules approach

Being declarative, rules have several benefits over imperative languages such as Java. The code is often smaller, easier to read, with a higher level of abstraction, and greater ease of later code extension. A downside is that rules can be difficult to combine with other languages or frameworks, and they do not fit for all tasks. Rules can only be used to specify some parts a system. We found that rules were a good fit for specifying the execution engine.

7.2 Rules

There are several rule languages proposed for the Semantic Web. Semantic Web Rule Language (SWRL), Description Logic Programs (DLP), and Jena rules are probably the three most known proposals.

SWRL [Horrocks et al. 2004] is a W3C member submission, with several partial implementations already. No full implementation exists as full support would lead to undecidability.

DLP [Grosof et al. 2003] is a newer proposal than SWRL. DLP is less expressive than SWRL, but decidable.

Jena [Carroll et al. 2004] is not a rule specification, but a Semantic Web Framework for Java. It provides a programmatic environment for RDF,

RDFS, OWL, and SPARQL. It also includes a mature rule engine that supports to use custom rules. Besides the normal possibility of inferencing new knowledge one can delete triples, and alter the RDF graph in other ways. We chose to use Jena rules because we have good experience with Jena, and the rule engine seemed to be powerful enough.

7.2.1 Triple pattern matching

There are several ways of understanding how rules work. One of them is to think of the rule engine as a triple pattern matching engine [Holger Knublauch 2006]. Let us look at the following example:

```
(?x childOf ?y), (?y hasBrother ?z) -> (?z uncleOf ?x).
```

A rule consists of two parts: a body (also called antecedent or premises) and a head (also called succedent or conclusion). In the above rule, `(?x childOf ?y), (?y hasBrother ?z)` is the body and `(?z uncleOf ?x)` is the head.

Different rule languages have different types of *built-ins*. Built-ins are special functions that can be found in the rule-language. Often found built-ins are equality functions, arithmetic functions, negation predicate, string tests, string manipulation functions, etc. Some rule-languages, including Jena, allow to create own built-ins[1].

A rule engine goes through the knowledge base, and tries to match all the premises in the head of the rule with triples. If it succeeds, it asserts the body of the rules. Rules may depend on each other. For instance, if we had a second rule that looked like this:

```
(?x childOf ?z), (?y childOf ?z), (?y sex "male"), (?x != ?y)
-> (?x hasBrother ?y)
```

And a knowledge base with the following triples:

```
(John childOf Brad), (Alex childOf Brad), (Jeff childOf Alex)
(John sex "male")
```

The rule engine would infer:

```
(John uncleOf Jeff), (Brad hasBrother John)
```

[1]For information about built-ins in Jena, see `http://jena.sourceforge.net/inference/`.

7.3 Rules in Jena

7.3.1 Testing platform

To create the rules, we needed a way of testing them. One solution would be to use an IDE that supports development of Jena-rules. One such IDE is TopBraid Composer[2]. Another solution would be to create a sample Java application, using Jena. We chose to do so, since we felt it gave us a better overview of what was happening.

7.3.2 Execution engine implementation

We have only implemented two rules. The complete source of the rules is found in appendix C on page 155.

In the execution example, Step 2 (figure 6.4(b) on page 83) one saw that the execution engine created a Task out of a Trigger. In listing 7.1 on the following page one can see the rule that actually does this.

In line 3–6 the types of our variables are specified. These lines are not strictly needed, as the domain and range of the properties below them would infer the same types. We left them there as the rule might be easier to understand with them than without them. Line 9–11 specifies that the sequence flow, activity start-event, and the trigger are connected in the way we want. In line 14–20 we see use of three different kinds of Jena built-ins. noValue asserts that it should not be possible to match the following triple. It is used to make sure that this rule is only triggered once for a given pair of Trigger and Activity instances. makeTemp(?task) creates a blank RDF-node, and now(?now) binds ?now to a timestamp with the current day and time.

The body of the rule is found in line 23–33. It should be self-explaining by looking at the well-commented code.

7.3.3 Completing the execution engine

Because of time limitations we have only implemented two of the rules needed for the execution engine. One might encounter some difficulties with the expressiveness of the rules when implementing the rest of the parts of the execution. However, we believe that it will be possible to complete the workflow-engine by creating the rest of the rules, at least if one allows creating additional built-ins. What is currently missing is rules to handle

[2]TopBraid Composer from TopQuadrant. `http://www.topquadrant.com/topbraid/composer/`.

Listing 7.1: *The rule for creating Tasks out of Triggers*

```
1  [processTrigger:
2  // Specify the types of our variables
3    (?sf         rdf:type              bpmn:SequenceFlow),
4    (?activity   rdf:type              bpmn:Activity),
5    (?startEvent rdf:type              bpmn:StartEvent),
6    (?trigger    rdf:type              execution:Trigger),
7
8  // Make sure that they are connected to each other
9    (?startEvent execution:hasTrigger ?trigger),
10   (?sf         bpmn:connectedFrom   ?startEvent),
11   (?sf         bpmn:connectedTo     ?activity),
12
13 // This trigger has not triggered this activity before
14   noValue(?trigger trigged ?activity),
15
16 // Create a new, blank, RDF-node.
17   makeTemp(?task),
18
19 // Create a new RDF-node with the current time.
20   now(?now),
21
22    ->
23
24 // Make sure this is not executed any more for this event
25   (?trigger trigged ?activity),
26
27 // Create the new Task and attach it to the activity and ↩
       the trigger
28   (?task rdf:type execution:Task),
29   (?activity execution:hasTask ?task),
30   (?task   execution:hasCause  ?trigger),
31
32 // Set timestamp for the trigger
33   (?task   execution:created  ?now)
34 ]
```

Listing 7.2: *Weekly P&I plan SPARQL query*

```
PREFIX bpmn: <http://www.ifi.uio.no/fredaleks/semtask/bpmn#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX semtask: <http://www.ifi.uio.no/fredaleks/semtask/semtask#>
SELECT ?Plan
WHERE {
        :productionOptimizationMeeting bpmn:hasConnectingTo ?connecter .
        ?connecter bpmn:connectedFrom ?Plan .
        ?Plan rdf:type semtask:URLBasedDocument .
}
```

different kinds of forks, joins and gateways. We estimate that only a couple of more rules will be needed to cover this. The final execution engine can probably be specified with less than a hundred effective lines of rules.

7.4 Fulfillment of the scenario

Now let us look at how this architecture and its partitioning of tasks and decisions can be used to fulfill the SemTask scenario described. Let us start with the automatic execution of activities and how SPARQL, or the Jena framework, can be used to bring up the relevant data concerning a particular process.

Automatic suggestions of relevant data

Let us exemplify by showing how the "Weekly P&I Plan" is automatically brought up in the "Production optimization meeting"-activity. We make the assumption that we are in the "Production optimization meeting" process. The SPARQL query given in listing 7.2 will list up all the relevant documents so that the program can automatically show the right information, and also bring up the necessary applications. The query returns: the Weekly P&I plan.

Adding inference support to SPARQL with the SemTask ontology one can do reasoning to find documents that are further down in the hierarchy (i.e., by subsumption get all instances of the subclasses of the class originally used in the query). In further extensions it is also possible to extend the class hierarchy introducing new and more specialized documents.

Regarding the scenario one can see that this fulfills the feature where the right PDP instance gets shown on the screen in the "Production optimization meeting".

Dataflow between the different tasks can help with automatic suggestions of relevant data. Once an activity is activated one can list all potential tasks that are associated with it, with the help of either a SPARQL query, or with Jena. Each task can then typically have a PDP instance URL, a textual description, and possibly an application to be automatically brought up.

Automatically handling activities

The rules form the heart of the execution engine. To exemplify and give a quick introduction to the syntax and semantics of the rules in Jena as well as introducing the thoughts behind the strategy of the SemTask execution let us take an example.

First let us look at how the rules are used regarding the triggered events. This is the first rule in appendix C on page 155. It is up to the Java part of SemTask to create the triggers in the diagrams, meaning that the "Weekday 9am" is trigged as a result of a function in the SemTask application that checks the time. The application adds the trigger RDF triple on to the event (this corresponds to figure 6.4(a) on page 83). Now the rule triggers, and automatically sees this event. resulting in a task being made. For the *noValue*, *makeTemp*, and *now* they are used to make sure the rule does not apply several times to the trigger, the making of a blank node and the making of a time stamp[3].

Automatically viewing of decisions/available tasks

With SemTask it is also easy for the participants such as Per in the SemTask scenario to automatically get on his screen the decisions that need his attention. An easy SPARQL query is sufficient. Per can then make the decision, and the rule engine can proceed with the execution of the workflow diagram.

In much the same way the users can always get an overview of the tasks that can be started the second they become available.

[3]See the Jena documentation for more information: `http://jena.sourceforge.net/inference/#rules`.

Monitoring, statistics, and history

In the scenario we saw how Mona was able to not only see what tasks Per was currently working on in the DPO-process but also look at statistics. Our execution model supports this, and the functionality can easily be implemented by making a predefined set of SPARQL queries.

Also following the fact that tasks are never deleted throughout the execution of a process one can easily produce statistics. This reflects the monitoring and optimization part of the BPM life cycle.

Given the scenario where a participant in the execution questions why a task was assigned to her, there is a direct link to the decision which was responsible for the making of the task. It is also possible to go further back in history of tasks and decisions by using the properties: *hasReason* and *isResultOf*. In other words, a history functionality is easily implemented where one can examine tasks with their decisions, the corresponding activities, and the data sources used.

Chapter 8

Conclusion and further work

8.1 Contributions and conclusion

SemTask has introduced several features that may facilitate work processes in to the domain of oil and gas production. Our contributions are summarized by the following list:

- Proposed an architecture for SemTask, an active support system for IO work processes based on Semantic Web technology
- Upgraded the graphical work process notation from SoluDyne to BPMN
- Created an OWL-DL ontology for BPMN, resulting in a standard storage format (serialization) for BPMN diagrams
- Described (parts of) Hydro's DPO process in BPMN/RDF
- Created a usage scenario for how SemTask could significantly aid operators in the DPO process
- Created an execution model for SemTask to connect tasks, data sources, and decisions
- Proposed an elegant execution engine relying on rules
- Described how the proposed execution model fulfill the main features of the DPO usage scenario

It has been shown how the SoluDyne notation has been replaced by the more expressive BPMN. It is then possible to capture more knowledge into the workflow system, which makes it easier for inexperienced operators to carry out tasks. The conversion from SoluDyne to BPMN was shown for the DPO

process, which illustrated the first step needed to go from solely passive to active human centric workflows.

Introducing execution support for key processes has benefits such as: increased effectiveness and safety, more control, and better quality. Time can be spent on other problems than the "trivial" parts of day-to-day operations, such as making the processes more effective by optimizing the workflow. This allows an iterative approach of gradual work process improvement.

We formalized BPMN as an OWL-DL ontology. BPMN is a workflow notation standard by OMG, adopted by many companies. Making a standard serializable format, based on ontologies, is therefore a contribution to the industry. This makes it possible for different workflow vendors to have a common serialization format. Using ontologies also opens for making semantic consistency checks of the diagrams.

The SemTask ontologies provide a good foundation for the application system. They show how Semantic Web technology can be used in the oil and gas domain. SemTask realizes the scenario mentioned for the DPO-process at StatoilHydro, and brings more control into the process. This may lead to faster and better decisions both by experienced and inexperienced operators.

The architecture is modular in the sense that there is a clear separation between the ontologies and their responsibilities: BPMN specification, data sources, states in a workflow diagram, and execution of a workflow diagram. These have been layered in a logical stack. StatoilHydro and other IO parties could partly use this solution by starting with the BPMN layer.

Extending the SemTask application by introducing ISO 15926 instead of the current data source ontology may be done independently of the other ontologies. Also, further extension of the rule would easily be done by adding rules without changing other parts of the system. This opens for an iterative upgrade of SemTask to include further features and to cover other aspects of oil and gas production.

Logical rules in Jena were chosen to implement SemTask's execution engine. Even though we did not write all the rules needed to complete the engine, using rules shows a good potential. They seem to give a very small and elegant code base.

SemTask illustrates how workflow and data can be merged in workflow diagrams. This could lead to new understanding of data as well as processes. In addition, related data sources are automatically suggested when working with tasks. This saves the operators' time, and help them deal with the information overload problem. SemTask also supports sending general messages between different tasks in the workflow.

8.2 Further work

We have seen how SemTask can be used to solving problems in the DPO process. In this chapter further extensions of the SemTask application will be examined.

8.2.1 Finishing SemTask

The first thing one might want to do is to finish the rule base. A basic framework of SemTask that combines the ontologies, the rules, and diagrams has been set up. However, this is not a finished product. It needs to be a tighter integration with PDP to simplify writing workflow diagrams and link PDP screens into tasks; for example.

A set of tools should be developed to aid the operators creating the workflows. The conversion between diagrams expressed in BPMN into RDF instances could for example be done using XSLT [Clark 1999]. A user friendly GUI on top would ease the use of the transformation application.

Much of the functionality of the monitoring functionality of SemTask can be accomplished with a set of predefined SPARQL queries. However, a GUI should be built on top of the monitoring functionality to make it easy to use.

After these initial steps are finished several other possible extensions of SemTask are possible. These parts are briefly discussed below.

8.2.2 Automation of tasks

As mentioned in section 2.2.2, the final phase of Integrated Operations is partial (and in some cases full) automation of work processes. SemTask does not automate any tasks in a workflow; it only helps operators by guiding them through a workflow, and suggesting data the operators could use. The internal SemTask model can support automation of activities in a workflow as an extension. To automate a task one would add the computer application that executes the given task as a *role* in the workflow. The application would then act as a "person" who consumes *Tasks* and creates *Decisions*.

Further investigation is needed to see whether the execution ontology requires adjustments to support integration with computer applications. Some way of adding computer-interpretable parameters to a decision is probably needed for the application to do its tasks. Relevant work in this direction is reported in [Schall et al. 2008].

8.2.3 Extension of the Data Ontology

An extension of the data ontology is needed for SemTask to be usable. One should investigate what type of reasoning is needed to create helpful suggestions for the operators. For instance, one should see what type of knowledge can be encoded in OWL-DL, and what type of knowledge can only be encoded in rules. As most of the rule-languages are undecidable, one should investigate what kinds of practical problems may occur when using rules.

8.2.4 Guarantees in a workflow diagram

Certain constraints of the workflow diagrams could be checked, to ensure that BPMN is followed, using languages such as Maude [Clavel et al. 2001]. The RDF instance of the workflow diagrams could be transformed into Maude syntax, and a predefined set of queries could be executed searching for illegal patterns (i.e., *DataObject* having a direct connection to another *DataObject*). This would be a useful tool to aid the inexperienced operators in making the diagrams. Introducing formal support and analysis to a notation have already been applied to UML [Durán et al. 2003].

If the entire organization starts to use SemTask, security considerations such as role based access control and separations of duty would have to be included. Previous work has already been done in this area [Kong et al. 2005] and can be extended.

Policy rules could also be included in the set of searches, e.g., the person who proposes an initiative cannot be the same who decides whether to implement it or not. Maude opens for making complex policy rules (i.e., formalization of access control policies [de Oliveira 2007].

8.2.5 Constrains checking

The *FormDocument* class as explained in section 6.5 on page 75 would typically include several fields to be filled out. A possible extension could be to include checks so that these fields always are filled out. A related extension could then provide easy access to help, by connecting experts or detailed documents to them.

8.2.6 Extending with other ontologies

As SemTask is built with ontologies, other ontologies may be included to enhance the functionality.

Organization modeling and skill finding

The roles (lanes) in a workflow diagram now only represent a single person. This is not as flexible as one would like, as certain tasks should either be done by someone with a specific organizational role, or by someone with a defined skill. Using ontologies as a part of skill-finding has already been done [Colucci et al. 2003; 2005], and it could be interesting to see how this would fit in the SemTask architecture.

Decision support

A given problem/task in an activity can have several solutions each connected to a given outcome. These decision situations can be highly complex with significant uncertainties. SemTask provides the reasons for the given task and relevant data. An extension to SemTask could be to provide decision support, analyzing different possible solutions and aiding the operators in making the right decision. Research has already been done in this area [Fjellheim et al. 2008], and can be integrated into SemTask.

Experience finding

Knowledge of previous operations can provide important information for making the right decisions on current ones. The knowledge should be recorded and easily accessible. Previous work has been done in knowledge transfer between drilling projects, through documented experiences, best practices, and expert references in the AKSIO project [Norheim and Fjellheim 2006]. This could provide a valuable extension to SemTask.

Acronyms

AKSIO	Active Knowledge Management for Integrated Operations
APOS	ArbeidsProsessOrientert Styringssystem (Work Process Oriented Administration System)
BPDM	Business Process Definition Metamodel
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMI	Business Process Management Initiative
BPMN	Business Process Modeling Notation
DAML	DARPA Agent Markup Language
DL	Description Logic
DLP	Description Logic Program
DNV	Det Norske Veritas
DPO	Daily Production Optimization
EOR	Enhanced Oil Recovery
G1/G2	Generation 1 / Generation 2 (in IO)
GAP	Groups, Algorithms and Programming
GUI	Graphical User Interface
IDE	Integrated Development Environment
IO	Integrated Operations
IP21	InfoPlus 21
IRC	Internet Relay Chat
ISO	International Standards Organization
N3	Notation3 (readable RDF syntax)
NCS	Norwegian Continental Shelf
NGL	Natural Gas Liquids
NOK	Norwegian Kroners
OIL	Ontology Inference Layer
OLF	Oljeindustriens Landsforening (The Norwegian Oil Industry Association)
OMG	Object Management Group
OWL	Web Ontology Language
OWL-AS	OWL Abstract Syntax
OWL-DL	OWL — Description Logic

PDP	Process Data Portal
POSC	Petrotechnical Open Standards Consortium
PRODML	Production Markup Language
RDF	Resource Description Framework
RDFS	RDF Schema
RuleML	Rule Markup Language
SBPM	Semantic Business Process Management
sBPMN	Semantic Business Process Modeling Notation
Sm3	Standard Cubic Meter
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SUPER	Semantics Utilised for Process management within and between EnteRprises
SWIG	Semantic Web Interest Group
SWRL	Semantic Web Rule Language
SWS	Semantic Web Services
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WITSML	Wellsite Information Transfer Standard Markup Language
WSDL	Web Services Description Language
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
XML	eXtensible Markup Language
XPDL	XML Process Definition Language
XSD	XML Schema Definition
XSLT	eXtensible Stylesheet Language Transformations
YAWL	Yet Another Workflow Language

Bibliography

Witold Abramowicz, Agata Filipowska, Monika Kaczmarek, and Tomasz Kaczmarek. Semantically enhanced Business Process Modelling Notation. In *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM)*, Innsbruck, Austria, April 2007.

Gustavo Alonso and Hans-Jörg Schek. Research Issues in Large Workflow Management Systems. In *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems*, Athens, Georgia, USA, May 1996.

Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, Second edition, March 2008. ISBN 978-0262012423.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: theory, implementation, and applications*. Cambridge University Press, September 2007. ISBN 978-0521876254.

Dave Beckett. RDF/XML Syntax Specification (Revised). W3C Recommendation, W3C (World Wide Web Consortium), February 2004. URL `http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/`.

Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, May 2001.

Dan Brickley and Ramanathan V. Guha. RDF vocabulary description language 1.0: RDF schema. W3C Recommendation, W3C (World Wide Web Consortium), February 2004. URL `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

Liliana Cabral, John Domingue, Enrico Motta, Terry R. Payne, and Farshad Hakimpour. Approaches to Semantic Web Services: an Overview and Comparisons. In *ESWS*, volume 3053, pages 225–239, 2004.

Jorge Cardoso. The Semantic Web Vision: Where Are We? *IEEE Intelligent Systems*, 22(5):84–88, 2007. ISSN 1541–1672. doi: 10.1109/MIS.2007.97.

Jeremy J. Carroll and Graham Klyne. Resource description framework (RDF): Concepts and abstract syntax. W3C Recommendation, W3C (World Wide Web Consortium), February 2004. URL http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: Implementing the Semantic Web Recommendations. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83, New York, NY, USA, 2004. ACM. ISBN 1-58113-912-8. doi: 10.1145/1013367.1013381.

James Clark. XSL Transformations (XSLT). W3C Recommendation, W3C (World Wide Web Consortium), November 1999. URL http://www.w3.org/TR/xslt.

Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 2001.

Simona Colucci, Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, Marina Mongiello, and Marco Mottola. A Formal Approach to Ontology-Based Semantic Match of Skills Descriptions. *Journal of Universal Computer Science*, 9(12):1437–1454, December 2003.

Simona Colucci, Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, Giacomo Piscitelli, and Stefano Coppi. Knowledge Based Approach to Semantic Composition of Teams in an Organization. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1314–1319, New York, NY, USA, 2005. ACM. ISBN 1-58113-964-0.

David Beckett. Turtle — Terse RDF Triple Language. Technical report, November 2007. URL http://www.dajobe.org/2004/01/turtle/. [Online; accessed 30-July-2008].

David Leal. ISO 15926 "Life Cycle Data for Process Plant": An Overview. *Oil & Gas Science and Technology*, 60(4):629–637, 2005.

Jos de Bruijn, Christoph Bussler, John Domingue, Dieter Fensel, Martin Hepp, Uwe Keller, Michael Kifer, Birgitta König-Ries, Jacek Kopecky, Rubén Lara, Holger Lausen, Eyal Oren, Axel Polleres, Dumitru Roman, James Scicluna, and Michael Stollberg. Web Service Modeling Ontology (WSMO). Member submission, W3C (World Wide Web Consortium), June 2005a. URL http://www.w3.org/Submission/2005/SUBM-WSMO-20050603/.

Jos de Bruijn, Dieter Fensel, Uwe Keller, Michael Kifer, Holger Lausen, Reto Krummenacher, Axel Polleres, and Livia Predoiu. Web Service

Modeling Language (WSML). Member Submisson, W3C (World Wide Web Consortium), June 2005b. URL http://www.w3.org/Submission/2005/SUBM-WSML-20050603/.

Anderson Santana de Oliveira. Rewriting-based access control policies. *Electron. Notes Theor. Comput. Sci.*, 171(4):59–72, 2007. ISSN 1571-0661. doi: 10.1016/j.entcs.2007.02.055.

Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in Description Logics. *Principles of Knowledge Representation*, pages 191–236, 1996. ISSN 1-57586-056-2.

Francisco Durán, Javier Herrador, and Antonio Vallecillo. Using UML and Maude for Writing and Reasoning about ODP Policies. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 15–25, Washington, DC, USA, October 2003. IEEE Computer Society. ISBN 0-7695-1933-4.

Lauri Eloranta, Eero Kallio, and Ilkka Terho. A Notation Evaluation of BPMN and UML Activity Diagrams. [Online; accessed 5-May-2008], 2006. URL http://www.soberit.hut.fi/T-86/T-86.5161/2006/BPMN_vs_UML_final.pdf.

Holger Endert, Tobias Küster, Benjamin Hirsch, and Sahin Albayrak. Mapping BPMN to Agents: An Analysis. In *Agent, Web Services, and Ontologies Integrated Methodologies*, pages 43–58, May 2007.

Energistics. PRODML Specifications, Version 1.0. Technical report, Energistics, November 2006. URL http://www.prodml.org/images/prodml/standards/v10/PRODMLV1.0_11172006.zip. [Online; accessed 2-June-2008].

Energistics. WITSML — Wellsite Information Transfer Standard Markup Language, 2008. URL http://www.witsml.org/.

Erlend Agøy Engum and Rolf Axel Wathne. Databaser for Dynamiske Arbeidsprosesser. Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet (NTNU), June 2004.

Roar A. Fjellheim, Reidar Bratvold, and Mike Herbert. CODIO — Collaborative Decisionmaking in Integrated Operations. In *Intelligent energy. Optimised efficency. Smarter business.*, Convertion Centre, Amsterdam RAI, The Netherlands, February 2008. Society of Petroleum Engineers.

Dimitrios Georgakopoulos, Mark F. Hornick, and Amit P. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, April 1995. ISSN 0926-8782. doi: 10.1007/BF01277643.

Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs With Description Logic. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 48–57. ACM, May 2003. ISBN 1-58113-680-3. doi: 10.1145/775152.775160.

Thomas R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal Human-Computer Studies*, 43 (5–6):907–928, November 1995. ISSN 1071-5819. doi: 10.1006/ijhc.1995.1081.

Jon Atle Gulla, Darijus Strasunskas, and Stein L. Tomassen. Semantic Interoperability in the Norwegian Petroleum Industry. *Proceedings of the 5th International Conference on Information Systems Technology and its Applications (ISTA 2006)*, P-84:81–94, 2006.

Martin Hepp and Dumitru Roman. An Ontology Framework for Semantic Business Process Management. In *Proceedings of the 8th International Conference Wirtschaftsinformatik 2007*. Universitaetsverlag Karlsruhe, February 2007. URL `http://www.heppnetz.de/files/hepp-roman-an-ontology-framework-for-SBPM-WI2007.pdf`.

Martin Hepp, Frank Leymann, John Domingue, Alexander Wahler, and Dieter Fensel. Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In *ICEBE '05: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 535–540, Washington DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2430-3. doi: 10.1109/ICEBE.2005.110.

Holger Knublauch. Semantic Web Rules; Tools and Languages — Jena Rules Walkthrough. Rule and Rule Markup Languages for the Semantic Web, Second International Coference, November 2006. URL `http://2006.ruleml.org/slides/tutorial-holger.pdf`. [Online; accessed 25-july-2008].

Bjørn Holst and Espen Nystad. Oil & Gas offshore/onshore Integrated Operations - introducing the Brage 2010+ project. In *Human Factors and Power Plants and HPRCT 13th Annual Meeting, 2007 IEEE 8th*, pages 357–359, August 2007. doi: 10.1109/HFPP.2007.4413233.

Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A practical guide to building OWL ontologies using the Protege-OWL plugin and CO-ODE tools Edition 1.0, 2004. URL `http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf`. [Online; accessed 13-May-2007].

Ian Horrocks. DAML+OIL: a Description Logic for the Semantic Web. *IEEE Data Engineering Bulletin*, 25(1):4–9, 2002.

Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, May 2004. URL http://www.w3.org/Submission/SWRL/.

Hydro. Hydros APOS (Arbeids Prosess Orientert Styring) documentation, January 2008. [Workflow diagrams, descriptions, documents, etc. in the SoluDyne system. Not publicly available].

ISO. ISO 15926: Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 1: Overview and fundamental principles. Technical report, The International Standardization Organization, July 2004. URL http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm%?csnumber=29556.

ISO. ISO 15926: Industrial automation systems and integration–Integration of life-cycle data for oil and gas production facilities–Part2: Data Model. Technical report, The International Standardization Organization, 2003. URL http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm%?csnumber=29557.

ISO. ISO 15926: Industrial automation systems and integration – Integration of life-cycle data for process plants including oil and gas production facilities – Part 4: Initial reference data . Technical report, The International Standardization Organization, 2007. URL http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm%?csnumber=41329.

Jae-Yoon Jung and Hoontae Kim and Suk-Ho Kang. Standards-based approaches to B2B workflow integration. *Computers & Industrial Engineering*, 51:321–334, October 2006. URL http://www.sciencedirect.com/science/article/B6V27-4KVXHGT-2/1/c7470c49eeb4d1cba3e6ea8126c8c0d4.

Jean-Jacques Dubray. BPMN 1.0 Meta Model, October 2007. URL http://www.wsper.org/bpmn10.html. [Online; accessed 19-November-2007.

Uwe Keller, Cristina Feier, Nathalie Steinmetz, and Holger Lausen. Report on reasoning techniques and prototype implementation for the WSML-Core and WSML-DL languages. *RW*^{2} *Project Deliverable*, July 2006.

Michel Klein, Dieter Fensel, Frank van Harmelen, and Ian Horrocks. The Relation Between Ontologies and XML Schemas. *Electronic Transactions on Artificial Intelligence. Special Issue on the 1st International Workshop "Semantic Web: Models, Architectures and Management"*, 5:65–94, October 2001.

Weiqiang Kong, Kazuhiro Ogata, and Kokichi Futatsugi. Formal Analysis of

Workflow Systems with Security Considerations. In *Seventeenth International Conference on Software Engineering and Knowledge Engineering*, pages 531–536, Howard International House,Taipei, Taiwan, Republic of China, July 2005.

Philip A. Laplante. *Dictionary of Computer Science, Engineering, and Technology*. CRC Press, December 2000. ISBN 978-0849326912.

Eve Maler, Jean Paoli, C. M. Sperberg-McQueen, François Yergeau, and Tim Bray. Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation, W3C (World Wide Web Consortium), August 2006. URL `http://www.w3.org/TR/2006/REC-xml-20060816`.

Robin Milner. *Communicating and Mobile Systems: The Pi Calculus*. Cambridge University Press, June 1999. ISBN 978-0521658690.

Ministry of Petroleum and Energy. Stortingsmelding 38 — del 3.3, 2007a. URL `http://www.regjeringen.no/nb/dep/oed/dok/regpubl/stmeld/20032004/Stmeld-nr-38-2003-2004-/3/3.html?id=404862`. [Online; accessed 21-February-2008].

Ministry of Petroleum and Energy. Beskrivelse av det sammenslåtte selskapet, 2007b. URL `http://www.regjeringen.no/nb/dep/oed/dok/regpubl/stprp/20062007/Stprp-nr-60-2006-2007-/6.html?id=462001`. [Online; accessed 29-January-2008].

Rick Morneau. <PRODML/> PRODuction xML Standard Overview, March 2007. URL `http://www.energistics.org/images/posc/meetings/mar07annual/mar07annual_prodml.ppt`. [Online; accessed 9-May-2008].

David Norheim and Roar A. Fjellheim. AKSIO — Active Knowledge Management in the Petroleum Industry. In *European Semantic Web Conference (ESWC)*, Budova, Montenegro, June 2006. URL `http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-194/paper3.pdf`.

Natalya F. Noy and Deborah L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology, March 2001. URL `http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html`. [Online; accessed 22-December-2007].

Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubezy, Ray W. Fergerson, and Mark A. Musen. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001. ISSN 1541-1672. doi: 10.1109/5254.920601.

OLF. Integrated Work Prcess: Future work processes on the NCS. Technical report, OLF (Oljeindustriens Landsforening), October 2005. URL `http:`

//www.olf.no/io/rapporter/?51638.pdf. [Online; accessed 17-September-2007].

OLF. Verdipotensialet for Integrerte operasjoner på Norsk Sokkel. Technical report, OLF (Oljeindustriens Landsforening), April 2006. URL http://www.olf.no/io/aktuelt/?32756.pdf. [Online; accessed 17-September-2007].

OLF. Oppdatering av verdipotensialet i IO. Technical report, OLF (Oljeindustriens Landsforening), December 2007. URL http://www.olf.no/io/rapporter/?52498.pdf. [Online; accessed 17-June-2008].

OMG. Business Process Modeling Notation (BPMN) Information, FAQ, 2005. URL http://www.bpmn.org/Documents/FAQ.htm. [Online; accessed 13-May-2008].

OMG. Business Process Modeling Notation Specification 1.1 - DRAFT. Technical report, OMG (Object Management Group, Inc.), July 2007. [Currently not publicly available].

OMG. BPMN Information Home, January 2008. URL http://www.bpmn.org/. [Online; accessed 26-May-2008].

OMG. Final Adopted BPMN 1.0 Specification. Technical report, OMG (Object Management Group, Inc.), February 2006. URL http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf.

Chun Ouyang, Will M.P. van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede. Translating BPMN to BPEL. [Online; accessed 2-May-2008], 2006. URL http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2006/BPM-06-02.pdf.

Patrick Hayes Peter F. Patel-Schneider and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation, W3C (World Wide Web Consortium), February 2004. URL http://www.w3.org/TR/owl-semantics/.

H. Sofia Pinto and João Pavão Martins. Reusing Ontologies. In *Proceedings of the AAAI2000 Spring Symposium Series, Workshop Bringing Knowledge to Business Processes*, pages 77–84. AAAI Press, March 2000.

Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. W3C Recommendation, W3C (World Wide Web Consortium), January 2008. URL http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

Bjørn Rasen. Time for integrated action, March 2008. URL

http://www.npd.no/English/Emner/IO/IO-forum/11.03.08+time+for+integrated+action.htm. [Online; accessed 7-July-2008].

Jan C. Recker and Jan Mendling:. On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. In Thibaud Latour and Michaël Petit, editors, *CAiSE 2006 Workshop Proceedings - Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD)*, pages 521–532, June 2006. ISBN 2870375255. URL http://wi.wu-wien.ac.at/home/mendling/publications/06-EMMSAD.pdf.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003. ISBN 0-13-080302-2.

Davide Sangiorgi and David Walker. *The* Π *– calculus A Theory of Mobile Processes*. Cambridge University Press, October 2003. ISBN 978-0521543279.

Ulrike Sattler, Frank Wolter, and Ian Horrocks. Composing and decomposing ontologies: a logic-based approach—A Case for Support. [Online; accessed 5-June-2008], December 2006a. URL http://www.csc.liv.ac.uk/~dirk/modularity/papers/conservative-epsrc-proposal.pdf.

Ulrike Sattler, Frank Wolter, Ian Horrocks, Borris Konev, and Bijan Parsia. Composing and decomposing ontologies: a logic-based approach — A Case for Support, 2006b. URL http://www.csc.liv.ac.uk/~dirk/modularity/papers/conservative-epsrc-proposal.pdf. [Online; accessed 12-March-2008].

Daniel Schall, Hong-Linh Truong, and Schahram Dustdar. Unifying Human and Software Services in Web-Scale Collaborations. *IEEE Internet Computing*, 12(3):62–68, 2008. ISSN 1089-7801. doi: 10.1109/MIC.2008.66.

Harold M. Schroder, Michael J. Driver, and Siegfried Streufert. *Human Information Processing: Individuals and Groups Functioning in Complex Social Situations*. Holt, Rinehart & Winston, Inc., 1967.

Dave Shipley, Ben Weltevrede, Alan Doniger, Hans Eric Klumpen, and Laurence Ormerod. Production Data Standards: The PRODML Business Case and Evolution. In *Intelligent Energy 2008 conference, Amsterdam, The Neherlands, 25-7 February 2008*. Society of Petroleum Engineers, February 2008.

Bruce Silver. What if BPMN Were a Modeling Language?, 2007. URL http://www.brsilver.com/wordpress/2007/04/30/what-if-bpmn-were-a-modeling-language/. [Online; accessed 22-October-2007].

Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007. ISSN 1570-8268. doi: 10.1016/j.websem.2007.03.004.

Peter Spyns, Robert Meersman, and Mustafa Jarrar. Data modelling versus Ontology engineering. *SIGMOD Record*, 31(4):12–17, 2002. ISSN 0163-5808. doi: 10.1145/637411.637413.

StatoilHydro. StatoilHydros Årsrapport for 2007, April 2008. URL `http://www.statoilhydro.com/no/NewsAndMedia/News/2008/Pages/2007report.aspx`. [Also available online].

Stephen A. White. Introduction to BPMN. Technical report, OMG (Object Management Group, Inc.), May 2004. URL `http://www.bpmn.org/Documents/Introduction%20to%20BPMN.pdf`. [Online; accessed 12-November-2007].

Darijus Strasunskas and Stein L. Tomassen. Scenario-Driven Information Retrieval: Supporting Rule-Based Monitoring of Subsea Operations. *Information Technology and Control*, 36(1A):87–92, 2007.

Hans Teijgeler. The Process Industries and the ISO 15926 Semantic Web. [Online - Updated 12 March 2008-edition; accessed 23-May-2008], August 2007. URL `http://www.infowebml.ws/Topics/papers/15926SW.htm`.

Tim Berners-Lee. Notation3 (N3) A readable RDF syntax. Technical report, W3C (World Wide Web Consortium), March 2006. URL `http://www.w3.org/DesignIssues/Notation3`.

Magne Valen-Sendstad. POSC Caesar/EPISTLE/ISO 15926 presentation at Det Norske Veritas 14 February 2008, February 2008. [Not publicly available].

Frank van Harmelen and Deborah L. McGuinness. OWL Web Ontology Language Overview. W3C Recommendation, W3C (World Wide Web Consortium), February 2004. URL `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

Wikipedia. Business process management — wikipedia, the free encyclopedia, 2008a. URL `http://en.wikipedia.org/w/index.php?title=Business_process_management&oldid=191665516`. [Online; accessed 24-February-2008].

Wikipedia. Upper ontology (information science) — wikipedia, the free encyclopedia, 2008b. URL `http://en.wikipedia.org/w/index.php?title=Upper_ontology_%28information_science%29&oldid=210976563`. [Online; accessed 24-July-2008].

Wikipedia. Business process modeling notation — wikipedia, the free encyclopedia, 2007. URL http://en.wikipedia.org/w/index.php?title=Business_Process_Modeling_Notation&oldid=158972395. [Online; accessed 26-September-2007].

Petia Wohed, Wil M.P. van der Aals, Marlon Dumas, Arthur H.M. ter Hofstede, and Nick Russell. *Lecture Notes in Computer Science*, volume 4102/2006, chapter On the Suitability of BPMN for Business Process Modelling, pages 161–176. Springer, 2006.

Peter Y.H. Wong and Jeremy Gibbons. A Process Semantics for BPMN. Submitted for publication. Extended version available online, 2008. URL http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmnsem.pdf.

YAWL Foundation. YAWL - Yet Another Workflow Language, 2007. URL http://www.yawlfoundation.org/theory/index.php. [Online; accessed 13-May-2008].

Øystein Fossen. Integrerte operasjoner — akselerert læring; Høydepunkter fra pågående OLF prosjekt, Stavanger 12. juni. OLFs konferanse om integrerte operasjoner 12. og 13. juni 2007, June 2007. URL http://www.olf.no/getfile.php/Konvertert/www.olf.no/Aktuelt/Dokumenter/OLFs%20IO-konferanse%202007%20-%20%C3%98ystein%20Fossen%2C%20Arbforskinst.pdf. [Online; accessed 25-july-2008].

Appendix

Appendix A

The SemTask OWL Ontologies

A.1 The BPMN Ontology

Listing A.1: *The BPMN Ontology in OWL-AS*

```
Namespace(rdf   = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Namespace(owl   = <http://www.w3.org/2002/07/owl#>)
Namespace(xsd   = <http://www.w3.org/2001/XMLSchema#>)
Namespace(rdfs  = <http://www.w3.org/2000/01/rdf-schema#>)
Namespace(bpmn  = <http://www.ifi.uio.no/fredaleks/semtask/bpmn#>)

Ontology( <http://www.ifi.uio.no/fredaleks/semtask/bpmn>

 ObjectProperty(bpmn:bpmnObjectProperty)
 ObjectProperty(bpmn:connected
  inverseOf(bpmn:hasConnecting)
  domain(bpmn:Connecting)
  range(unionOf(bpmn:Flow bpmn:Artifact)))
 ObjectProperty(bpmn:connectedFrom
  inverseOf(bpmn:hasConnectingFrom)
  domain(bpmn:Connecting))
 ObjectProperty(bpmn:connectedTo Functional
  inverseOf(bpmn:hasConnectingTo)
  domain(bpmn:Connecting))
 ObjectProperty(bpmn:hasActivityTaskType Functional
  domain(unionOf(owl:Thing bpmn:Activity))
  range(bpmn:ActivityTaskType))
 ObjectProperty(bpmn:hasConnecting
  inverseOf(bpmn:connected)
```

```
  domain(unionOf(bpmn:Flow bpmn:Artifact))
  range(bpmn:Connecting))
ObjectProperty(bpmn:hasConnectingFrom
  inverseOf(bpmn:connectedFrom))
ObjectProperty(bpmn:hasConnectingTo InverseFunctional
  inverseOf(bpmn:connectedTo))
ObjectProperty(bpmn:hasLane InverseFunctional
  inverseOf(bpmn:isInPool)
  domain(bpmn:Pool)
  range(bpmn:Lane))
ObjectProperty(bpmn:hasObject InverseFunctional
  inverseOf(bpmn:isInLane)
  domain(bpmn:Lane)
  range(unionOf(bpmn:Connecting bpmn:Flow)))
ObjectProperty(bpmn:hasStartEvent Functional ↩
    InverseFunctional
  domain(bpmn:SubProcess)
  range(bpmn:StartEvent))
ObjectProperty(bpmn:isInLane Functional
  inverseOf(bpmn:hasObject)
  domain(unionOf(bpmn:Connecting bpmn:Flow))
  range(bpmn:Lane))
ObjectProperty(bpmn:isInPool Functional
  inverseOf(bpmn:hasLane)
  domain(bpmn:Lane)
  range(bpmn:Pool))

DatatypeProperty(bpmn:bpmnDatatypeProperty)
DatatypeProperty(bpmn:hasCondition Functional
  domain(unionOf(bpmn:ConditionalSequenceFlow ↩
      bpmn:NormalSequenceFlow))
  range(xsd:string))
DatatypeProperty(bpmn:hasMessage Functional
  domain(unionOf(bpmn:MessageFlow bpmn:NormalSequenceFlow)↩
      )
  range(xsd:string))
DatatypeProperty(bpmn:hasName Functional
  domain(bpmn:BPMNElement)
  range(xsd:string))

Class(bpmn:Activity partial
  bpmn:Flow)
Class(bpmn:ActivityTaskType partial
  bpmn:BPMNElement)
EnumeratedClass(bpmn:ActivityTaskType bpmn:None ↩
    bpmn:Manual bpmn:User bpmn:Receive bpmn:Script ↩
    bpmn:Service bpmn:Send bpmn:Reference)
Class(bpmn:AdHocActivity partial
  bpmn:Activity)
```

```
Class(bpmn:AdHocSubProcess partial
 bpmn:AdHocActivity
 bpmn:SubProcess)
Class(bpmn:Annotation partial
 bpmn:Artifact)
Class(bpmn:Artifact partial
 bpmn:BPMNElement)
Class(bpmn:Association partial
 bpmn:Connecting)
Class(bpmn:BPMNElement partial)
Class(bpmn:CancelEndEvent partial
 bpmn:EndEvent
 bpmn:CancelEvent)
Class(bpmn:CancelEvent partial
 bpmn:Event)
Class(bpmn:CancelIntermediateEvent partial
 bpmn:IntermediateEvent
 bpmn:CancelEvent)
Class(bpmn:ComensationIntermediateEvent partial
 bpmn:CompensationEvent
 bpmn:IntermediateEvent)
Class(bpmn:CompensationActivity partial
 bpmn:Activity)
Class(bpmn:CompensationEndEvent partial
 bpmn:CompensationEvent
 bpmn:EndEvent)
Class(bpmn:CompensationEvent partial
 bpmn:Event)
Class(bpmn:CompensationSubProcess partial
 bpmn:SubProcess
 bpmn:CompensationActivity)
Class(bpmn:CompensationTask partial
 bpmn:CompensationActivity
 bpmn:Task)
Class(bpmn:ConditionalSequenceFlow partial
 bpmn:SequenceFlow)
Class(bpmn:Connecting complete
 unionOf(bpmn:Association bpmn:SequenceFlow ↩
     bpmn:MessageFlow))
Class(bpmn:Connecting partial
 bpmn:BPMNElement
 restriction(bpmn:connectedTo maxCardinality(1))
 restriction(bpmn:connected cardinality(2)))
Class(bpmn:DataBasedXorMergeGateway partial
 bpmn:XorMergeGateway)
Class(bpmn:DataObject partial
 bpmn:Artifact)
Class(bpmn:DefaultSequenceFlow partial
 bpmn:SequenceFlow)
```

```
Class(bpmn:EndEvent partial
 bpmn:Event)
Class(bpmn:ErrorEndEvent partial
 bpmn:ErrorEvent
 bpmn:EndEvent)
Class(bpmn:ErrorEvent partial
 bpmn:Event)
Class(bpmn:ErrorIntermediateEvent partial
 bpmn:ErrorEvent
 bpmn:IntermediateEvent)
Class(bpmn:Event partial
 bpmn:Flow)
Class(bpmn:EventBasedXorMergeGateway partial
 bpmn:XorMergeGateway)
Class(bpmn:ExceptionEndEvent partial
 bpmn:EndEvent
 bpmn:ExceptionEvent)
Class(bpmn:ExceptionEvent partial
 bpmn:Event)
Class(bpmn:ExceptionIntermediateEvent partial
 bpmn:IntermediateEvent
 bpmn:ExceptionEvent)
Class(bpmn:Flow complete
 intersectionOf(restriction(bpmn:isInLane cardinality(1))
     unionOf(bpmn:Event bpmn:Gateway bpmn:Activity)))
Class(bpmn:Flow partial
 bpmn:BPMNElement)
Class(bpmn:Gateway partial
 bpmn:Flow)
Class(bpmn:Group partial
 bpmn:Artifact)
Class(bpmn:Grouping complete
 unionOf(bpmn:Pool bpmn:Lane))
Class(bpmn:Grouping partial
 bpmn:BPMNElement)
Class(bpmn:IntermediateEvent partial
 bpmn:Event)
Class(bpmn:Lane partial
 bpmn:Grouping
 restriction(bpmn:isInPool cardinality(1)))
Class(bpmn:LinkEndEvent partial
 bpmn:EndEvent
 bpmn:LinkEvent)
Class(bpmn:LinkEvent partial
 bpmn:Event)
Class(bpmn:LinkIntermediateEvent partial
 bpmn:IntermediateEvent
 bpmn:LinkEvent)
Class(bpmn:LinkStartEvent partial
```

```
  bpmn:StartEvent
  bpmn:LinkEvent)
 Class(bpmn:LoopedActivity partial
  bpmn:Activity)
 Class(bpmn:LoopedSubProcess partial
  bpmn:LoopedActivity
  bpmn:SubProcess)
 Class(bpmn:LoopedTask partial
  bpmn:LoopedActivity
  bpmn:Task)
 Class(bpmn:MergeGateway partial
  bpmn:Gateway)
 Class(bpmn:MessageEndEvent partial
  bpmn:EndEvent
  bpmn:MessageEvent)
 Class(bpmn:MessageEvent partial
  bpmn:Event)
 Class(bpmn:MessageFlow partial
  restriction(bpmn:connectedFrom cardinality(1))
  bpmn:Connecting
  restriction(bpmn:connectedTo cardinality(1)))
 Class(bpmn:MessageIntermediateEvent partial
  bpmn:IntermediateEvent
  bpmn:MessageEvent)
 Class(bpmn:MessageStartEvent partial
  bpmn:StartEvent
  bpmn:MessageEvent)
 Class(bpmn:MultipleEndEvent partial
  bpmn:MultipleEvent
  bpmn:EndEvent)
 Class(bpmn:MultipleEvent partial
  bpmn:Event)
 Class(bpmn:MultipleInstanceActivity partial
  bpmn:Activity)
 Class(bpmn:MultipleInstanceSubProcess partial
  bpmn:SubProcess
  bpmn:MultipleInstanceActivity)
 Class(bpmn:MultipleInstanceTask partial
  bpmn:MultipleInstanceActivity
  bpmn:Task)
 Class(bpmn:MultipleIntermediateEvent partial
  bpmn:IntermediateEvent
  bpmn:MultipleEvent)
 Class(bpmn:MutipleStartEvent partial
  bpmn:MultipleEvent
  bpmn:StartEvent)
 Class(bpmn:NormalSequenceFlow partial
  bpmn:SequenceFlow)
 Class(bpmn:OrMergeGateway partial
```

```
 bpmn:Gateway)
Class(bpmn:ParalellForkGateway complete
 intersectionOf(restriction(bpmn:hasConnectingFrom ↩
     minCardinality(2)) bpmn:ParalellForkOrJoinGateway))
Class(bpmn:ParalellForkOrJoinGateway partial
 bpmn:Gateway)
Class(bpmn:ParalellJoinGateway complete
 intersectionOf(restriction(bpmn:hasConnectingTo ↩
     minCardinality(2)) bpmn:ParalellForkOrJoinGateway))
Class(bpmn:Pool partial
 bpmn:Grouping
 restriction(bpmn:hasLane minCardinality(1)))
Class(bpmn:RuleEndEvent partial
 bpmn:EndEvent
 bpmn:RuleEvent)
Class(bpmn:RuleEvent partial
 bpmn:Event)
Class(bpmn:RuleStartEvent partial
 bpmn:StartEvent
 bpmn:RuleEvent)
Class(bpmn:SequenceFlow partial
 restriction(bpmn:connectedFrom cardinality(1))
 bpmn:Connecting
 restriction(bpmn:connectedTo cardinality(1)))
Class(bpmn:StartEvent partial
 bpmn:Event)
Class(bpmn:SubProcess partial
 restriction(bpmn:hasStartEvent cardinality(1))
 bpmn:Activity)
Class(bpmn:Task partial
 restriction(bpmn:hasActivityTaskType someValuesFrom(↩
     bpmn:ActivityTaskType))
 bpmn:Activity)
Class(bpmn:TerminateEndEvent partial
 bpmn:TerminateEvent
 bpmn:EndEvent)
Class(bpmn:TerminateEvent partial
 bpmn:Event)
Class(bpmn:TimerEvent partial
 bpmn:Event)
Class(bpmn:TimerIntermediateEvent partial
 bpmn:IntermediateEvent
 bpmn:TimerEvent)
Class(bpmn:TimerStartEvent partial
 bpmn:TimerEvent
 bpmn:StartEvent)
Class(bpmn:XorMergeGateway partial
 bpmn:Gateway)
Class(owl:Thing partial)
```

```
Individual(bpmn:Manual
 type(bpmn:ActivityTaskType))
Individual(bpmn:None
 type(bpmn:ActivityTaskType))
Individual(bpmn:Receive
 type(bpmn:ActivityTaskType))
Individual(bpmn:Reference
 type(bpmn:ActivityTaskType))
Individual(bpmn:Script
 type(bpmn:ActivityTaskType))
Individual(bpmn:Send
 type(bpmn:ActivityTaskType))
Individual(bpmn:Service
 type(bpmn:ActivityTaskType))
Individual(bpmn:User
 type(bpmn:ActivityTaskType))

Datatype(xsd:string)

DisjointClasses(bpmn:TimerEvent bpmn:ExceptionEvent)
DisjointClasses(bpmn:MultipleEvent bpmn:TimerEvent)
DisjointClasses(bpmn:CompensationEvent ↩
    bpmn:TerminateEvent)
DisjointClasses(bpmn:TimerEvent bpmn:LinkEvent)
DisjointClasses(bpmn:Event bpmn:Gateway)
DisjointClasses(bpmn:XorMergeGateway ↩
    bpmn:ParalellForkOrJoinGateway)
DisjointClasses(bpmn:IntermediateEvent bpmn:StartEvent)
DisjointClasses(bpmn:MergeGateway ↩
    bpmn:ParalellForkOrJoinGateway)
DisjointClasses(bpmn:CancelEvent bpmn:ExceptionEvent)
DisjointClasses(bpmn:MultipleEvent bpmn:CancelEvent)
DisjointClasses(bpmn:TerminateEvent bpmn:ExceptionEvent)
DisjointClasses(bpmn:MultipleEvent bpmn:TerminateEvent)
DisjointClasses(bpmn:DataBasedXorMergeGateway ↩
    bpmn:EventBasedXorMergeGateway)
DisjointClasses(bpmn:Pool bpmn:Lane)
DisjointClasses(bpmn:LinkEvent bpmn:CancelEvent)
DisjointClasses(bpmn:MessageEvent bpmn:RuleEvent)
DisjointClasses(bpmn:RuleEvent bpmn:CancelEvent)
DisjointClasses(bpmn:CompensationEvent bpmn:MessageEvent)
DisjointClasses(bpmn:TerminateEvent bpmn:LinkEvent)
DisjointClasses(bpmn:TimerEvent bpmn:CancelEvent)
DisjointClasses(bpmn:Connecting bpmn:Grouping)
DisjointClasses(bpmn:AdHocActivity ↩
    bpmn:CompensationActivity)
DisjointClasses(bpmn:MergeGateway bpmn:XorMergeGateway)
DisjointClasses(bpmn:LinkEvent bpmn:ExceptionEvent)
```

```
DisjointClasses(bpmn:TimerEvent bpmn:TerminateEvent)
DisjointClasses(bpmn:Connecting bpmn:Artifact)
DisjointClasses(bpmn:DataObject bpmn:Annotation)
DisjointClasses(bpmn:Grouping bpmn:Artifact)
DisjointClasses(bpmn:MultipleInstanceActivity ↩
    bpmn:CompensationActivity)
DisjointClasses(bpmn:CompensationEvent bpmn:RuleEvent)
DisjointClasses(bpmn:MultipleEvent bpmn:LinkEvent)
DisjointClasses(bpmn:MessageEvent bpmn:ExceptionEvent)
DisjointClasses(bpmn:MultipleEvent bpmn:MessageEvent)
DisjointClasses(bpmn:Connecting bpmn:Flow)
DisjointClasses(bpmn:ConditionalSequenceFlow ↩
    bpmn:NormalSequenceFlow)
DisjointClasses(bpmn:TerminateEvent bpmn:CancelEvent)
DisjointClasses(bpmn:StartEvent bpmn:EndEvent)
DisjointClasses(bpmn:AdHocActivity ↩
    bpmn:MultipleInstanceActivity)
DisjointClasses(bpmn:MessageEvent bpmn:LinkEvent)
DisjointClasses(bpmn:Flow bpmn:Grouping)
DisjointClasses(bpmn:Association bpmn:MessageFlow)
DisjointClasses(bpmn:SequenceFlow bpmn:MessageFlow)
DisjointClasses(bpmn:NormalSequenceFlow ↩
    bpmn:DefaultSequenceFlow)
DisjointClasses(bpmn:TimerEvent bpmn:MessageEvent)
DisjointClasses(bpmn:OrMergeGateway ↩
    bpmn:ParalellForkOrJoinGateway)
DisjointClasses(bpmn:Flow bpmn:Artifact)
DisjointClasses(bpmn:Event bpmn:Activity)
DisjointClasses(bpmn:LoopedActivity ↩
    bpmn:CompensationActivity)
DisjointClasses(bpmn:Flow bpmn:ActivityTaskType)
DisjointClasses(bpmn:ConditionalSequenceFlow ↩
    bpmn:DefaultSequenceFlow)
DisjointClasses(bpmn:RuleEvent bpmn:ExceptionEvent)
DisjointClasses(bpmn:MultipleEvent bpmn:RuleEvent)
DisjointClasses(bpmn:CompensationEvent ↩
    bpmn:ExceptionEvent)
DisjointClasses(bpmn:Association bpmn:SequenceFlow)
DisjointClasses(bpmn:CompensationEvent bpmn:MultipleEvent ↩
    )
DisjointClasses(bpmn:LoopedActivity bpmn:AdHocActivity)
DisjointClasses(bpmn:LinkEvent bpmn:RuleEvent)
DisjointClasses(bpmn:Group bpmn:Annotation)
DisjointClasses(bpmn:CompensationEvent bpmn:CancelEvent)
DisjointClasses(bpmn:Gateway bpmn:Activity)
DisjointClasses(bpmn:IntermediateEvent bpmn:EndEvent)
DisjointClasses(bpmn:CompensationEvent bpmn:LinkEvent)
DisjointClasses(bpmn:XorMergeGateway bpmn:OrMergeGateway)
DisjointClasses(bpmn:MessageEvent bpmn:CancelEvent)
```

```
DisjointClasses(bpmn:TimerEvent bpmn:RuleEvent)
DisjointClasses(bpmn:CompensationEvent bpmn:TimerEvent)
DisjointClasses(bpmn:TerminateEvent bpmn:MessageEvent)
DisjointClasses(bpmn:MergeGateway bpmn:OrMergeGateway)
DisjointClasses(bpmn:TerminateEvent bpmn:RuleEvent)
DisjointClasses(bpmn:SubProcess bpmn:Task)
DisjointClasses(bpmn:LoopedActivity ➲
    bpmn:MultipleInstanceActivity)
DisjointClasses(bpmn:MultipleEvent bpmn:ExceptionEvent)
DisjointClasses(bpmn:DataObject bpmn:Group)
EquivalentClasses(unionOf(bpmn:ConditionalSequenceFlow ➲
    bpmn:NormalSequenceFlow))

SubPropertyOf(bpmn:hasConnectingTo bpmn:hasConnecting)
SubPropertyOf(bpmn:connectedTo bpmn:connected)
SubPropertyOf(bpmn:hasName bpmn:bpmnDatatypeProperty)
SubPropertyOf(bpmn:isInLane bpmn:bpmnObjectProperty)
SubPropertyOf(bpmn:connected bpmn:bpmnObjectProperty)
SubPropertyOf(bpmn:hasLane bpmn:bpmnObjectProperty)
SubPropertyOf(bpmn:hasActivityTaskType ➲
    bpmn:bpmnObjectProperty)
SubPropertyOf(bpmn:hasConnecting bpmn:bpmnObjectProperty)
SubPropertyOf(bpmn:hasCondition bpmn:bpmnDatatypeProperty➲
    )
SubPropertyOf(bpmn:hasStartEvent bpmn:bpmnObjectProperty)
SubPropertyOf(bpmn:connectedFrom bpmn:connected)
SubPropertyOf(bpmn:hasConnectingFrom bpmn:hasConnecting)
SubPropertyOf(bpmn:isInPool bpmn:bpmnObjectProperty)
SubPropertyOf(bpmn:hasMessage bpmn:bpmnDatatypeProperty)
SubPropertyOf(bpmn:hasObject bpmn:bpmnObjectProperty)

DifferentIndividuals(bpmn:None bpmn:Manual bpmn:User ➲
    bpmn:Receive bpmn:Script bpmn:Service bpmn:Send ➲
    bpmn:Reference)

)
```

A.2 The SemTask Ontology

Listing A.2: *The Semantic Task Support Ontology in OWL-AS*

```
Namespace(rdf          = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Namespace(owl          = <http://www.w3.org/2002/07/owl#>)
Namespace(xsd          = <http://www.w3.org/2001/XMLSchema#>)
Namespace(rdfs         = <http://www.w3.org/2000/01/rdf-schema#>)
Namespace(bpmn         = <http://www.ifi.uio.no/fredaleks/semtask/bpmn#>)
Namespace(semtask      = <http://www.ifi.uio.no/fredaleks/semtask/semtask#>)

Ontology( <http://www.ifi.uio.no/fredaleks/semtask/semtask>

 Annotation( owl:imports <http://www.ifi.uio.no/fredaleks/semtask/bpmn>)


 ObjectProperty(bpmn:hasConnectingFrom)
 ObjectProperty(semtask:hasField InverseFunctional
  inverseOf(semtask:isInForm)
  domain(semtask:FormDocument)
  range(semtask:Field))
 ObjectProperty(semtask:isInForm Functional
  inverseOf(semtask:hasField)
  domain(semtask:Field)
  range(semtask:FormDocument))
 ObjectProperty(semtask:semtaskObjectProperty)
 ObjectProperty(rdf:rest)
 ObjectProperty(owl:oneOf)

 DatatypeProperty(semtask:hasName Functional
  domain(semtask:Field)
  range(xsd:string))
 DatatypeProperty(semtask:isBeginningEvent Functional
  domain(bpmn:StartEvent)
  range(xsd:boolean))
 DatatypeProperty(semtask:programParameters Functional
  domain(semtask:ApplicationDocument)
  range(xsd:string))
 DatatypeProperty(semtask:semtaskDatatypeProperty)
 DatatypeProperty(semtask:url Functional
```

```
 domain(unionOf(semtask:ApplicationDocument ↩
     semtask:URLBasedDocument))
 range(xsd:string))
DatatypeProperty(rdf:first)

Class(bpmn:Activity complete
 unionOf(bpmn:SubProcess bpmn:Task))
Class(bpmn:ConditionalSequenceFlow partial)
Class(bpmn:DataObject partial)
Class(bpmn:DefaultSequenceFlow partial)
Class(bpmn:Gateway partial)
Class(bpmn:NormalSequenceFlow partial)
Class(bpmn:SequenceFlow complete
 unionOf(bpmn:ConditionalSequenceFlow ↩
     bpmn:NormalSequenceFlow bpmn:DefaultSequenceFlow))
Class(bpmn:StartEvent partial)
Class(bpmn:SubProcess partial)
Class(bpmn:Task partial)
Class(semtask:ApplicationDocument partial
 restriction(semtask:programParameters minCardinality(1))
 semtask:Document
 restriction(semtask:url minCardinality(1)))
Class(semtask:DecisionGateway complete
 intersectionOf(bpmn:Gateway restriction(↩
     bpmn:hasConnectingFrom allValuesFrom(↩
     bpmn:ConditionalSequenceFlow)) restriction(↩
     bpmn:hasConnectingFrom minCardinality(2))))
Class(semtask:DecisionGateway partial
 semtask:SemTaskElement)
Class(semtask:Document complete
 unionOf(semtask:ApplicationDocument semtask:FormDocument↩
      semtask:URLBasedDocument))
Class(semtask:Document partial
 semtask:SemTaskElement
 bpmn:DataObject)
Class(semtask:Field partial
 semtask:SemTaskElement)
Class(semtask:FormDocument partial
 semtask:Document)
Class(semtask:SemTaskElement partial)
Class(semtask:URLBasedDocument partial
 semtask:Document
 restriction(semtask:url minCardinality(1)))
Class(owl:Thing partial)

Datatype(xsd:boolean)
Datatype(xsd:string)

DisjointClasses(semtask:DecisionGateway semtask:Field)
```

```
  DisjointClasses(semtask:ApplicationDocument ↩
      semtask:URLBasedDocument)
  DisjointClasses(semtask:Document semtask:DecisionGateway)
  DisjointClasses(semtask:Document semtask:Field)
  DisjointClasses(semtask:ApplicationDocument ↩
      semtask:FormDocument)
  DisjointClasses(semtask:FormDocument ↩
      semtask:URLBasedDocument)

  SubPropertyOf(semtask:hasName ↩
      semtask:semtaskDatatypeProperty)
  SubPropertyOf(semtask:programParameters ↩
      semtask:semtaskDatatypeProperty)
  SubPropertyOf(semtask:isInForm ↩
      semtask:semtaskObjectProperty)
  SubPropertyOf(semtask:hasField ↩
      semtask:semtaskObjectProperty)
  SubPropertyOf(semtask:isBeginningEvent ↩
      semtask:semtaskDatatypeProperty)
  SubPropertyOf(semtask:url semtask:semtaskDatatypeProperty↩
      )

)
```

A.3 The Data Ontology

Listing A.3: *The Data Ontology in OWL-AS*

```
Namespace(rdf   = <http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Namespace(owl   = <http://www.w3.org/2002/07/owl#>)
Namespace(xsd   = <http://www.w3.org/2001/XMLSchema#>)
Namespace(rdfs  = <http://www.w3.org/2000/01/rdf-schema#>)
Namespace(data  = <http://www.ifi.uio.no/fredaleks/semtask/data#>)

Ontology( <http://www.ifi.uio.no/fredaleks/semtask/data>

 ObjectProperty(data:artifactBelongingTo Functional
  inverseOf(data:hasArtifact)
  domain(data:TopsideArtifact)
  range(data:Platform))
 ObjectProperty(data:dataObjectProperty)
 ObjectProperty(data:hasArtifact InverseFunctional
  inverseOf(data:artifactBelongingTo)
  domain(data:Platform)
  range(data:TopsideArtifact))
 ObjectProperty(data:hasSensor InverseFunctional
  inverseOf(data:isLocatedOn)
  domain(unionOf(data:Well data:TopsideArtifact))
  range(data:Sensor))
 ObjectProperty(data:hasWell InverseFunctional
  inverseOf(data:wellBelongingTo)
  domain(data:Platform)
  range(data:Well))
 ObjectProperty(data:isLocatedOn Functional
  inverseOf(data:hasSensor)
  domain(data:Sensor)
  range(unionOf(data:Well data:TopsideArtifact)))
 ObjectProperty(data:relevantArtifact Transitive Symmetric
  inverseOf(data:relevantArtifact)
  domain(data:TopsideArtifact)
  range(data:TopsideArtifact))
 ObjectProperty(data:relevantData
  domain(data:DataElement)
  range(data:DataElement))
 ObjectProperty(data:relevantSensor Symmetric
  inverseOf(data:relevantSensor)
  domain(data:Sensor)
  range(data:Sensor))
 ObjectProperty(data:relevantWell Transitive Symmetric
  inverseOf(data:relevantWell)
  domain(data:Well)
```

```
 range(data:Well))
ObjectProperty(data:wellBelongingTo Functional
 inverseOf(data:hasWell)
 domain(data:Well)
 range(data:Platform))
ObjectProperty(rdf:rest)
ObjectProperty(owl:oneOf)

DatatypeProperty(data:dataDatatypeProperty)
DatatypeProperty(data:hasDescription Functional
 domain(data:DataElement)
 range(xsd:string))
DatatypeProperty(data:hasPDPURL
 domain(data:DataElement)
 range(xsd:string))
DatatypeProperty(rdf:first)

Class(data:ContinuousSensor partial
 data:Sensor)
Class(data:DataElement partial
 owl:Thing
 restriction(data:hasPDPURL minCardinality(1)))
Class(data:DiscreteSensor partial
 data:Sensor)
Class(data:FlowSensor partial
 data:ContinuousSensor)
Class(data:InjectionWell partial
 data:Well)
Class(data:Platform partial
 data:DataElement)
Class(data:PressureSensor partial
 data:DiscreteSensor)
Class(data:ProductionWell partial
 data:Well)
Class(data:Sensor partial
 data:DataElement
 restriction(data:isLocatedOn cardinality(1)))
Class(data:TempSensor partial
 data:DiscreteSensor)
Class(data:TopsideArtifact partial
 data:DataElement)
Class(data:VolumeSensor partial
 data:DiscreteSensor)
Class(data:Well complete
 unionOf(data:InjectionWell data:ProductionWell))
Class(data:Well partial
 data:DataElement)
Class(owl:Thing partial)
```

```
Datatype(xsd:string)

DisjointClasses(data:InjectionWell data:ProductionWell)
DisjointClasses(data:PressureSensor data:VolumeSensor)
DisjointClasses(data:Platform data:TopsideArtifact)
DisjointClasses(data:Platform data:Sensor)
DisjointClasses(data:Platform data:Well)
DisjointClasses(data:DiscreteSensor data:ContinuousSensor
    )
DisjointClasses(data:TempSensor data:PressureSensor)
DisjointClasses(data:Sensor data:TopsideArtifact)
DisjointClasses(data:Well data:TopsideArtifact)
DisjointClasses(data:TempSensor data:VolumeSensor)
DisjointClasses(data:Well data:Sensor)

SubPropertyOf(data:hasDescription
    data:dataDatatypeProperty)
SubPropertyOf(data:relevantSensor data:relevantData)
SubPropertyOf(data:wellBelongingTo
    data:dataObjectProperty)
SubPropertyOf(data:relevantData data:dataObjectProperty)
SubPropertyOf(data:isLocatedOn data:dataObjectProperty)
SubPropertyOf(data:relevantWell data:relevantData)
SubPropertyOf(data:artifactBelongingTo
    data:dataObjectProperty)
SubPropertyOf(data:hasPDPURL data:dataDatatypeProperty)
SubPropertyOf(data:hasWell data:dataObjectProperty)
SubPropertyOf(data:hasSensor data:dataObjectProperty)
SubPropertyOf(data:hasArtifact data:dataObjectProperty)
SubPropertyOf(data:relevantArtifact data:relevantData)

)
```

A.4 The Execution Ontology

Listing A.4: *The Execution Ontology in OWL-AS*

```
Namespace(rdf          = <http://www.w3.org/1999/02/22-
    rdf-syntax-ns#>)
Namespace(owl          = <http://www.w3.org/2002/07/owl#>
    )
Namespace(xsd          = <http://www.w3.org/2001/
    XMLSchema#>)
Namespace(rdfs         = <http://www.w3.org/2000/01/rdf-
    schema#>)
Namespace(execution    = <http://www.ifi.uio.no/fredaleks
    /semtask/execution#>)
Namespace(bpmn         = <http://www.ifi.uio.no/fredaleks
    /semtask/bpmn#>)
Namespace(data         = <http://www.ifi.uio.no/fredaleks
    /semtask/data#>)
Namespace(semtask      = <http://www.ifi.uio.no/fredaleks
    /semtask/semtask#>)

Ontology( <http://www.ifi.uio.no/fredaleks/semtask/
    execution>

 Annotation( owl:imports <http://www.ifi.uio.no/fredaleks/
    semtask/data>)
 Annotation( owl:imports <http://www.ifi.uio.no/fredaleks/
    semtask/semtask>)


 ObjectProperty(execution:belongsToFilledForm Functional
  inverseOf(execution:hasFilledField)
  domain(execution:FilledField)
  range(execution:FilledForm))
 ObjectProperty(execution:decisionBelongsTo Functional
  inverseOf(execution:hasDecision)
  domain(execution:Decision)
  range(bpmn:Flow))
 ObjectProperty(execution:executionObjectProperty)
 ObjectProperty(execution:hasDecision InverseFunctional
  inverseOf(execution:decisionBelongsTo)
  domain(bpmn:Flow)
  range(execution:Decision))
 ObjectProperty(execution:hasDocument
  domain(execution:Decision)
  range(semtask:Document))
 ObjectProperty(execution:hasFilledField InverseFunctional
  inverseOf(execution:belongsToFilledForm)
  domain(execution:FilledForm)
```

```
  range(execution:FilledField))
ObjectProperty(execution:hasFilledForm InverseFunctional
  inverseOf(execution:usedInDecision)
  domain(execution:Decision)
  range(execution:FilledForm))
ObjectProperty(execution:hasReason InverseFunctional
  domain(execution:Task)
  range(unionOf(execution:Decision execution:Trigger)))
ObjectProperty(execution:hasTask InverseFunctional
  inverseOf(execution:taskBelongsTo)
  domain(bpmn:Flow)
  range(execution:Task))
ObjectProperty(execution:hasTrigger InverseFunctional
  inverseOf(execution:triggerBelongsTo)
  domain(bpmn:Event)
  range(execution:Trigger))
ObjectProperty(execution:isOfFormType Functional
  domain(execution:FilledForm)
  range(semtask:FormDocument))
ObjectProperty(execution:isResultOf Functional
    InverseFunctional
  inverseOf(execution:resultsIn)
  domain(execution:Decision)
  range(execution:Task))
ObjectProperty(execution:resultsIn Functional
    InverseFunctional
  inverseOf(execution:isResultOf)
  domain(execution:Task)
  range(execution:Decision))
ObjectProperty(execution:taskBelongsTo Functional
  inverseOf(execution:hasTask)
  domain(execution:Task)
  range(bpmn:Flow))
ObjectProperty(execution:triggerBelongsTo Functional
  inverseOf(execution:hasTrigger)
  domain(execution:Trigger)
  range(bpmn:Event))
ObjectProperty(execution:usedInDecision
  inverseOf(execution:hasFilledForm)
  domain(execution:FilledForm)
  range(execution:Decision))
ObjectProperty(execution:usesData
  domain(unionOf(execution:Decision execution:Task))
  range(datexecution:DataElement))

DatatypeProperty(execution:createdTime Functional
  domain(execution:Task)
  range(xssemtask:dateTime))
DatatypeProperty(execution:description Functional
```

```
  domain(unionOf(execution:Decision execution:Task))
  range(xssemtask:string))
 DatatypeProperty(execution:executionDatatypeProperty)
 DatatypeProperty(execution:triggerDescription Functional
  domain(execution:Trigger)
  range(xssemtask:string))

 Class(bpmn:Event partial)
 Class(bpmn:Flow partial)
 Class(datexecution:DataElement partial)
 Class(execution:Decision partial
  execution:ExecutionElement)
 Class(execution:ExecutionElement partial)
 Class(execution:FilledField partial
  execution:ExecutionElement)
 Class(execution:FilledForm partial
  execution:ExecutionElement)
 Class(execution:Task partial
  execution:ExecutionElement)
 Class(execution:Trigger partial
  execution:ExecutionElement
  restriction(execution:triggerBelongsTo cardinality(1)))
 Class(execution:Trigger partial
  annotation(rdfs:comment "Class representing a triggering
      of an event"^^http://www.w3.org/2001/XMLSchema#
     string)
)
 Class(semtask:Document partial)
 Class(semtask:FormDocument partial)

 AnnotationProperty(rdfs:comment)

 Datatype(xssemtask:string)
 Datatype(xssemtask:dateTime)

 DisjointClasses(execution:FilledForm
    execution:FilledField)
 DisjointClasses(execution:FilledForm execution:Task)
 DisjointClasses(execution:Decision execution:FilledField)
 DisjointClasses(execution:Decision execution:Task)
 DisjointClasses(execution:FilledForm execution:Trigger)
 DisjointClasses(execution:Task execution:FilledField)
 DisjointClasses(execution:Decision execution:Trigger)
 DisjointClasses(execution:FilledField execution:Trigger)
 DisjointClasses(execution:Task execution:Trigger)
 DisjointClasses(execution:Decision execution:FilledForm)

 SubPropertyOf(execution:triggerBelongsTo
    execution:executionObjectProperty)
```

```
SubPropertyOf(execution:resultsIn ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:hasDecision ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:hasFilledForm ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:hasTask ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:belongsToFilledForm ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:hasDocument ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:hasReason ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:isResultOf ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:decisionBelongsTo ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:usesData ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:usedInDecision ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:hasTrigger ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:isOfFormType ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:createdTime ↩
    execution:executionDatatypeProperty)
SubPropertyOf(execution:taskBelongsTo ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:triggerDescription ↩
    execution:executionDatatypeProperty)
SubPropertyOf(execution:hasFilledField ↩
    execution:executionObjectProperty)
SubPropertyOf(execution:description ↩
    execution:executionDatatypeProperty)

)
```

Appendix B

The RDF Documents

B.1 The Daily Production Optimization

Listing B.1: *Daily Production Optimization in Notation3*

```
#Converted with http://www.mindswap.org/2002/rdfconvert/

@prefix : <#> .
@prefix bpmn: <http://www.ifi.uio.no/fredaleks/semtask/
    bpmn#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix protege: <http://protege.stanford.edu/plugins/owl/
    protege#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix semtask: <http://www.ifi.uio.no/fredaleks/semtask/
    semtask#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix xsp: <http://www.owl-ontologies.com/2005/08/07/xsp
    .owl#> .

:actionList a semtask:FormDocument;
  bpmn:hasConnectingFrom :
      actionListToExecutionOfUpdatedPlan;
  bpmn:hasConnectingTo :
      productonOptimzationMeetingToActionList;
  bpmn:hasName "Action List"^^xsd:string .

:actionListToExecutionOfUpdatedPlan a bpmn:Association;
  bpmn:connectedFrom :actionList;
  bpmn:connectedTo :executionOfUpdatedPlan .
```

```
:approvedToStoringTestInProdTekDatabase a bpmn:
    ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionApproved;
  bpmn:connectedTo :storingTestInProdTekDatabase;
  bpmn:hasCondition "Yes"^^xsd:string .

:approvedToWellTesting a bpmn:ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionApproved;
  bpmn:connectedTo :wellTesting;
  bpmn:hasCondition "No"^^xsd:string .

:bestPracticeDocumentForWellAnalysis a semtask:
    URLBasedDocument;
  bpmn:hasConnectingFrom :
      bestPracticeDocumentForWellAnalysisToWellAnalysis;
  bpmn:hasName "Best Practice Document for Well Analysis"
      ^^xsd:string;
  semtask:url "http://en.wikipedia.org/wiki/Best_Practice"
      ^^xsd:string .

:bestPracticeDocumentForWellAnalysisToWellAnalysis a bpmn:
    Association;
  bpmn:connectedFrom :bestPracticeDocumentForWellAnalysis;
  bpmn:connectedTo :wellAnalysis .

:checklistForSystemOptimizing a semtask:URLBasedDocument;
  bpmn:hasConnectingFrom :
      checklistForSystemOptimizingToExecutionOfOptimization
      ;
  bpmn:hasName "Checklist for System Optimizing"^^xsd:
      string;
  semtask:url "http://en.wikipedia.org/wiki/Checklist"^^
      xsd:string .

:checklistForSystemOptimizingToExecutionOfOptimization a
    bpmn:Association;
  bpmn:connectedFrom :checklistForSystemOptimizing;
  bpmn:connectedTo :executionOfOptimization .

:continuousProductionSupervising a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :
      continuousProductionSupervisingToFork,
 :continuousProductionSupervisingToOptmizationPotential;
  bpmn:hasConnectingTo :
      forkContinuousProductionSupervisingToContinuous-
      ProductionSupervising,
 :productionOptimizationMeetingTo-
     ContinuousProductionSupervising;
```

```
  bpmn:hasName "Continuous production supervising"^^xsd:string;
  bpmn:isInLane :executing_Technical_manager_operation .

:continuousProductionSupervisingToFork a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :continuousProductionSupervising;
  bpmn:connectedTo :forkContinuousProductionSupervisingAnd-PropositionForInitiative .

:continuousProductionSupervisingToOptmizationPotential a bpmn:Association;
  bpmn:connectedFrom :continuousProductionSupervising;
  bpmn:connectedTo :optimizationPotential .

:creatingNetworkModel a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :creatingNetworkModelToDecision,
 :creatingNetworkModelToNetworkModel;
  bpmn:hasConnectingTo :processCapacitiesToCreatingNetworkModel,
 :wellDataToCreatingNetworkModel;
  bpmn:hasName "Creating network model"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:creatingNetworkModelToDecision a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :creatingNetworkModel;
  bpmn:connectedTo :decisionModelWithTestOrProductionData .

:creatingNetworkModelToNetworkModel a bpmn:Association;
  bpmn:connectedFrom :creatingNetworkModel;
  bpmn:connectedTo :networkModel .

:decisionApproved a semtask:DecisionGateway;
  bpmn:hasConnectingFrom :approvedToStoringTestInProdTekDatabase,
 :approvedToWellTesting;
  bpmn:hasConnectingTo :wellTestingToApproved;
  bpmn:isInLane :executing_Production_engineer_onshore .

:decisionImplementInitiative a semtask:DecisionGateway;
  bpmn:hasConnectingFrom :implementInitiativeToExecutionOfUpdatedPlan,
 :implementInitiativeToOptimalProduction;
  bpmn:hasConnectingTo :propositionForInitiativeToDecision;
  bpmn:isInLane :organizer_Production_engineer_onshore .
```

```
:decisionModelOK a semtask:DecisionGateway;
  bpmn:hasConnectingFrom :decisionToEvaluationOfInputData,
  :decisionToExecutionOfOptimzation;
  bpmn:hasConnectingTo :
      validatingModelWithProductionDataToDecision,
  :validatingModelWithTestDataToDecision;
  bpmn:isInLane :executing_Production_engineer_onshore .

:decisionModelWithTestOrProductionData a semtask:
    DecisionGateway;
  bpmn:hasConnectingFrom :
      decisionToValidatingModelWithProductionData,
  :decisionToValidatingModelWithTestData;
  bpmn:hasConnectingTo :creatingNetworkModelToDecision;
  bpmn:isInLane :executing_Production_engineer_onshore .

:decisionMoreThan180DaysSinceLastWellExamination a semtask
    :DecisionGateway;
  bpmn:hasConnectingFrom :decisionToWellExamination,
  :decisionToWellInitativeSuggestion;
  bpmn:hasConnectingTo :wellAnalysisToDecision;
  bpmn:hasName "More than 180 days since last well
      examiniation"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:decisionNeedForWellTest a semtask:DecisionGateway;
  bpmn:hasConnectingFrom :needForWellTestToWellAnalysis,
  :needForWellTestToWellTesting;
  bpmn:hasConnectingTo :trendingOfWellDataToDecision;
  bpmn:isInLane :executing_Production_engineer_onshore .

:decisionToEvaluationOfInputData a bpmn:
    ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionModelOK;
  bpmn:connectedTo :evaluationOfInputData;
  bpmn:hasCondition "No"^^xsd:string .

:decisionToExecutionOfOptimzation a bpmn:
    ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionModelOK;
  bpmn:connectedTo :executionOfOptimization;
  bpmn:hasCondition "Yes"^^xsd:string .

:decisionToValidatingModelWithProductionData a bpmn:
    ConditionalSequenceFlow;
  bpmn:connectedFrom :
      decisionModelWithTestOrProductionData;
  bpmn:connectedTo :validatingModelWithProductionData;
  bpmn:hasCondition "Use production data"^^xsd:string .
```

```
:decisionToValidatingModelWithTestData a bpmn:ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionModelWithTestOrProductionData;
  bpmn:connectedTo :validatingModelWithTestData;
  bpmn:hasCondition "Use test data"^^xsd:string .

:decisionToWellExamination a bpmn:ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionMoreThan180DaysSinceLastWellExamination;
  bpmn:connectedTo :wellExamination;
  bpmn:hasCondition "Yes"^^xsd:string .

:decisionToWellInitativeSuggestion a bpmn:ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionMoreThan180DaysSinceLastWellExamination;
  bpmn:connectedTo :wellInitiativeSuggestion;
  bpmn:hasCondition "No"^^xsd:string .

:estimatedRisk a semtask:Field;
  semtask:hasName "Estimated risk (1-5)"^^xsd:string;
  semtask:isInForm :optimizationPotential .

:estimatedValue a semtask:Field;
  semtask:hasName "Estimated Value in NOK"^^xsd:string;
  semtask:isInForm :optimizationPotential .

:evaluateSubseaWellInitiative bpmn:hasConnectingTo :followUpToEvaluateSubseaWellInitiative;
  bpmn:hasName "Evaluate Subsea Well Initiative"^^xsd:string;
  bpmn:hasStartEvent :needForWellAnalysis;
  bpmn:isInLane :executing_Production_engineer_onshore .

:evaluationOfInputData a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :evaluationOfInputDataToWellData;
  bpmn:hasConnectingTo :decisionToEvaluationOfInputData;
  bpmn:hasName "Evaluation of Input Data"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:evaluationOfInputDataToWellData a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :evaluationOfInputData;
  bpmn:connectedTo :wellData .

:executing_Process_Operation_engineer_onshore a bpmn:Lane;
```

```
    bpmn:hasName "Executing (Process/Operation engineer, ↩
        onshore)"^^xsd:string;
    bpmn:hasObject :↩
        forkTopsideProductionOptimizationAndPropositionFor-↩
        Initative,
  :topsideProductionOptimization;
    bpmn:isInPool :production_Optimization .

:executing_Production_engineer_onshore a bpmn:Lane;
    bpmn:hasName "Executing (Production engineer onshore)"^^↩
        xsd:string;
    bpmn:hasObject :creatingNetworkModel,
  :decisionApproved,
  :decisionModelOK,
  :decisionModelWithTestOrProductionData,
  :decisionMoreThan180DaysSinceLastWellExamination,
  :decisionNeedForWellTest,
  :evaluateSubseaWellInitiative,
  :evaluationOfInputData,
  :executionOfOptimization,
  :followUp,
  :forkProductionFollowUpAndPropositionForInitiative,
  :needForNetworkOptimizing,
  :needForWellAnalysis,
  :optimizedModelCalculations,
  :processCapacities,
  :productionFollowUp,
  :startProductionFollowUp,
  :storingTestInProdTekDatabase,
  :systemOptimization,
  :trendingOfWellData,
  :updatingAllModelsThatUseTestData,
  :validatingModelWithProductionData,
  :validatingModelWithTestData,
  :wellAnalysis,
  :wellData,
  :wellExamination,
  :wellInitiativeSuggestion,
  :wellTesting;
    bpmn:isInPool :production_Optimization .

:executing_Technical_manager_operation a bpmn:Lane;
    bpmn:hasName "Executing (Tehnical manager, operation)"^^↩
        xsd:string;
    bpmn:hasObject :continuousProductionSupervising,
  :executionOfUpdatedPlan,
  :forkContinuousProductionSupervisingAnd-↩
      PropositionForInitiative;
    bpmn:isInPool :production_Optimization .
```

```
:executionOfOptimization a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :>
      executionOfOptimzationToOptimizedModelCalculations;
  bpmn:hasConnectingTo :>
      checklistForSystemOptimizingToExecutionOfOptimization >
      ,
 :decisionToExecutionOfOptimzation;
  bpmn:hasName "semtask of Optimization"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:executionOfOptimzationToOptimizedModelCalculations a bpmn>
    :NormalSequenceFlow;
  bpmn:connectedFrom :executionOfOptimization;
  bpmn:connectedTo :optimizedModelCalculations .

:executionOfUpdatedPlan a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :>
      executionOfUpdatedPlanToPropsitionForInitiative;
  bpmn:hasConnectingTo :actionListToExecutionOfUpdatedPlan >
      ,
 :implementInitiativeToExecutionOfUpdatedPlan,
 :newInitiativeToExecutionOfUpdatedPlan,
 :productionOptimizationMeetingToExecutionOfUpdatedPlan;
  bpmn:hasName "semtask of Updated Plan"^^xsd:string;
  bpmn:isInLane :executing_Technical_manager_operation .

:executionOfUpdatedPlanToPropsitionForInitiative a bpmn:>
    NormalSequenceFlow;
  bpmn:connectedFrom :executionOfUpdatedPlan;
  bpmn:connectedTo :propositionForInitiative .

:followUp a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :>
      followUpToEvaluateSubseaWellInitiative,
 :followUpToSystemOptimization;
  bpmn:hasName "Follow-up"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:followUpToEvaluateSubseaWellInitiative a bpmn:>
    NormalSequenceFlow;
  bpmn:connectedFrom :followUp;
  bpmn:connectedTo :evaluateSubseaWellInitiative .

:followUpToSystemOptimization a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :followUp;
```

```
  bpmn:connectedTo :systemOptimization .

:forkContinuousProductionSupervisingAndProposition-
    ForInitiative a bpmn:ParalellForkGateway;
  bpmn:hasConnectingFrom :
      forkContinuousProductionSupervisingToContinuous-
      ProductionSupervising,
 :forkContinuousProductionSupervisingToPropsition-
     ForInitiative;
  bpmn:hasConnectingTo :
      continuousProductionSupervisingToFork;
  bpmn:isInLane :executing_Technical_manager_operation .

:forkContinuousProductionSupervisingToContinuous-
    ProductionSupervising a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :
      forkContinuousProductionSupervisingAndProposition-
      ForInitiative;
  bpmn:connectedTo :continuousProductionSupervising .

:forkContinuousProductionSupervisingToPropsition-
    ForInitiative a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :
      forkContinuousProductionSupervisingAnd-
      PropositionForInitiative;
  bpmn:connectedTo :propositionForInitiative .

:forkProductionFollowUpAndProposition-ForInitiative a bpmn
    :ParalellForkGateway;
  bpmn:hasConnectingFrom :
      forkProductionFollowUpToProductionFollowUp,
 :forkProductionFollowUpToPropsition-ForInitiative;
  bpmn:hasConnectingTo :productionFollowUpToFork;
  bpmn:isInLane :executing_Production_engineer_onshore .

:forkProductionFollowUpToProductionFollowUp a bpmn:
    NormalSequenceFlow;
  bpmn:connectedFrom :
      forkProductionFollowUpAndPropositionForInitiative;
  bpmn:connectedTo :productionFollowUp .

:forkProductionFollowUpToPropsitionForInitiative a bpmn:
    NormalSequenceFlow;
  bpmn:connectedFrom :
      forkProductionFollowUpAndPropositionForInitiative;
  bpmn:connectedTo :propositionForInitiative .

:forkTopsideProductionOptimizationAnd-
    PropositionForInitative a bpmn:ParalellForkGateway;
```

```
  bpmn:hasConnectingFrom :>
      forkTopsideProductionOptimizationTo->
      PropsitionForInitiative;
  bpmn:hasConnectingTo :>
      topsideProductionOptimizationToFork;
  bpmn:isInLane :>
      executing_Process_Operation_engineer_onshore .

:forkTopsideProductionOptimizationTo->
    PropsitionForInitiative a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :forkTopsideProductionOptimizationAnd >
      -ProposionForInitative;
  bpmn:connectedTo :propositionForInitiative .

:forkValidatingModelWithProductionDataAnd->
    ValidatingModelWithTestData a bpmn:ParalellForkGateway >
    .

:implementInitiativeToExecutionOfUpdatedPlan a bpmn:>
    ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionImplementInitiative;
  bpmn:connectedTo :executionOfUpdatedPlan;
  bpmn:hasCondition "Yes"^^xsd:string .

:implementInitiativeToOptimalProduction a bpmn:>
    ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionImplementInitiative;
  bpmn:connectedTo :optimalProduction;
  bpmn:hasCondition "No"^^xsd:string .

:longDescription a semtask:Field;
  semtask:hasName "Long description"^^xsd:string;
  semtask:isInForm :optimizationPotential .

:needForAnalysisToTrendingOfWellData a bpmn:>
    NormalSequenceFlow;
  bpmn:connectedFrom :needForWellAnalysis;
  bpmn:connectedTo :trendingOfWellData .

:needForNetworkOptimizing a bpmn:StartEvent;
  bpmn:hasConnectingFrom :>
      needForNetworkOptimizingToProcessCapacities,
 :needForNetworkOptimizingToWellData;
  bpmn:hasName "Need for Network Optimizing"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:needForNetworkOptimizingToProcessCapacities a bpmn:>
    NormalSequenceFlow;
  bpmn:connectedFrom :needForNetworkOptimizing;
```

```
  bpmn:connectedTo :processCapacities .

:needForNetworkOptimizingToWellData a bpmn:
    NormalSequenceFlow;
  bpmn:connectedFrom :needForNetworkOptimizing;
  bpmn:connectedTo :wellData .

:needForWellAnalysis a bpmn:StartEvent;
  bpmn:hasConnectingFrom :
      needForAnalysisToTrendingOfWellData;
  bpmn:hasName "Need for Well Analysis"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:needForWellTestToWellAnalysis a bpmn:
    ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionNeedForWellTest;
  bpmn:connectedTo :wellAnalysis;
  bpmn:hasCondition "No"^^xsd:string .

:needForWellTestToWellTesting a bpmn:
    ConditionalSequenceFlow;
  bpmn:connectedFrom :decisionNeedForWellTest;
  bpmn:connectedTo :wellTesting;
  bpmn:hasCondition "Yes"^^xsd:string .

:networkModel a bpmn:DataObject;
  bpmn:hasConnectingFrom :
      networkModelToValidatingModelWithProductionData,
 :networkModelToValidatingModelWithTestData;
  bpmn:hasConnectingTo :creatingNetworkModelToNetworkModel
      ;
  bpmn:hasName "Network Model"^^xsd:string .

:networkModelToValidatingModelWithProductionData a bpmn:
    Association;
  bpmn:connectedFrom :networkModel;
  bpmn:connectedTo :validatingModelWithProductionData .

:networkModelToValidatingModelWithTestData a bpmn:
    Association;
  bpmn:connectedFrom :networkModel;
  bpmn:connectedTo :updatingAllModelsThatUseTestData .

:newInitiative a semtask:FormDocument;
  bpmn:hasConnectingFrom :
      newInitiativeToExecutionOfUpdatedPlan;
  bpmn:hasConnectingTo :
      propositionForInitiativeToNewInitiative;
  bpmn:hasName "New Initiative"^^xsd:string .
```

```
:newInitiativeToExecutionOfUpdatedPlan a bpmn:Association;
  bpmn:connectedFrom :newInitiative;
  bpmn:connectedTo :executionOfUpdatedPlan .

:optimalProduction a bpmn:TerminateEndEvent;
  bpmn:hasConnectingTo :
      implementInitiativeToOptimalProduction;
  bpmn:hasName "Optimal Production"^^xsd:string;
  bpmn:isInLane :organizer_Production_engineer_onshore .

:optimizationPotential a semtask:FormDocument;
  bpmn:hasConnectingFrom :
      optmizationPotentialToPropositionForInitiative;
  bpmn:hasConnectingTo :
      continuousProductionSupervisingToOptmizationPotential
      ,
 :productonFollowUpToOptmizationPotential;
  bpmn:hasName "Optimization Potential"^^xsd:string;
  semtask:hasField :estimatedRisk,
 :estimatedValue,
 :longDescription,
 :shortDescription .

:optimizedModelCalculations a bpmn:TerminateEndEvent;
  bpmn:hasConnectingTo :
      executionOfOptimzationToOptimizedModelCalculations;
  bpmn:isInLane :executing_Production_engineer_onshore .

:optmizationPotentialToPropositionForInitiative a bpmn:
    Association;
  bpmn:connectedFrom :optimizationPotential;
  bpmn:connectedTo :propositionForInitiative .

:organizer_Production_engineer_onshore a bpmn:Lane;
  bpmn:hasName "Organizaer (production engineer, noshore)"
      ^^xsd:string;
  bpmn:hasObject :decisionImplementInitiative,
 :optimalProduction,
 :productionOptimizationMeeting,
 :propositionForInitiative,
 :weekday_9am;
  bpmn:isInPool :production_Optimization .

:processCapacities a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :
      processCapacitiesToCreatingNetworkModel;
```

```
  bpmn:hasConnectingTo :>
      needForNetworkOptimizingToProcessCapacities;
  bpmn:hasName "Process Capacities"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:processCapacitiesToCreatingNetworkModel a bpmn:>
    NormalSequenceFlow;
  bpmn:connectedFrom :processCapacities;
  bpmn:connectedTo :creatingNetworkModel .

:productionFollowUp a bpmn:LoopedSubProcess;
  bpmn:hasConnectingFrom :productionFollowUpToFork,
 :productonFollowUpToOptmizationPotential;
  bpmn:hasConnectingTo :>
      forkProductionFollowUpToProductionFollowUp;
  bpmn:hasName "Production Follow-up"^^xsd:string;
  bpmn:hasStartEvent :startProductionFollowUp;
  bpmn:isInLane :executing_Production_engineer_onshore .

:productionFollowUpToFork a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :productionFollowUp;
  bpmn:connectedTo :>
      forkProductionFollowUpAndPropositionForInitiative .

:productionIncreasingInitiativeReport a semtask:>
    FormDocument;
  bpmn:hasConnectingFrom :>
      productionIncreasingInitiativeReportTo->
      PropositionForInitiative;
  bpmn:hasConnectingTo :topsdeProductionOptimizationTo->
      ProductionIncreasingInitiativeReport;
  bpmn:hasName "Production Increasing Initiative Report"^^>
      xsd:string .

:productionIncreasingInitiativeReportTo->
    PropositionForInitiative a bpmn:Association;
  bpmn:connectedFrom :productionIncreasingInitiativeReport>
      ;
  bpmn:connectedTo :propositionForInitiative .

:productionOptimizationMeeting a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :productionOptimizationMeetingTo->
      ContinuousProductionSupervising,
 :productionOptimizationMeetingToExecutionOfUpdatedPlan,
 :productionOptimizationMeetingToProductionFollowUp,
 :productionOptimizationMeetingToTopsideProduction->
     Optimization,
 :productonOptimzationMeetingToActionList;
```

```
  bpmn:hasConnectingTo :
      weekday9amToProductionOptimizationMeeting ,
 :weeklyPlanToProductonOptimzationMeeting;
  bpmn:hasName "Production Optimization Meeting"^^xsd:
      string;
  bpmn:isInLane :organizer_Production_engineer_onshore .

:productionOptimizationMeetingToContinuous-
    ProductionSupervising a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :productionOptimizationMeeting;
  bpmn:connectedTo :continuousProductionSupervising .

:productionOptimizationMeetingToExecutionOf-UpdatedPlan a
    bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :productionOptimizationMeeting;
  bpmn:connectedTo :executionOfUpdatedPlan .

:productionOptimizationMeetingToProduction-FollowUp a bpmn
    :NormalSequenceFlow;
  bpmn:connectedFrom :productionOptimizationMeeting;
  bpmn:connectedTo :topsideProductionOptimization .

:productionOptimizationMeetingToTopsideProduction-
    Optimization a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :productionOptimizationMeeting;
  bpmn:connectedTo :topsideProductionOptimization .

:production_Optimization a bpmn:Pool;
  bpmn:hasLane :
      executing_Process_Operation_engineer_onshore ,
 :executing_Production_engineer_onshore ,
 :executing_Technical_manager_operation ,
 :organizer_Production_engineer_onshore;
  bpmn:hasName "Production Optimization"^^xsd:string .

:productonFollowUpToOptmizationPotential a bpmn:
    Association;
  bpmn:connectedFrom :productionFollowUp;
  bpmn:connectedTo :optimizationPotential .

:productonOptimzationMeetingToActionList a bpmn:
    Association;
  bpmn:connectedFrom :productionOptimizationMeeting;
  bpmn:connectedTo :actionList .

:propositionForInitiative a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :
      propositionForInitiativeToDecision ,
```

```
  :propositionForInitiativeToNewInitiative;
   bpmn:hasConnectingTo :>
       executionOfUpdatedPlanToPropsitionForInitiative,
  :forkContinuousProductionSupervisingTo-:>
      PropsitionForInitiative,
  :forkProductionFollowUpToPropsitionForInitiative,
  :forkTopsideProductionOptimizationTo-:>
      PropsitionForInitiative,
  :optmizationPotentialToPropositionForInitiative,
  :productionIncreasingInitiativeReportToProposition-:>
      ForInitiative;
   bpmn:hasName "Proposition for Initiative"^^xsd:string;
   bpmn:isInLane :organizer_Production_engineer_onshore .

:propositionForInitiativeToDecision a bpmn:>
    NormalSequenceFlow;
   bpmn:connectedFrom :propositionForInitiative;
   bpmn:connectedTo :decisionImplementInitiative .

:propositionForInitiativeToNewInitiative a bpmn:>
    Association;
   bpmn:connectedFrom :propositionForInitiative;
   bpmn:connectedTo :newInitiative .

:recommendedPracticeForWellTesting a semtask:>
    URLBasedDocument;
   bpmn:hasConnectingFrom :>
       recommendedPracticeForWellTestingToWellTesting;
   bpmn:hasName "Recommended Practice for Well Te"^^xsd:>
       string;
   semtask:url "http://en.wikipedia.org/wiki/Well_Testing"^^xsd:string .

:recommendedPracticeForWellTestingToWellTesting a bpmn:>
    Association;
   bpmn:connectedFrom :recommendedPracticeForWellTesting;
   bpmn:connectedTo :wellTesting .

:shortDescription a semtask:Field;
   semtask:hasName "Short description of the optimization >
       potential"^^xsd:string;
   semtask:isInForm :optimizationPotential .

:startProductionFollowUp a bpmn:StartEvent;
   bpmn:isInLane :executing_Production_engineer_onshore .

:startProductionFollowUpToFollowUp a bpmn:>
    NormalSequenceFlow;
   bpmn:connectedFrom :startProductionFollowUp;
```

```
  bpmn:connectedTo :followUp .

:storingTestInProdTekDatabase a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :
      storingTestInProdTekDatabaseToUpdating-
      AllModelsThatUseTestData;
  bpmn:hasConnectingTo :
      approvedToStoringTestInProdTekDatabase,
 :wellTestResultsToStoringTestInProdtekDatabase;
  bpmn:hasName "Storing Test in ProdTek Database"^^xsd:
      string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:storingTestInProdTekDatabaseToUpdating-
    AllModelsThatUseTestData a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :storingTestInProdTekDatabase;
  bpmn:connectedTo :updatingAllModelsThatUseTestData .

:systemOptimization a bpmn:SubProcess;
  bpmn:hasConnectingTo :followUpToSystemOptimization;
  bpmn:hasName "System Optimization"^^xsd:string;
  bpmn:hasStartEvent :needForNetworkOptimizing;
  bpmn:isInLane :executing_Production_engineer_onshore .

:topsdeProductionOptimizationToProductionIncreasing-
    InitiativeReport a bpmn:Association;
  bpmn:connectedFrom :topsideProductionOptimization;
  bpmn:connectedTo :productionIncreasingInitiativeReport .

:topsideProductionOptimization a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :
      topsdeProductionOptimizationToProductionIncreasing-
      InitiativeReport,
 :topsideProductionOptimizationToFork;
  bpmn:hasConnectingTo :
      productionOptimizationMeetingToProductionFollowUp,
 :productionOptimizationMeetingToTopside-
     ProductionOptimization;
  bpmn:hasName "Topside Production Optimization"^^xsd:
      string;
  bpmn:isInLane :
      executing_Process_Operation_engineer_onshore .

:topsideProductionOptimizationToFork a bpmn:
    NormalSequenceFlow;
  bpmn:connectedFrom :topsideProductionOptimization;
```

```
  bpmn:connectedTo :>
      forkTopsideProductionOptimizationAndProposition->
      ForInitative .

:trendingOfWellData a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :trendingOfWellDataToDecision;
  bpmn:hasConnectingTo :>
      needForAnalysisToTrendingOfWellData;
  bpmn:hasName "Trending of Well Data"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:trendingOfWellDataToDecision a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :trendingOfWellData;
  bpmn:connectedTo :decisionNeedForWellTest .

:updatingAllModelsThatUseTestData a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :>
      updatingAllModelsThatUseTestDataToWellAnalysis;
  bpmn:hasConnectingTo :>
      networkModelToValidatingModelWithTestData,
 :storingTestInProdTekDatabaseToUpdating->
     AllModelsThatUseTestData;
  bpmn:hasName "Updating all Models that Use Test Data"^^:>
      xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:updatingAllModelsThatUseTestDataToWellAnalysis a bpmn:>
    NormalSequenceFlow;
  bpmn:connectedFrom :updatingAllModelsThatUseTestData;
  bpmn:connectedTo :wellAnalysis .

:validatingModelWithProductionData a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :>
      validatingModelWithProductionDataToDecision;
  bpmn:hasConnectingTo :>
      decisionToValidatingModelWithProductionData,
 :networkModelToValidatingModelWithProductionData;
  bpmn:hasName "Validating Model with Production Data"^^:>
      xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:validatingModelWithProductionDataToDecision a bpmn:>
    NormalSequenceFlow;
  bpmn:connectedFrom :validatingModelWithProductionData;
  bpmn:connectedTo :decisionModelOK .
```

```
:validatingModelWithTestData a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :>
      validatingModelWithTestDataToDecision;
  bpmn:hasConnectingTo :>
      decisionToValidatingModelWithTestData;
  bpmn:hasName "Validating Model with Test Data"^^xsd:>
      string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:validatingModelWithTestDataToDecision a bpmn:>
    NormalSequenceFlow;
  bpmn:connectedFrom :validatingModelWithTestData;
  bpmn:connectedTo :decisionModelOK .

:weekday9amToProductionOptimizationMeeting a bpmn:>
    NormalSequenceFlow;
  bpmn:connectedFrom :weekday_9am;
  bpmn:connectedTo :productionOptimizationMeeting .

:weekday_9am a bpmn:TimerStartEvent;
  bpmn:hasConnectingFrom :>
      weekday9amToProductionOptimizationMeeting;
  bpmn:hasName "Weekday 9am"^^xsd:string;
  bpmn:isInLane :organizer_Production_engineer_onshore;
  semtask:isBeginningEvent "true"^^xsd:boolean .

:weeklyPlanToProductonOptimzationMeeting a bpmn:>
    Association;
  bpmn:connectedFrom :weekly_P_and_I_plan;
  bpmn:connectedTo :productionOptimizationMeeting .

:weekly_P_and_I_plan a semtask:URLBasedDocument;
  bpmn:hasConnectingFrom :>
      weeklyPlanToProductonOptimzationMeeting;
  bpmn:hasName "Weekly P&I-plan"^^xsd:string;
  semtask:url "http://en.wikipedia.org/wiki/Plan"^^xsd:>
      string .

:wellAnalysis a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :wellAnalysisToDecision;
  bpmn:hasConnectingTo :>
      bestPracticeDocumentForWellAnalysisToWellAnalysis,
 :needForWellTestToWellAnalysis,
 :updatingAllModelsThatUseTestDataToWellAnalysis;
  bpmn:hasName "Well Analysis"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .
```

```
:wellAnalysisToDecision a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :wellAnalysis;
  bpmn:connectedTo :
      decisionMoreThan180DaysSinceLastWellExamination .

:wellData a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :wellDataToCreatingNetworkModel;
  bpmn:hasConnectingTo :evaluationOfInputDataToWellData,
 :needForNetworkOptimizingToWellData;
  bpmn:hasName "Well Data"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:wellDataToCreatingNetworkModel a bpmn:NormalSequenceFlow;
  bpmn:connectedFrom :wellData;
  bpmn:connectedTo :creatingNetworkModel .

:wellExamination a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :
      wellExaminationToWellInitativeSuggestion;
  bpmn:hasConnectingTo :decisionToWellExamination;
  bpmn:hasName "Well Examination"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:wellExaminationToWellInitativeSuggestion a bpmn:
    NormalSequenceFlow;
  bpmn:connectedFrom :wellExamination;
  bpmn:connectedTo :wellInitiativeSuggestion .

:wellInitiativeSuggestion a bpmn:TerminateEndEvent;
  bpmn:hasConnectingTo :decisionToWellInitativeSuggestion,
 :wellExaminationToWellInitativeSuggestion;
  bpmn:hasName "Well Initiative Suggestion"^^xsd:string;
  bpmn:isInLane :executing_Production_engineer_onshore .

:wellTestResults a bpmn:DataObject;
  bpmn:hasConnectingFrom :
      wellTestResultsToStoringTestInProdtekDatabase;
  bpmn:hasName "Well Test Results"^^xsd:string .

:wellTestResultsToStoringTestInProdtekDatabase a bpmn:
    Association;
  bpmn:connectedFrom :wellTestResults;
  bpmn:connectedTo :storingTestInProdTekDatabase .

:wellTesting a bpmn:Task;
  bpmn:hasActivityTaskType bpmn:User;
  bpmn:hasConnectingFrom :wellTestingToApproved;
```

```
    bpmn:hasConnectingTo :approvedToWellTesting,
  :needForWellTestToWellTesting,
  :recommendedPracticeForWellTestingToWellTesting;
    bpmn:hasName "Well Testing"^^xsd:string;
    bpmn:isInLane :executing_Production_engineer_onshore .

:wellTestingToApproved a bpmn:NormalSequenceFlow;
    bpmn:connectedFrom :wellTesting;
    bpmn:connectedTo :decisionApproved .

bpmn:BPMNElement rdfs:comment ""^^xsd:string .
```

#ENDS

B.2 Data Sources

Listing B.2: *A sample of data sources in Notation3*

```
@prefix : <#> .
@prefix data: <http://www.ifi.uio.no/fredaleks/semtask/data#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix protege: <http://protege.stanford.edu/plugins/owl/protege#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix xsp: <http://www.owl-ontologies.com/2005/08/07/xsp.owl#> .

<> a owl:Ontology;
 owl:imports <http://www.ifi.uio.no/fredaleks/semtask/data> .
[  a owl:AllDifferent;
 owl:distinctMembers  (
    :temp1
    :temp2
    :temp3
    :temp4
    :temp5
    :temp6
    :temp7
    :temp8  ) ].

:flow1 a data:FlowSensor;
 data:hasPDPURL "http://pdp/sensors/flow1"@en;
 data:isLocatedOn :injectionWellA1;
 data:relevantSensor :flow5 .

:flow2 a data:FlowSensor;
 data:hasPDPURL "http://pdp/sensors/flow2"@en;
 data:isLocatedOn :productionWellB1;
 data:relevantSensor :flow5 .

:flow3 a data:FlowSensor;
 data:hasPDPURL "http://pdp/sensors/flow3"@en;
 data:isLocatedOn :productionWellB2;
 data:relevantSensor :flow5 .

:flow4 a data:FlowSensor;
 data:hasPDPURL "http://pdp/sensors/flow4"@en;
 data:isLocatedOn :productionWellB3;
```

```
 data:relevantSensor :flow5 .

:flow5 a data:FlowSensor;
 data:hasDescription "Flow  before liquid gets seperated"^^xsd:string;
 data:hasPDPURL "http://pdp/sensors/flow5"@en;
 data:isLocatedOn :separator1;
 data:relevantSensor :flow1,
    :flow2,
    :flow3,
    :flow4 .

:gasTank1 a data:TopsideArtifact;
 data:artifactBelongingTo :osebergEast;
 data:hasPDPURL "http://pdp/osebergEast/artifacts/gasTank1"@en;
 data:hasSensor :pressure2,
    :temp1;
 data:relevantArtifact :gasTank2,
    :oilTank1 .

:gasTank2 a data:TopsideArtifact;
 data:artifactBelongingTo :osebergEast;
 data:hasPDPURL "http://pdp/osebergEast/artifacts/gasTank2"@en;
 data:hasSensor :pressure4,
    :temp2;
 data:relevantArtifact :gasTank1 .

:injectionWellA1 a data:InjectionWell;
 data:hasPDPURL ""^^xsd:string;
 data:hasSensor :flow1,
    :pressure1,
    :temp3;
 data:relevantWell :productionWellB3;
 data:wellBelongingTo :osebergEast .

:oilTank1 a data:TopsideArtifact;
 data:artifactBelongingTo :osebergEast;
 data:hasPDPURL ""@en;
 data:hasSensor :temp4 .

:osebergEast a data:Platform;
 data:hasArtifact :gasTank1,
    :gasTank2,
    :oilTank1,
    :separator1;
 data:hasPDPURL "http://pdp/osebergEast/"@en;
 data:hasWell :injectionWellA1,
```

```
    :productionWellB1 ,
    :productionWellB2 ,
    :productionWellB3 .

:pressure1 a data:PressureSensor;
 data:hasPDPURL "http://pdp/sensors/pressure1"@en;
 data:isLocatedOn :injectionWellA1;
 data:relevantSensor :temp2 .

:pressure2 a data:PressureSensor;
 data:hasPDPURL "http://pdp/sensors/pressure2"@en;
 data:isLocatedOn :gasTank1;
 data:relevantSensor :temp1 .

:pressure3 a data:PressureSensor;
 data:hasPDPURL "http://pdp/sensors/pressure3"@en;
 data:isLocatedOn :productionWellB1 .

:pressure4 a data:PressureSensor;
 data:hasPDPURL "http://pdp/sensors/pressure4"@en;
 data:isLocatedOn :gasTank2 .

:productionWellB1 a data:ProductionWell;
 data:hasPDPURL ""^^xsd:string;
 data:hasSensor :flow2 ,
    :pressure3 ,
    :temp5;
 data:relevantWell :productionWellB2;
 data:wellBelongingTo :osebergEast .

:productionWellB2 a data:ProductionWell;
 data:hasPDPURL ""^^xsd:string;
 data:hasSensor :flow3 ,
    :temp6;
 data:relevantWell :productionWellB1;
 data:wellBelongingTo :osebergEast .

:productionWellB3 a data:ProductionWell;
 data:hasPDPURL ""^^xsd:string;
 data:hasSensor :flow4 ,
    :temp7;
 data:relevantWell :injectionWellA1;
 data:wellBelongingTo :osebergEast .

:separator1 a data:TopsideArtifact;
 data:artifactBelongingTo :osebergEast;
 data:hasPDPURL "http://pdp/osebergEast/artifacts/
    separator1"@en;
 data:hasSensor :flow5 ,
```

```
    :temp8 .

:temp1 data:hasPDPURL "http://pdp/sensors/temp1"@en;
 data:isLocatedOn :gasTank1;
 data:relevantSensor :pressure2 .

:temp2 a data:TempSensor;
 data:hasPDPURL "http://pdp/sensors/temp2"@en;
 data:isLocatedOn :gasTank2;
 data:relevantSensor :pressure1 .

:temp3 a data:TempSensor;
 data:hasPDPURL "http://pdp/sensors/temp3"@en;
 data:isLocatedOn :injectionWellA1 .

:temp4 a data:TempSensor;
 data:hasPDPURL "http://pdp/sensors/temp4"@en;
 data:isLocatedOn :oilTank1 .

:temp5 a data:TempSensor;
 data:hasPDPURL "http://pdp/sensors/temp5"@en;
 data:isLocatedOn :productionWellB1 .

:temp6 a data:TempSensor;
 data:hasPDPURL "http://pdp/sensors/temp6"@en;
 data:isLocatedOn :productionWellB2 .

:temp7 a data:TempSensor;
 data:hasPDPURL "http://pdp/sensors/temp7"@en;
 data:isLocatedOn :productionWellB3 .

:temp8 a data:TempSensor;
 data:hasPDPURL "http://pdp/sensors/temp8"@en;
 data:isLocatedOn :separator1 .
```

#ENDS

Appendix C

Jena Rules

Listing C.1: *Two execution rules in Jena*

```
//prefixes
@prefix bpmn: <http://www.ifi.uio.no/fredaleks/semtask/
    bpmn#>
@prefix data: <http://www.ifi.uio.no/fredaleks/semtask/
    data#>
@prefix semtask: <http://www.ifi.uio.no/fredaleks/semtask/
    semtask#>
@prefix execution: <http://www.ifi.uio.no/fredaleks/
    semtask/execution#>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

//rule that processes of triggered events
[processTrigger:
// Specify the types of our variables
  (?sf          rdf:type              bpmn:SequenceFlow),
  (?activity    rdf:type              bpmn:Activity),
  (?startEvent  rdf:type              bpmn:StartEvent),
  (?trigger     rdf:type              execution:Trigger),

// Make sure that they are connected to each other
  (?startEvent  execution:hasTrigger  ?trigger),
  (?sf          bpmn:connectedFrom    ?startEvent),
  (?sf          bpmn:connectedTo      ?activity),

// This trigger has not triggered this activity before
  noValue(?trigger trigged ?activity),

// Create a new, blank, RDF-node.
  makeTemp(?task),

// Create a new RDF-node with the current time.
```

```
  now(?now),
   ->
// Print debug information
  print("RULE: processTrigger. Activity", ?activity, "
     Event:", ?startEvent),

// Make sure this is not executed any more for this event
  (?trigger trigged ?activity),

// Create the new Task and attach it to the activity and
    the trigger
  (?task rdf:type execution:Task),
  (?activity execution:hasTask ?task),
  (?task  execution:hasCause  ?trigger),

// Set timestamp for the trigger
  (?task  execution:created  ?now)
]

//rule that creates a task out of a decision
[processDecision:
  (?sf                rdf:type     bpmn:SequenceFlow),
  (?decisionActivity  rdf:type     bpmn:Activity),
  (?taskActiviy       rdf:type     bpmn:Activity),
  (?decision          rdf:type     execution:Decision),

  (?sf        bpmn:connectedFrom   ?decisionActivity),
  (?sf        bpmn:connectedTo     ?taskActivity),
  (?decision  execution:belongsTo  ?decisionActivity),

  noValue(?decision trigged ?taskActivity),
  makeTemp(?task),
  now(?now),
   ->
  print("RULE: processDecision. from activity",
     ?decisionActivity, " to activity:", ?taskActiviy),

  (?task          rdf:type           execution:Task),
  (?taskActivity  execution:hasTask  ?task),

  //make sure this is not executed any more for this event
  (?decision      trigged            ?taskActivity),

  //Create a new Task based on the Decision
  (?task          execution:created  ?now),
  (?task          execution:hasCause ?decision)
]
```